

Operating System

GCOS 8 Operating System

Programmer's Guide

**Bull NovaScale 9000 Series Assembly
Instructions**

GCOS 8

Operating System

GCOS 8 Operating System

Programmer's Guide

Bull NovaScale 9000 Series Assembly Instructions

GCOS 8

Subject: Assembly Instructions programmer's guide for the Bull NovaScale 9000 Series large-system computer systems.

Special instructions: This is the first version of 67 A2 RJ78.

Software supported: GCOS 8 System Release SR5.2 and later

Date: November 2003

Bull S.A.
CEDOC
Atelier de reprographie
357, Avenue Patton BP 20845
49008 ANGERS Cedex 01
FRANCE

Bull HN Information Systems Inc.
Publication Order Entry
FAX: (800) 611-6030
MA30/415
300 Concord Rd.
Billerica, MA 01821
U.S.A.

Copyright © Bull HN Information Systems Inc., 2003

All Rights Reserved

All trademarks, service marks, and company names are the property of their respective owners.

Suggestions and criticisms concerning the form, the content, and the presentation of this manual are invited. Documentation comments or changes can be reported using Software Technical Action Requests (STARs) via local account procedures.

BULL DISCLAIMS THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND MAKES NO EXPRESS WARRANTIES EXCEPT AS MAY BE STATED IN ITS WRITTEN AGREEMENT WITH AND FOR ITS CUSTOMER. BULL DOES NOT WARRANT THAT USE OF THE SOFTWARE PRODUCTS WILL BE UNINTERRUPTED OR THAT THE SOFTWARE PRODUCTS ARE ERROR-FREE. In no event is Bull liable to anyone for any indirect, special, or consequential damages.

The information and specifications in this document are subject to change without notice. Consult your marketing representative for product or service availability.

Preface

About this Manual

This manual contains information that enables the user to code programs in symbolic machine language which is then translated into binary machine instructions.

The manual is directed to users who are experienced in coding within the environment of a large-scale computer installation. Considerable knowledge and practical experience is required to use address modification with indirection, hardware indicators, fault interrupts and recovery routines, macro operations, pseudo-operations, and other features normally encountered in a large computer system with a flexible repertoire for instructions under the control of a master executive program. The reader should also be familiar with the two's-complement number system.

This manual includes processor capabilities, modes of operation, detailed descriptions of machine instructions, virtual memory addressing, paging, and the representation of data. Programmers who are responsible for analyzing conditions that cause system failures should find this manual especially useful.

For related information, see the *GCOS 8 GMAP Assembler User's Guide*, Order No. DH01.

Disclaimer for this Manual

Although the GCOS 8 system supports the execution of programs that employ the Extended Segment (ES) and Extended Instruction (EI) Segment modes described in this manual, the current release of GMAP does not support assembly of the instructions identified for use in ES/EI mode only. These instructions are available with GMAPV and other compilers in this release. (Refer to Section 7 for a complete list of ES/EI mode instructions.)

Some obsolete instructions and mnemonics from other platforms will cause Illegal Procedure (IPR) faults and others will result in unexpected code generation.

GCOS 8 Documentation

GCOS 8 documentation is distributed on the Bull CD-ROM product, CD-DOC II. Any documents that are updated after a CD-DOC version is distributed are available in Portable Document Format (PDF) from the Bull Internet CD-DOC site at:

http://www.enterprise.bull.com/cd_doc/

Reporting Documentation Discrepancies

Report documentation discrepancies via the site's normal problem-reporting process.

Document Corrections

Document corrections made after a CD-DOC version is distributed can be accessed via a link on the Bull Internet CD-DOC site.

No new document corrections are included in this revision.

Bull Hardware Platform

This document may have generic references to a Bull NovaScale 9000 Series hardware platform. If so, such references are applicable to all models of the following Bull large-system computers.

Hardware Model

Corresponding Software

Bull NovaScale 9000 Series (9000V)

GCOS 8 System Release 5.2 (SR5.2)
or later

NOTE: The name in parenthesis — i.e., 9000V — is used in the GCOS 8 software and in problem reporting as the internal equivalent of the external model name.

Contact your marketing representative for more information about NovaScale 9000 hardware models.

Table of Contents

1.	Introduction.....	1-1
1.1	Processor Features.....	1-2
1.1.1	Functional Units.....	1-2
1.1.2	Address Modification.....	1-3
1.1.3	Faults And Interrupts.....	1-3
1.1.4	Execution Of Interrupts.....	1-4
1.2	Operating Modes.....	1-4
1.2.1	Processor Modes Of Operation.....	1-5
1.2.2	Segmentation Modes.....	1-7
1.2.3	Memory Addressing Modes.....	1-8
1.2.3.1	Virtual Memory Paging.....	1-9
1.2.3.2	Absolute Mode.....	1-10
1.2.3.3	Reserve Memory Space.....	1-10
1.3	Virtual Machine Operational Modes.....	1-10
1.4	Interval Timer.....	1-10
2.	Representation of Data.....	2-1
2.1	Formats.....	2-1
2.2	Position Numbering.....	2-1
2.3	The Machine Word.....	2-2

2.4	Character Strings	2-3
2.4.1	Character Positions.....	2-3
2.4.2	Bit Positions.....	2-4
2.5	Literals	2-4
2.6	Binary Numbers.....	2-5
2.6.1	Fixed-Point Numbers	2-5
2.6.2	Floating-Point Numbers.....	2-7
2.6.3	Quadruple-Precision Numbers	2-8
2.6.4	Normalized Binary Floating-Point Numbers	2-9
2.6.5	Hexadecimal Floating-Point Numbers.....	2-10
2.6.6	Binary Representation Of Fractional Values	2-11
2.7	Decimal Numbers	2-11
2.7.1	Decimal Data Character Codes	2-12
2.7.2	Floating-Point Decimal Numbers	2-13
2.7.3	Decimal Number Ranges	2-13
3.	Memory Organization	3-1
3.1	General Description	3-1
3.2	Main Memory (MM) References.....	3-1
3.2.1	Main Memory Real Addresses	3-1
3.2.2	Store Into Instruction Stream - Single CPU	3-2
3.3	Virtual Memory.....	3-3
3.3.1	Working Spaces.....	3-4
3.3.2	Page Tables.....	3-4
3.3.3	Segments.....	3-5
3.3.4	Descriptors.....	3-6
3.3.4.1	Standard Descriptor.....	3-8
3.3.4.2	Standard Descriptor with Working Space Number.....	3-10
3.3.4.3	Super Descriptor.....	3-11
3.3.4.4	Super Descriptor with Working Space Number.....	3-12
3.3.4.5	Extended Descriptor	3-13
3.3.4.6	Extended Descriptor with Working Space Number	3-14

Table of Contents

3.3.5	Domains	3-15
3.3.5.1	Entry Descriptor.....	3-17
3.3.5.2	Special Entry Descriptor.....	3-18
3.3.5.3	Dynamic Linking Descriptor.....	3-19
3.3.5.4	Shrinking.....	3-19
4.	Processor Accessible Registers	4-1
4.1	Accumulator Register (A)	4-4
4.2	Accumulator-Quotient Register (AQ)	4-5
4.3	Address Match Register (AMR).....	4-6
4.4	Address Registers (AR _n).....	4-7
4.5	Argument Stack Register (ASR)	4-9
4.6	Calendar Clock (CC).....	4-10
4.7	Data Stack Address Register (DSAR).....	4-11
4.8	Data Stack Descriptor Register (DSDR).....	4-12
4.9	Debug Mode Register (DMR).....	4-13
4.10	Exponent Register (E)	4-14
4.11	Exponent-Accumulator-Quotient Register (EAQ)	4-15
4.12	Fault Register (FLTR).....	4-16
4.13	General Index Registers (GX _n)	4-17
4.14	IC History Registers (ICHR).....	4-18

4.15	Index Registers (X_n)	4-19
4.16	Indicator Register (IR)	4-20
4.17	Instruction Counter (IC)	4-25
4.18	Instruction Segment Register (ISR)	4-26
4.19	Instruction Segment Identity Register - SEGID(IS)	4-27
4.20	Interrupt Registers (INTR_p)	4-28
4.21	Linkage Segment Register (LSR)	4-29
4.22	Low Operand Register (LOR)	4-30
4.23	Option Register (OR)	4-31
4.24	Page Directory Base Register (PDBR)	4-32
4.25	Parameter Segment Register (PSR)	4-34
4.26	Quotient Register (Q)	4-35
4.27	Safe Store Register (SSR)	4-36
4.28	Segment Descriptor Registers (DR_n)	4-37
4.29	Segment Identity Registers (SEGID_n)	4-38
4.30	Stack Control Register (SCR)	4-40
4.31	Timer Register (TR)	4-41
4.32	Virtual Machine Timer Register (VMTR)	4-42

Table of Contents

4.33	Working Space Registers (WSRn)	4-43
5.	Address Modification and Development.....	5-1
5.1	Address Modification Features	5-1
5.2	Address Generation In The NS Mode	5-1
5.2.1	Basic Modification	5-2
5.2.2	Indirect Addressing	5-2
5.2.3	Tag Field	5-3
5.2.4	Types Of Address Modification	5-4
5.2.4.1	Register (R)	5-5
5.2.4.2	Register then Indirect (RI)	5-8
5.2.4.3	Indirect Then Register (IR)	5-10
5.2.4.4	Indirect Then Tally (IT)	5-13
5.2.5	Variations Under IT Modification	5-16
5.2.5.1	Fault (T) = F Variation	5-16
5.2.5.2	Character Indirect (T) = CI Variation	5-16
5.2.5.3	Sequence Character (T) = SC Variation	5-17
5.2.5.4	Sequence Character Reverse (T) = SCR Variation	5-18
5.2.5.5	Indirect (T) = I Variation	5-19
5.2.5.6	Increment Address, Decrement Tally (T) = ID Variation	5-20
5.2.5.7	Decrement Address, Increment Tally (T) + DI Variation	5-21
5.2.5.8	Increment Address, Decrement Tally, and Continue (T) = IDC Variation	5-22
5.2.5.9	Decrement Address, Increment Tally, and Continue (T) = DIC Variation	5-23
5.2.5.10	Add Delta (T) = AD Variation	5-24
5.2.5.11	Subtract Delta (T) = SD Variation	5-25
5.2.6	Address Modification Octal Codes	5-25
5.2.7	Address Modification Flowchart	5-26
5.2.8	Floatable Code	5-27
5.2.9	Address Modification With Address Registers	5-27
5.2.9.1	Single-Word Address Modification	5-28
5.2.9.2	Multiword Address Modification	5-31
5.2.9.3	Multiword Modification Field	5-32
5.2.10	Operand Descriptors	5-35
5.2.10.1	Bit String Operand Descriptor	5-35
5.2.10.2	Alphanumeric Operand Descriptors	5-35
5.2.10.3	Numeric Operand Descriptors	5-36
5.2.10.4	Indirect Word	5-39
5.2.10.5	Operand Descriptor Address Preparation Flowchart	5-40
5.2.10.6	Operand Descriptor Bit String Address Preparation	5-42
5.2.10.7	Operand Descriptor Alphanumeric/Numeric Address Preparation	5-43
5.2.10.8	Operand Descriptor Address Preparation - Bit Operations	5-46
5.2.10.9	Operand Descriptor Address Preparation - Character Operations	5-48

5.3	Address Generation In ES/EI Modes	5-49
5.3.1	Instruction Address Field	5-49
5.3.2	Address Modification With No AR Indicated	5-49
5.3.3	Address Modification With AR Indicated	5-51
5.3.4	Tag Field Modification.....	5-53
5.3.5	IC Modification.....	5-55
5.3.5.1	ES Mode.....	5-55
5.3.5.2	EI Mode	5-56
5.3.6	Operand Descriptor Modification	5-57
5.4	Address Development.....	5-59
5.4.1	Virtual Memory Addressing.....	5-59
5.4.1.1	Operand Address Procedure.....	5-60
5.4.1.2	Instruction Address Procedure	5-60
5.4.2	Virtual Address Generation For NS Mode	5-61
5.4.2.1	Standard Descriptor NS Mode	5-61
5.4.2.2	Super Descriptor NS Mode.....	5-62
5.4.2.3	Extended Segment Descriptor NS Mode	5-64
5.4.3	Virtual Address Generation For ES/EI Modes	5-65
5.4.3.1	Standard Descriptor ES Mode.....	5-65
5.4.3.2	Extended Segment Descriptor ES Mode.....	5-66
5.4.3.3	Virtual Address Generation in EI Mode	5-67
5.4.4	Working Space Zero	5-68
5.5	Paging.....	5-69
5.5.1	Page Table Directory Word Format – SV Mode.....	5-70
5.5.2	Page Table Directory Word Format - SVMX Mode	5-71
5.5.3	Page Table Base Word Format	5-72
5.5.4	Page Table Word Format	5-74
5.5.5	Dense Page Table.....	5-76
5.5.6	Section Table	5-78
5.5.7	Address Truncation.....	5-81
5.5.8	Bounds Checking	5-81
5.5.8.1	Word and Double-Word Operations	5-81
5.5.8.2	Byte Operations.....	5-82
5.5.8.3	Bit Strings and Table of Translate Instruction	5-83
5.5.8.4	Bound Check Equations	5-83
5.1.9	Multiprocessor Memory Management.....	5-84
6.	Faults and Interrupts	6-1
6.1	Description of Faults and Interrupts	6-1
6.2	Fault Procedure	6-2

Table of Contents

6.3	Fault Processing.....	6-4
6.3.1	Processor Halt Conditions	6-8
6.3.2	Fault Descriptions	6-9
6.3.2.1	STUP Fault.....	6-9
6.3.2.2	EXF Fault.....	6-10
6.3.2.3	ONC Fault.....	6-11
6.3.2.4	LUF Fault.....	6-12
6.3.2.5	MEMSYS Fault.....	6-13
6.3.2.6	DVCF Fault.....	6-15
6.3.2.7	OFL Fault.....	6-16
6.3.2.8	CMD Fault.....	6-17
6.3.2.9	MSG Fault.....	6-18
6.3.2.10	BND Fault	6-19
6.3.2.11	MME Fault	6-20
6.3.2.12	DRL Fault.....	6-21
6.3.2.13	IPR Fault.....	6-22
6.3.2.14	FTAG Fault	6-23
6.3.2.15	SCL1 Fault.....	6-24
6.3.2.16	DYN Fault	6-25
6.3.2.17	MWS Fault.....	6-26
6.3.2.18	MSCT Fault	6-27
6.3.2.19	MPF Fault	6-28
6.3.2.20	SCL2 Fault.....	6-29
6.3.2.21	SSSF Fault	6-30
6.3.2.22	CON Fault.....	6-32
6.3.2.23	TRO Fault	6-33
6.3.2.24	SDF Fault.....	6-34
6.3.2.25	MEEP Fault	6-35
6.3.2.26	ADT Fault.....	6-36
6.3.3	Segment Descriptor Flag Faults	6-37
6.3.4	Page Table Word Control Field Faults	6-39
6.4	CPU Cache	6-41
6.5	Interrupt Procedure	6-41
6.5.1	Interrupt Timing	6-41
6.5.1.1	Single-Word Instructions	6-42
6.5.1.2	Multiword Instructions.....	6-42
6.5.1.3	Climb Instruction.....	6-42
6.5.2	Interrupt Processing	6-43
6.5.3	Instruction Counter Value Stored At Interrupt	6-44

7.	Machine Instruction Functions	7-1
7.1	Single-Word Instructions	7-2
7.1.1	Address Register Instructions	7-2
7.1.2	Boolean Operations	7-3
7.1.3	Comparison Operations.....	7-3
7.1.4	Data Movement Instructions	7-3
7.1.5	Data Shifting Instructions.....	7-3
7.1.6	Effective Address to Register Instructions.....	7-4
7.1.7	Fixed-Point Arithmetic Instructions	7-4
7.1.8	Floating-Point Arithmetic Instructions.....	7-5
7.1.9	Quadruple-Precision Floating-Point Instructions	7-5
7.1.10	Privileged Master Mode Instructions.....	7-6
7.1.11	Miscellaneous Instructions	7-6
7.1.12	Random Number Instructions.....	7-6
7.1.13	Special Processor Instructions.....	7-6
7.2	Multiword Instructions	7-7
7.2.1	Alphanumeric Instructions	7-7
7.2.2	Numeric Instructions.....	7-7
7.2.3	Bit String Instructions.....	7-7
7.2.4	Conversion Instructions	7-8
7.2.5	Edited Move Instructions.....	7-8
7.2.6	Multiword Instruction Capabilities.....	7-9
7.3	Address Register Instructions	7-10
7.3.1	Address Register Load	7-11
7.3.2	Address Register Store.....	7-11
7.3.3	Alter Address Register Contents	7-11
7.3.4	Special Address Register Instructions	7-13
7.4	Boolean Operation Instructions.....	7-14
7.4.1	Boolean Expressions	7-14
7.4.2	Evaluation Of Boolean Expressions.....	7-14
7.4.3	Boolean AND.....	7-15
7.4.4	Boolean OR	7-15
7.4.5	Boolean EXCLUSIVE OR.....	7-15
7.4.6	Boolean COMPARATIVE AND.....	7-16
7.4.7	Boolean COMPARATIVE NOT AND	7-16

Table of Contents

7.5	Fixed-Point Instructions	7-17
7.5.1	Data Movement Load	7-17
7.5.2	Data Movement Store.....	7-17
7.5.3	Data Movement Shift.....	7-18
7.5.4	Fixed-Point Addition	7-18
7.5.5	Fixed-Point Subtraction	7-18
7.5.6	Fixed-Point Multiplication.....	7-19
7.5.7	Fixed-Point Division	7-19
7.5.8	Fixed-Point Comparison	7-19
7.5.9	Fixed-Point Negate	7-19
7.6	Floating-Point Instructions.....	7-20
7.6.1	Data Movement Load	7-20
7.6.2	Data Movement Store.....	7-20
7.6.3	Floating-Point Addition.....	7-20
7.6.4	Floating-Point Subtraction	7-20
7.6.5	Floating-Point Multiplication	7-21
7.6.6	Floating-Point Division	7-21
7.6.7	Floating-Point Comparison	7-21
7.6.8	Floating-Point Negate	7-21
7.6.9	Floating-Point Normalize	7-21
7.6.10	Floating-Point Round.....	7-21
7.6.11	Floating-Point Truncate Fraction.....	7-21
7.7	Quadruple-Precision Instructions	7-22
7.8	Multiword Instructions	7-23
7.8.1	Multiword Instruction Format.....	7-23
7.8.2	Operand Descriptors And Indirect Words	7-25
7.8.3	Alphanumeric Instructions	7-25
7.8.3.1	Alphanumeric Operand Descriptor Format.....	7-26
7.8.3.2	Alphanumeric Compare.....	7-28
7.8.3.3	Alphanumeric Move.....	7-28
7.8.4	Character Move To/From Register Instructions.....	7-28
7.8.4.1	Operand Descriptor for Character Move Instructions.....	7-28
7.8.4.2	Character Move Instruction Repertoire	7-29
7.8.5	Numeric Instructions.....	7-30
7.8.5.1	Numeric Operand Descriptor Format	7-30
7.8.5.2	Numeric Compare	7-32
7.8.5.3	Numeric Move	7-32
7.8.6	Bit String Instructions.....	7-33
7.8.6.1	Bit String Operand Descriptor Format	7-34
7.8.6.2	Bit String Combine.....	7-35
7.8.6.3	Bit String Compare	7-35
7.8.6.4	Bit String Set Indicators	7-35
7.8.7	Data Conversion Instructions	7-35

7.8.8	Arithmetic Instructions	7-35
7.8.8.1	Decimal Addition.....	7-35
7.8.8.2	Decimal Subtraction	7-36
7.8.8.3	Decimal Multiplication.....	7-36
7.8.8.4	Decimal Division	7-36
7.9	Micro Operations for Edit Instructions MVE, MVNE, MVNEX	7-37
7.9.1	Micro Operation Sequence	7-37
7.9.2	Edit Insertion Tables	7-38
7.9.3	MVNE, MVE, And MVNEX Differences.....	7-39
7.9.3.1	Numeric Edit (MVNE and MVNEX)	7-39
7.9.3.2	Alphanumeric Edit (MVE)	7-40
7.9.4	Micro Operation Repertoire.....	7-40
7.9.5	Micro Operations Descriptions	7-41
7.9.5.1	CHT Micro Operation.....	7-42
7.9.5.2	ENF Micro Operation.....	7-43
7.9.5.3	IGN Micro Operation.....	7-44
7.9.5.4	INSA Micro Operation.....	7-45
7.9.5.5	INSB Micro Operation.....	7-46
7.9.5.6	INSM Micro Operation	7-47
7.9.5.7	INSN Micro Operation	7-48
7.9.5.8	INSP Micro Operation.....	7-49
7.9.5.9	LTE Micro Operation	7-50
7.9.5.10	MFLC Micro Operation	7-51
7.9.5.11	MFLS Micro Operation	7-52
7.9.5.12	MORS Micro Operation	7-54
7.9.5.13	MSES Micro Operation.....	7-55
7.9.5.14	MVC Micro Operation	7-56
7.9.5.15	MVZA Micro Operation	7-57
7.9.5.16	MVZB Micro Operation	7-58
7.9.5.17	SES Micro Operation.....	7-59
7.9.6	Micro Operation Code Assignment Map.....	7-60
7.9.7	Terminating Micro Operations	7-60
7.10	Virtual Memory Instructions	7-61
7.10.1	Descriptor Register Instructions	7-61
7.10.2	Domain Transfer (CLIMB)	7-61
7.10.3	Privileged Instructions.....	7-61
7.10.3.1	Clear Associative Memory.....	7-62
7.10.3.2	Diagnostic.....	7-62
7.10.3.3	Register Load	7-62
7.10.3.4	Register Store.....	7-62
7.10.3.5	Memory Control	7-62
7.10.3.6	System Control.....	7-63
7.10.3.7	Pointer Register Instructions	7-63

Table of Contents

7.11	ES And EI Mode Instructions	7-64
7.11.1	Register to Register Instructions.....	7-64
7.11.1.1	RR Type Instruction.....	7-64
7.11.1.2	Movement and Arithmetic Instructions	7-66
7.11.1.3	Shift Instructions	7-66
7.11.2	Fixed-Point Instructions	7-66
7.12	Transfer Instructions.....	7-67
7.12.1	Conditional Transfer	7-67
7.12.2	Unconditional Transfer	7-67
7.13	Miscellaneous Instructions	7-68
7.13.1	Option Register Instructions.....	7-68
7.13.2	Binary-To-BCD Conversion.....	7-68
7.13.3	Execute Instructions	7-68
7.13.4	Gray-To-Binary Conversion.....	7-68
7.13.5	Programmed Fault.....	7-68
7.13.6	No Operation.....	7-69
7.13.7	Repeat Instructions	7-69
7.13.8	Pointer And Length Instructions	7-69
7.13.9	Read Calendar Clock.....	7-69
7.13.10	Read Processor Number.....	7-69
7.13.11	Random Number.....	7-69
7.14	Coding Limitations	7-70
7.14.1	Operand During IT Modification.....	7-70
7.14.2	DU/DL Modification of Conditional Transfer of Control Instructions	7-70
7.14.3	Accepting an Interrupt during Instruction Overlap Processing	7-70
7.14.4	Shutdown Fault and Safe Store Stack Fault.....	7-70
7.14.5	Notes on the Read and Clear Type Instructions	7-71
7.14.6	Indirect Modification of Conditional Transfer of Control Instructions	7-71
7.14.7	Operand Store-Compare (alteration of prefetched operand by preceding store instruction).....	7-72
7.14.8	Overlapped Operand of the MLR and MRL Instructions	7-72
7.14.9	Overlapped Operand of the CSL and CSR Instructions.....	7-72
7.14.10	Bound Check of a Multiword Instruction	7-72
7.14.11	Result of Fault Detection in the MLR/MRL Instruction	7-73
7.14.12	Segment Boundary for Bit String or 6-Bit Character Operand.....	7-73
7.14.13	Effective Address (EA) Wraparound Detection.....	7-73
7.14.14	Incorrect Bounds Fault During Multiword Instructions.....	7-74
7.14.15	Prepage Check in a Multiword Instruction.....	7-75
7.14.16	Modification of the "Instruction Stream" Using MLR, MRL, or MTM	7-75
7.15	NovaScale 9000 Instruction Repertoire.....	7-76

8.	Machine Instruction Descriptions (A-B).....	8-1
8.1	Format of Instruction Descriptions	8-2
8.2	Abbreviations And Symbols.....	8-5
8.3	Common Attributes Of Instructions	8-9
8.3.1	Illegal Modification	8-9
8.3.2	Parity Indicator	8-9
8.4	Instruction Word Formats.....	8-9
8.4.1	Single-Word Instructions.....	8-9
8.4.2	Multiword Instructions.....	8-11
8.4.3	Address Register Special Arithmetic Instructions	8-13
8.4.4	Character Move To/From Register Instructions	8-15
8.4.5	Register to Register Instructions.....	8-16
8.5	Instruction Repertoire	8-17
8.6	Machine Instruction Descriptions (A).....	8-18
8.6.1	A4BD/A4BDX.....	8-18
8.6.2	A6BD/A6BDX.....	8-21
8.6.3	A9BD/A9BDX.....	8-24
8.6.4	AAR _n	8-26
8.6.5	ABD/ABDX.....	8-28
8.6.6	ACKS.....	8-31
8.6.7	AD2D	8-32
8.6.8	AD2DX.....	8-35
8.6.9	AD3D	8-38
8.6.10	AD3DX.....	8-42
8.6.11	ADA	8-45
8.6.12	ADAQ	8-46
8.6.13	ADE	8-48
8.6.14	ADL	8-50
8.6.15	ADLA.....	8-52
8.6.16	ADLAQ	8-53
8.6.17	ADLQ.....	8-55
8.6.18	ADLR.....	8-56
8.6.19	ADLX _n	8-58
8.6.20	ADQ	8-60
8.6.21	ADRR	8-61
8.6.22	ADTCS	8-63
8.6.23	ADX _n	8-65
8.6.24	ALR	8-67
8.6.25	ALS.....	8-69

Table of Contents

8.6.26	ANA	8-71
8.6.27	ANAQ	8-72
8.6.28	ANQ	8-73
8.6.29	ANRR	8-74
8.6.30	ANSA	8-76
8.6.31	ANSQ	8-77
8.6.32	ANSX_n	8-78
8.6.33	ANX_n	8-80
8.6.34	AOS	8-82
8.6.35	ARAn	8-83
8.6.36	ARL	8-85
8.6.37	ARN_n	8-87
8.6.38	ARS	8-89
8.6.39	ASA	8-91
8.6.40	ASQ	8-92
8.6.41	ASX_n	8-93
8.6.42	AWCA	8-95
8.6.43	AWCQ	8-97
8.6.44	AWD/AWDX	8-99
8.6.45	AWkD	8-102
8.7	Machine Instruction Description (B)	8-104
8.7.1	BCD	8-104
8.7.2	BNCT	8-108
8.7.3	BTD	8-110
9.	Machine Instruction Descriptions (C-D)	9-1
9.1	Machine Instruction Descriptions (C)	9-1
9.1.1	CAMP	9-1
9.1.2	CANA	9-4
9.1.3	CANAQ	9-5
9.1.4	CANQ	9-6
9.1.5	CANX_n	9-7
9.1.6	CBMX_n	9-9
9.1.7	CCAC	9-11
9.1.8	CIOC	9-12
9.1.9	CLIMB	9-15
9.1.1.1	Inward CLIMB (CALL/ICLIMB) C field bits 22 and 23 = 00	9-19
9.1.1.2	Outward CLIMB (RET/OCLIMB) C Field Bits 22 and 23 = 01	9-36
9.1.1.3	Lateral Transfer (LTRAS/GCLIMB) C Field Bits 22 and 23 = 10	9-39
9.1.1.4	Lateral Transfer (PCLIMB/LTRAD) C Field Bits 22 and 23 = 11	9-40
9.1.10	CMG	9-46
9.1.11	CMK	9-47
9.1.12	CMPA	9-49
9.1.13	CMPAQ	9-51
9.1.14	CMPB	9-53
9.1.15	CMPBX	9-56

9.1.16	CMPC	9-58
9.1.17	CMPCT	9-61
9.1.18	CMPN	9-64
9.1.19	CMPNX.....	9-67
9.1.20	CMPQ	9-69
9.1.21	CMPX _n	9-71
9.1.22	CMRR	9-73
9.1.23	CNAA	9-75
9.1.24	CNAAQ.....	9-76
9.1.25	CNAQ	9-77
9.1.26	CNAX _n	9-78
9.1.27	CSL.....	9-80
9.1.28	CSR	9-83
9.1.29	CWL.....	9-86
9.2	Machine Instruction Descriptions (D).....	9-88
9.2.1	DFAD.....	9-88
9.2.2	DFCMG	9-90
9.2.3	DFCMP	9-92
9.2.4	DFDI	9-94
9.2.5	DFDV	9-96
9.2.6	DFLD	9-98
9.2.7	DFLP	9-99
9.2.8	DFMP	9-101
9.2.9	DFRD.....	9-103
9.2.10	DFSB	9-105
9.2.11	DFSBI.....	9-107
9.2.12	DFST	9-109
9.2.13	DFSTR.....	9-110
9.2.14	DIAG	9-112
9.2.15	DIS.....	9-117
9.2.16	DIV.....	9-119
9.2.17	DIVN	9-121
9.2.18	DLY.....	9-123
9.2.19	DRL	9-124
9.2.20	DTB	9-125
9.2.21	DTRACE.....	9-130
9.2.22	DUFA.....	9-131
9.2.23	DUFM	9-133
9.2.24	DUFS	9-135
9.2.25	DV2D	9-137
9.2.26	DV2DX.....	9-140
9.2.27	DV3D	9-143
9.2.28	DV3DX.....	9-149
9.2.29	DVF.....	9-152
9.2.30	DVRR.....	9-154
9.2.31	DVRRN	9-156

Table of Contents

10.	Machine Instruction Descriptions (E-G).....	10-1
10.1	Machine Instruction Descriptions (E).....	10-1
10.1.1	EAA	10-1
10.1.2	EAQ	10-3
10.1.3	EAX _n	10-5
10.1.4	EPAT	10-7
10.1.5	EPPR _n	10-9
10.1.6	ERA	10-11
10.1.7	ERAQ	10-12
10.1.8	ERQ	10-13
10.1.9	ERRR.....	10-14
10.1.10	ERSA.....	10-16
10.1.11	ERSQ.....	10-18
10.1.12	ERSX _n	10-20
10.1.13	ERX _n	10-22
10.2	Machine Instruction Descriptions (F).....	10-24
10.2.1	FAD	10-24
10.2.2	FCMG	10-26
10.2.3	FCMP	10-28
10.2.4	FDI.....	10-30
10.2.5	FDV.....	10-32
10.2.6	FLD.....	10-34
10.2.7	FLP	10-35
10.2.8	FMP	10-37
10.2.9	FNEG.....	10-39
10.2.10	FNO	10-41
10.2.11	FRAN.....	10-43
10.2.12	FRD	10-46
10.2.13	FSB.....	10-48
10.2.14	FSBI.....	10-50
10.2.15	FST	10-52
10.2.16	FSTR	10-53
10.2.17	FSZN	10-55
10.2.18	FTR.....	10-56
10.3	Machine Instruction Descriptions (G)	10-58
10.3.1	GLDD.....	10-58
10.3.2	GLLR.....	10-60
10.3.3	GLLS	10-63
10.3.4	GLR	10-66
10.3.5	GLRL.....	10-69
10.3.6	GLRS.....	10-72
10.3.7	GLS	10-75
10.3.8	GRL	10-78

10.3.9	GRS	10-81
10.3.10	GSTD.....	10-84
10.3.11	GTB	10-86
11.	Machine Instruction Descriptions (L-M).....	11-1
11.1	Machine Instruction Descriptions (L).....	11-1
11.1.1	LAREG	11-1
11.1.2	LAR _n	11-4
11.1.3	LCA	11-7
11.1.4	LCAQ.....	11-8
11.1.5	LCCL	11-10
11.1.6	LCQ	11-12
11.1.7	LCTR	11-14
11.1.8	LCX _n	11-15
11.1.9	LDA	11-17
11.1.10	LDAB.....	11-18
11.1.11	LDAC.....	11-20
11.1.12	LDAH.....	11-22
11.1.13	LDAQ.....	11-24
11.1.14	LDAS	11-25
11.1.15	LDCR.....	11-27
11.1.16	LDD _n	11-29
11.1.17	LDDR.....	11-58
11.1.18	LDDSA	11-60
11.1.19	LDDSD	11-62
11.1.20	LDE.....	11-64
11.1.21	LDEA _n	11-65
11.1.22	LDI	11-67
11.1.23	LDMB	11-69
11.1.24	LDO	11-71
11.1.25	LDP _n	11-73
11.1.26	LDPR	11-78
11.1.27	LDPS	11-80
11.1.28	LDQ	11-82
11.1.29	LDQB.....	11-83
11.1.30	LDQC.....	11-85
11.1.31	LDQH.....	11-87
11.1.32	LDRR.....	11-89
11.1.33	LDSS	11-91
11.1.34	LDT.....	11-93
11.1.35	LDWS	11-94
11.1.36	LDX _n	11-96
11.1.37	LIMR	11-98
11.1.38	LLAR	11-100
11.1.39	LLPNR.....	11-101
11.1.40	LLR.....	11-103
11.1.41	LLS.....	11-105
11.1.42	LLUF.....	11-107

Table of Contents

11.1.43	LPDBR	11-108
11.1.44	LPL	11-110
11.1.45	LREG	11-112
11.1.46	LRL	11-114
11.1.47	LRMB	11-116
11.1.48	LRS	11-117
11.1.49	LVMMR	11-119
11.1.50	LVMTR	11-120
11.1.51	LXL _n	11-121
11.2	Machine Instruction Descriptions (M)	11-123
11.2.1	MLR	11-123
11.2.2	MME	11-127
11.2.3	MP2D	11-128
11.2.4	MP2DX	11-131
11.2.5	MP3D	11-133
11.2.6	MP3DX	11-137
11.2.7	MPF	11-140
11.2.8	MPRR	11-142
11.2.9	MPRS	11-145
11.2.10	MPX	11-148
11.2.11	MPY	11-150
11.2.12	MRL	11-152
11.2.13	MTM	11-155
11.2.14	MTR	11-157
11.2.15	MVE	11-160
11.2.16	MVN	11-164
11.2.17	MVNE	11-168
11.2.18	MVNEX	11-172
11.2.19	MVNX	11-175
11.2.20	MVT	11-180
12.	Machine Instruction Descriptions (N-R)	12-1
12.1	Machine Instruction Descriptions (N)	12-1
12.1.1	NAR _n	12-2
12.1.2	NEG	12-4
12.1.3	NEGL	12-6
12.1.4	NOP	12-8
12.2	Machine Instruction Descriptions (O)	12-10
12.2.1	ORA	12-10
12.2.2	ORAQ	12-11
12.2.3	ORQ	12-12
12.2.4	ORRR	12-13
12.2.5	ORSA	12-15
12.2.6	ORSQ	12-16

12.2.7	ORSX _n	12-17
12.2.8	ORX _n	12-19
12.2.9	OWA	12-21
12.2.10	OWQ	12-23
12.2.11	OWRES	12-25
12.3	Machine Instruction Descriptions (P)	12-26
12.3.1	PAS	12-26
12.3.2	PULS1	12-29
12.3.3	PULS2	12-31
12.4	Machine Instruction Descriptions (Q)	12-33
12.4.1	QFAD	12-33
12.4.2	QFLD	12-35
12.4.3	QFMP	12-37
12.4.4	QFSB	12-39
12.4.5	QFST	12-41
12.4.6	QFSTR	12-42
12.4.7	QLR	12-44
12.4.8	QLS	12-46
12.4.9	QRL	12-48
12.4.10	QRS	12-49
12.4.11	QSMP	12-50
12.5	Machine Instruction Descriptions (R)	12-52
12.5.1	RBFF	12-52
12.5.2	RCCL	12-54
12.5.3	RCRES	12-55
12.5.4	RDTCS	12-57
12.5.5	RET	12-58
12.5.6	RICHR	12-62
12.5.7	RIMR	12-63
12.5.8	RIW	12-65
12.5.9	RLPNR	12-67
12.5.10	RPD	12-69
12.5.11	RPL	12-76
12.5.12	RPN	12-83
12.5.13	RPT	12-84
12.5.14	RRES	12-90

Table of Contents

13.	Machine Instruction Descriptions (S).....	13-1
13.1	Machine Instruction Descriptions (S).....	13-1
13.1.1	S4BD/S4BDX.....	13-1
13.1.2	S6BD/S6BDX.....	13-3
13.1.3	S9BD/S9BDX.....	13-5
13.1.4	SACK.....	13-7
13.1.5	SAR _n	13-8
13.1.6	SAREG.....	13-10
13.1.7	SB2D.....	13-12
13.1.8	SB2DX.....	13-16
13.1.9	SB3D.....	13-18
13.1.10	SB3DX.....	13-22
13.1.11	SBA.....	13-25
13.1.12	SBAQ.....	13-26
13.1.13	SBAR.....	13-27
13.1.14	SBD/SBDX.....	13-29
13.1.15	SBLA.....	13-31
13.1.16	SBLAQ.....	13-33
13.1.17	SBLQ.....	13-35
13.1.18	SBLR.....	13-37
13.1.19	SBLX _n	13-39
13.1.20	SBQ.....	13-41
13.1.21	SBRR.....	13-42
13.1.22	SBX _n	13-44
13.1.23	SCD.....	13-46
13.1.24	SCDR.....	13-49
13.1.25	SCM.....	13-52
13.1.26	SCMR.....	13-55
13.1.27	SCTR.....	13-58
13.1.28	SDR _n	13-59
13.1.29	SFR.....	13-62
13.1.30	SICHR.....	13-63
13.1.31	SIW.....	13-65
13.1.32	SLAR.....	13-67
13.1.33	SPDBR.....	13-68
13.1.34	SPL.....	13-70
13.1.35	SREG.....	13-72
13.1.36	SRMB.....	13-74
13.1.37	SSA.....	13-75
13.1.38	SSQ.....	13-77
13.1.39	SSX _n	13-79
13.1.40	STA.....	13-81
13.1.41	STAB.....	13-82
13.1.42	STAC.....	13-83
13.1.43	STACQ.....	13-85
13.1.44	STAH.....	13-87
13.1.45	STAQ.....	13-88

13.1.46	STAS	13-89
13.1.47	STBA	13-91
13.1.48	STBQ	13-93
13.1.49	STC1	13-95
13.1.50	STC2	13-97
13.1.51	STCA	13-99
13.1.52	STCQ	13-101
13.1.53	STD _n	13-103
13.1.54	STDSA	13-105
13.1.55	STDSD	13-107
13.1.56	STE	13-108
13.1.57	STI	13-109
13.1.58	STMB	13-112
13.1.59	STO	13-114
13.1.60	STP _n	13-116
13.1.61	STPS	13-118
13.1.62	STQ	13-120
13.1.63	STQB	13-121
13.1.64	STQH	13-122
13.1.65	STSS	13-123
13.1.66	STT	13-125
13.1.67	STWS	13-126
13.1.68	STX _n	13-128
13.1.69	STZ	13-130
13.1.70	SVMMR	13-131
13.1.71	SVMOS	13-132
13.1.72	SVMTR	13-133
13.1.73	SWCA	13-134
13.1.74	SWCQ	13-136
13.1.75	SWD/SWDX	13-138
13.1.76	SXL _n	13-140
13.1.77	SZN	13-142
13.1.78	SZNC	13-143
13.1.79	SZTL	13-145
13.1.80	SZTR	13-148
14.	Machine Instruction Descriptions (T-U)	14-1
14.1	Machine Instruction Descriptions (T)	14-2
14.1.1	TCT	14-2
14.1.2	TCTR	14-6
14.1.3	TEO	14-8
14.1.4	TEU	14-11
14.1.5	TMI	14-14
14.1.6	TMOZ	14-17
14.1.7	TNC	14-20
14.1.8	TNZ	14-23
14.1.9	TOV	14-26
14.1.10	TPL	14-29

Table of Contents

14.1.11	TPNZ	14-32
14.1.12	TRA	14-35
14.1.13	TRC	14-38
14.1.14	TRCT _n	14-41
14.1.15	TRTF	14-45
14.1.16	TRTN	14-48
14.1.17	TSS.....	14-51
14.1.18	TSX _n	14-54
14.1.19	TSYNC	14-57
14.1.20	TTF	14-58
14.1.21	TTN.....	14-61
14.1.22	TXIP.....	14-64
14.1.23	TZE.....	14-65
14.2	Machine Instruction Descriptions (U).....	14-68
14.2.1	UFA	14-68
14.2.2	UFM	14-70
14.2.3	UFS.....	14-72
14.2.4	UFTR	14-74
15.	Machine Instruction Descriptions (V-X).....	15-1
15.1	Machine Instruction Descriptions (V).....	15-1
15.1.1	VMME.....	15-2
15.2	Machine Instruction Descriptions (W and X).....	15-3
15.2.1	WRES	15-3
15.2.2	WSYNC	15-5
15.2.3	XEC	15-6
15.2.4	XED	15-8
15.2.5	XRAN.....	15-11

Appendices

A.	NovaScale 9000 Machine Instructions	A-1
B.	Operation Code Maps.....	B-1
C.	ASCII Sequence	C-1
D.	EBCDIC Sequence	D-1
E.	GBCD Sequence	E-1
F.	HBCD Sequence	F-1
Index	i-1

Table of Contents

Figures

3-1	Layout Of Segments On Pages.....	3-5
3-2	Domain Of Noncontiguous Segments	3-15
3-3	Shrunk Descriptor For Corresponding New Segment.....	3-20
4-1	Accumulator Register (A) Format.....	4-4
4-2	Accumulator-Quotient Register (AQ) Format	4-5
4-3	Address Match Register (AMR) Format	4-6
4-4	Address Register (AR _n) Format (NS Mode).....	4-7
4-5	Address Register (AR _n) Format (ES/EI Mode)	4-8
4-6	Argument Stack Register (ASR) Format	4-9
4-7	Data Stack Address Register (DSAR) Format	4-11
4-8	Data Stack Descriptor Register (DSDR) Format.....	4-12
4-9	Exponent Register (E) Format.....	4-14
4-10	Exponent-Accumulator-Quotient Register (EAQ) Format	4-15
4-11	General Index Registers (GX _n) Format.....	4-17
4-12	IC History Registers (ICHR) Format.....	4-18
4-13	Index Register (X _n) Format	4-19
4-14	Indicator Register (IR) Format.....	4-20
4-15	Instruction Counter (IC) Format.....	4-25
4-16	Instruction Segment Register (ISR) Format	4-26
4-17	Instruction Segment Identity Register - SEGID(IS) Format.....	4-27
4-18	Interrupt Registers (INTR _p) Format.....	4-28
4-19	Linkage Segment Register (LSR) Format	4-29
4-20	Low Operand Register Format	4-30
4-21	Option Register (OR) Format	4-31
4-22	Page Directory Base Register (PDBR) Format	4-32
4-23	Parameter Segment Register (PSR) Format	4-34
4-24	Quotient Register (Q) Format.....	4-35
4-25	Safe Store Register (SSR) Format.....	4-36
4-26	Segment Identity Register (SEGID _n) Format.....	4-38
4-27	Timer Register (TR) Format	4-41
4-28	Virtual Machine Timer Register (VMTR) Format.....	4-42
4-29	Working Space Register (WSR _n) Format, SV Mode	4-43
5-1	Address Modification Flowchart	5-26
5-2	Flowchart for Operand Descriptor Address Preparation	5-41
5-3	Virtual Address Generation Using Standard Descriptor (NS Mode)	5-61
5-4	BASE For Standard Descriptor	5-62
5-5	Bounds For Standard Descriptor	5-63
5-6	Virtual Address Generation Using Extended Segment Descriptor (NS Mode) ...	5-64
5-7	Virtual Address Generation Using Standard Descriptor (ES Mode)	5-65
5-8	Virtual Address Generation Using Extended Segment Descriptor (ES Mode) ...	5-66
5-9	Virtual Address Generation Using Standard Descriptor (EI Mode)	5-67
5-10	Virtual Address Check.....	5-68
5-11	Page Table Directory Word (PTDW) Format in SV mode.....	5-70
5-12	Page Table Directory Word (PTDW) Format in SVMX mode	5-71
5-13	Page Table Base Word (PBW) Format.....	5-72
5-14	Main Memory Page Table Word (PTW) Format.....	5-74
5-15	Virtual Address	5-76

5-16	Address Mapping Using a Dense Page Table	5-76
5-17	PTDW Address.....	5-77
5-18	PTW Address	5-77
5-19	Word Address.....	5-77
5-20	Virtual Address	5-78
5-21	Address Mapping Using a Section Table	5-79
5-22	PBW Address	5-79
5-23	PTW Address	5-80
5-24	Word Address.....	5-80
7-1	Single-word Instruction with Address Modification.....	7-10
7-2	Alter Address Register Contents	7-11
7-3	Special Address Register Instructions.....	7-13
7-4	Multiword Instruction Format	7-23
7-5	Operand Descriptor Indirect Word Format	7-25
7-6	Alphanumeric Operand Descriptor Format.....	7-26
7-7	Character Move Descriptor Format	7-28
7-8	Numeric Operand Descriptor Format	7-30
7-9	Bit String Operand Descriptor Format	7-34
7-10	Micro Operation (MOP) Character Format	7-37
8-1	Single-Word Instruction Format.....	8-9
8-2	Multiword Instruction Format	8-11
8-3	Address Register Special Arithmetic Instruction Format.....	8-13
8-4	Character Move To/From Register Instruction Format.....	8-15
8-5	Register To Register Instruction Format	8-16
8-6	AWkD Instruction Format	8-102
9-1	Safe Store Stack Format	9-25

Tables

1-1	Status of Processor Mode Determinants.....	1-5
1-2	Memory Addressing Modes.....	1-9
2-1	Ranges Of Fixed-Point Numbers	2-6
2-2	Ranges of Binary Floating-Point Numbers	2-10
4-1	Processor Accessible Registers.....	4-2
5-1	Address Modification Octal Codes	5-25
5-2	Register Codes.....	5-33
5-3	Bound Check Equations	5-83
6-1	Processor Faults By Priority	6-3
7-1	Alphanumeric Character Number (CN) Codes.....	7-26
7-2	Alphanumeric Data Type (TA) Codes	7-27
7-3	Default Edit Insertion Table Characters for MVE and MVNX.....	7-38
7-4	Edit Insertion Table Entries for MVNEX	7-39
7-5	Micro Operation Code Assignment Map	7-60
8-1	BINARY-TO-BCD CONVERSION CONSTANTS	8-106
B-1	Operation Code Map (Bit 27 = 0)	B-2
B-2	Operation Code Map (Bit 27 = 1)	B-4

1. Introduction

This section of the Programmer's Guide provides an introduction to the essential characteristics of the central processors for Bull NovaScale™ 9000 Series systems, organized as follows:

- Section 1.1, Processor Features
- Section 1.2, Operating Modes
- Section 1.3, Virtual Machine Operational Modes
- Section 1.4, Interval Timer

This manual contains a set of machine instructions used on Bull hardware and operating systems. The systems are highly modular, allowing system configuration to be matched to the work load mix.

Each processor module in the system has full program execution capability. The processors conduct all actual computational processing (data movement, arithmetic, logic, comparison, and control operations) within the information system. The processors contain several special features that make significant contributions to multiprogramming, high throughput, and rapid turnaround. These features are under the control of the operating system, which maintains automatic supervision and complete control of the multiprogramming/multi-processing environment.

The CPU emulator emulates the DPS 9000/TA200 CPU instruction set and processor behavior except as noted in this document.

™ NovaScale is a trademark of Bull S.A.

1.1 Processor Features

A processor contains the following general features:

1. Memory protection to place access restrictions on specified segments;
2. Ability to interrupt program execution in response to an external signal (e.g., I/O termination), to save processor status, and to restore the status at a later time without loss of program continuity;
3. Ability to fetch and buffer instructions;
4. Cyclical instruction development, address preparation, paging, and operation execution; (While one instruction is being executed, instruction decoding, address preparation, and paging for the operands of the next instructions are taking place in the processor, so the processor is highly pipelined.)
5. A high level of interleaved direct main memory accesses;
6. Ability to hold recently referenced operands and instructions in a high-speed cache memory;
7. An Extended Segment (ES) addressing mode that addresses very large virtual memory segments and that includes a set of general register to register instructions;
8. An extended address paging mode that permits access of real memory configurations of up to 1 gigabyte;
9. Quad-precision arithmetic operations with exponents handled as powers of 16.

1.1.1 Functional Units

The processor consists of the following functional units:

- an instruction prefetch and decoding unit;
- a memory buffer unit, including operand and instruction caches;
- a microprogram control store unit; and
- execution units consisting of basic, floating-point, and extended instruction control units.

1.1.2 Address Modification

The address modification capability enables the user to dynamically develop an address contained in an instruction (or indirect word). Before each main memory access, two major phases of address preparation take place:

1. The address is modified by register or indirect word content if specified by the instruction word or indirect word;
2. The address is modified by translating (mapping) a virtual memory address into an absolute address for accessing main memory (i.e., no user control).

Indirection can also be used to modify the address, which leads to repetitions of the same type or to other types of modification before accessing main memory for an operand.

1.1.3 Faults And Interrupts

The processor detects certain illegal instructions, faulty communication with main memory, programmed faults, certain external events, and arithmetic faults. Many of the processor fault conditions are deliberately caused by the software and do not necessarily involve error conditions. The processor communicates with the other system modules (I/O processors and other processors) by setting and answering external interrupts. When a fault or interrupt is recognized, a "trap" results. When the processor responds to a fault or interrupt, control is transferred to an operating system module through an interdomain transfer using an entry descriptor obtained from a fixed memory location.

The locations in real memory containing the entry descriptors for interrupt, fault, and system entry (PMME) are listed below.

Type	Location
Interrupt	30-31 (octal)
Fault	32-33 (octal)
System Entry	34-35 (octal)

Interrupts and certain low-priority faults are recognized only at specific times during program execution. If, at these times, bit 28 in the instruction word is set ON, the trap is inhibited and program execution continues. The interrupt or fault signal is saved for future recognition and is reset only when the trap is recognized.

1.1.4 Execution Of Interrupts

In a multiprogramming/multiprocessing computer system, both the hardware and software must be freed from the burden of checking other components of the system, either for completion of, or requests for, service. To free the system, all active modules that have completed assigned tasks, or that require service, generate faults or interrupts to the normal flow of instructions in a processor. Typically, the Input-Output Processor (IOP) sends an interrupt to the processor after completing an I/O service (movement of data from a peripheral to main memory). Each system controller has its program interrupt cells connected in a priority sequence.

Normally, after each instruction word pair in the processor is executed, a check is made for the presence of an interrupt. If no interrupts are present, or if interrupts have been inhibited, instruction execution continues in the normal sequence. If one or more interrupts are present (and not inhibited), the system controller reports the identity of the highest priority cell that is set and then resets that interrupt cell. This procedure causes the processor to execute an inward CLIMB. The processor servicing an interrupt may load the interrupt enable registers with suitable combinations of bits to prevent any undesired interrupts and to prevent other processors from being interrupted. Servicing of the interrupt can then proceed without use of the interrupt inhibit bit. The processor can be protected against undesirable interrupts but can be interrupted, in turn, by enabled, higher-priority interrupts.

Each Input/Output Processor will generate interrupts to indicate a number of events:

1. successful completion of a requested I/O action,
2. unsuccessful initiation of a requested I/O action,
3. special interrupts (e.g., unit becoming READY), and
4. error conditions.

1.2 Operating Modes

Two types of modes determine the operation of the CPU:

1. Privileged Master, Master, and Slave modes determine the **processor mode of operation**;
2. NS (Normal Segment), ES (Extended Segment), and EI (Extended Instruction Segment) **segmentation modes** determine whether 18-bit or 36-bit registers are used and determine the method to be used to generate effective and virtual addresses; and

1.2.1 Processor Modes Of Operation

The three processor modes of operation are 1) **Privileged Master** Mode, 2) **Master** Mode, and 3) **Slave** Mode. The determinants involved in defining these processor modes are the master mode bit in the indicator register, the privileged bit in the instruction segment register (ISR), and the housekeeping bit in the page table word (PTW) for the instruction.

The status of the determinants for each mode is shown in Table 1-1.

Table 1-1. Status of Processor Mode Determinants

Determinants	Processor Modes ^a		
	Privileged Master	Master	Slave
Master Mode Bit in Indicator Register	ON	ON	OFF
Privileged Bit in Instruction Segment Register	ON	OFF	OFF
Housekeeping Bit in Page Table Word for the Instruction	ON ^b	ON/OFF	OFF

- a All other combinations are illegal and result in a Class 1 Security Fault.
- b When working space zero is referenced, the housekeeping bit is assumed to be ON, and the processor addresses memory through working space zero page tables.

A fault or an interrupt causes the processor to enter Privileged Master Mode. If the processor is in Privileged Master Mode, an instruction can change to Master mode by transferring to a segment marked *non-privileged*. The reverse is also true when transferring to a segment marked *privileged*. The use of a CLIMB instruction between Master and Privileged Master modes, like the transfer, not only allows a change of processor execution modes but also a change of domains. Refer to the CLIMB instruction definition (documented later in the manual) for a detailed description of the variations of changes in domains.

The Master mode bit in the indicator register can be turned ON when:

1. an interrupt or fault occurs;
2. execution of the PMME version of the CLIMB instruction occurs, which causes a system entry; and
3. execution of the OCLIMB version of the CLIMB instruction occurs, where the master mode bit of the restored indicator register is ON.

The following mode-dependent processor functions are listed by mode. None of these functions is permitted in Slave mode.

Functions Allowed in Master and Privileged Master Modes

1. access through working space register zero
2. reading operands from a housekeeping page of segment descriptor type T = 0, 2, 4, 6, 12, or 14
3. executing instructions from housekeeping pages of type T = 0 segments
4. executing a CLIMB (ICLIMB or GCLIMB) not invoking a system entry option (PMME)
5. executing a transfer to a privileged executable segment

Functions Allowed Only in Privileged Master Mode

1. executing Privileged Master mode instructions (e.g., load working space registers)
2. executing Privileged Master mode options of the LDD_n, LDP_n, or CLIMB instructions, such as copying the safe store register (SSR) to a descriptor register (DR_n)
3. writing on housekeeping pages of type T = 0, 2, 4, 6, 12 or 14 segments using instructions other than CLIMB, SDR_n, STD_n

1.2.2 Segmentation Modes

The NS (Normal Segment), ES (Extended Segment), and EI (Extended Instruction Segment) modes are specified with bit 24 of the Instruction Segment Register (ISR):

1. when ISR bit 24 = 0, NS mode;
2. when ISR bit 24 = 1 and ISR type T = 0, ES mode;
3. when ISR bit 24 = 1 and ISR type T = 12, EI mode.

ISR bit 24 may be altered only with the CLIMB instruction.

Processor operations differ between NS, ES, and EI modes for the following items:

- the number of bits in the index and the address registers, Instruction Counter (IC), and the Page Directory Base Register (PDBR),
- the method used to generate effective and virtual address,
- the execution of some instructions, and
- the additional register instructions available in ES and EI modes.

1.2.3 Memory Addressing Modes

Three types of memory addressing exist in the DPS 9000G and DPS 9000TA:

1. **Virtual memory** which is mapped to a **real** (physical) memory address,
2. **Absolute mode** which is used only when working space zero is referenced, and
3. **Reserve memory** which is reserved for special use.

SV mode

Standard virtual, standard real memory mode. Virtual addressing is limited to 512 working spaces and real memory addressing is limited to 1 GB.

SVMX mode

Standard virtual, real memory extended mode. Virtual addressing is limited to 512 working spaces and real memory addressing is extended to 16 GB.

1.2.3.1 Virtual Memory Paging

Virtual memory paging mode is an integral part of the address translation process for mapping a virtual memory address to a real memory address. Each of the 512 working spaces is supported by a page table. The location of the page table supporting a particular working space (WS) is found by using the 9-bit (SV or SVMX mode) working space (WS) number to index a 512-word table (SV or SVMX mode) that contains a section table containing the real memory address of the page table.

Table 1-2. Memory Addressing Modes

Memory Addressing Modes	512 Workspaces (9-bit WS)	
Maximum Real Memory	1GB (256 MW)	SV Standard Virtual, Standard Real Memory
	16 GB (4 GW)	SVMX Standard Virtual, Real Memory Extended

Each of the 512 working spaces is supported by one page table or one section table. The location of the page table or section table supporting a given WS is indicated by a 9-bit WS number in SV mode. This WS number indexes the page table directory (PTD), a 512-word table that contains the real memory address of a page table or section table. The section table consists of up to 4K words and includes the real memory address of the page table. The individual words in the section table are called page table base words (PBW). When paging is performed, the section table allows the page table to be divided and distributed throughout memory.

1.2.3.2 Absolute Mode

Absolute addressing mode is defined as any reference to $WSN = 0$ when the Processor is in Privileged Master Mode (i.e., $IR(28) = 1$ and $ISR(26) = 1$). The CPU hardware actually uses paging when generating absolute mode addresses. When in absolute mode the hardware will reference a Page Table which translates the Virtual Address to a real memory address. The $WSN = 0$ Page Table is prepared by the maintenance system when the system is initialized. In Absolute Addressing mode all pages are given housekeeping privilege regardless of the value of PTW bit 32.

To reference working space zero, the CPU must be in Privileged Master Mode with the privileged bit of the Instruction Segment Register (ISR) ON. If these conditions are not satisfied, a Command fault occurs when an attempt is made to reference working space zero.

1.2.3.3 Reserve Memory Space

Reserve memory space is space in main memory specifically reserved by the Service Processor (SP). It is reserved at system startup for use by the operating system software, SCU firmware, CPU firmware.

The operating system software can access reserve memory space only through the RRES (Read Reserve Memory), WRES (Write Reserve Memory), RCRES (Read and Clear Reserve Memory), and OWRES (OR Write Reserve Memory instructions. These instructions generate a real memory address by adding the effective address to the reserve memory base register. In this operation, the segment descriptor is not used, and the result is not paged. The operating system can use RRES and WRES instructions to access the configuration information needed for startup and page management.

1.3 Virtual Machine Operational Modes

NOTE: VMF not implemented on the V9000 platform. Attempts to enter VMM or VMOS mode causes IPR fault.

1.4 Interval Timer

The processor contains a timer that provides a program interrupt (timer runout fault) at the end of a variable interval. The timer is loaded by the operating system and can be set to a maximum of approximately four minutes total elapsed time.

2. Representation of Data

This section of the Programmer's Guide provides a description of data representation, organized as follows:

- Section 2.1, Formats
- Section 2.2, Position Numbering
- Section 2.3, The Machine Word
- Section 2.4, Character Strings
- Section 2.5, Literals
- Section 2.6, Binary Numbers
- Section 2.7, Decimal Numbers

2.1 Formats

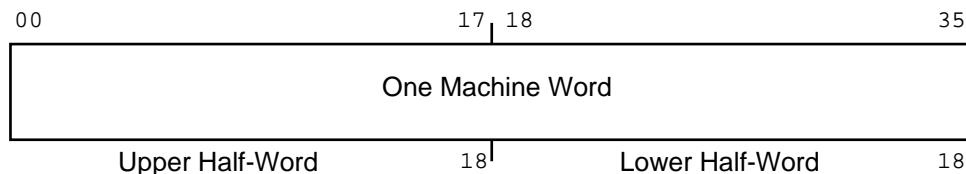
The processor is functionally organized to process 36-bit groupings of information called words. Special features are also included for ease in manipulating 4-bit groups, 6-bit groups, 9-bit groups, 18-bit groups, and 72-bit double-precision groups. These bit groupings are used by the hardware and software to represent a variety of forms of information.

2.2 Position Numbering

The numbering of bit positions, character positions, words, etc., starts with zero and increases from left to right as in conventional alphanumeric text.

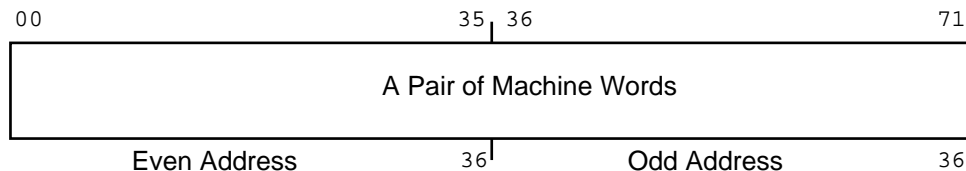
2.3 The Machine Word

The machine word consists of 36 bits arranged in the following way.



Data transfers between the processor and memory are double-word-oriented. For single-precision data, 36 bits are used at a time, and two parallel 36-bit words are used for double-precision data. When words are transferred to a memory unit, Error Detection and Correction (EDAC) bits are added to each word pair before the words are stored. When words are requested from a memory unit, the EDAC bits are read from memory, verified, and removed before sending the word pair to the processor.

The processor has many built-in features for the efficient transfer and processing of pairs of words. In transferring a pair of words to or from memory, a pair of memory locations is accessed. Their addresses consist of an even number and the next higher odd number. A pair of machine words is arranged as illustrated on the next page.



Either of the two addresses may be used as the effective address (Y) when addressing such a pair of memory locations in an instruction intended for handling pairs of machine words.

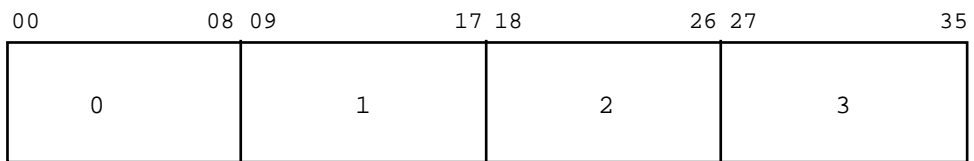
Thus, if Y is even, the pair of locations (Y, Y+1) is accessed. If Y is odd, the pair of locations (Y-1, Y) is accessed. The term "Y-pair" is used for each such pair of addresses. Preferred coding practice refers to the even address; the GMAP assembler issues a warning diagnostic if Y is odd in an instruction intended for handling pairs of machine words.

2.4 Character Strings

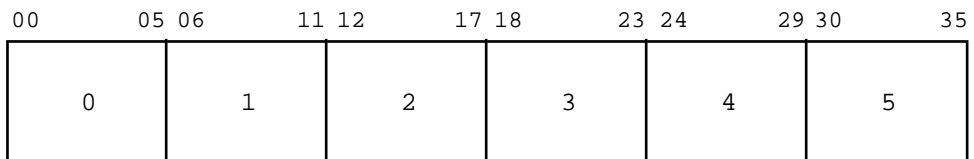
2.4.1 Character Positions

Alphanumeric data is represented by 9-bit, 6-bit, or 4-bit characters. A machine word contains either four, six, or eight characters, respectively. The character positions within the word are listed below.

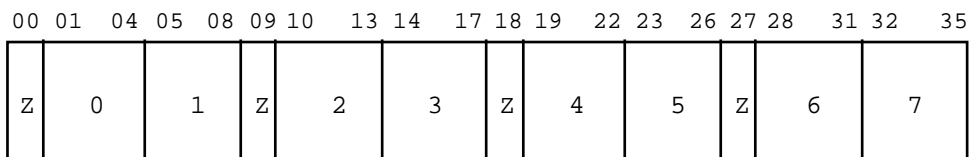
9-Bit Characters (Bytes)



6-Bit Characters



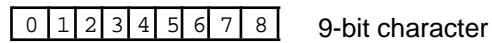
4-Bit Characters (Packed Decimal)



The Z represents the zero bit value. Other numbers in the fields represent the character positions.

2.4.2 Bit Positions

The following illustration indicates the bit positions within a character.



Thus, both bit and character positions increase from left to right as in normal reading.

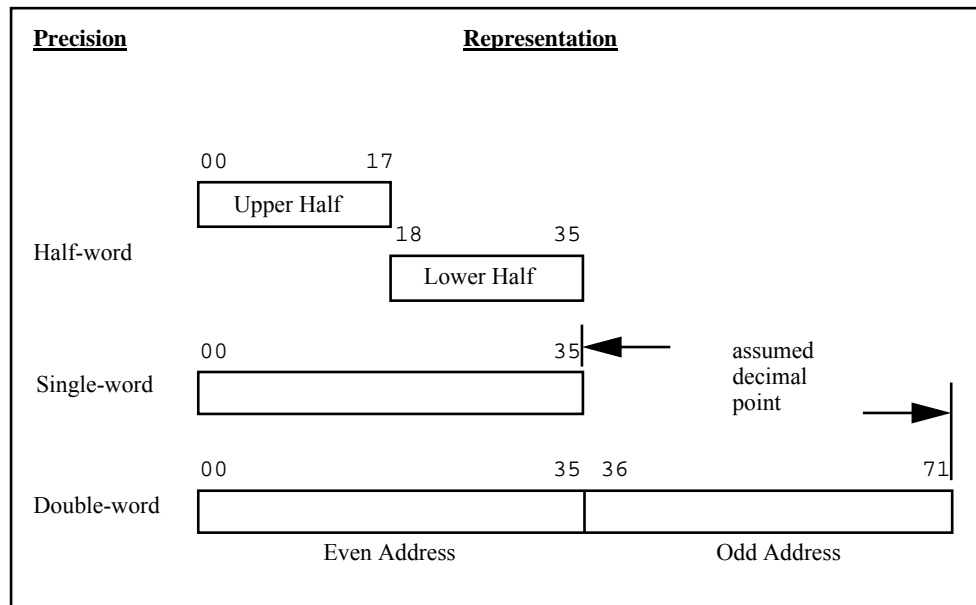
2.5 Literals

For information about literals, refer to the *GMAP Assembler User's Guide* (Order No. DH01).

2.6 Binary Numbers

2.6.1 Fixed-Point Numbers

Binary fixed-point numbers are represented with half-word, single-word, and double-word precision (as shown below).



Instructions can be divided into two groups according to the way in which the operand is interpreted: 1) the "logic" group and 2) the "algebraic" group.

For logic operations, operands and results are regarded as unsigned, positive binary numbers. In the cases of addition and subtraction, the occurrence of an overflow is indicated by the carry out of the most significant (leftmost) bit position:

Addition if the carry out of the leftmost bit position equals 1 (Carry indicator ON), the sum is above the range;

Subtraction if the carry out of the leftmost bit position equals 0 (Carry indicator OFF), the difference is below the range.

In the case of comparisons, the zero and carry indicators show the relation.

For algebraic operations, operands and results are regarded as signed binary numbers, and the leftmost bit is used as a sign bit (a 0 being plus and 1 minus). When the sign is positive, all the bits represent the real value of the number. When the sign is negative, they represent the complement of the real value of the number.

With addition and subtraction, the occurrence of an overflow is indicated by the carries into and out of the leftmost bit position (the sign position). If the carry into the leftmost bit position does not equal the carry out of that position, then overflow has occurred. If overflow has been detected and the sign bit equals 0, the result is below range; if overflow has occurred and the sign bit equals 1, the result is above range.

In integral arithmetic, the location of the decimal point is assumed to the right of the least significant bit position, that is, depending on the precision, to the right of bit position 35 or 71 (17 for upper half-word).

The number ranges for the various cases of precision, interpretation, and arithmetic are given in Table 2-1.

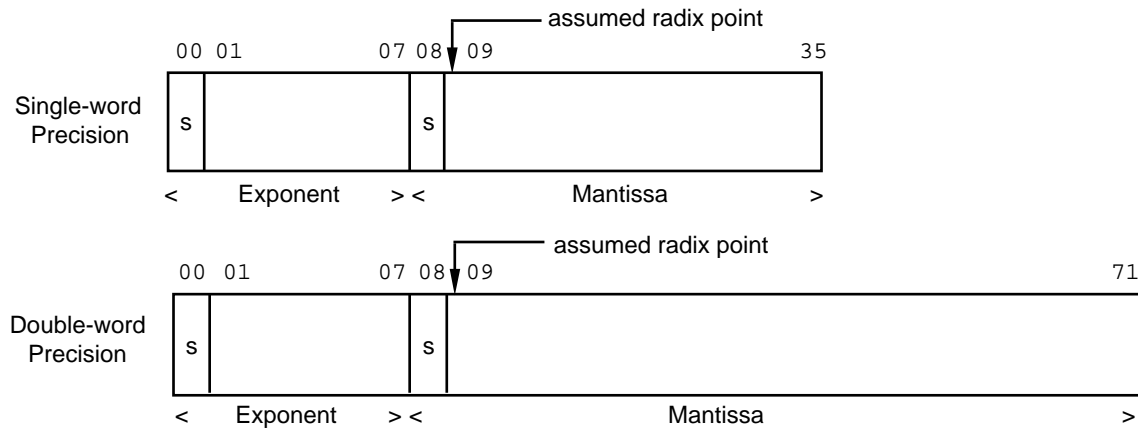
Table 2-1. Ranges Of Fixed-Point Numbers

Inter-pretation	Arithmetic	Precision		
		Half-word (Xn, Y0 ...17)	Single-word (A,Q,Y)	Double-word (AQ, Y-pair)
Algebraic	Integral	$-2^{17} \leq N \leq (2^{17} - 1)$	$-2^{35} \leq N \leq (2^{35} - 1)$	$-2^{71} \leq N \leq (2^{71} - 1)$
	Fractional	$-1 \leq N \leq (1 - 2^{-17})$	$-1 \leq N \leq (1 - 2^{-35})$	$-1 \leq N \leq (1 - 2^{-71})$
Logic	Integral	$0 \leq N \leq (2^{18} - 1)$	$0 \leq N \leq (2^{36} - 1)$	$0 \leq N \leq (2^{72} - 1)$
	Fractional	$0 \leq N \leq (1 - 2^{-18})$	$0 \leq N \leq (1 - 2^{-36})$	$0 \leq N \leq (1 - 2^{-72})$

2.6.2 Floating-Point Numbers

Binary floating-point numbers are represented with single-word and double-word precision. The upper 8 bits represent the integral exponent to the base 2 in two's complement form, and the lower 28 or 64 bits represent the fractional mantissa in two's complement form.

The format for a binary floating-point number is given below.



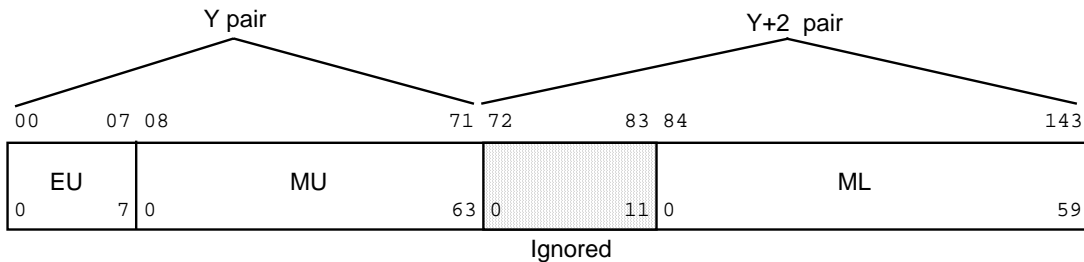
Before performing floating-point additions or subtractions, the processor aligns the number that has the smaller exponent. To maintain accuracy, the lowest permissible exponent of -128, together with the mantissa of zero, has been defined as the machine representation of the number zero (which has no unique floating-point representation). Whenever a floating-point operation yields an untruncated resultant mantissa equal to zero (71 bits plus sign because of extended precision), the exponent is automatically set to -128.

2.6.3 Quadruple-Precision Numbers

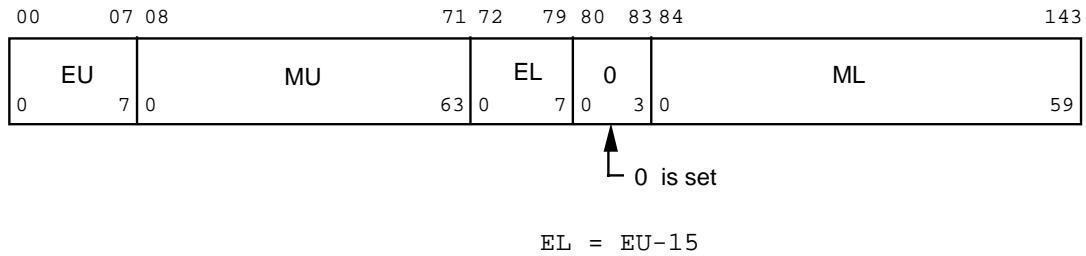
The data format used in quadruple-precision arithmetic is illustrated below.

NOTE: The format of data to be used in an operation is somewhat different from that of data to be stored after the operation.

The format for data when an operand in main memory is used as arithmetic data is structured in the following way:



The format for data when the result is stored in main memory is given below.



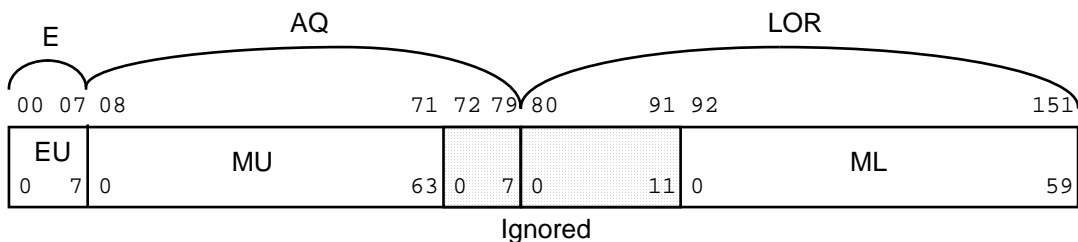
NOTE: In these formats,

<u>Exponent</u>	<u>Mantissa</u>
EU = E-upper	MU = M-upper
EL = E-lower	ML = M-lower

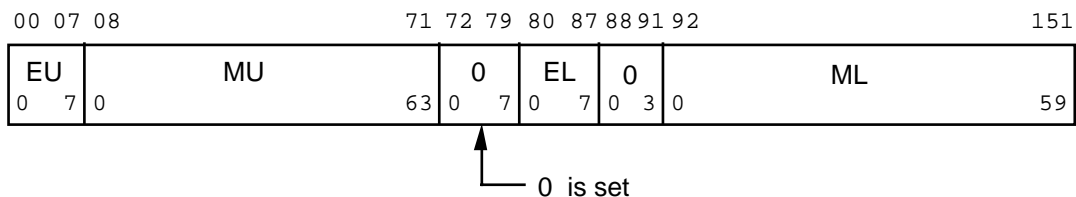
- The data in memory must reside on a double-word boundary.
- The four words of data may span two pages.

The registers E, AQ, and LOR are used for quadruple-precision arithmetic. The format for data used as operation data is structured as follows:

Representation of Data



The format when the result is loaded into C(EAQ, LOR) is structured as follows:



Field Values

- EU High-order exponent
- MU High-order mantissa
- EL Low-order exponent
- ML Low-order mantissa

Quadruple-precision value: $N = (M_U + M_L) \cdot 16^{EU}$

The quadruple-precision instructions operate with the exponent as a hexadecimal exponent, regardless of the value of bit 32 of the indicator register (IR).

2.6.4 Normalized Binary Floating-Point Numbers

For normalized binary floating-point numbers, the binary point is placed at the left of the most significant bit of the mantissa (to the right of the sign bit). Numbers are normalized by shifting the mantissa to the left (and correspondingly adjusting the exponent) until no leading zeros are present in the mantissa for positive numbers, or until no leading ones are present in the mantissa for negative numbers. Zeros fill in the vacated bit positions on the right.

The number ranges resulting from the various cases of precision, normalization, and sign are given in Table 2-2.

Table 2-2. Ranges of Binary Floating-Point Numbers

	Sign	Single Precision	Double Precision
Normalized	Positive	$-2^{-129} \leq N \leq (1-2^{-27}) 2^{127}$	$2^{129} \leq N \leq (1-2^{-63}) 2^{127}$
	Negative	$(-1+2^{-26}) 2^{-129} \geq N \geq -2^{127}$	$(-1+2^{-62}) 2^{-129} \geq N \geq -2^{127}$
Unnormalized	Positive	$2^{-155} \leq N \leq (1-2^{-27}) 2^{127}$	$2^{-191} \leq N \leq (1-2^{-63}) 2^{127}$
	Negative	$-2^{-155} \geq N \geq -2^{127}$	$-2^{-155} \geq N \geq -2^{127}$

NOTE: The floating-point number zero is not included in the table.

2.6.5 Hexadecimal Floating-Point Numbers

The hexadecimal option may be used in floating-point operations to declare hexadecimal constants, either explicitly or by default. The term hexadecimal refers to a floating-point format where the mantissa is a binary number, while the exponent represents a power of 16 (2^{*4}). The mantissa is shifted by the number of places for 4-bit groups, as required by the exponent.

When decimal data is declared in source images, the characters "X" or "XD" are specified in the variable field of the DEC pseudo-operation in place of "E" or "D" to indicate single- or double-precision hexadecimal floating-point binary data, respectively. (Refer to the *GCOS 8 OS GMAP User's Guide*.) These characters control the computation of the exponent, the positioning of the binary mantissa, and the storage required by the data. When reading the converted data, the user should be aware that the exponent represents a power of 16, so a normalized positive mantissa may have as many as three leading binary zeros.

The hexadecimal floating-point mode is enabled only when bit 32 of the Indicator Register is set to 1. After the hexadecimal floating-point mode is requested, the user controls the floating-point mode through the Indicator Register. If bit 32 of the Indicator Register is not set to 1, the floating-point mode will be binary.

If a decimal point is present in the variable field of the DEC pseudo-operation and no other controls are defined, the mechanism defaults to floating-point format. The HXFLPT pseudo-operation alters the default mechanism to hexadecimal floating-point format. The default mechanism may be further controlled by including the ON, OFF, SAVE, or RESTORE options in the variable field of the HXFLPT pseudo-operation. (Refer to the *GCOS 8 OS GMAP User's Guide* for additional information.)

2.6.6 Binary Representation Of Fractional Values

A decimal fraction of a given number of digits cannot necessarily be represented exactly by a binary fraction of any finite number of bits. Consider, for example, the value 1/5, which is represented in decimal notation as 0.2. As a four-bit **binary** fraction, 1/5 becomes $(.0011)_2$ or 3/16. In eight bits, it becomes $(.00110011)_2$ or 51/256. In fact, the exact value must be written as:

$$(0.2)_{10} = (0.0011)_2 \dots,$$

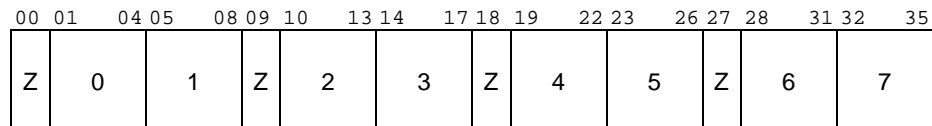
which means that the bit pattern 0011 in the binary expansion keeps repeating indefinitely. If the decimal value 0.2 is converted to a binary expansion of 71 bits and then is converted back, the one-digit result would be 0.1, quite different from 0.2. The four-digit result would be 0.1999, which is almost (but not quite) equal to 0.2. If computations were involved instead of only conversions, the imprecision in the decimal result could be perpetuated.

Various adjustments can be made to binary fractional values to make exact decimal results highly probable. Binary integer notation can be used to represent all values, whether integral or fractional, but this method may make multiplication or division of an operand by a power of 10 necessary in the course of a computation.

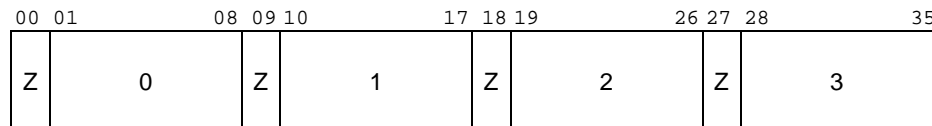
2.7 Decimal Numbers

Scaled decimal numbers that are used directly in hardware arithmetic commands are expressed as decimal digits in either the 4-bit or 9-bit character format. They are expressed as unsigned numbers or as signed numbers using a separate sign character.

Decimal data use the following formats:



Packed Decimal (4-bit)



ASCII (9-Bit)

The 'Z' represents the bit value 0 while other numbers in the fields represent the character positions.

2.7.1 Decimal Data Character Codes

During arithmetic operations, decimal digits and signs are checked by the hardware as 4-bit data (the 4 least significant bits from a 9-bit numeric). The following interpretations are made.

Bit Pattern for Character	Interpreted As:	Illegal Procedure (IPR) if:
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001	0 1 2 3 4 5 6 7 8 9	found where descriptor specifies sign
1010 1011 1100 1101 1110 1111	+ + + - + +	found where descriptor specifies digits

The following codes are generated for output signs (9-bit zones are created by prefixing binary 00010). The octal values are listed below.

	Plus	Minus
4-bit	14 (13)	15
9-bit	053	055

For several numeric instructions, a sign value of 13 can be optionally generated.

2.7.2 Floating-Point Decimal Numbers

The format for a floating-point decimal number expressed in 9-bit characters is provided in the following illustration.

SIGN	$10^n \dots 10^2$	10^1	10^0	0	EXPONENT
------	-------------------	--------	--------	---	----------

where SIGN can start at any legal 9-bit character boundary

In 4-bit character notation, four formats for floating point decimal numbers exist:

										8-bit
0	SIGN	$10^n \dots$	0	10^3	10^2	0	10^1	10^0	0	EXPONENT

< Even character boundary, odd number of digits (number of digits = n+1)

								4-bit	4-bit		
SIGN	0	$10^n \dots$	10^3	0	10^2	10^1	0	10^0	EXPO	0	NENT

< Odd character boundary, odd number of digits (number of digits = n+1)

The 8-bit exponent field, which now spans two character positions, is interpreted in the same way as in 9-bit character mode. The other two formats are formed with n+1 even. This process effectively exchanges the two exponent representations in the formats shown.

2.7.3 Decimal Number Ranges

The number ranges for decimal numbers are listed below.

1. Fixed-Point Unsigned Integer: $0 \dots 10$ to the power of 63
2. Fixed-point Signed Integer: ± 10 to the power of 62
3. Floating-point (implicitly signed)
 - a) 9-bit format range:
 $\pm 10^{61} * 10^{+127-128}$
 - b) 4-bit format range:
 $\pm 10^{60} * 10^{+127-128}$
 - c) Zero:
 $\pm 0 * 10^{+127-128}$

Notes

3. Memory Organization

This section of the Programmer's Guide describes memory organization, organized as follows:

- Section 3.1, General Description
- Section 3.2, Main Memory (MM) References
- Section 3.3, Virtual Memory

3.1 General Description

The Central Processing Units (CPUs), I/O Central (IOCs), and SP Agent access emulated DPS9000 Memory in performing the emulation of a NovaScale 9000 system.

3.2 Main Memory (MM) References

3.2.1 Main Memory Real Addresses

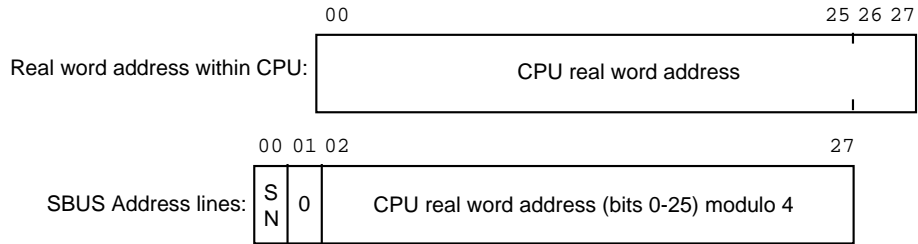
The V9000 generates a 28-bit real memory word (30-bit byte) address in Standard Virtual Storage (SV) mode. The CPU hardware does not check or notify software of any address generation process which exceeds 28 bits.

Memory accesses are done in 16-word blocks. A block is transmitted to/from the CPU in four consecutive 4-word sub-block cycles. A 28-bit System Bus Address (SBA) is used to specify the first sub-block of the 16-word block to be transmitted. The remaining sub-blocks are transmitted as described below:

1 st sub-block	2 nd sub-block	3 rd sub-block	4 th sub-block
00	01	10	11
01	00	11	10
10	11	00	01
11	10	01	00

Bit 0 of the SBA is a System Number (SN) which is set by the Service Processor (SP). Bit 1 of the SBA is always forced to a value of zero by the CPU. Bits 2-27 of the SBA are the upper 26 bits of the 28-bit real memory address. The system number allows the hardware to be split into two physically different systems.

The relationship of the CPU real word address to the data sent out on the System Bus address lines is:



The SCU determines how addresses are routed to the various Memory units attached to the SCU memory bus. This is done via configuration tables contained in the SCU and set by the SP. There are no software instructions for accessing the SCU memory configuration registers.

3.2.2 Store Into Instruction Stream - Single CPU

Within a single CPU there shall be no sequence restrictions as to where stores are directed except as defined in a given instruction specification. Within a single CPU a store instruction can modify:

- a) any instruction or operand of an instruction following or preceding the store instruction;
- b) any instruction or operand of an XEC, XED*, RPT, RPD*, or RPL instruction;
- c) any instruction or operand of IDC or DIC address modification.

***NOTE:** The first instruction of an XED pair, or the first instruction of an RPD pair cannot store into the second instruction of the pair.

3.3 Virtual Memory

Virtual memory (VM) provides an extremely large, directly addressable memory space (2^{43} bytes) and a complement of registers and instructions to manage virtual address space. To provide for efficient management and control, the VM space is divided into a number of equal working spaces. The working spaces are further divided into variable sizes called "segments". A segment within a working space is described by a "segment descriptor," which has a base relative to the origin of the working space and a bound relative to the base, together with control information. Thus, for all memory references, virtual memory addresses are prepared relative to a particular working space and to a particular segment base within the working space. These virtual memory addresses are then mapped to real memory addresses by a hardware algorithm, of which memory paging is an integral part.

A high level of security, based on virtual memory management with working spaces and segment descriptors, is provided between simultaneously executed procedures by using hardware registers and instructions. To access (generate a memory address for) an area of VM, a process (used here to mean the smallest working unit of software) must have a segment descriptor that "frames" the particular segment of VM and gives the desired permission for using this segment of VM (that is, Read, Write, or Execute permission). A process cannot create a segment descriptor or change the base and bound to access an area of VM not enclosed by the area originally "framed" or increase the permissions field. Therefore, a process is limited to accessing only those areas of VM described by segment descriptors that are available to the process. Segment descriptors are passed to a process either by the operating system or by another process. (All descriptors are created by the operating system but may be passed by one process to another process.)

In the most secure form of operation, segment descriptors are passed to a process only through one or more of the three segment descriptor "stacks" maintained in main memory. Each of these stack areas of memory is defined by a special hardware register. A unique transfer of domain (CLIMB) instruction is provided that allows the process to specify which descriptors in the stacks are to be passed to another process. Then, during the execution of this instruction, the descriptor stack registers are manipulated by the hardware to pass descriptors as specified by the process performing the transfer.

The hardware environment for the virtual memory is composed of four elements: working spaces, domains, segments, and pages. The working spaces and pages are physical elements, whereas the segments and domains are logical elements. These elements are treated as separate components of the virtual memory but must be interpreted in the context of the whole environment since they are closely related in their interaction with each other.

3.3.1 Working Spaces

The virtual memory is divided into 512 (0 through 511) equal working spaces (WS) of 2^{34} bytes, each of which is divided into fixed-length parts called pages. These pages are used for memory management and have a fixed size of 1024 words (4096 bytes) each. Working space numbers (WSN) used to generate a particular virtual memory address are obtained from one of eight working space registers (WSR) or a segment descriptor register (DR_n).

NOTE: Historically, virtual memory included reference to working space quarters, described in this manual as working spaces. The concept of working space quarter is not used by any software implementation, and the phrase is not mentioned elsewhere in this manual. The hardware has not been changed.

3.3.2 Page Tables

Each working space has an associated page table that identifies the real memory allocation. The page table for each working space is located in real memory by a pointer that resides in a section table (SCT). The directory has 512 entries, and the pointer to the directory is stored in the page directory base register (PDBR).

Directory entries are pointers to section tables. The section table (SCT) consists of up to 4K words called page table base words (PBW) that allow page tables to be divided and distributed throughout the memory. The PDBR or the SCT can only be altered in the Privileged Master mode.

In a memory operation, a virtual address is automatically transformed to a real address by the hardware. The virtual address has three components: a working space number (WSN), a page number, and a page byte number (commonly called an offset).

3.3.3 Segments

Another division of the working space is the segment. Each segment is a logical entity of variable length and may be as small as one byte or as large as 2^{32} bytes. Consequently, a segment may reside on a portion of a page or span several pages (see Figure 3-1). Segments are described with two-word (72-bit) segment descriptors. When a virtual address is generated, the segment descriptor is located in the segment descriptor register. Segments in virtual memory are specified with a base value which is relative to from the origin of the WS, and a bound which is relative to or from the base.

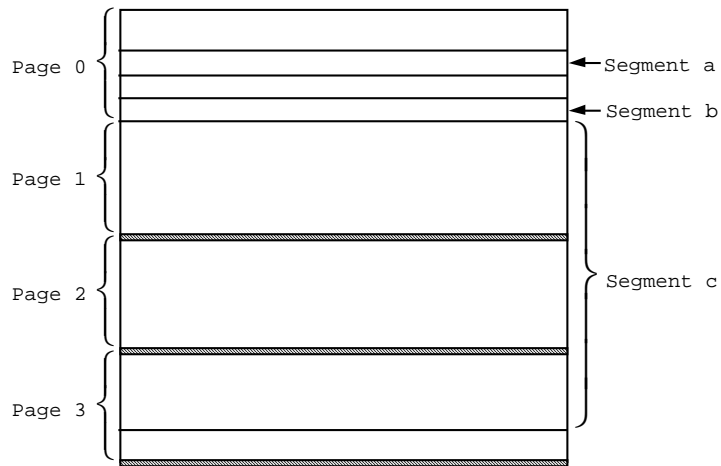
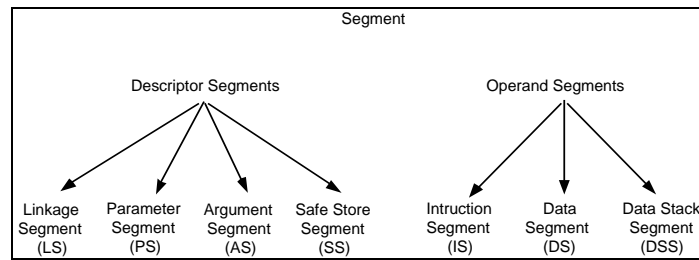


Figure 3-1. Layout Of Segments On Pages

To understand the relationship between pages and segments, the structure of a working space must be examined. The combination of a working space number and offset within the related working space is called a virtual address. Pages of 1K size are ordered sequentially by page number within a working space. Each page is represented by a page table word (PTW) that points to a real page, if that page is in memory.

A segment is a logical sequence of virtual addresses, starting from a base and of a size equal to the bound of that segment. The base and bound of a segment are contained in a system protected, two-word structure called a segment descriptor. A segment may be small and contained anywhere within a page, or it may span several pages, irrespective of page boundaries.

A segment is characterized by its elements and the form of access to these elements, which can be Execute, Read, or Write. Segments are classified either as descriptor segments or operand segments. The descriptor segments that contain valid descriptors as part of their contents may be used as linkage, parameter, argument, or safe store segments. The operand segments may be instruction-only, data-only, instruction and data segments, or data stack segments, as illustrated in the following diagram.



A segment of either class may also be loaded into one of the eight operand descriptor registers (DR_n).

3.3.4 Descriptors

A descriptor consists of a 72-bit word-pair and locates a segment in virtual memory. When the processor hardware obtains a descriptor from memory, the processor assumes that the descriptor is located on an even-word boundary and ignores the least significant bit of the virtual word address. If a descriptor is stored from a register, the processor hardware stores on an even-word boundary.

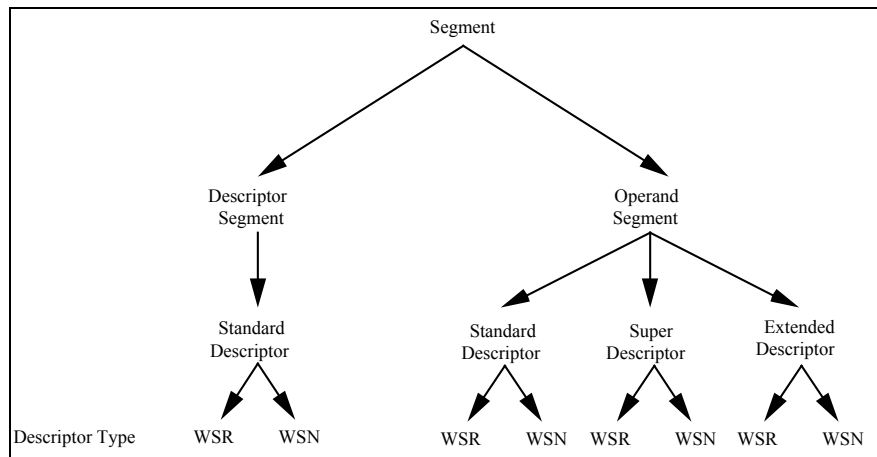
To allow a process to have access to a segment, a copy of the descriptor must be obtained to locate the segment in virtual memory. The descriptor also delimits, through a set of flags, what forms of access to the segment are available.

Twelve types of descriptors are available. Those segments containing instructions, data, or a combination of both are commonly called operand segments and have descriptors that are either type 0, 2, 4, 6, 12, or 14 to indicate operand storage. The segments containing only descriptors, that is, descriptor segments, have descriptors that are either type 1 or 3 to indicate descriptor storage. Operand memory references are always accomplished through operand segment descriptors, usually to nonhousekeeping pages, whereas descriptor references are made only through descriptor segment descriptors to housekeeping pages.

Memory Organization

The remaining five descriptors are used only during the execution of the special transfer-of-domain (CLIMB) instruction. The descriptor types are listed below.

<u>Type</u>	<u>Descriptor</u>	<u>Contents</u>
0	Standard	Instructions/Data
2	Standard with WSN	Data
4	Super	Data
6	Super with WSN	Data
12		Extended Data
14		Extended with WSN Data
1	Standard	Descriptors
3	Standard with WSN	Descriptors
5	Dynamic Linking	}
7	Special Entry	}
8	Entry	} Used only with Climb
9	Entry	}
11	Entry	}

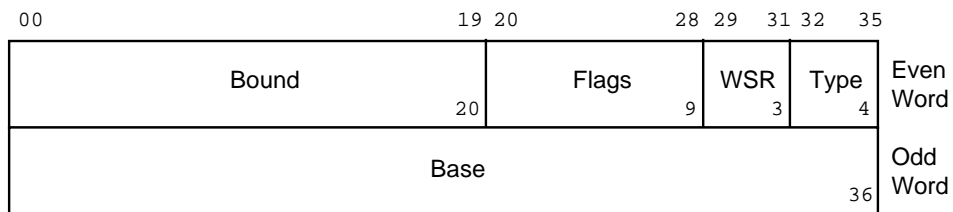


Instructions such as LDSS and LDAS that load segment descriptors from operand segments to registers and instructions such as TSS and STPS that store segment descriptors in operand memory areas access segments of type 0, 2, 4, 6, 12, or 14. In these instances, instruction operand memory addresses must specify operands in operand segments. An Illegal Procedure (IPR) fault occurs when operand or indirect word addresses are generated which specify segment descriptors of other than those types. This procedure has two exceptions.

1. Segment descriptor types 1 and 3 specify segments that include segment descriptors. The CLIMB, SDR_n, LDP_n, LDD_n, and STD_n instructions access segment descriptor segments to load or store segment descriptors. These segment descriptor segments must be located in housekeeping pages. An IPR fault occurs when either a segment descriptor is accessed with an instruction other than one of the five mentioned above or when one of these instructions is used to access a segment descriptor in a an operand segment that is not located in a housekeeping page.
2. Instructions such as LDD_n can access both operand segments and segment descriptor segments because LDD_n performs different operations with each access. These instructions indirectly access segment descriptors through operand segments. The safe store stack contains data other than segment descriptors. However, it is specified with type 1 or 3 segment descriptors. The safe store stack does not contain operand data and cannot be accessed except with Privileged Master Mode. Using this mode, the segment descriptor for the safe store stack can be obtained and converted to a type 0 or 2 segment descriptor. (Refer to the LDD_n instruction description in Section 8.)

3.3.4.1 Standard Descriptor

The format of the standard descriptor is given below.



Bound

A 20-bit field that is the maximum valid byte address within the segment. Bits 0-17 are the word address, and bits 18-19 are the 9-bit byte address. The bound is relative to the base. A zero bound indicates a 1-byte segment if bit 27 is 1.

Memory Organization

Flags A 9-bit field that describes the access privileges as well as other control information associated with the descriptor:

Bit	Flag Code	Meaning
20	R	Read 0 Read not allowed 1 Read allowed
21	W	Write 0 Write not allowed 1 Write allowed
22	S	Store by STDn 0 Descriptor may not be stored in a type 1 or 3 segment by the STDn instruction. 1 Descriptor may be stored in a type 1 or 3 segment by the STDn instruction.
23	C	Cache Use Control 0 Cache is not used for fetches through this descriptor. 1 Cache is used for all memory references through this descriptor.
24	X	NS/ES Mode 0 NS Mode 1 ES Mode
25	E	Execute 0 Execute not allowed 1 Execute allowed
26	P	Privilege 0 Privilege Master Mode not required for execution 1 Privilege Master Mode required for execution
27	B	Bound Validity 0 Bound is not valid; segment is empty 1 Bound field is maximum valid address
28	A	Available Segment 0 Segment not available; references not allowed 1 Segment available; references are allowed

WSR A 3-bit field that specifies which of the eight working space registers to use with this descriptor. The working space register supplies the working space number (WSN).

Type A 4-bit field that defines the descriptor type. Two types for standard descriptors exist:

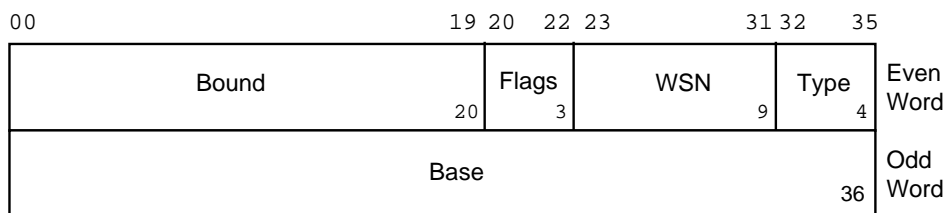
Type 0: the descriptor "frames" instruction/operand space; and

Type 1: the descriptor "frames" an address space containing descriptors.

Base A 36-bit virtual byte address that is relative to the working space defined in the WSR. Bits 0-33 are a 34-bit word address, and bits 34-35 represent a 9-bit byte within the word.

3.3.4.2 Standard Descriptor with Working Space Number

The format of the standard descriptor with working space number (WSN) is given below.



This format is the same as that for the standard descriptor except that the flags field has been truncated to allow the descriptor to contain the actual working space number rather than point to a working space register. The three flag bits are the same as the corresponding flag bits of the standard descriptor. The state of the truncated flags is assumed as follows:

- Flags**
- 1) Execute not allowed (NE)
 - 2) Not privileged (NP)
 - 3) Bound valid (B)
 - 4) Segment available (A)
 - 5) Bypass cache honored

WSN The actual working space number

Type The two types of the standard descriptor with WSN are
 Type = 2 the descriptor "frames" operand space, and
 Type = 3 the descriptor "frames" an address space containing descriptors.

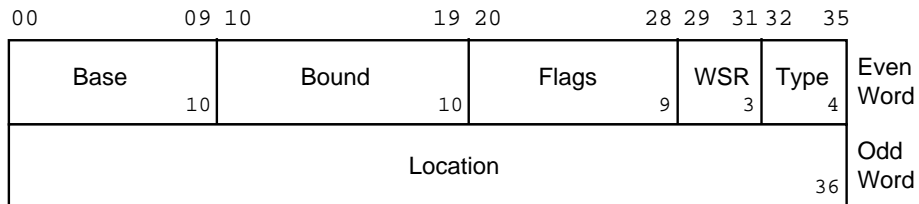
Memory Organization

3.3.4.3 Super Descriptor

When segments larger than 256K (2^{18}) words are required, super-descriptors are used to define the large segments. The definitions of the flags, WSR, WSN, and type fields of the super-descriptor are the same as those of the standard descriptor. The base and bound fields are automatically extended on the right to a length of 36 bits. The base is extended with zeros, and the bound is extended with ones.

Therefore, a super descriptor with base, location, and bound of zero describes a segment that begins at location zero of a working space and extends 2^{26} bytes (16 million words). A super descriptor with a base of 1, location of zero, and a bound of 3 describes a segment that starts at location 2^{26} and extends 2^{28} bytes (64 million words).

The format of the super descriptor is given below.



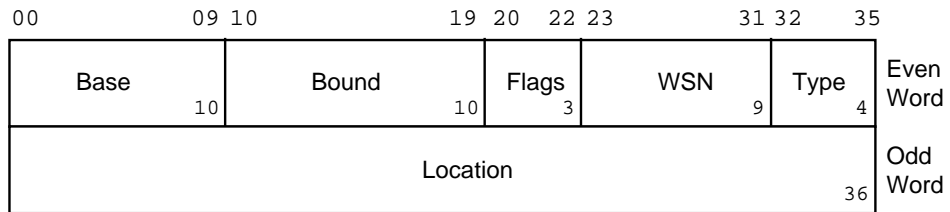
- Base** A 10-bit virtual address (unit 2^{26} bytes) within a working space. The 10-bit base is converted to a 36-bit base (unit 1 byte) by extending to the right by 26 zero bits.
- Bound** A 10-bit virtual address (unit 2^{26} bytes) that is the maximum valid address within the segment. Conversion to a 36-bit bound (unit 1 byte) is accomplished by extending the 10-bit field to the right by 26 one bits. The bound is relative to the base.
- Flags** A field that describes the access privileges associated with the descriptor (identical to the flags field for the standard descriptor).
- WSR** A 3-bit field that specifies which of the eight working space registers to use with this descriptor (identical to the WSR field for the standard descriptor, except bit 25 Must Be Zero (Execute Not Allowed)).
- Type** A 4-bit field that defines the type for the super descriptor.
Type = 4 The descriptor "frames" operand space.

Location A 36-bit byte virtual address relative to the base, that is, an offset from the 10-bit base. The area framed by the super descriptor extends from (Base + Location) through (Base + Bound).

NOTE: If an attempt is made to use a super descriptor in the ES mode, an IPR fault occurs.

3.3.4.4 Super Descriptor with Working Space Number

The format of the super descriptor with working space number (WSN) is given below.



This format is the same as that for the super descriptor except the truncated flags field contains three bits that are defined identically as the corresponding three bits of the standard descriptor. The state of the truncated flags is assumed as follows:

- Flags**
- 1) Execute not allowed (NE)
 - 2) Not privileged (NP)
 - 3) Bound valid (B)
 - 4) Segment available (A)
 - 5) Bypass cache honored

WSN The actual working space number

Type A 4-bit field that defines the descriptor type as "super with WSN"

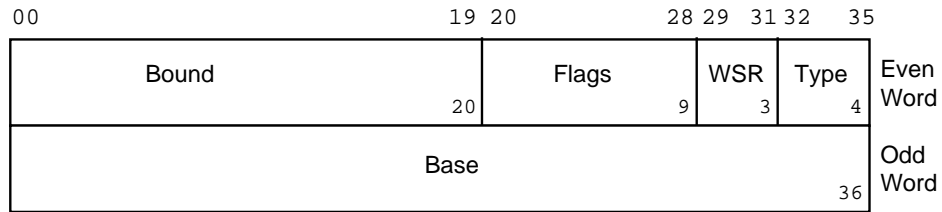
Type = 6 The descriptor "frames" operand space

NOTE: If an attempt is made to use a super descriptor with WSN in the ES mode, an IPR fault occurs.

Memory Organization

3.3.4.5 Extended Descriptor

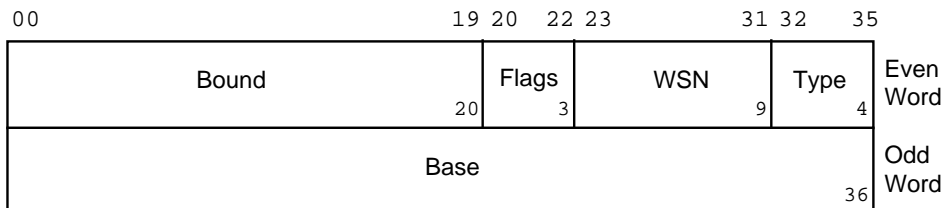
The format of the extended descriptor is given below.



- Bound** A 20-bit field that is the maximum valid byte address within the segment, modulo 2^{12} bytes (2^{10} words). In other words, the bound is in terms of 4096 byte pages. It is converted to a 36-bit byte bound by extending to the right of the 20-bit field by 12 1-bits and adding four zero-bits in the high-order. The bound is relative to the base.
- Flags** The same as in the standard descriptor
- WSR** The same as in the standard descriptor
- Type** Indicates the type for the descriptor
 Type = 12(10) for the extended descriptor
- Base** The same as in the standard descriptor

3.3.4.6 Extended Descriptor with Working Space Number

The format of the standard descriptor with working space number (WSN) is given below.



This format is nearly the same as for the Extended Descriptor (T = 12(10)), except that the flag field is shorter and a working space number (WSN) is specified.

Flags The three bits of the flag field are the same as the corresponding standard descriptor flag bits. The state of the truncated flags is assumed as follows:

- 1) Execute bit allowed
- 2) Not privileged (NP)
- 3) Bound valid (B)
- 4) Segment available (A)
- 5) Bypass cache honored

WSN The actual working space number

Type Indicates the type of the descriptor
 T = 14(10) indicates an Extended descriptor with WSN

3.3.5 Domains

Another logical element of the virtual environment is the domain. A domain is equal to those items that currently can be accessed. The domain exists as the primary mechanism for security protection. The domain is a flexible and temporary range of operation that may encompass several noncontiguous segments in one or more working spaces (see Figure 3-2). Two or more domains may interact by including the same segment. Each domain contains exactly one linkage segment to define the domain. A change of domain implies a change of linkage segment and vice versa. The linkage segment contains descriptors for the segments constituting the domain. Descriptors for the domain may be in descriptor segments described in the linkage segment, in descriptor registers, or in the parameter segment.

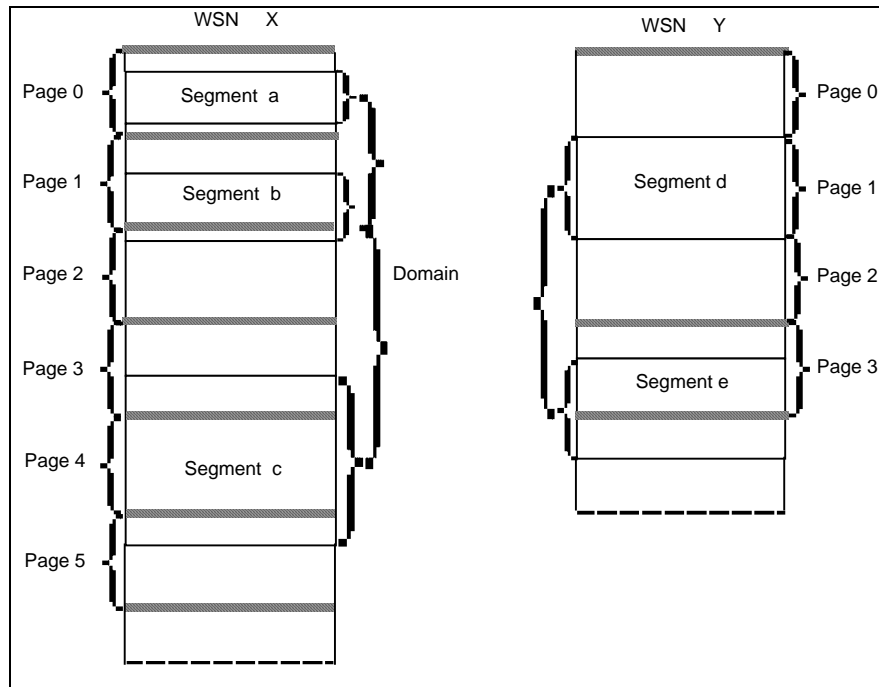


Figure 3-2. Domain Of Noncontiguous Segments

The safe store stack and the data stack segments are also associated with the process. The safe store stack is always used (except for GCLIMB and PCLIMB) in a change of domain, but a new domain may or may not choose to access a different portion of the data stack segment. It does not have access to that portion used by the calling domain.

Normally, a change of domain is accomplished through a succession of operations that are associated with the ICLIMB instruction. Starting with two separate domains, which for convenience are referred to as calling domain and called domain, the entry descriptor accessed in the calling domain describes the called-domain linkage segment and identifies a specific initial instruction in an instruction segment described in that linkage segment. The contents of the calling domain's registers (LSR, ASR, PSR, and DSAR), as well as those of any other registers specified by the type of entry descriptor, are safe stored.

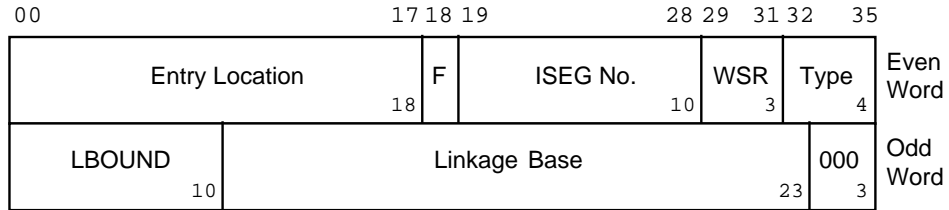
The change-of-domain CLIMB instruction indicates whether there are parameters and the number of arguments. The arguments may be either vectors or descriptors. (Refer to the discussion of LDDn instruction in Section 11.) If the arguments are vectors, descriptors are prepared for the vectors, stored in the parameter segment of the called domain, and the argument segment becomes empty.

The source of the list of vectors or descriptors is given as the contents of pointer register zero. (Descriptor register zero identifies the segment in which the list occurs and indicates whether vectors or descriptors are listed. Address register zero gives the offset in that segment of the list.) On change-of-domain return, the contents of the calling-domain's domain registers and any other register contents that were safe stored are restored.

Memory Organization

3.3.5.1 Entry Descriptor

An entry descriptor is required to call a new domain. The entry descriptor describes the linkage segment that defines the new domain, a segment containing instructions to be initially executed in the domain, and an offset relative to the origin of that segment to which control is transferred. The entry descriptor is used with the CLIMB instruction and has the following format:



Entry Location An 18-bit word address that is loaded into the instruction counter when the entry descriptor is used as an argument of the CLIMB instruction. The entry location is relative to the base of the new instruction segment.

F Bit 18 is the "store" permission bit and is interpreted the same as flag bit 22 of the standard and super descriptors.

ISEG No. The number of the descriptor to be loaded into the instruction segment register (ISR). The ISEG number is expressed in units of descriptors and is an index relative to the new linkage segment base. The ISEG number is extended with three zeros to be expressed in bytes and is also used in loading the SEGID (IS) register as follows:

Bits 0 - 1 = 11
Bits 2 -11 = ISEG No.

WSR The working space register containing the number of the working space to which the linkage base is relative

Type A 4-bit field that defines the entry descriptor type
Type = 8, 9, or 11; each number has a special meaning for the CLIMB instruction (determining the registers to be saved in the safe store stack upon change of domain)

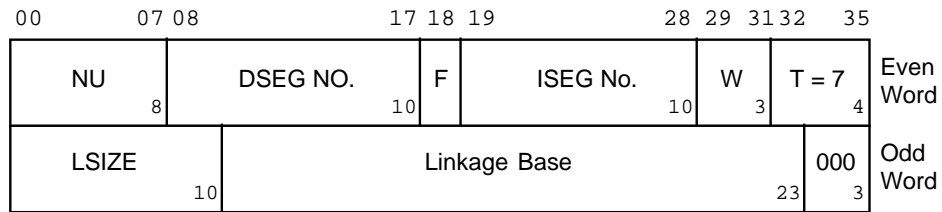
LBOUND The bound of the linkage segment expressed in units of descriptors. To form a standard descriptor bound, bound = 0000000||LBOUND||111.

Linkage base

The virtual starting address of the linkage segment relative to the working space defined by the working space register pointed to by the WSR field. When an entry descriptor is utilized, the associated linkage segment must be contained in the first 2**26 bytes of the working space. The last three bits of the linkage base are shown as zeros since the linkage segment must start on a double-word boundary. In actual practice, the hardware ignores the contents of these three bits.

3.3.5.2 Special Entry Descriptor

When the entry point is beyond the first 256K of the segment, the following special entry descriptor (T = 7) is used by the CLIMB instruction to transfer to EI mode.



This descriptor can only be loaded in the Descriptor Register n (DRn).

Explanation

NU Not used

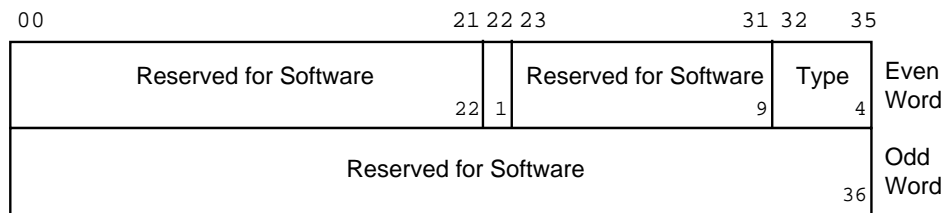
DSEG NO. This field is the descriptor number of a type 12 segment descriptor which defines the 34 bit entry location which is loaded into the Instruction Counter. The DSEG.NO is expressed in units of segment descriptor (i.e., modulo 2 words) and is the relative address from the base of the linkage segment.

F, ISEG NO., W, LBOUND, LINKAGE BASE These fields are the same as those for the entry descriptor of T = 8, 9, and 11.

3.3.5.3 Dynamic Linking Descriptor

The dynamic linking descriptor has a double-word format with a type field of T=5 entered in bits 32-35 of the even word. Bits 0-21, 23-31, and 36-71 are used to define how the linkage is to be resolved. Bit 22 indicates store permission. A dynamic linking fault occurs when the CLIMB instruction attempts to address through a dynamic linking descriptor. Any attempt by the STDn instruction to store a dynamic linking descriptor with the store permission bit (bit 22) of word 1 equal to zero in a type T=1 or 3 segment causes an SCL2 fault.

The dynamic linking descriptor has the following format:



Type A 4-bit field that defines the dynamic linking descriptor
Type = 5

NOTE: The software usually replaces this descriptor with a Type = 11 entry descriptor while processing a dynamic linking fault.

3.3.5.4 Shrinking

A feature commonly used to provide descriptor access control is called shrinking. This feature offers the only means available to the Slave mode for the creation of descriptors. In this process, a new descriptor of decreased scope is formed in one of the descriptor registers from a descriptor already available. In essence, a new subordinate segment identified by the shrunken descriptor is formed (see Figure 3-3).

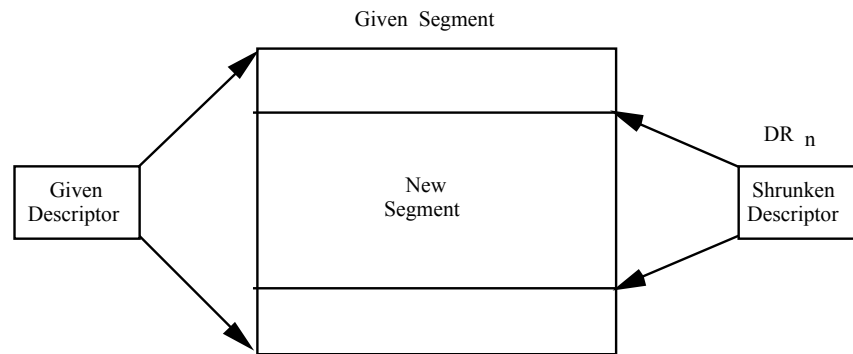


Figure 3-3. Shrunken Descriptor For Corresponding New Segment

Shrinking is used in a number of ways:

- to prepare parameter descriptors for another domain,
- to facilitate access to portions of the domain, and
- to restrict access to specific shared portions of the domain.

Shrinking operations may be performed on both standard and super descriptors, but the result is always a standard descriptor. A shrunken descriptor may be stored in a descriptor segment on a housekeeping page or in the descriptor stack addressable by the Argument Stack Register (ASR). Storing requires the descriptor to be stored to have store permission.

Shrinking uses a Load Descriptor Register n (LDDn) instruction, or a domain call, or the transfer version of the CLIMB instruction (ICLIMB or PCLIMB). In each instance, operands are used to define the shrinking operation in terms of a base address, size, and segment. The operands are called vectors and each is composed of two contiguous words. Each vector specifies one of the following functions to be performed by the instruction: copy descriptor, normal shrink, or data stack shrink. An operand of a LDDn instruction may be in the same segment as the LDDn instruction or in another segment. If the operand is in a descriptor segment, it is a descriptor, not a vector, and replacement occurs rather than shrinking.

A companion of the vector is an internal offset (a combination of a segment identifier (SEGID) and an address value) called a pointer. A pointer, in NS mode, is a 36-bit operand with sufficient information to identify an operand within a domain. Since a pointer is relative to a domain, it can be used only to address operands within its domain. Pointers for one domain cannot be used in another domain, but pointers can be exchanged and used by several instruction segments within a domain.

A pointer in both ES and EI modes is a 2-word construct containing the same information of segment identifier (SEGID) and address offset value.

4. Processor Accessible Registers

This section of the Programmer's Guide describes Processor accessible registers, organized as follows:

- Section 4.1, Accumulator Register (A)
- Section 4.2, Accumulator-Quotient Register (AQ)
- Section 4.3, Address Match Register (AMR)
- Section 4.4, Address Registers (AR_n)
- Section 4.5, Argument Stack Register (ASR)
- Section 4.6, Calendar Clock (CC)
- Section 4.7, Data Stack Address Register (DSAR)
- Section 4.8, Data Stack Descriptor Register (DSDR)
- Section 4.9, Debug Mode Register (DMR)
- Section 4.10, Exponent Register (E)
- Section 4.11, Exponent-Accumulator-Quotient Register (EAQ)
- Section 4.12, Fault Register (FLTR)
- Section 4.13, General Index Registers (GX_n)
- Section 4.14, IC History Registers (ICHR)
- Section 4.15, Index Registers (X_n)
- Section 4.16, Indicator Register (IR)
- Section 4.17, Instruction Counter (IC)
- Section 4.18, Instruction Segment Register (ISR)
- Section 4.19, Instruction Segment Identity Register - SEGID(IS)
- Section 4.20, Interrupt Registers (INTR_p)
- Section 4.21, Linkage Segment Register (LSR)
- Section 4.22, Low Operand Register (LOR)
- Section 4.23, Option Register (OR)
- Section 4.24, Page Directory Base Register (PDBR)
- Section 4.25, Parameter Segment Register (PSR)
- Section 4.26, Quotient Register (Q)
- Section 4.27, Safe Store Register (SSR)
- Section 4.28, Segment Descriptor Registers (DR_n)
- Section 4.29, Segment Identity Registers (SEGID_n)
- Section 4.30, Stack Control Register (SCR)
- Section 4.31, Timer Register (TR)
- Section 4.32, Virtual Machine Timer Register (VMTR)
- Section 4.33, Working Space Registers (WSR_n)

A processor register is a hardware assembly that holds information for use in some specified manner. An accessible register is a register whose contents are available to the user. Some accessible registers are explicitly addressed by particular instructions, and some are implicitly referenced during the execution of instructions. Some are used in both ways. The accessible registers are listed in Table 4-1. Refer to Sections 8–15, "Machine Instruction Descriptions", for a discussion of each instruction to determine the way in which the registers are used.

Table 4-1. Processor Accessible Registers

Register Name	Mnemonic	Length (# bits)	Quantity
Accumulator Register	A	36	1
Accumulator-Quotient Register	AQ	72	1
Address Match Register	AMR	52	1
Address Registers	ARn	24/36	8
Argument Stack Register	ASR	72	1
Calendar Clock ²	CCL	52	1
Data Stack Address Register	DSAR	17	1
Data Stack Descriptor Register	DSDR	72	1
Debug Mode Register	DMR	6	1
Exponent Register	E	8	1
Exponent-Accumulator-Quotient Register ¹	EAQ	80	1
Fault Register	FLTR	72	1
General Index Registers	GXn	36	8
IC History Register	ICHR	72	1024
Index Registers	Xn	18	8
Indicator Register	IR	18	1
Instruction Counter	IC	18/34 *	1
Instruction Segment Register	ISR	72	1
Instruction Segment Identity Register	SEGID(IS)	12	1
Interrupt Registers	INTRp ³	9	16
Linkage Segment Register	LSR	72	1
Low Operand Register	LOR	72	1
Option Register	OR	2	1
Page Directory Base Register	PDBR	18/36 ²	1
Parameter Segment Register	PSR	72	1
Quotient Register	Q	36	1
Safe Store Register	SSR	72	1
Segment Descriptor Registers	DRn	72	8
Segment Identity Registers	SEGIDn	12	8
Stack Control Register	SCR	2	1
Timer Register	TR	27	1
Upper Limit Address Register	ULAR	16	1
Virtual Machine Timer Register	VMTR	27	1
Working Space Registers	WSRn	9/18 ³	8

* 34 bits in EI mode.

Processor Accessible Registers

- 1 These registers are not separate physical assemblies but combinations of their constituent registers.
- 2 In SV mode, PDBR length is 18 bits.
- 3 In SV mode, WSR_n length is 9 bits. In VR mode, WSR_n length is 18 bits.

In the descriptions that follow, the diagrams given for register formats do not imply that a physical assembly possessing the pictured bit pattern actually exists. The diagram is a graphic representation of the form of the register data as it appears in memory when the register contents are stored or of how data bits must be assembled for loading into the register.

If the diagrams contain the character "x" or "0", the value of the bit in the position shown is irrelevant to the register. Bits pictured as "x" are not changed in the receiving cell when the register is stored. Bits pictured as "0" are set to 0 in the receiving cell when the register is stored. Neither "x" bits nor "0" bits are loaded into the register.

Accumulator Register (A)

4.1 Accumulator Register (A)

Format:

36 bits

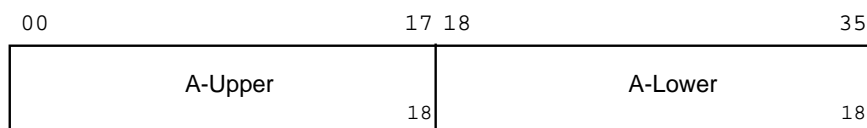


Figure 4-1. Accumulator Register (A) Format

Description:

A 36-bit physical register

Function:

In fixed-point binary instructions, holds operands and results

In floating-point binary instructions, holds the most significant part of the mantissa and the result

In shifting instructions, holds original data and shifted results

In address preparation, may hold two logically independent offsets, A-upper and A-lower, or an extended range bit- or character-string length

Accumulator-Quotient Register (AQ)

4.2 Accumulator-Quotient Register (AQ)

Format:

72 bits

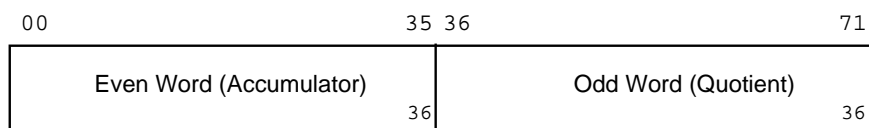


Figure 4-2. Accumulator-Quotient Register (AQ) Format

Description:

A combination of the accumulator (A) and quotient (Q) registers

Function:

In fixed-point binary instructions, holds double-precision operands and results.

In floating-point binary instructions, holds the mantissa and the result.

In shifting instructions, holds original data and shifted results.

Address Match Register (AMR)

4.3 Address Match Register (AMR)

Format:

52 bits

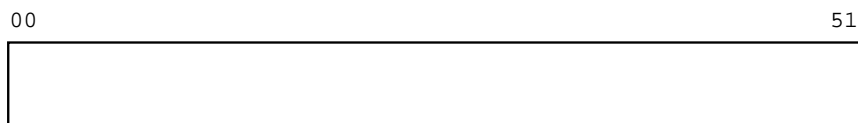


Figure 4-3. Address Match Register (AMR) Format

Description:

Contains Virtual Byte Address or Real Memory Byte Address dependent on addressing and debug mode setting.

Function:

When $DMR(1) = 0$, the AMR contains a 52-bit Virtual Byte Address. In SV or SVMX mode the upper 9 bits of AMR must be set to zero.

When $DMR(1) = 1$ the lower 34 bits of AMR specify a real memory byte address and the upper 18 bits must be set to zero.

Address Registers (AR_n)

4.4 Address Registers (AR_n)

Format:

24 bits each(NS Mode)

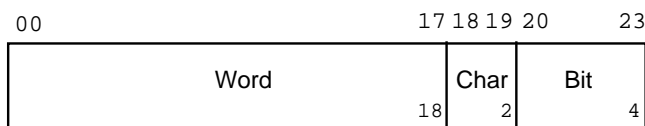


Figure 4-4. Address Register (AR_n) Format (NS Mode)

Description:

Eight 24-bit physical registers numbered 0 through 7 that are associated with the operand descriptor registers (DR_n) and that allow address modification on a word, character, or bit basis

Function:

The address registers provide address modification to the word, byte, and bit level:

- Word 18 bits (0-17); a word offset within the segment described by the associated operand descriptor register
- Char 2 bits; designates one of the four 9-bit characters (bytes) of which the word is composed
- Bit 4 bits; designates one of the 9 bits within the character.

Address Registers (AR_n)

Format:

36 bits each(ES/EI Mode)

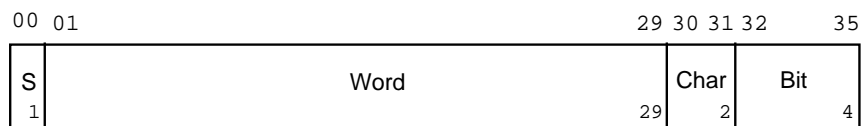


Figure 4-5. Address Register (AR_n) Format (ES/EI Mode)

Description:

Eight 36-bit physical registers numbered 0 through 7 that are associated with the operand descriptor registers (DR_n) and that allow addressing on a word, character, or bit basis.

Function:

In ES/EI mode, each address register is extended to 36 bits. The AR_n is as given in two's complement form, with bit 0 as sign bit. In the effective address generation, bit 0 is extended 4 bits to the left.

Word	29 bits (1-29); a word offset within the segment described by the associated operand descriptor register
Char	2 bits; designates one of the four 9-bit characters (bytes) of which the word is composed
Bit	4 bits; designates one of the 9 bits within the character

Argument Stack Register (ASR)

4.5 Argument Stack Register (ASR)

Format:

72 bits

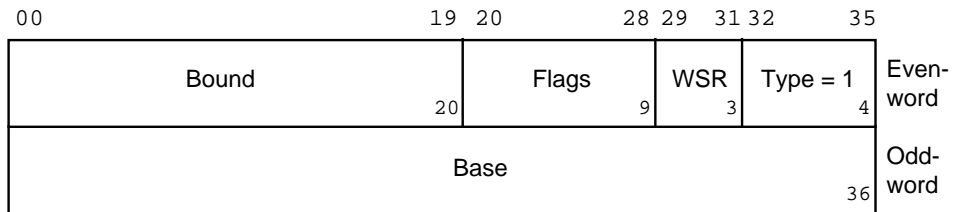


Figure 4-6. Argument Stack Register (ASR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that describes (or frames) the argument stack of the current domain of the currently executing process

Function:

Instructions are provided for loading (Privileged Master Mode) and storing the argument stack register. The argument stack register is utilized by and may have its contents changed by the hardware during the execution of a Save Descriptor Register (SDR_n) or CLIMB instruction. When the bound field of the ASR is loaded, bits 0-6 are forced to zero. If flag-bit 27 = 1 (bound valid), bits 17-19 are forced to 111. Thus, the size of the argument stack is effectively limited to 1024 descriptors.

Calendar Clock (CCL)

4.6 Calendar Clock (CC)

Format:

52 bits

Description:

The DPS 9000TA CC is a 52-bit register located in the SCU that is incremented every microsecond. The V9000 converts the second count and the microsecond count returned by LINUX's "gettimeofday()" function to microseconds. V9000 emulates the LCCL instruction by saving the difference between the microseconds specified in the LCCL operand and the microseconds from the current "gettimeofday()" function.

Data Stack Address Register (DSAR)

4.7 Data Stack Address Register (DSAR)

Format:

17 bits

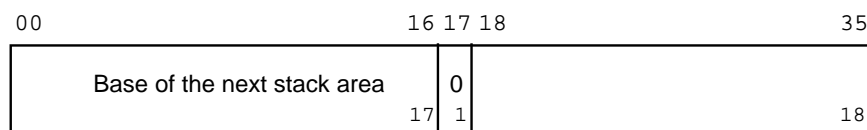


Figure 4-7. Data Stack Address Register (DSAR) Format

Description:

A 17-bit special-purpose index register that points to the next available double-word location within the data stack area of memory framed by the Data Stack Descriptor Register (DSDR); bit 17 is always zero

Function:

Privileged Master Mode instructions (LDDSA and STDSA) are available for loading and storing the Data Stack Address Register. The contents of the DSAR may be altered during the execution of the Load Descriptor Register (LDDn) instruction, Load Data Stack Address Register (LDDSA) instruction, or CLIMB instruction.

Data Stack Descriptor Register (DSDR)

4.8 Data Stack Descriptor Register (DSDR)

Format:

72 bits

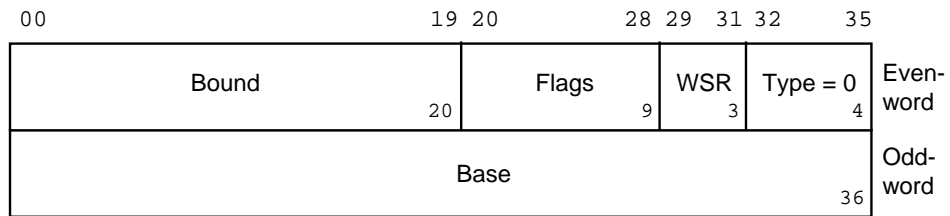


Figure 4-8. Data Stack Descriptor Register (DSDR) Format

Description:

A 72-bit register located in the virtual unit that holds a type 0 standard descriptor that frames the data stack area of memory for the current process

Function:

Privileged Master Mode instructions (LDDSD and STDSD) are available for loading and storing the data stack descriptor register. The contents of the data stack descriptor register are used by the hardware when the vector of the Load Descriptor Register (LDD_n) or CLIMB instruction indicates that a working data stack descriptor is to be generated.

Debug Mode Register (DMR)**4.9 Debug Mode Register (DMR)****Format:**

6 bits

Description:

The Debug Mode Register is used to enable/disable address traps on Virtual or Real addresses, on Store or Read, on Instruction or Operand, and enable or disable WIC faults.

Function:

Bit(s)	Function
0	0 Disable Address trap 1 Enable Address trap
1	0 Virtual address type in Address Match Register 1 Real address type in Address Match Register
2	0 Operand 1 Instruction (ignore bits 3-4)
3-4	00 Unused/disabled 01 Read for basic operation; controller 1 for EIS operation 10 Store for basic operation; controller 2 for EIS operation 11 Read or Store
5	0 WIC fault disabled 1 WIC fault enabled

Exponent Register (E)

4.10 Exponent Register (E)

Format:

8 bits

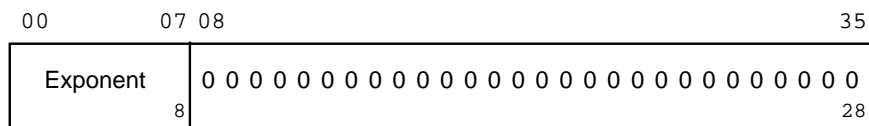


Figure 4-9. Exponent Register (E) Format

Description:

An 8-bit physical register

Function:

In floating-point binary instructions, holds the exponent value.

Exponent-Accumulator-Quotient Register (EAQ)

4.11 Exponent-Accumulator-Quotient Register (EAQ)

Format:

80 bits



Figure 4-10. Exponent-Accumulator-Quotient Register (EAQ) Format

Description:

A combination of the exponent (E), accumulator (A), and quotient (Q) registers

Although the combined register has a total of 80 bits, only 72 are involved in transfers to and from main memory. The low-order 8 bits are discarded on store and zero-filled on load (that is, Q-register bits 28-35 are zero on load; bits 64-71 of the AQ Register are ignored). See "Floating-Point Arithmetic Instructions" documented in Section 7.

Function:

In floating-point binary instructions, holds operands and results.

Fault Register (FLTR)

4.12 Fault Register (FLTR)

Format:

72 bits

Description:

The Fault register is set by hardware at the time of a fault.

Function:

The Fault register can be accessed via the Store Fault Register instruction. The FLTR content is set with only those bits associated with the fault type altered.

DPS 9000G2/DPS 9000TA/V9000 Systems:

Bit(s)	Function
0	if = 1 on IPR: Illegal Op Code
1-5	* reserved *
6	if = 1 on IPR: Illegal Decimal Digit
7	* reserved *
8	* unused *
9	if = 1 on SCL2: Attempted write to page whose write enable bit is not set
10	* unused * Forced to zero.
11-17	* reserved for hardware use *
18-23	* unused * Forced to zero.
24-71	* reserved *

General Index Registers (GX_n)

4.13 General Index Registers (GX_n)

Format:

36 bits (ES/EI Mode)



Figure 4-11. General Index Registers (GX_n) Format

Description:

Eight 36-bit physical registers numbered 0 through 7 used in ES/EI mode only; general register data may occupy the entire 36-bit operand

Function:

May be used as a data operand register with fixed-point operations

However, in the ES mode, GX_n registers may be used as the single-precision operand register.

IC History Registers (ICHR)

4.14 IC History Registers (ICHR)

Format:

72 bits each

	00 01 02	17 18	35 36	47 48	71
NS:	00	zeros	From IC value	From SEGID(IS)	zeros
EI:	00	Transfer From IC value		From SEGID(IS)	zeros

Figure 4-12. IC History Registers (ICHR) Format

Description:

The V9000 ICHR is a 1024 by 2-word circular table. Each double word entry identifies the source IC and SEGID value of an instruction causing a transfer. Each entry may specify the destination IC and ISR or WSRn as well as the source IC of the transfer.

The V9000 ICHR is not locked on fault. However, the 16 most recent entries are copied to a locked ICHR buffer on fault if ICHR is currently unlocked when the fault occurs.

Function:

The ICHR consists of 1024 double-word entries where each entry consists of:

<u>Bit(s)</u>	<u>Function</u>
0-1	must be zero
2-17 (NS)	zeros
18-35 (NS)	From IC value
2-35 (EI)	Transfer From IC value
36-47	From SEGID(IS)
48-71	zeros

Index Registers (X_n)

4.15 Index Registers (X_n)

Format:

18 bits each (NS Mode)

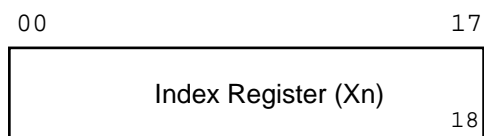


Figure 4-13. Index Register (X_n) Format

Description:

Eight 18-bit physical registers numbered 0 through 7

Index register data may occupy the position of either an upper or lower 18-bit half-word operand.

Indicator Register (IR)

<u>Key (bit)</u>	<u>Indicator Name</u>	<u>Action</u>
c (20)	Carry	<p>This indicator is set ON for any of the following conditions:</p> <ul style="list-style-type: none"> • If a bit propagates leftward out of bit 0 of the main binary adder for any binary or left-shifting instruction; • If $value1 \leq value2$ for a decimal numeric comparison instruction; or • If $char1 \leq char2$ for a decimal alphanumeric comparison instruction. • Otherwise, it is set OFF.
d (21)	Overflow	<p>This indicator is set ON if the arithmetic range of a register is exceeded in a fixed-point binary instruction or if the target string of a decimal numeric instruction is too small to hold the integral part of the result. It remains ON until reset by the Transfer On Overflow (TOV) instruction or reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator below.)</p>
e (22)	Exponent Overflow	<p>This indicator is set ON if the exponent of the overflow result of a floating-point binary or decimal numeric instruction is greater than +127. It remains ON until reset by the Transfer On Exponent Overflow (TEO) instruction or reset by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator below.)</p>
f (23)	Exponent Underflow	<p>This indicator is set ON if the exponent of the underflow result of a floating-point binary or decimal numeric instruction is less than -128. It remains ON until reset by the Transfer On Exponent Underflow (TEU) instruction or by some other instruction that loads the IR. The event that sets this indicator ON may also cause an overflow fault. (See overflow mask indicator.)</p>

Indicator Register (IR)

<u>Key (bit)</u>	<u>Indicator Name</u>	<u>Action</u>
g (24)	Overflow Mask	This indicator is set to ON or OFF only by the LDI, RET, and CLIMB instructions. When set ON, it inhibits the generation of the fault for those events that normally cause an overflow fault. When the overflow mask is ON, no overflow fault is generated if either the overflow or the exponent overflow indicator is set to ON status. When the overflow mask is set OFF, an overflow fault is generated if either the overflow or the exponent overflow indicator is set to ON status. If the overflow mask indicator is set OFF after an overflow event occurs, an overflow fault does not occur, even though the indicator for that event is still set ON. The state of the overflow mask indicator does not affect the setting, testing, or storing of any other indicator or the overflow fault caused by the truncation indicator.
h (25)	Tally Runout	This indicator is set OFF at initialization of any tallying operation. It is then set ON for any of the following conditions: <ul style="list-style-type: none"> • If any Repeat instruction terminates because of tally exhaust; • If a Repeat Link (RPL) instruction terminates because of a zero link address; • If a tally exhaust is detected for an Indirect then Tally modifier; the instruction is executed whether or not tally exhaust occurs; or • If a string scanning instruction reaches the end of the string without finding a match condition.
i (26)	* unused *	This indicator has no meaning to the hardware.
j (27)	* ignored *	This indicator is ignored.
k (28)	Master Mode	This indicator is set ON for an interrupt acceptance, a fault acceptance, execution of a PMME instruction, and the execution of an OCLIMB instruction (when the master mode bit of the indicator register to be restored is ON).

Indicator Register (IR)

<u>Key (bit)</u>	<u>Indicator Name</u>	<u>Action</u>
1 (29)	Truncation	<p>This indicator is affected only by multiword instructions. It is set to ON during string instructions when the source string length is greater than the destination string length, and set to OFF when the reverse is true. For decimal arithmetic instructions, it is set to ON when there are no rounding specifications, and the lowest digit, or more of the result, is truncated. It is set to OFF when the reverse is true. The bit is not set if both the truncated value and the result are zero. When the highest nonzero digit is lost, the Overflow Indicator is set ON.</p>
m (30)	Multi-word Instruction Interrupt	<p>This indicator is set OFF by the execution of the SPL instruction and by the end of execution of all interrupt multiword instructions. The indicator has meaning only when determining the proper restart resequence for an interrupted multiword instruction. This indicator is set ON by the following conditions:</p> <ul style="list-style-type: none"> • When any fault or interrupt occurs during the execution of a multiword instruction (except CLIMB and vector instructions); or • When any fault or interrupt occurs during the execution of a vector instruction. <p>The ON state of this indicator is used during the CLIMB instruction (after a fault or interrupt); for example, to save the pointers and lengths data in order to resume the instruction.</p>

Indicator Register (IR)

<u>Key (bit)</u>	<u>Indicator Name</u>	<u>Action</u>
n (31)	Exponent Underflow Mask	<p>This indicator can be set ON or OFF only by the LDI, RET, or CLIMB instructions. When the exponent mask underflow mask is set ON, no overflow fault is generated when the exponent underflow indicator is set to ON status. In this instance, if the exponent underflow indicator is set to ON with binary or hexadecimal floating-point instructions (including ADE), the exponent of the result is set to -128, the mantissa of the result is 0, the zero indicator is set to ON, the negative indicator set to OFF, and instruction execution is continued. (See Note below.)</p> <p>With instructions having decimal floating-point data as results, when the exponent underflow mask indicator is ON and the exponent underflow indicator is set to ON, the exponent of the result is stored as +127, and the mantissa of the result is stored as +0.</p> <p>An overflow fault does not occur when the overflow mask indicator is ON, even when the exponent underflow mask indicator is set to OFF and the exponent underflow indicator is set to ON.</p> <p>The status of the exponent underflow mask indicator does not affect the setting, testing, or storing of the exponent underflow indicator.</p> <p>NOTE: The A and Q registers remain unchanged when the exponent underflow mask is set ON by an ADE instruction.</p>
p (32)	Hexadecimal Exponent Mode	<p>This indicator is set ON or OFF only by the instructions that load the IR.</p> <p>NOTE: When set ON, it causes the floating-point instructions to be executed in the hexadecimal exponent mode.</p>
q (33)	Fixed-Point Overflow Mask	<p>This indicator is used to mask fixed-point binary and decimal overflows.</p>
(34-35)	unused	<p>Bits 34-35 must be zero (MBZ).</p>

Instruction Counter (IC)

4.17 Instruction Counter (IC)

Format:

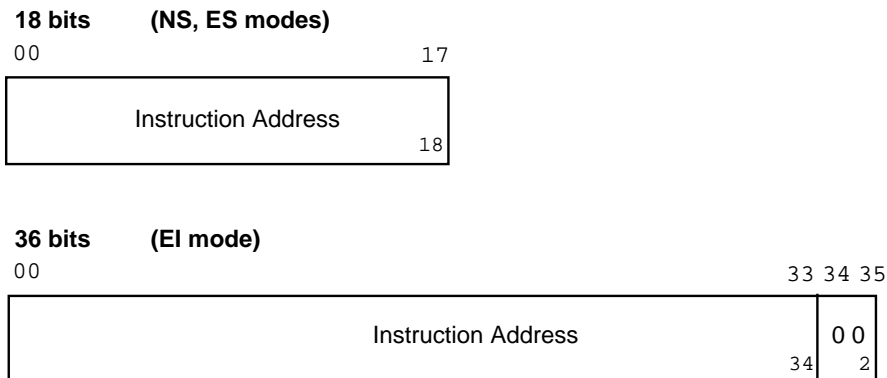


Figure 4-15. Instruction Counter (IC) Format

Description:

An 18-bit physical register(NS, ES modes)

A 36-bit physical register(EI mode)

Function:

Holds the address of the current instruction being executed

The IC is incremented by 1 by the control unit for the sequential execution of single-word instructions or by the appropriate amount (2, 3, or 4) for multiword instructions. The content of the IC is changed by a transfer-of-control instruction or by a fault or interrupt.

A description of faults and interrupts is contained in Section 6.

Instruction Segment Register (ISR)

4.18 Instruction Segment Register (ISR)

Format:

72 bits

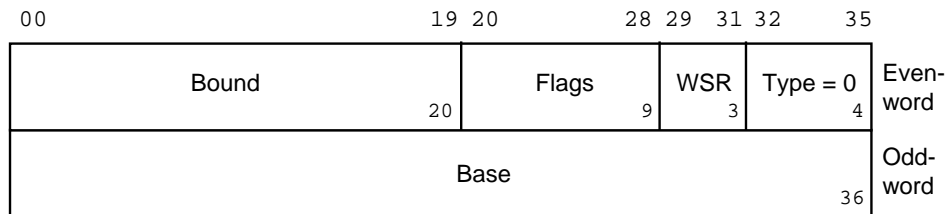


Figure 4-16. Instruction Segment Register (ISR) Format

Description:

A 72-bit register that holds a type 0 standard descriptor (or type 12 for EI mode) that describes the current instruction segment for the current domain of the currently executing process

Function:

The instruction segment register may not be loaded or stored directly. The register is loaded during the execution of a CLIMB or transfer instruction with bit 29 ON. The ISR may be stored indirectly by moving its contents to an operand descriptor register (DR_n) and then storing DR_n . If bit 29 of an instruction word is zero or the AR bit in the MF field of a multiword instruction is zero, the instruction segment register is used in forming the virtual address of the operand. The base and bound values placed in the ISR are constrained: the 5 least significant bits of the base field must be zero and the 5 least significant bits of the bound field must be ones.

ES mode is in effect when ISR bit 24 = 1. EI mode is in effect when ISR bit 24 = 1 and the descriptor type is 12 (max bound = 4 GB). The IC is extended to 34 bits.

NOTE: When ISR bit 24 = 0 and the ISR type is 12, an IPR fault occurs.

Instruction Segment Identity Register - SEGID (IS)

4.19 Instruction Segment Identity Register - SEGID(IS)

Format:

12 bits



Figure 4-17. Instruction Segment Identity Register - SEGID(IS) Format

Description:

A 12-bit register that is associated with the instruction segment register (ISR) in the same manner that a SEGID_n register is associated with an operand descriptor register (DR_n); points to the source of the descriptor in the ISR.

Function:

The instruction segment identity register may not be loaded or stored directly. It is loaded with the identity of the source of the descriptor when a transfer or CLIMB instruction loads the Instruction Segment Register (ISR). The S and D field codes used in these registers indicate the origin of the descriptor. See the SEGID_n codes.

Interrupt Registers (INTR_p)

4.20 Interrupt Registers (INTR_p)

Format:

9 bits each

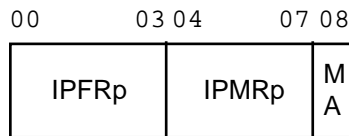


Figure 4-18. Interrupt Registers (INTR_p) Format

Description:

CPUs contain a set of 16 Interrupt Registers (INTR_p). Each interrupt register is associated with one of 16 System Identification numbers (p = SID) with SID = 0 reserved for the Host System and the remaining assigned to up to 15 Guest Operating Systems.

Function:

Each Interrupt register (INTR_p) consists of three parts: a 4-bit Interrupt Present Flag register (IPFR_p), a 4-bit Interrupt Mask Flag register (IPMR_p) and mask all flag (ALLF_p).

Bit(s)	Function
0	IPFR _p (0) = 1 if a Fault type interrupt is present
1	IPFR _p (1) = 1 if a Terminate type interrupt is present
2	IPFR _p (2) = 1 if a Marker type interrupt is present
3	IPFR _p (3) = 1 if a Special type interrupt is present
4-7	IPMR _p (0-3) Mask corresponding to the four Interrupt Present flags IPFR _p (0-3). If an IPMR _p bit is set, that corresponding interrupt type is masked off.
8	ALLF _p If = 0; all types of interrupt for system p are ignored independent of the content of IPFR _p or IPMR _p .

Linkage Segment Register (LSR)

4.21 Linkage Segment Register (LSR)

Format:

72 bits

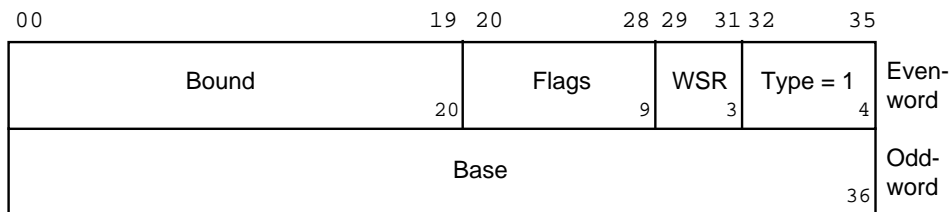


Figure 4-19. Linkage Segment Register (LSR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that describes the linkage segment of the current domain of the currently executing process

Function:

The LSR is loaded only by executing a CLIMB instruction. It may be stored by transferring its contents to an operand descriptor register (DR_n) and then storing DR_n. When the bound field of the LSR is loaded, bits 0-6 are forced to zero and bits 17-19 are forced to 111. Thus, the size of the linkage segment is effectively limited to 1024 descriptors.

Low Operand Register (LOR)

4.22 Low Operand Register (LOR)

Format:

72 bits

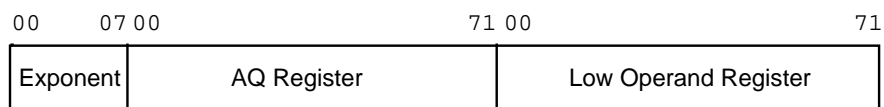


Figure 4-20. Low Operand Register Format

Description:

The low operand register (LOR) functions in combination with the exponent (E), accumulator (A), and quotient (Q) registers in quadruple-precision floating-point operations.

Function:

The 72-bit low operand register is used for the lower mantissa of quadruple-precision (four words) with floating-point operations.

Option Register (OR)

4.23 Option Register (OR)

Format:

2 bits

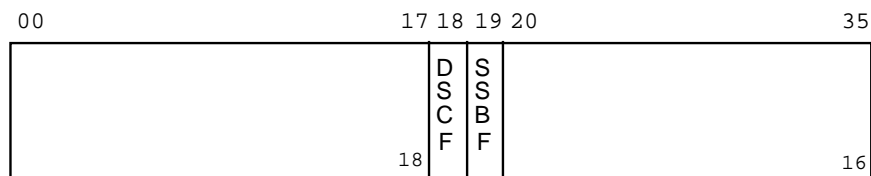


Figure 4-21. Option Register (OR) Format

Description:

A 2-bit register located in the virtual unit that controls the clearing of data stack space, bypassing the safe store portion of an inward CLIMB (ICLIMB) instruction, and bypassing cache memory

Bit 18 is the Data Stack Clear Flag (DSCF) and bit 19 is the Safe Store Bypass Flag (SSBF).

Function:

The option register is loaded with the Load Option Register (LDO) instruction and stored with the Store Option Register (STO) instruction.

Page Directory Base Register (PDBR)

4.24 Page Directory Base Register (PDBR)

Format:

18 bits in SV mode

36 bits in SVMX mode

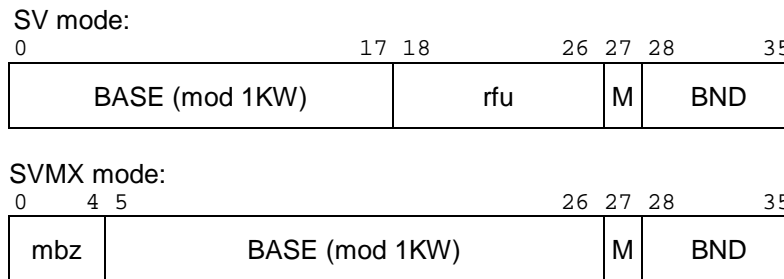


Figure 4-22. Page Directory Base Register (PDBR) Format

Description:

In SV mode: A 36-bit, modulo 1024-word register that contains the 18-bit base address of the Working Space Page Table Directory (WSPTD) or Directory Base Table (DBT), a 1-bit M indicator, and an 8-bit Bound specification of the WSPTD or DBT. Bits 27-35 are ignored in SV mode.

In SVMX mode: A 36-bit, modulo 1024-word register that contains a 5-bit RFU, the 22-bit base address, a 1-bit M indicator, and an 8-bit Bound specification. Bits 27-35 are ignored in SVMX mode.

Page Directory Base Register (PDBR)

Function:

SV mode:

Bit(s)	Function
0-17	BASE Base address of the WSPTD or DBT in units of 1024 words. The BASE field with 10 zeroes appended on the right form a 28-bit real word address.
18-26	RFU Reserved for future use. Must Be Zero
27	M bit (ignored in SV mode) 0 = DD mode (WSPTD) 1 = ID mode (DBT)
28-35	BND (ignored in SV mode) Bound specification of the WSPTD(M=0) or DBT(M=1). When M=0 the bound of the WSPTD is BNDf with 11 ones appended on the right. When M=1 the bound of the DBT is BNDf with 1 one appended on the right.

SVMX mode:

Bit(s)	Function
0-4	RFU Reserved for future use
5-26	BASE Base address in units of 1024 words. The BASE field with 10 zeroes appended on the right form a 32-bit real word address.
27	M bit (ignored in SVMX mode) 0 = DD mode (WSPTD) 1 = ID mode (DBT)
28-35	BND (ignored in SVMX mode) Bound specification.

Privileged Master Mode instructions (LPDBR, SPDBR) are available for loading and storing the page directory base register.

Parameter Segment Register (PSR)

4.25 Parameter Segment Register (PSR)

Format:

72 bits

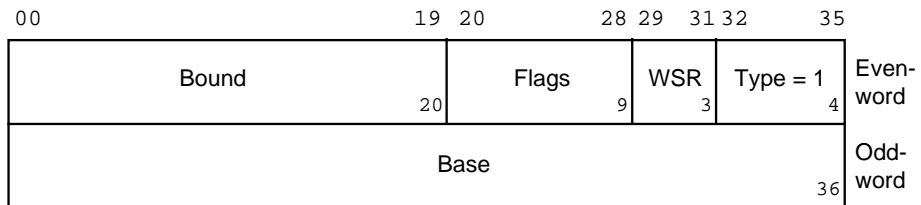


Figure 4-23. Parameter Segment Register (PSR) Format

Description:

A 72-bit register that holds a type 1 standard descriptor that frames the parameter segment of the current domain of the currently executing process

Function:

Instructions are provided for loading (Privileged Master Mode) and storing the parameter segment register. The parameter stack register is utilized by and may have its contents changed by the hardware during the execution of the CLIMB instruction. When the bound field of the PSR is loaded, bits 0-6 are forced to zero; if flag-bit 27 = 1 (bound valid), bits 17-19 are forced to 111. Thus, the size of the parameter segment is effectively limited to 1024 descriptors.

Quotient Register (Q)

4.26 Quotient Register (Q)

Format:

36 bits

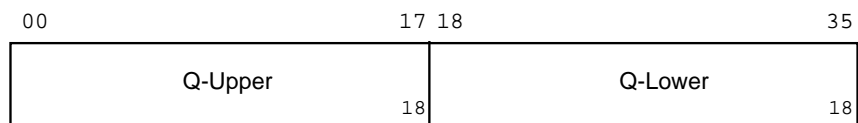


Figure 4-24. Quotient Register (Q) Format

Description:

A 36-bit physical register

Function:

In fixed-point binary instructions, holds operands and results

In floating-point binary instructions, holds the least significant part of the mantissa

In shifting instructions, holds original data and shifted results

In address preparation, may hold two logically independent offsets, Q-upper and Q-lower, or an extended range bit- or character-string length.

Safe Store Register (SSR)

4.27 Safe Store Register (SSR)

Format:

72 bits

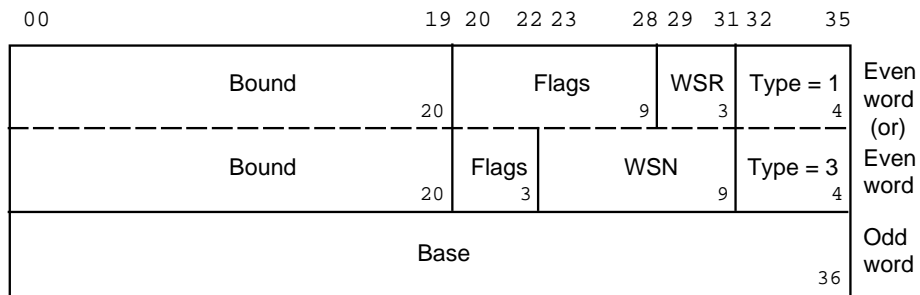


Figure 4-25. Safe Store Register (SSR) Format

Description:

A 72-bit register located in the virtual unit that holds either a Type 1 or 3 standard descriptor that describes the safe store stack of the current process

NOTE: The format for a Type 3 descriptor differs in that the Flags field is truncated at bit 22 to allow the descriptor to contain the actual working space number (WSN) rather than point to a Working Space Register (WSR).

Function:

The safe store register describes the safe store stack of the current process. The SSR is loaded and stored with the Privileged Master mode instructions LDSS and STSS. A 2-bit hardware Stack Control Register (SCR) is associated with the safe store register. This register's content determines the size of the safe store frame. (Refer to the SCR section below.)

Segment Descriptor Registers (DR_n)

4.28 Segment Descriptor Registers (DR_n)

Format:

72 bits each

Description:

Eight 72-bit registers that hold segment descriptors describing address space contained within the current domain of the currently executing process

The format of the descriptors matches the content of the type fields. Type fields 0, 2, 4, 6, 12, and 14 are used for operand segments and type fields 1 and 3 are used for descriptor segments.

Function:

Instructions are available for loading and storing the segment descriptor registers and for modifying their contents. A segment descriptor register is invoked for developing a virtual operand address when bit 29 of the instruction is 1. Address bits 0, 1, and 2 specify which one of the combined segment descriptor register (DR_n) and address register n (AR_n) is to be used. Each of these eight segment descriptor registers is associated with a corresponding address register. For example, an AR3 modification refers to the segment with the contents of DR3 in its descriptor. For multiword instructions, the use of AR_n and the associated DR_n is specified by the AR bit in the MF field. Refer to "Multiword Modification Field" documented in Section 5.

A segment descriptor must not be loaded with an operand descriptor intended for use with a multiword instruction.

Segment Identity Registers (SEGID_n)

4.29 Segment Identity Registers (SEGID_n)

Format:

12 bits each

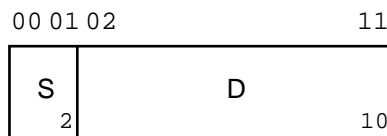


Figure 4-26. Segment Identity Register (SEGID_n) Format

Description:

Eight 12-bit registers that have a one-to-one correspondence with the operand descriptor registers (DR_n). The segment identity registers point to the source of the descriptor in the DR_n.

Function:

The Load Pointer Register (LDP_n) and Store Pointer (STP_n) instructions are available for directly loading and storing the segment identity registers. The S and D field codes used in these registers indicate the origin of the descriptor (S = segment, D = descriptor offset).

When S = 0

The D field indicates the location of the segment descriptor loaded into the DR_n.

Segment Identity Registers (SEGID_n)

For D = 1760 through 1777 (octal), the selected register is copied into the DR_n.

D = 1760	Undefined
D = 1761	The segment descriptor type field is changed. *
D = 1762	Instruction Segment Register (ISR)
D = 1763	Data Stack Descriptor Register (DSDR)
D = 1764	Safe Store Register (SSR)
D = 1765	Linkage Segment Register (LSR)
D = 1766	Argument Stack Register (ASR)
D = 1767	Parameter Segment Register (PSR)
D = 1770	DR0, Descriptor Register 0 }
D = 1771	DR1, Descriptor Register 1 }
D = 1772	DR2, Descriptor Register 2 }
D = 1773	DR3, Descriptor Register 3 } Self-Identifying
D = 1774	DR4, Descriptor Register 4 }
D = 1775	DR5, Descriptor Register 5 }
D = 1776	DR6, Descriptor Register 6 }
D = 1777	DR7, Descriptor Register 7 }

NOTE: * When S = 0 with D = 1761, 1763, and 1764, a Command fault occurs unless the CPU is in the Privileged Master Mode.

When S = 0 with D = 1761 in the Privileged Master Mode and the type of the segment descriptor in the DR_n is T = 1 or 3, this segment descriptor type is changed to 0 or 2, respectively. SEGID_n is set to be self-identifying. No fault occurs and no operation is performed with the LDD_n instruction when the type in the DR_n is not T = 1 or 3.

For D = 0000 through 1757 (octal), the descriptor in DR_n was loaded from the parameter segment and D was the index to the desired descriptor.

When S = 2

The descriptor DR_n was loaded from the argument stack using D as the index to the descriptor.

When S = 1 or 3

The descriptor in DR_n was loaded from the linkage segment using D as the index to the descriptor.

Stack Control Register (SCR)

4.30 Stack Control Register (SCR)

Format:

2 bits (internal)

Description:

An internal register that controls the size of the safe store frame

Function:

The SCR is initialized by execution of the LDSS instruction. This register contains the code indicating the size of the last safe store frame as. (Refer to the discussion of the Safe Store Register.)

If SCR = 10 and ISR(24) = 1 (ES or EI mode), safe store size = 80 words. ARn is entered in words 64-71 and zeros in bits 00-23 of words 16-23. GXn is entered in words 56-63 and words 40-43 are unpredictable.

If SCR = 11 and ISR(24) = 0 (NS mode), safe store size = 64 words. Xn is entered in words 40-43 and words 56-63 are unpredictable.

If SCR = 01, safe store size = 24 words. When the mode before the execution of the CLIMB is ES or EI mode, bits 00-23 of words 16-23 are zeros.

If SCR = 00, safe store size = 16 words.

When a special entry descriptor (T = 7) is specified by the SD field of the CLIMB, the value set in SCR = 11 on a NS to EI transition or SCR = 10 on an ES or EI transition to the EI mode.

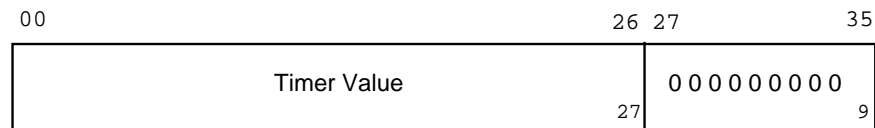
New SCR values are determined by the final segment descriptor and the ISR(24) value before starting the CLIMB:

Type Field	ISR(24)	New SCR
0 or 8	x	00
9	x	01
10 or 12	1	00
11	1	10
11	0	11

When the frame size is 64 words, the actual number of words stored is 48.

Timer Register (TR)**4.31 Timer Register (TR)****Format:**

27 bits

**Figure 4-27. Timer Register (TR) Format****Description:**

A 27-bit settable, free running clock

The value decrements at a rate of 512 KHz. Its range is 1.953125 microseconds to approximately 4.37 minutes.

Function:

The TR may be loaded with any convenient value with the Load Timer Register (LDT) instruction. When the value next passes through zero, a timer runout fault is signaled. If the processor is in Slave mode with interrupts not inhibited or is stopped at an uninhibited Delay Until Interrupt Signal (DIS) instruction, the fault occurs immediately. If the processor is in Master or Privileged Master mode or has interrupts inhibited, the fault is delayed until the processor returns to Slave mode or stops at an uninhibited Delay Until Interrupt Signal (DIS) instruction.

Virtual Machine Timer Register (VMTR)

4.32 Virtual Machine Timer Register (VMTR)

Format:

27 bits

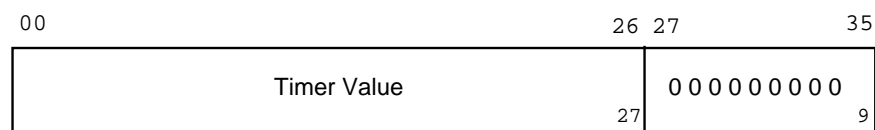


Figure 4-28. Virtual Machine Timer Register (VMTR) Format

Description:

A 27-bit settable, free running clock

The value decrements at a rate of 512 kHz. Its range is 1.953125 microseconds to approximately 4.37 minutes.

Function:

The TR may be loaded with any convenient value with the Load Timer Register (LDT) instruction. When the value next passes through zero, a timer runout fault is signaled. If the processor is in Slave mode with interrupts not inhibited or is stopped at an uninhibited Delay Until Interrupt Signal (DIS) instruction, the fault occurs immediately. If the processor is in Master or Privileged Master mode or has interrupts inhibited, the fault is delayed until the processor returns to Slave mode or stops at an uninhibited Delay Until Interrupt Signal (DIS) instruction.

Working Space Registers (WSR_n)

4.33 Working Space Registers (WSR_n)

Format:

In SV mode, 9 bits each



Figure 4-29. Working Space Register (WSR_n) Format, SV Mode

Description:

Eight 9-bit (SV mode) registers located in the virtual unit, each of which holds a working space (WS) number that is used to form a virtual address

Function:

A working space register is referred to by the WSR field of a descriptor. The LDWS and STWS instructions are used to load and store the working space registers, respectively. To execute these two instructions, the processor must be in Privileged Master mode. When the processor is initialized and cleared, working space register 0 is set to all zeros. The working space registers provide the means for sharing and isolating working spaces.

Notes

5. Address Modification and Development

This section of the Programmer's Guide describes address modification and development, organized as follows:

- Section 5.1, Address Modification and Development
- Section 5.2, Address Generation In The NS Mode
- Section 5.3, Address Generation In ES/EI Modes
- Section 5.4, Address Development
- Section 5.5, Paging

5.1 Address Modification Features

Address modification features permit the user to alter an address contained in an instruction (or in an indirect word referenced by an instruction). The address modification procedure is generally directed by the tag field of the instruction or indirect word. Address generation differs between the Normal Segmentation (NS) mode, the Extended Segmentation (ES) and Extended Instruction Segmentation (EI) modes. The general definition of each of these segment modes is:

Segment Mode	Max. Instruction Segment	Max. Data Segment	IC length (bits)	Xn length (bits)	ARn word field (bits)
NS	1 MB	1 MB	18	18	18
ES	1 MB	4 GB	18	36	30
EI	4 GB	4 GB	34	36	30

5.2 Address Generation In The NS Mode

The aspects of address generation in the NS mode which are discussed in this section are: basic modification, indirect addressing, tag fields, types of address modification, modification octal codes, modification flowchart, floatable code, address modification with address registers, and operand descriptors.

5.2.1 Basic Modification

Address modification is performed in four basic ways: Register (R), Register Then Indirect (RI), Indirect Then Register (IR), Indirect Then Tally (IT). A fifth way, address register modification, is discussed later in this section under "Address Modification with Address Registers". Each of these basic types has a number of variations in which selectable registers can be substituted for R in R, RI, and IR and in which various tallying or other substitutions can be made for T in IT. I indicates indirect address modification and is represented by the asterisk placed in the variable field of the program statement as *R or R* when IR or RI is specified. To indicate IT modification, only the substitution for T appears in the variable field; the asterisk is not used.

5.2.2 Indirect Addressing

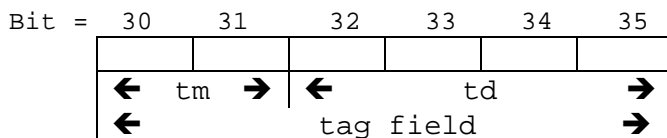
Generally, in indirect addressing, the content of bits 0-17 in the word addressed by the instruction address (y) is treated as another address, rather than as the operand of the instruction. Indirect address modification is performed by the hardware whenever called for by a program instruction. When I modification is called for by a program instruction, an indirect word is always obtained from memory. This indirect word may call for I modification again, or it may specify the effective address (Y) to be used for the original instruction. Indirect addressing for RI, IR, and IT modification is indicated by a binary 1 in either position of the tag modifier field (bit positions 30 and 31) of an instruction or indirect word.

NOTE: A "1" in bit position 30 or 31 of an indirect word does not necessarily mean further indirection.

5.2.3 Tag Field

An address modification procedure generally takes place as directed by the tag field of an instruction and the tag field of an indirect word. Repeat mode instructions and character store instructions do not provide for address modification.

The tag field consists of two parts: tag modifier (tm) and tag designator (td) (as illustrated below).



In the illustration of the tag field,

Tm specifies one of four possible modification types: Register (R), Register Then Indirect (RI), Indirect Then Register (IR), and Indirect Then Tally (IT);

Td specifies the activity for each modification type;
 In the case of tm = R, RI, or IR; td is the register designator and generally specifies the register to be used in indexing;
 In the case of tm = IT; td is the tally designator and specifies the tallying in detail.

The following table shows the valid mnemonics for address modification and their relationship to the classes R, RI, IR, and IT.

td	tm=00 R	tm=01 RI	tm=11 IR	tm=10 IT
00	Blank	*		
00	N	N*	*N	F
01	AU	AU*	*AU	-
02	QU	QU*	*QU	-
03	DU	-	*DU	-
04	IC	IC*	*IC	SD
05	AL	AL*	*AL	SCR
06	QL	QL*	*QL	-
07	DL	-	*DL	-
10	0	0*	*0	CI
11	1	1*	*1	I
12	2	2*	*2	SC
13	3	3*	*3	AD
14	4	4*	*4	DI
15	5	5*	*5	DIC
16	6	6*	*6	ID
17	7	7*	*7	IDC

5.2.4 Types Of Address Modification

The four basic types of modification, their mnemonic substitutions as used in the variable field of the program statement, and their binary forms are presented in the following illustration.

Modification Type	Variable Field	Binary Forms	Example								
		<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">30</td> <td style="text-align: center;">31</td> <td style="text-align: center;">32</td> <td style="text-align: center;">35</td> </tr> <tr> <td style="text-align: center;">tm</td> <td colspan="3" style="text-align: center;">td</td> </tr> </table>	30	31	32	35	tm	td			
30	31	32	35								
tm	td										
R	BETA, (R)	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> </tr> </table>	0	0	1	1	0	1	BETA, 5		
0	0	1	1	0	1						
RI	BETA, (R)*	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table>	0	1	1	0	1	0	BETA, 2*		
0	1	1	0	1	0						
IR	BETA, *(R)	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> <td style="text-align: center;">1</td> </tr> </table>	1	1	1	1	1	1	BETA, *7		
1	1	1	1	1	1						
IT	BETA, (T)	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> <td style="text-align: center;">1</td> <td style="text-align: center;">0</td> </tr> </table>	1	0	1	0	1	0	BETA, SC		
1	0	1	0	1	0						

The parentheses enclosing R and T indicate that substitutions should be made by the user for them (as explained under the separate discussions of R, IR, RI, and IT modification below). Binary equivalents of the substitution are used in the tm subfield.

5.2.4.1 Register (R)

The processor performs register address modification whenever an R-type variation is coded. The assembler places binary zeros in both positions of the tm subfield of the instruction. Accordingly, 1 of 16 variations under R are performed by the processor, depending upon the bit configurations generated by the assembler, and placed in the designator subfield (td) of the general instruction. The 16 variations, their mnemonic substitutions used on the assembler coding sheet, the td field binary forms presented to the processor, and the effective address Y generated by the processor are indicated in the following illustration:

Modification Variation	Mnemonic Substitution	Binary Form (td field)	Effective Address
(R) = X0	0	1000	$Y = y + C(X0)$
(R) = X1	1	1001	$Y = y + C(X1)$
(R) = X2	2	1010	$Y = y + C(X2)$
(R) = X3	3	1011	$Y = y + C(X3)$
(R) = X4	4	1100	$Y = y + C(X4)$
(R) = X5	5	1101	$Y = y + C(X5)$
(R) = X6	6	1110	$Y = y + C(X6)$
(R) = X7	7	1111	$Y = y + C(X7)$
(R) = A ₀₋₁₇	AU	0001	$Y = y + C(A_{0-17})$
(R) = A ₁₈₋₃₅	AL	0101	$Y = y + C(A_{18-35})$
(R) = Q ₀₋₁₇	QU	0010	$Y = y + C(Q_{0-17})$
(R) = Q ₁₈₋₃₅	QL	0110	$Y = y + C(Q_{18-35})$
(R) = IC	IC	0100	$Y = y + C(IC)$
direct upper	DU	0011	bits 0-17 of operand = y; bits 18-35 of operand = 0
direct lower	DL	0111	bits 0-17 of operand = 0; bits 18-35 of operand = y
(R) = None	Blank or N	0000	$Y = y$
(R) = Any	Any defined		
symbolic	symbol*		
index register			
* Symbol must be defined as one of the index registers by using an applicable pseudo-operation (EQU or BOOL).			

R modification allows for the use of the instruction address field as the operand. This procedure is called direct operand address modification and can be divided into two types: Direct Upper (DU) and Direct Lower (DL). With the DU variation, the address field of the instruction serves as bit positions 0-17 of the operand, and zeros serve as bit positions 18-35 of the operand. With the DL variation, the address field of the instruction serves as bit positions 18-35 of the operand, and zeros serve as bit positions 0-17 of the operand.

IC modification should only be used with an absolute operand. A relative operand that has IC modification is flagged with a possible relocation error (R) by the assembler.

The following examples show how R-type modification variations are entered and how they affect effective addresses.

EXAMPLES

	1	8	16	Effective Address
(1)		EAX0 LDA	1 B,0	Y=B+1
(2)		LDA LDA	=2,DL C,AL	Y=C+2
(3)		EAQ LDA	3 M,QU	Y=M+3
(4)	ABC	LDA	-2,IC	Y=ABC-2
(5)	XYZ	LDA	*,DU	operand ⁰⁻¹⁷ =XYZ, operand ¹⁸⁻³⁵ =0
(6)		EAX7 LDA	ABC 1,7	Y=ABC+1
(7)		LDA	2,DL	operand ⁰⁻¹⁷ =0, operand ¹⁸⁻³⁵ =2
(8)		LDA	B	Y=B
(9)		LDA	B,N	Y=B
(10)		EAX LDA	ALPHA,10 C,ALPHA	
	ALPHA	EQU	2	Y=C+10

Coding examples of R-type modification

1. **(R) = N**

```
ALPHA LDA     ADRES1,N
```

is equivalent to:

```
ALPHA LDA     ADRES1
```

No address modification results; ADRES1 is the effective operand.

Address Modification and Development

2. (R) = Xn where n = 0 to 7

ALPHA LDA ADRES2,5

X5 contains the value 2.

```
ADRES2 DEC 12
        OCT 7777
        OCT 123456765432
```

ADRES2+2 becomes the effective address, and its contents (octal 123456765432) are loaded into the A-register.

	A-register	X5
Before:	773412315026	000002
After:	123456765432	000002

3. (R) = AU, AL, QU, QL

ALPHA LDA ADRES3,QU

Bits 0-17 of the Q-register contain the value 3.

```
ADRES3 DEC 10
        OCT 12
        OCT 14
        OCT 16
```

ADRES3+3 becomes the effective address and its contents (octal 16) are loaded into the A-register.

	A-register	Q-register
Before:	123456765432	000003 123456
After:	000000000016	000003 123456

4. (R) = DU, DL

ALPHA LDA ADRES4,DU

No memory access to modify ADRES4 exists. The address represented by the symbol ADRES4 is placed in bits 0-17 of the A-register, and bits 18-35 are filled with zeros.

ADRES4 OCT 10 (assume ADRES4 is at location 001002₈)

Before:	000000000016
After:	00100200000

A simple program segment, the movement of 50 words from ABC to XYZ, may help illustrate the power of address modification.

Without Address Modification			With Address Modification		
1	8	16	1	8	16
-----			-----		
START	LDX1	=0B17, DU	START	LDX1	0, DU
LDA	ABC			LDA	ABC, 1
STA	XYZ			STA	XYZ, 1
LDA	=1B17			ADLX1	1, DU
ASA	START+1			CMPX1	50, DU
ASA	START+2			TNC	START+1
ADLX1	=1B17				
CMPX1	=50B17				
TNC	START+1				

5.2.4.2 Register then Indirect (RI)

Register Then Indirect address modification is a combination in which both indexing (register modification) and indirect addressing are performed. For indexing modification under RI, the mnemonic substitutions for R are the same as those given under the discussion of register (R) modification, except that DU and DL are invalid for RI usage. For indirect addressing (I), the processor interprets the contents of the operand address associated with the original instruction or with an indirect word.

Under RI modification, the effective address Y is found by first performing the specified register modification on the operand address of the instruction. The result of this R modification under RI is the address of an indirect word, which is then retrieved.

After the indirect word has been accessed from memory and decoded, the processor carries out the address modification specified by this indirect word. If the indirect word specifies RI, IR, or IT modification (any type specifying indirection), the indirect sequence is continued. When an indirect word is found that specifies R modification, the processor performs R modification, using the register specified by the rd field of this last-encountered indirect word and the address field of the same word, to form the effective address Y.

When used with Register Then Indirect modification (RI), the variations DU and DL of register modification (R) cause an Illegal Procedure (IPR) fault.

To refer to an indirect word from the instruction itself, without including register modification of the operand address, the "no modification" variation should be specified. Under RI modification, this specification is indicated by placing only an asterisk (*) in the tag position.

Address Modification and Development

The following examples illustrate the use of RI modification, including the use of (R) = N (no register modification). The asterisk appearing in the modifier subfield is the assembler symbol for I (Indirect). The address-subfield, single-symbol expressions shown are not intended to be realistic coding examples but to show the relation between operand addresses, indirect addressing, and register modification.

EXAMPLES

	1	8	16	Modification Type	Effective Address
(1)		EAA	1		
		EAX1	2		
		STA	Z, AU*	(RI)	Y=B+2
		ORG	Z+1		
		ARG	B, 1	(R)	
(2)		EAQ	3		
		MPY	Z, *	(RI)	Y=B+3
	Z	ARG	B, QU	(R)	
(3)		EAX3	3		
		EAX5	5		
		STQ	Z, *	(RI)	Y=M
	Z	ARG	B, 5*	(RI)	
		ORG	B+5		
		ARG	C, 3*	(RI)	
		ORG	C+3		
		ZERO	M	(R)	

Coding examples of RI modification

1. (RI) = N*

```
ALPHA LDA ADRES1, N*
```

is equivalent to:

```
ALPHA LDA ADRES, *
```

The indirect word at ADRES1 is obtained; if this indirect word specifies further indirect modification, the process continues until an indirect word is obtained with (R) modification.

2. (RI) = (X_n)* where n = 0 to 7

```

EAX5 5
EAX2 2
ALPHA LDA ADRES2, 5*
```

The indirect word at ADRES2+5 is obtained. If the indirect word at this location is:

```
LDQ ADRES3, 2
```

the effective address is ADRES3+2.

5.2.4.3 Indirect Then Register (IR)

Indirect Then Register address modification is a combination in which both indirect addressing and indexing (register modification) are performed. However, IR modification is not a simple inverse of RI; several important differences exist.

Under IR modification, the processor first fetches an indirect word from the memory location specified by the address field y of the machine instruction. The $C(R)$ of IR are safe stored for use in making the final index modification to develop the effective address Y . The address modification, if any, specified by this first indirect word is then examined. If this modification is again IR, the register from the last IR modification is safe stored and used for final effective address.

If the indirect sequence produces an RI indirect word, the R-type modification is performed immediately to form another address, but the I of this RI treats the contents of the address as an indirect word. At this point the new indirect word might initiate an RI loop. However, if IR modification is specified, an IPR fault occurs. When this loop is broken, the remaining modification type is either R or IT.

When either R or IT is encountered, it is treated as type R, where R is the last safe stored $C(R)$ of an IR modification. At this point the safe stored $C(R)$ is combined with the y of the indirect word that produced R or IT, and the effective address Y is developed.

If an indirect modification without register modification is desired, the "no modification" variation (N) of register modification should be specified in the instruction. This variation normally will be entered on coding sheets as *N in the modifier part of the variable field. (The entry * alone is equivalent to N* under RI modification and must be used in that way.)

NOTE: As described above, if IR modification is detected twice in an indirect modification chain, an IPR fault occurs.

Coding examples of IR modification

1. **(IR) = *N**

```
ALPHA   LDA    ADRES1, *N
```

The indirect word at ADRES1 is obtained. If the indirect word at this location is:

```
ADRES1  LDQ    ADRES2
```

The effective address is:

```
ADRES2
```

2. **IR and then R or IT**

(IR) = *(X_n) where n = 0 to 7
 EAX5 15

```
ALPHA   LDA    ADRES1, *5
```

The indirect word at ADRES1 is obtained. If the indirect word is:

```
ADRES1  LDQ    ADRES2, (R)
```

or

```
ADRES1  LDQ    ADRES2, (T)
```

the effective address is:

```
ADRES2+15
```

3. **IR and then RI**

(IR) = *(X_n) where n = 0 to 7

```
          EAX5    16
          EAX2    17
ALPHA   LDA    ADRES1, *5
ADRES1  LDQ    ADRES2, 2*
          .
          .
          .
          LDA    ADRES4       ( in ADRES2+17)
```

the effective address is:

```
ADRES4+16
```

The following examples illustrate the use of IR-type modification, intermixed with R and RI types, under the several conditions noted on the previous page.

EXAMPLES

1	8	16	Modification Type	Effective Address
(1)		LDQ 1,DL		
		LDA Z,*QL	(IR)	Y=M+1
	Z	ARG M	(R)	
(2)		EAX3 2		
		EAX5 3		
	ABC	LDA Z,*3	(IR)	Y=C+2
	Z	ARG B,5*	(RI)	
		ORG B+3		
		ARG C,IC	(R)	
(3)		EAX3 8		
		LDQ 9,DL		
		LDA Z,*DL	(IR)	C(A(18-35))=M
	Z	ARG B,3*	(R)	
		ORG B+8		
		ARG M,QL	(R)	
(4)		LDA 10,DL		
		LDA Z,*AL	(IR)	Y=B+10
	Z	ARG B,AD	(IT)	
(5)		EAX3 11		
		LDA Z,*N	(IR)	Y=B
	Z	ARG B,3	(R)	
(6)		EAX5 13		
		LDA Z,*	(RI)	Y=M+13
	Z	ARG B,*5	(IR)	
	B	ARG M,DU	(R)	
(7)		EAX1 14		
		LDA X,*	(RI)	Y=Z+14
	X	ARG B,*1	(IR)	
	B	ARG Z,ID	(IT)	
	Z	TALLY A,10	(IT)	

5.2.4.4 Indirect Then Tally (IT)

Indirect Then Tally address modification is a combination in which both indirect addressing and automatic incrementing/decrementing of fields in the indirect word are performed as hardware features, thus relieving the user of these responsibilities. The automatic tallying and other functions of IT modification allow processing of tabular data in memory, provide a means for working upon character data, and allow termination on user-selectable numeric tally conditions. If an unassigned IT tag is used, an Illegal Procedure (IPR) fault occurs.

The following table shows the variations under IT modification. The mnemonic substitution for IT is (T). The designator I for indirect addressing in IT is not represented. (Note that one of the substitutions for T is I.)

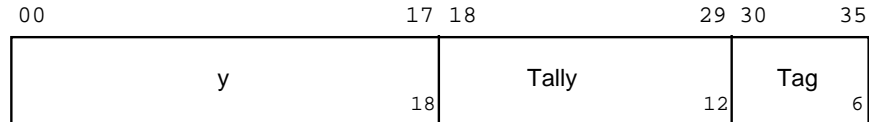
Variation	Mnemonic Substitution	Binary form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
Fault	F	0000	None. A Fault Tag fault is generated. The indirect word is not examined.
Character	CI	1000	None; applies to TALLY, TALLYB.
Sequence Character	SC	1010	Obtain the operand address from the tally word, then add 1 to the character position value in the tag field and subtract 1 from the tally count field; add 1 to the address field and set the character position value to zero when the character position crosses a word boundary; applies to TALLY, TALLYB.
Sequence Character Reversed	SCR	0101	Subtract 1 from the character position value in the tag field and add 1 to the tally count field;
Indirect	I	1001	None. The operand address is the word to which the tally word address field refers; applies to all tally pseudo-operations.
Increment address, Decrement tally	ID	1110	Obtain the operand address from the tally word; add 2 to the address field and subtract 1 from the tally count field; applies to all tally pseudo-operations.

Variation	Mnemonic Substitution	Binary form (td Field)	Effect on Processor and Indirect (Tally) Word for Each Reference
Decrement address Increment tally	DI	1100	Subtract 1 from the address field, add 1 to the tally count field, and then obtain the operand address from the tally word; applies to all tally pseudo-operations.
Increment address, Decrement tally, and Continue	IDC	1111	Obtain the operand address from the tally word; add 1 to the address field and subtract 1 from the tally count field. Additional address modification will be performed as specified by the tag field; applies to TALLYC.
Decrement address, Increment tally, and Continue	DIC	1101	Subtract 1 from the address field, add 1 to the tally count field, obtain the operand address from the tally word. Additional address modification will be performed as specified by the tag field; applies to TALLYC.
Add Delta	AD	1011	Obtain the operand address from the tally word, add an increment to the address field, and subtract 1 from the tally count field; applies to TALLYD.
Subtract Delta	SD	0100	Subtract an increment from the address field, add 1 to the tally count field, and then obtain the operand address from the tally word; applies to TALLYD.

Indirect Word Format

The location of the indirect word is specified by the address field (y) of the instruction or previous indirect word (IDC or DIC). IT modification causes the indirect word to be fetched and interpreted as specified by the td subfield of the instruction or previous indirect word that referred to the indirect word.

The format of the indirect word is:



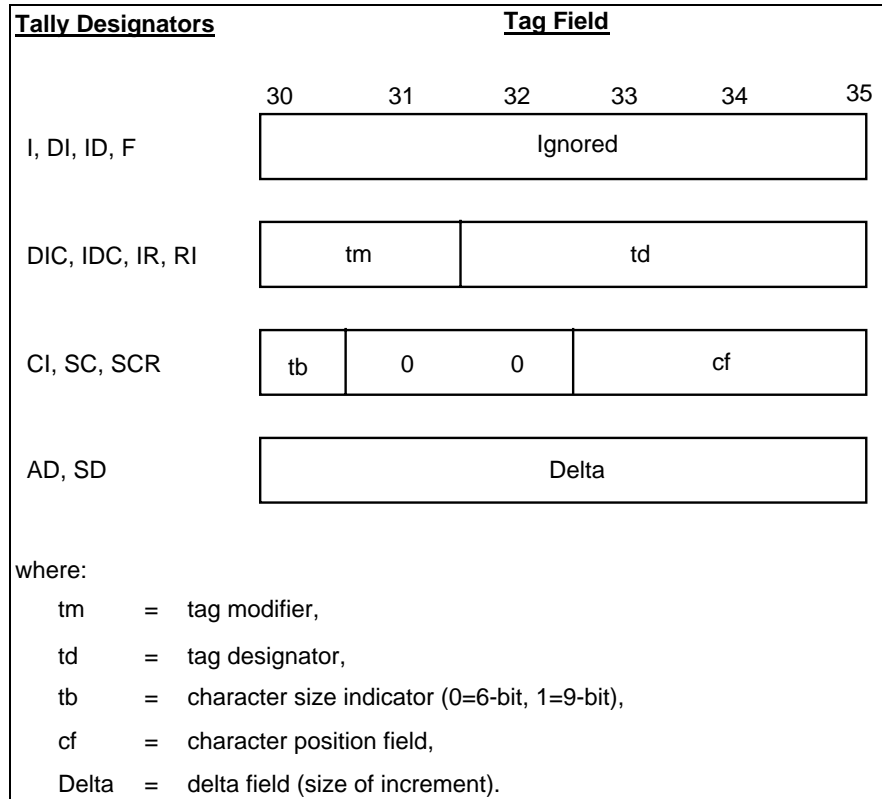
where:

y = address field

Tally = tally field

Tag = tag field

Depending upon the prior tally designator, the tag field for the indirect word is used in one of the following ways:



5.2.5 Variations Under IT Modification

5.2.5.1 Fault (T) = F Variation

The Fault variation enables the user to force program transfers to operating system routines or to corrective routines during the execution of an address modification sequence by causing a Fault Tag fault. (This fault will usually indicate some abnormal condition for which the user desires protection.)

5.2.5.2 Character Indirect (T) = CI Variation

The Character Indirect (CI) variation is provided for operations on the A register or Q register in situations where repeated reference to a single character in memory is required. The character size field (tb) of the indirect word specifies the character size.

For this variation, the effective address is the address field of the CI indirect word obtained through the tentative operand address of the instruction or preceding indirect word that specified the CI variation. The character position field (cf) of the indirect word is used to specify the character to be involved in the operation.

This variation is similar to the SC variation except that no incrementing or decrementing of the address, tally, or character position is performed. Some examples are given below.

	1	8	16	Modification Type	Effective Address	Character Position
(1)	Z	LDA TALLY	Z,CI B,,4	(IT) 6-bit char.	Y=B addressing	4

	1	8	16			
(2)	ADDR	LDA TALLY	ADDR,CI ADD,,3	6-bit char.	addressing	
or	ADDR	TALLYB	ADD,,3	9-bit char.	addressing	

The effective address is ADDR. The character in character position 3 is loaded into the A-register in character position 5 for 6-bit characters or into position 3 for 9-bit characters. The remainder of the A-register is loaded with all zero bits.

5.2.5.3 Sequence Character (T) = SC Variation

The Sequence Character (SC) variation is provided for sequential access to 6-bit or 9-bit characters. The character size field (tb) of the indirect word is used to specify the character size. Processor instructions that do not allow SC operations are so indicated in the descriptions for the individual instructions. The operand address is obtained from the address field of the indirect word referenced by the word containing the SC tag.

Characters are operated on in sequence from left to right within the machine word. The character position field (cf) of the indirect word is used to specify the character position to be involved in the operation. The Tally Runout indicator is set when the tally field of the indirect word reaches 0. The coding below provides an example.

1	8	16	32

	LDA	A, SC	
A	TALLY	TABLE, 70, 4	6-bit char. addressing
TABLE	BSS	13	

In this example, 70 is the count and 4 designates the character position of the tally start.

For register loads using the SC variation, a character is fetched from the indicated position of the memory location and is written into the lower end of the register; the remaining bits of the register are set to zero. For stores under the SC variation, a character is fetched from the lower end of the register and written into the indicated position in the memory location. The remaining character positions in the memory location remain unchanged.

The tally field of the indirect word is used to count the number of times a reference is made to a character. Each time an SC reference is made to the indirect word, the tally is decremented by 1, and the character position is incremented by 1 to specify the next character position. The tally runout indicator is set when the tally reaches 0. When character position 5 (for 6-bit characters) or 3 (for 9-bit characters) is incremented, it is changed to position 0, and the address field of the indirect word is incremented by 1. All incrementing and decrementing are done after the effective address has been provided for the current instruction execution.

The effect of successive references using SC modification is shown in the following examples.

1	8	16	Effective Address	Character Position	Reference
	LDA	Z, SC	B	0	1
Z	TALLY	B, 80, 0	B	1	2
B	BSS	14	.	.	.
			.	.	.
			B	5	6
			B+1	0	7
			.	.	.
Tally Runout indicator is set on the 80th reference			.	.	.
			.	.	.
			B+n	0	6n+1
			.	.	.
			.	.	.

1	8	16	
ADD1	LDA	ADDR, SC	
	TTF	ADD1	
	.		
	.		
ADDR	TALLY	ADD, 12, 13	(6-bit characters)
	or		
ADDR	TALLYB	ADD, 12, 13	(6-bit characters)
ADD	BSS	4	

The first effective address is ADD. The character in character position 3 is loaded into the A-register in position 5 (for 6-bit characters) or into position 3 (for 9-bit characters). The second reference will load ADD character 4 (if 6-bit) or ADD+1 character 0 (if 9-bit), etc. The tally is decremented from 12 to 0. The destination in the A-register does not change.

5.2.5.4 Sequence Character Reverse (T) = SCR Variation

The SCR variation is the reverse of SC. The character position is decremented by 1 and the tally is incremented by 1 before the indirect word address field and character position are used as the operand character address. When the character position attempts to go negative, it is set to the maximum value (3 or 5), and the address is decremented by 1.

5.2.5.5 Indirect (T) = I Variation

The Indirect (I) variation of IT modification is in effect a subset of the ID and DI variations described below in that all three -- I, ID, and DI -- make use of one indirect word in order to refer to the operand. The I variation is functionally unique, however, in that the indirect word accessed by an instruction remains unaltered. No incrementing/decrementing of the address field or tally occurs. Since the tag field of the indirect word under I is not interrogated, this word will always terminate the indirect chain.

The following differences in the coding and the effects of *, *N, and I should be observed:

1. RI modification is coded as R* for all cases, excluding R=N.
 For R=N under RI, the modifier subfield can be written as N* or as * alone, according to preference.
 When N* or just * is coded, the assembler generates a machine word with octal 20 in bit positions 30-35. Octal 20 causes the processor to add 0 to the address field y of the word containing the N* or * and then to access the indirect word at memory location y.
2. IR modification is coded as *R for all cases, including R=N.
 For R=N under IR, the modifier subfield must be written as *N.
 When *N is coded, the assembler generates octal 60 in bit positions 30-35 of the associated machine word. Octal 60 causes the processor to (1) retrieve the indirect word at the location (y) specified by the machine word, and (2) effectively safe store zeros (for possible final index modification of the last indirect word).
3. IT modification is coded using only a variation designator (I, ID, DI, SC, SCR, CI, AD, SD, F, IDC, or DIC), that is, no asterisk (*) is written. Thus, a written IT address modification appears as ALPH,DI; BETA,AD; and so on.
 For the variation I under IT, the assembler generates a machine word with octal 51 in bit positions 30-35. Octal 51 causes the processor to examine *one* indirect word to be retrieved from memory to obtain the effective address Y.

1	8	16	Modification Type	Effective Address
	EAX5	1		
	LDA	Z, I	(IT)	Y=B
Z	ARG	B, *5	(IR)	

5.2.5.6 Increment Address, Decrement Tally (T) = ID Variation

The ID variation under IT modification provides automatic (hardware) incrementing or decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the ID variation, the effective address is the address field of the indirect word obtained through the tentative operand address of the instruction or preceding indirect word, whichever specified the ID variation. Each time such a reference is made to the indirect word, the address field of the indirect word is incremented by 1 and the tally portion of the indirect word is decremented by 1. The incrementing and decrementing are performed after the effective address is provided for the instruction operation. When the tally reaches zero, the Tally Runout indicator is set.

The following examples show the effect of ID.

1	8	16	Modification Type	Effective Address	Reference

Z	LDA	Z, ID	(IT)	B	1
B	TALLY	B, 12	word addressing	B+1	2
B	BSS	12	.	.	

The Tally Runout Indicator is set on the 12th reference.

1	8	16	-----		
ADRES	LDA	ADRES, ID			
	TTF	ADRES1			
	.				
	.				
ADRES2	TALLY	ADRES3, 10	word addressing		
ADRES3	BSS	10			

The first effective address is ADRES3; the second is ADRES3 plus 1, etc. The tally is decremented from 10 to zero. The TTF instruction checks the Tally Runout indicator. If the tally is not zero, transfer is made to ADRES1. If the tally is zero, processing continues with the instruction following TTF. Without the TTF instruction, only one effective address is obtained.

5.2.5.7 Decrement Address, Increment Tally (T) + DI Variation

The DI variation under IT modification provides automatic (hardware) incrementing and decrementing of an indirect word that is best used for processing tabular operands (data located at consecutive memory addresses). The indirect word always terminates the indirect chain.

In the DI variation, the effective address is the modified address field (1 less than the value before modification) of the indirect word obtained via the tentative operand address of the instruction or preceding indirect word, whichever one specified the DI variation. Each time a DI reference is made to the indirect word, the address field of the indirect word is decremented by 1 and the tally portion is incremented by 1. When the tally is incremented from 7777 to 0, the tally runout indicator is set. The incrementing and decrementing are performed before providing the effective address for the current instruction operation.

The effect of DI is shown in the following examples.

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z,DI	(IT)	B-1	1
Z	TALLY	B,-18	word addressing	B-2	2
				.	.
B	BFS	18		.	.
				B- <u>n</u>	<u>n</u>

Tally Runout indicator is set on the 18th reference. There, the 12-bit tally field in the indirect word overflows and becomes all zeros.

1	8	16	Modification Type	Effective Address	Reference
ADRES1	LDA	ADRES2,DI			
.	TTF	ADRES1			
.					
ADRES2	TALLY	ADRES3,-10	word addressing		
ADRES3	BFS	10			

The first effective address is ADRES3 -1; the second is ADRES3 -2, etc. The tally increases from -10 to 0.

5.2.5.8 Increment Address, Decrement Tally, and Continue (T) = IDC Variation

The IDC variation under IT modification functions in a manner similar to the ID variation except that, in addition to automatic incrementing/decrementing, it permits the user to continue the indirect chain in obtaining the instruction operand. Where the ID variation is useful for processing tabular data, the IDC variation permits processing of scattered data by a table of indirect pointers. Specifically, the ID portion of this variation provides the ability to sequentially step through a table, and the C portion (continuation) allows indirection through the tabular items. The tabular items may be data pointers, subroutine pointers, or a transfer vector.

The address and tally fields are used as described under the ID variation. The tag field uses the set of variations for instruction address modification under the following restrictions: no variation is permitted that requires an indexing modification in the IDC cycle since the indexing adder is being used by the tally phase of the operation. Thus, permissible variations are any allowable form of IT or IR, but if RI or R is used, R must equal N (RI and R forced to N).

The effect of successive references using IDC modification is indicated in the following examples.

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z, IDC	X	1	
Z	TALLYC	B, 10, I	Y	2	
B	ARG	X	Z	3	
	ARG	Y	.	.	
	ARG	Z	.	.	

The Tally Runout indicator is set on the 10th reference.

1	8	16	32
ADRES1	LDA	ADRES2, IDC	
	TTF	ADRES1	
ADRES2	TALLYC	ADRES3, 4, *	word addressing and indirect
ADRES3	ARG	AD1	
	ARG	AD2	
	ARG	AD3	
	ARG	AD4	

AD1 is the first effective address; AD2 is the second; AD3, the third; and AD4, the fourth.

5.2.5.9 Decrement Address, Increment Tally, and Continue (T) = DIC Variation

The DIC variation under IT modification performs in much the same way as the DI variation except that, in addition to automatic decrementing or incrementing, it allows the user to continue the indirect chain in obtaining an instruction operand. The continuation function of DIC operates in the same manner and under the same restrictions as IDC except that (1) it increments in the reverse direction, and (2) decrementing/incrementing is performed before obtaining the effective address from the tally word. (Refer to the first example under IDC and work from the bottom of the table to the top.) DIC is especially useful in processing last-in, first-out lists (see the following examples).

1	8	16	Modification Type	Effective Address	Reference
	LDA	Z, DIC	(IT)		
Z	TALLYC	B, -10, I	(IT)	Y	1
	ARG	Z		X	2
	ARG	X		Z	3
	ARG	Y		.	.
				.	.
B	NULL				

Assuming an initial tally of -10, the Tally Runout indicator is set on the 10th reference. There, the 12-bit tally field in the indirect word overflows and becomes all zeros.

1	8	16	32
ADRES1	LDA	ADRES2, DIC	
	TTF	ADRES1	
ADRES2	TALLYC	ADRES3, -4, *N	word addressing and indirect
	ARG	AD4, *	
	ARG	AD3	
	ARG	AD2, *N	
	ARG	AD1, *N	
ADRES3	BSS	1	
AD1	ARG	A	
AD2	ARG	B	
AD4	ARG	C	

A is the first effective address; B is the second; AD3, the third; and C, the fourth.

5.2.5.10 Add Delta (T) = AD Variation

The Add Delta (AD) variation is provided for programming situations where tabular data to be processed is stored at equally spaced locations (such as data items), each occupying two or more consecutive memory addresses. It functions in a manner similar to the ID variation, but the incrementing (delta) of the address field is selectable by the user.

Each time such a reference is made to the indirect word, the address field of the indirect word is increased by delta, and the tally portion of the indirect word is decremented by 1. The addition of delta and decrementing are done after the effective address is provided for the instruction operation.

The following examples show the effect of successive references using AD modification.

1	8	16	Modification Type	Effective Address	Reference
	LDAQ	Z,AD	(IT)	B	1
Z	ETALLY	B,20,2		B+2	2
B	EBSS	40		B+4	3
				.	.
				.	.
				B+2n	n+1

The Tally Runout indicator is set on the 20th reference.

1	8	16	32
ADRES1	LDAQ	ADRES2,AD	
	TTF	ADRES1	
	.		
	.		
ADRES2	ETALLYD	ADRES3,10,2	word addressing with DELTA
ADRES3	EBSS	20	

The first effective address is ADRES3; the second is ADRES3+2. The tally decreases from 10 to 0.

5.2.5.11 Subtract Delta (T) = SD Variation

The Subtract Delta (SD) variation is useful in processing tabular data in a manner similar to the AD variation except that the table can be scanned easily from back to front using a programmer-specified increment. The effective address from the indirect word is decreased by delta, and the tally is increased by 1 each time an SD reference is made to the indirect word. This procedure is done before supplying the operand address to the current instruction, making the SD variation analogous to the DI variation.

5.2.6 Address Modification Octal Codes

Address modification and 2-digit octal codes for each type of modification are listed in Table 5-1.

Table 5-1. Address Modification Octal Codes

		LOW ORDER OCTAL DIGIT							
		0	1	2	3	4	5	6	7
H I G H	0	N	AU	QU	DU	IC	AL	QL	DL
	1	0	1	2	3	4	5	6	7
O R D E R	2	N*	AU*	QU*		IC*	AL*	QL*	
	3	0*	1*	2*	3*	4*	5*	6*	7*
O C T A L	4	F				SD	SCR		
	5	CI	I	SC	AD	DI	DIC	ID	IDC
D I G I T	6	*N	*AU	*QU	*DU	*IC	*AL	QL	*DL
	7	*0	*1	*2	*3	*4	*5	*6	*7

5.2.7 Address Modification Flowchart

The process of address modification is illustrated in flowchart form in Figure 5-1. (Address register modification is not included in this example.)

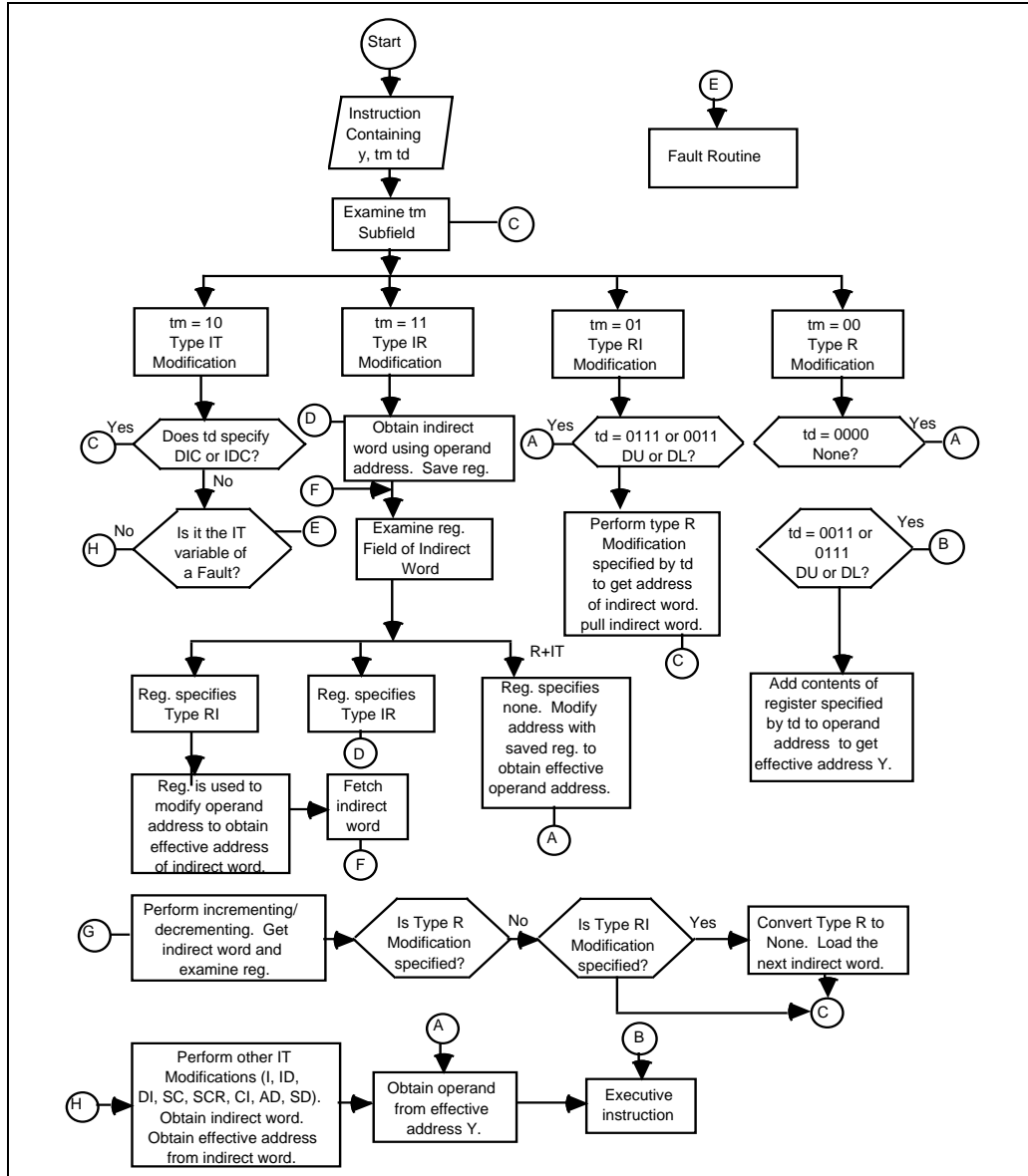


Figure 5-1. Address Modification Flowchart

5.2.8 Floatable Code

Program statements may be written in floatable code. Such statements may then be executed from any location in memory without relocation at load time. Floatable code is created by use of instruction counter (IC) modification in all references to locations within a program. Thus, to transfer to location SYM, the following statement can be written:

```
TRA    SYM-*,IC
```

or

```
TRA    SYM,$
```

The assembler accepts the currency symbol (\$) as a valid IC register designator. The following tag fields in a machine instruction are permitted:

<u>Mnemonic</u>	<u>Octal Code</u>
\$	04
\$*	24

The assembler computes the difference between the value of the address location argument of the variable field and the current location as the content of the address field of the instruction word. The IC is then supplied for modification. *\$ is illegal and will be assembled as *IC.

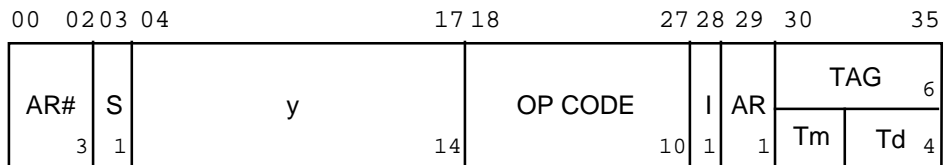
NOTE: The FLOAT pseudo-operation or \$ modification does not apply when used with SYMREF symbols or within the range of a BLOCK pseudo-operation.

5.2.9 Address Modification With Address Registers

The address register format allows addressing on a character or bit basis and is used by the character and bit manipulation instructions of the processor. When an address register is used to modify an address in which character and/or bit addressing is not used, the character and bit positions of the address register are ignored. Address registers (AR_n) provide a second-level indexing capability.

5.2.9.1 Single-Word Address Modification

When an address register is to be used in address preparation, its application is specified in the instruction word. All single-word instructions to which address modification is applicable have the same instruction word format.



- AR#** Address register number, if bit 29 = 1
- S** Sign bit, if bit 29 = 1
- y** Address field bits 0-17 or bits 3-17, depending on the state of bit 29; must be an absolute value if AR mode is used
- OP CODE** 10-bit operation code field
- I** Program interrupt inhibit bit
- AR** Address register bit:
- If bit 29 = 1, use address register specified in bits 0, 1, and 2 of y field for address modification and use operand descriptor register specification in bits 0,1, and 2 of y field as the segment descriptor; bit 3 (sign) is then extended to bits 0, 1, and 2;
 - If bit 29 = 0, no address register modification is performed and the ISR is used as the segment descriptor
- TAG** Tag field: used to control address modification
- Tm (Bits 30-31): type of address modification
 - Td (Bits 32-35): index register or modification variation designator

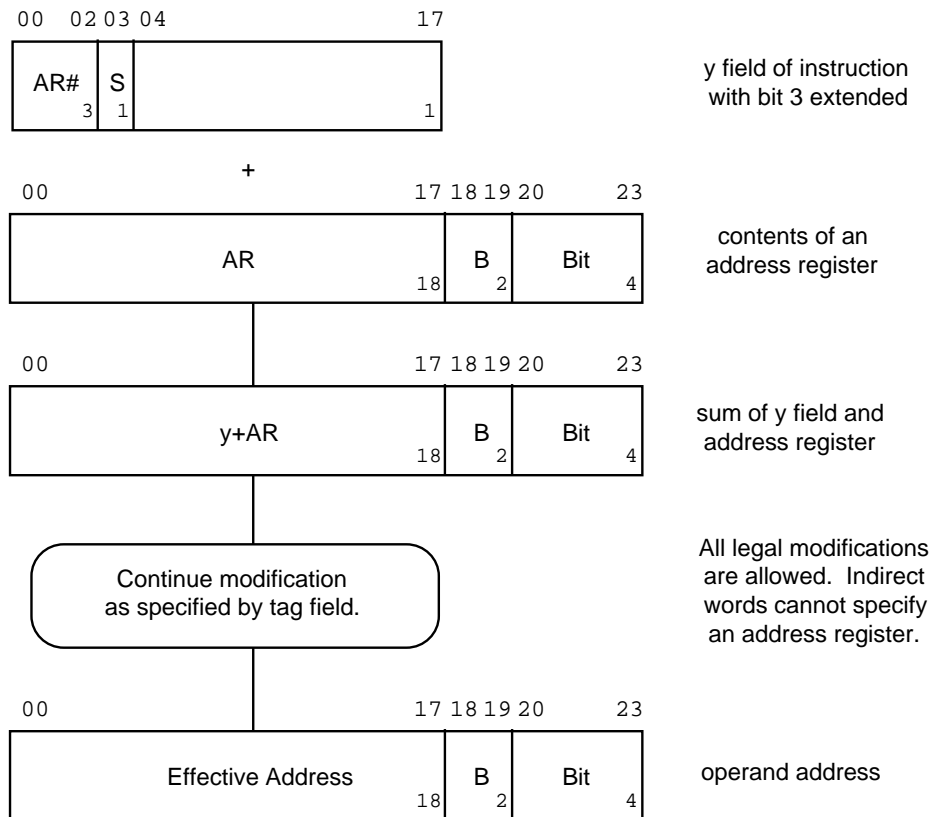
NOTE: With some instructions, certain address modification is not permitted, and if such modification is specified, an Illegal Procedure fault (IPR) occurs. (Refer to the individual instruction specifications in Sections 8 through 15.)

Address Modification and Development

The address preparation for a single-word instruction with bit 29 = 1 is listed below.

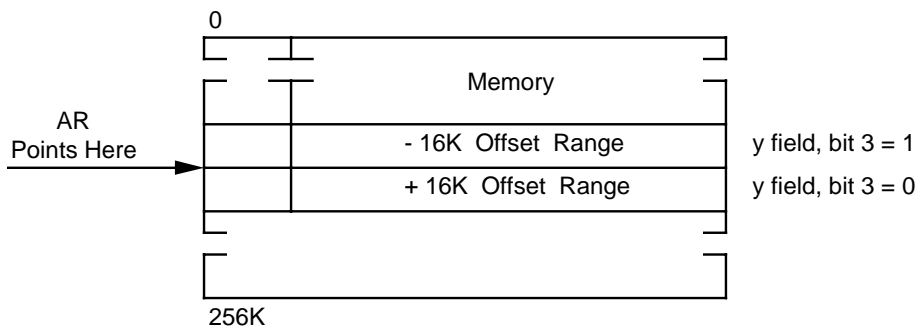
1. The three most-significant bits of y (0, 1, 2) are decoded to determine which of the eight address registers is to be used.
2. Bit 3 of the y field is extended to fill bit positions 2, 1, and 0, thus forming a two's complement signed number.
3. The two's complement y field is then added to the contents of the specified address register. The character and bit positions of the address register are ignored, and the contents of the address register remain unchanged.
4. Address modification continues as specified by the tag field of the instruction word.

Address preparation is described as follows:



When bit 29 = 0, the first step of the address modification procedure using the address register is omitted. The tag field specifies the only address modification performed.

When an address register is specified, extending bit 3 of the y field to form a two's complement signed number effectively designates bit 3 as a sign bit. This extension leaves 14 bits, 4 through 17, with which to designate an address offset. Thus, an address offset with values between -2^{14} and $2^{14}-1$ can be specified. An address register, then, contains a complete 18-bit memory address, which may be offset $\pm 16K$ by the partial address contained in the y field of the instruction (as shown on the next page).

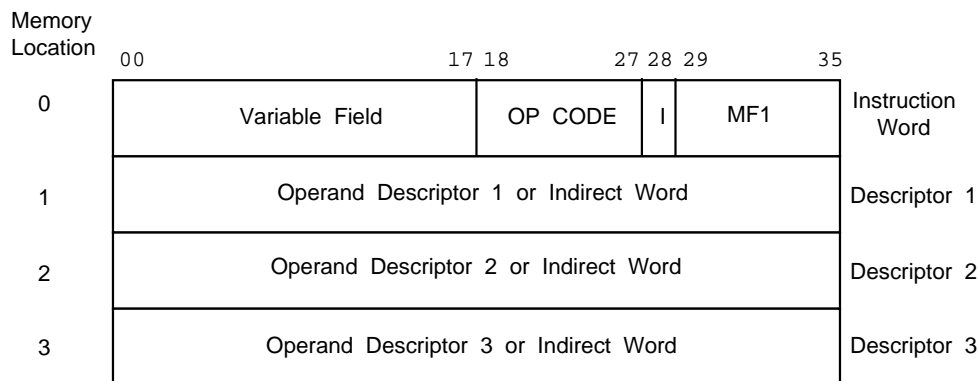


Coding Examples

1. LDQ 4,N,2
Effective Address = 4 + bits 0-17 of C(AR2)
2. LDQ -4,N,2
Effective Address = -4 + bits 0-17 of C(AR2)

5.2.9.2 Multiword Address Modification

The general format of a multiword instruction is given below.



where:

Variable Field contains additional information concerning the operation to be performed, depending on the particular instruction. When descriptors 2 and 3 are present, most instructions provide a corresponding MF2 (bits 11-17) and MF3 (bits 2-8) within the variable field to describe the address modification to be performed on these operands, when present. Exceptions to these structures are the CMPCT, MVT, SCD, SCDR, SCM, SCMR, TCT, and TCTR instructions.

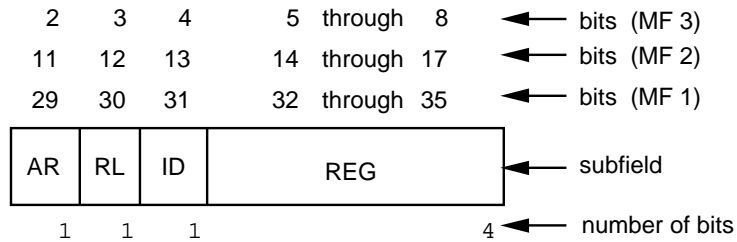
OP CODE is the 10-bit operation code field. Octal representation consists of three octal digits corresponding to bit positions 18-26 and a 1 for bit position 27.

I is the program interrupt inhibit bit.

MF1 is the Modification Field 1 (MF1) which describes address modification that is to be performed for descriptor 1.

5.2.9.3 Multiword Modification Field

Each modification field (MF) contained in a multiword instruction is a 7-bit field specifying address modification to be performed on the operand descriptors. The modification field is interpreted in the following way.



AR=	Address Register Specifier
0	No address register used.
1	Bits 0-2 of the operand descriptor address field specify the address register to be used in computing the effective address of the operand. Bits 0 - 2 also specify the operand descriptor register that defines the segment containing the operand.
RL=	Register or Length
0	Operand length is specified in the N field (bits 32-35) of the operand descriptor.
1	The length of operand is contained in the register that is specified by code in the N field (bits 32-35) of the operand descriptor, in the machine format of REG (the coding format is different).
ID =	Indirect Operand Descriptor
0	The operand descriptor follows the instruction word in its sequential memory location.
1	The operand descriptor location contains an indirect word that points to the operand descriptor. Only one level of indirection is allowed.
REG	Address Modification Register
	Selects the address modification register for R-type modification of the operand descriptor address field. The REG codes are approximately the same as the single-word modifications. In addition, for indirect string length specification (RL = 1), the N field codes are similar to the REG field. A comparison of these codes is shown in Table 5-2.

Table 5-2. Register Codes

Octal Code	REG in MF (1)	REG in Indirect Word when ID = 1 (2)	Bits 32-35 of N when RL = 1	Td Field of Tag
0000	None	None	IPR Fault	None
0001	AU	AU	AU	AU
0010	QU	QU	QU	QU
0011	DU	IPR Fault	IPR Fault	DU
0100	IC	IC	IPR Fault	IC
0101	A (3)	A (3)	A (3)	AL
0110	Q (3)	Q (3)	Q (3)	QL
0111	IPR Fault	IPR Fault	IPR Fault	DL
1000	X0	X0	X0	X0
1001	X1	X1	X1	X1
1010	X2	X2	X2	X2
1011	X3	X3	X3	X3
1100	X4	X4	X4	X4
1101	X5	X5	X5	X5
1110	X6	X6	X6	X6
1111	X7	X7	X7	X7

Notes for Table 5-2:

1. The register content is interpreted as character or bit index. For alphanumeric descriptor, this index is the number of 9-bit, 6-bit, or 4-bit characters, depending data type specified in the descriptor. For a numeric descriptor, it is the number of 9-bit or 4-bit characters, also depending on the data type specified. For a bit descriptor, it is the number of bits.
2. Register contents are interpreted as a word index.
3. The A- and Q-registers provide for indexing number greater than $2^{18}-1$. When the A or Q register is specified, the number of right-justified bits for indexing depends on the type of unit reference that is specified in the operand referring to the A- or Q-register:
 - 18 bits for full-word (36-bits) operations,
 - 21 bits for 9-bit and 6-bit character operations,
 - 22 bits for 4-bit character operations,
 - 24 bits for bit operations.

All addressing is "modulo addressing". For example, when software indexes backwards by N words, it actually indexes forward by $2^{18}-N$ words. This same method is also used in character and bit indexing.

Unit	No. Units/Word	No. to Effectively Yield -N
Word	1	$2^{18} - N$
9-bit	4	$4 * 2^{18} - N$ ($2^{20} - N$)
4-bit	8	$8 * 2^{18} - N$ ($2^{21} - N$)
6-bit	6	$6 * 2^{18} - N$
1 bit	36	$36 * 2^{18} - N$

For indexing 9-bit, 6-bit, 4-bit, and 1-bit characters, A and Q can be loaded with 4, DU; 6,DU; 8,DU; or 36,DU; respectively. Then N can be subtracted.

The index register designations may be specified by a symbol defined by the user to have a value in the octal range of 0, 1, ...,7. An indirect descriptor may be designated (e.g., 10, 11,...,17) when the RL usage is in a descriptor that does not immediately follow the multiword instruction.

Examples

1	8	16

XA	BOOL	17
	MLR	(0,1), (0,1)
	ADSC9	A, 0, XA
	ADSC9	B, 0, XA

This procedure is used to specify a move of the number of characters specified by the current value of index register 7. Similarly, the following code provides for the sending address of the move to be specified indirectly in the word labeled LA:

1	8	16

	MLR	(0,1,1), (0,1)
	ARG	LA
	ADSC9	B, 0, XA
		.
LA	ADSC9	A, 0, XA

As a precaution, all index register symbols should be defined with octal values in the range 10, 11,...,17 because the assembler uses only the low-order 3 bits (in all contexts except the indirect descriptor, where the symbol cannot be identified from context as an index register designation).

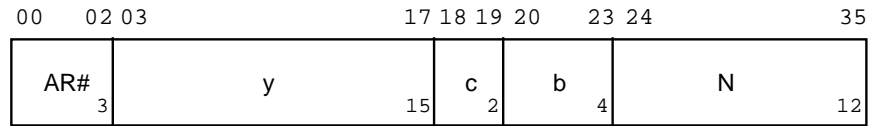
The content of the IC is always interpreted as a word address when used in address modification. During the entire execution of a multiword instruction, the IC points to the instruction word. Thus, if IC address modification is involved with a descriptor word, the instruction word address is used.

Specifying DU or DL type address modification in the REG field of an indirect operand descriptor is illegal and causes an IPR fault. DU address modification is legal for MF2 of the SCD, SCDR, SCM, and SCMR instructions. For all other instructions, an IPR fault occurs.

5.2.10 Operand Descriptors

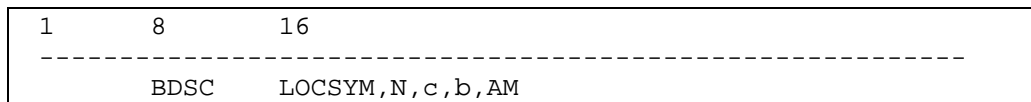
The operand descriptors describe the data to be used in the operation and provide the basic address for obtaining the data from memory. A unique operand descriptor format is required for each of the three data types: bit string, alphanumeric, and numeric.

5.2.10.1 Bit String Operand Descriptor

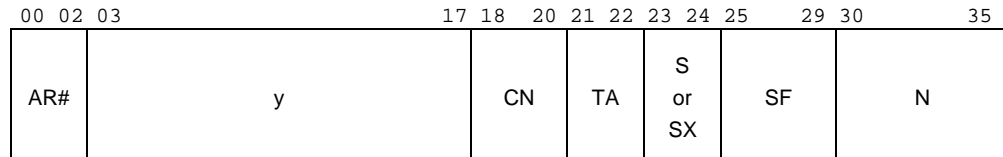


The coding format for the bit string descriptor (BDSC) is given below.

BDSC Bit descriptor

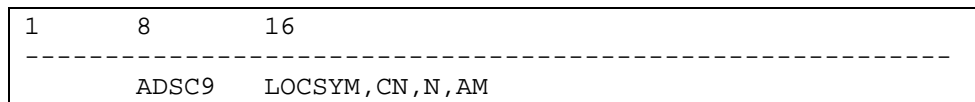


5.2.10.2 Alphanumeric Operand Descriptors



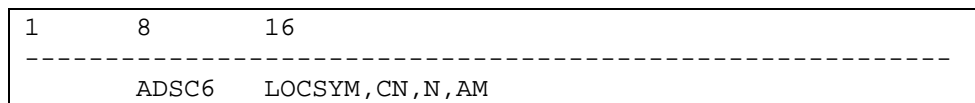
Coding formats for the alphanumeric descriptors are listed below.

1. **ADSC9 - ASCII alphanumeric descriptor**



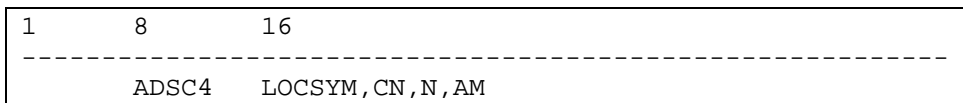
ADSC9 sets the TA field for 9-bit ASCII characters.

2. **ADSC6 BCI alphanumeric descriptor**



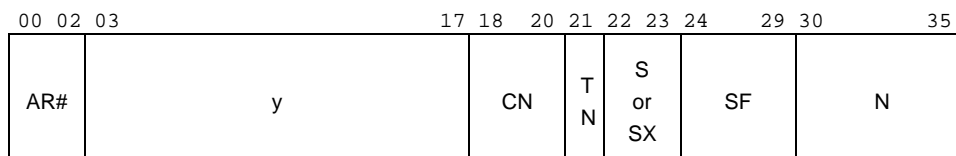
ADSC6 sets the TA field for 6-bit BCI characters.

3. **ADSC4 Packed decimal alphanumeric descriptor**



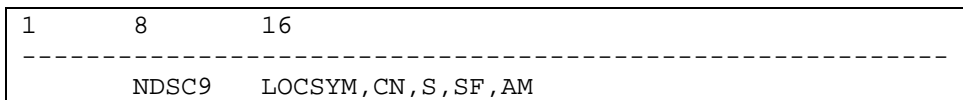
ADSC4 sets the TA field for 4-bit packed decimal characters.

5.2.10.3 Numeric Operand Descriptors



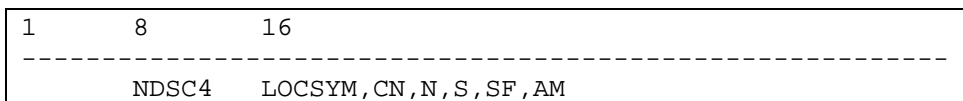
Coding formats for the numeric descriptors are listed below.

1. **NDSC9 ASCII numeric descriptor**



NDSC9 sets the TN field for 9-bit ASCII characters.

2. **NDSC4 Packed decimal numeric descriptor**



NDSC4 sets the TN field for 4-bit packed decimal characters.

Legend for the machine and coding formats of the descriptors

y = starting data word address;
 18 bits (0-17) if address register not specified in MF
 15 bits (3-17) if address register specified in MF, with bit 3 extended, i.e., if bit 3 is zero, bits 0-2 are also considered to be zero; if bit 3 is 1, bits 0-2 are also considered to be 1s.

Address Modification and Development

c = starting character position within a word of 9-bit characters;

<u>Code</u>	<u>Char</u>
00	0
01	1
10	2
11	3

b = starting bit position within a 9-bit character

<u>Code</u>	<u>Bit</u>
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5 All other combinations of
0110	6 these 4 bits are illegal
0111	7 codes and will cause an
1000	8 IPR fault.

N = either the number of characters or bits in the data string if RL = 0 in MF or a 4-bit code (bits 32-35) that specifies a register (see Table 5-2) that contains the number of characters or bits if RL = 1 in MF

CN = starting character number within the data word specified by the starting data word address; legal codes for the CN depends on the data type as shown below; coding entry is by the character shown under CN Character

<u>Data Type</u>	<u>CN Character</u>	<u>Legal Codes</u>	<u>Illegal Codes</u>
9-bit	0	000	001
	1	010	011
	2	100	101
	3	110	111
6-bit	0	000	110
	1	001	111
	2	010	
	3	011	
	4	100	
4-bit	5	101	
	0	000	
	1	001	
	2	010	
	3	011	
	4	100	
	5	101	
6	110		
7	111		

TA = a code that defines which type of alphanumeric character is used in the data

<u>Code</u>	<u>Data Type</u>
00	9-bit
01	6-bit
10	4-bit
11	illegal - causes IPR fault

TN = a code that defines which type of numeric character is specified

<u>Code</u>	<u>Data Type</u>
0	9-bit (unpacked data)
1	4-bit (packed data)

<u>S =</u>	<u>Character</u>	<u>Code</u>	<u>Description</u>
	0	00	Floating point, leading sign
	1	01	Scaled fixed-point, leading sign
	2	10	Scaled fixed-point, trailing sign
	3	11	Scaled fixed-point, unsigned

SX = sign and scaling (for X operation codes)

if TN = 0 (unpacked data)

00	leading sign, overpunched, scaled
01	leading sign, separate, scaled
10	trailing sign, separate, scaled
11	trailing sign, overpunched, scaled

if TN = 1 (packed data)

00	leading sign, overpunched, scaled
01	leading sign, separate, scaled
10	trailing sign, separate, scaled
11	trailing sign, overpunched, scaled

SF = scaling factor

a two's complement binary number that gives the scale point position for scaled decimal numbers

The decimal point is assumed to be immediately to the right of the least-significant digit. The scaling factor is treated as a power of ten exponent where a positive number moves the scaled decimal point to the right and a negative number moves it to the left. Since the SF field is 6 bits, the largest number expressible is $M \times 10^{+31}$ and the smallest number is $M \times 10^{-32}$, where M is the value of the data described by the numeric operand descriptor.

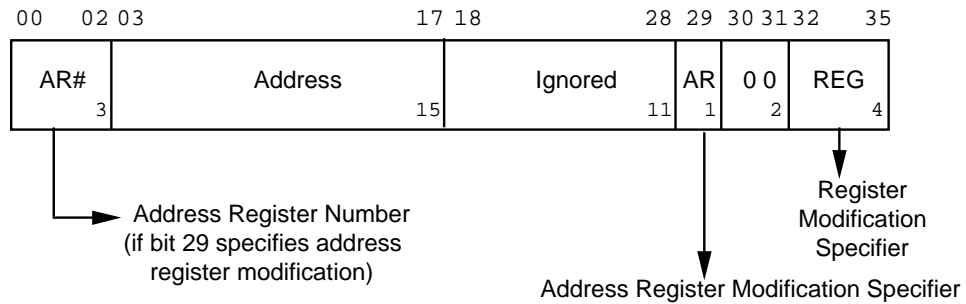
This field is ignored if S = 00.

Example: If data = 12345, S is not 00, and SF = -3, the value is 12.345.

AM = address register modification, used when AR = 1 in MF field

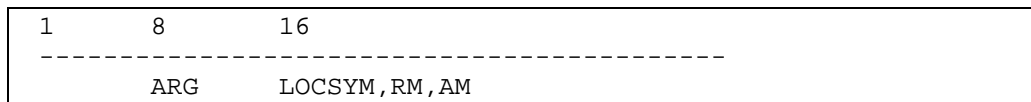
5.2.10.4 Indirect Word

The basic instruction word containing the operation code is followed by either zero, two, or three descriptor words, with the number of descriptor words being determined by the particular instruction. The descriptor words contain either the operand descriptor or an indirect word that points to the operand descriptor. When an indirect word points to the descriptor, the format of the indirect word is structured as shown below.



The AR and REG fields are identical in function to the corresponding modification fields in the instruction word, except that the register content specified by the REG field of an indirect word is interpreted as word index only.

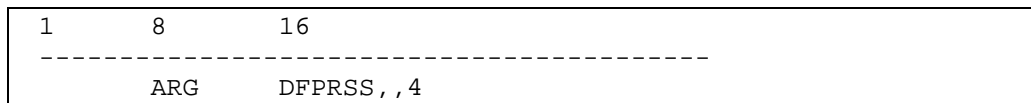
Indirect words can be generated with the ARG pseudo-operation:



where

- LOCSYM address
- RM register modification
- AM address register modification

Example



5.2.10.5 Operand Descriptor Address Preparation Flowchart

A flowchart of the operations involved in operand descriptor address preparation is shown in Figure 5-2. The chart depicts the address preparation for operand descriptor 1 of a multiword instruction as described by modification field 1 (MF1). A similar type address preparation would be carried out for each operand descriptor as specified by its MF code.

1. The multiword instruction is obtained from memory.
2. The indirect (ID) bit of MF1 is queried to determine if the descriptor for operand 1 is present or is an indirect word.
3. This step is reached only if an indirect word was in the operand descriptor location. Address modification for the indirect word is now performed. If the AR bit of the indirect word is 1, address register modification step 4 is performed.
4. The *y* field of the indirect word is added to the contents of the specified address register.
5. A check is now made to determine if the REG field of the indirect word specifies that a register type modification be performed.
6. The indirect address as modified by the address register is now modified by the contents of the specified register, producing the effective address of the operand descriptor.
7. The operand descriptor is obtained from the location determined by the generated effective address in item 6.
8. Modification of the operand descriptor address begins. This step is reached directly from 2 if no indirection is involved. The AR bit of MF1 is checked to determine if address register modification is specified.
9. Address register modification is performed on the operand descriptor as described under "Address Modification with Address Registers" above. The character and bit positions of the specified address register are used in one of two ways, depending on the type of operand descriptor, i.e., whether the type is a bit string, a numeric, or an alphanumeric descriptor.
10. The REG field of MF1 is checked for a legal code. If DU is specified in the REG field of MF2 in one of the four multiword instructions (SCD, SCDR, SCM, or SCMR) for which DU is legal, the CN field is ignored and the character or characters are arranged within the 18 bits of the word address portion of the operand descriptor.
11. The count contained in the register specified by the REG field code is appropriately converted and added to the operand address.
12. The operand is retrieved from the calculated effective address location.

Address Modification and Development

Operand descriptor address preparation is illustrated in the flowchart in Figure 5-2. Procedures for the preparation of bit string addresses and alphanumeric/ numeric addresses follow.

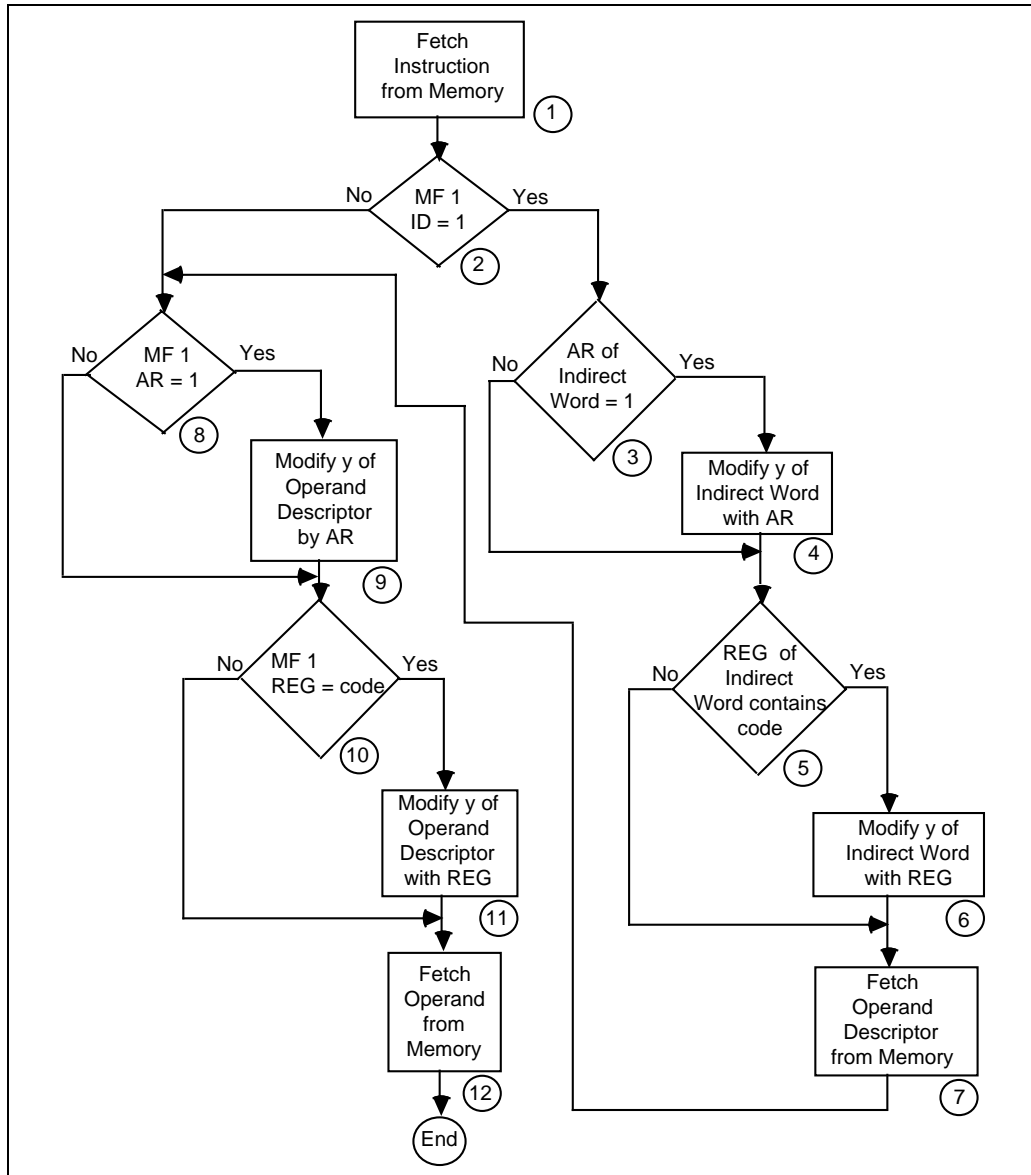
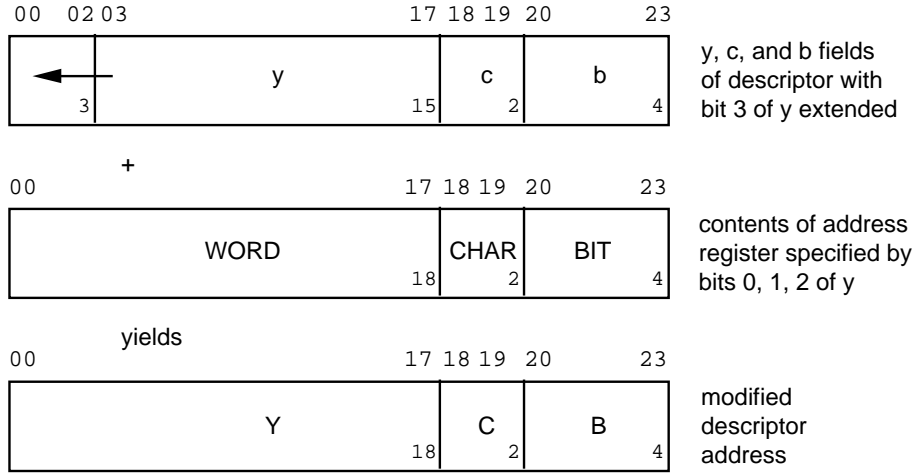


Figure 5-2. Flowchart for Operand Descriptor Address Preparation

5.2.10.6 Operand Descriptor Bit String Address Preparation



where:

$Y = WORD + y$

$C = CHAR + c$

$B = BIT + b$

- If (BIT + b) exceeds 8, a carry is generated to character position
 C and $B = (BIT + b) - 9$

```

      BIT = 7
      b   = 5
      -----
      BIT + b = 12   carry 1 to C and B = 12 - 9 = 3
    
```

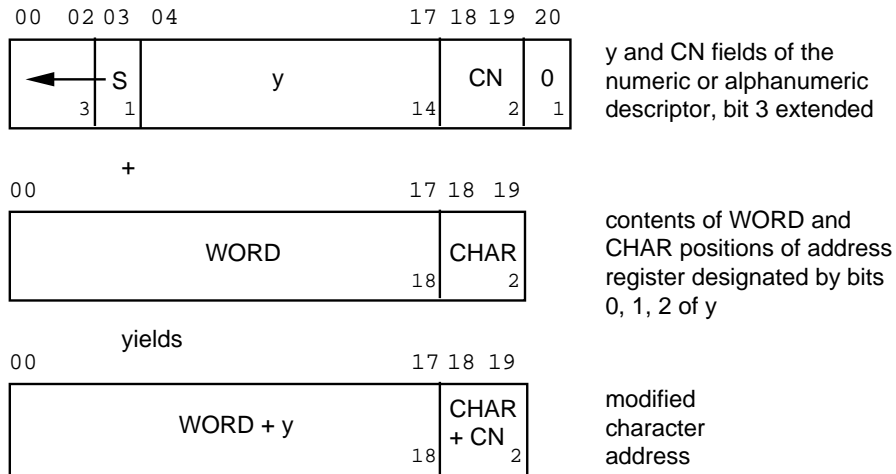
- If (CHAR + c + carry from B) exceeds 3, a carry is generated to the word address and $C = (CHAR + c + carry from B) - 4$:

```

      CHAR = 2
      c    = 3
      carry + 1
            = 6,   carry 1 to word address and
                  C = 6 - 4 = 2
    
```


5.2.10.7 Operand Descriptor Alphanumeric/Numeric Address Preparation

First, the data type designator (TA for alphanumeric, TN for numeric) is checked to determine the character size. If the data is in 9-bit characters, then the descriptor address and CN fields can be added directly to the address register contents.



Bits 20-23 of the address register are ignored. CHAR is added to bits 18 and 19 of CN. Bit 20 of the descriptor is zero and is not used. If CHAR + CN is greater than 3, a carry is generated to WORD + y and CHAR + CN = (CHAR + CN) - 4.

If the data is in 4- or 6-bit characters, the 9-bit character representation contained in the CHAR and BIT portions of the specified address register is interpreted to determine the corresponding 4- or 6-bit character position within the memory word. Translation to a 4-bit character location can be accomplished in the following way:

$$C = 2 (\text{CHAR}) + [(\text{BIT} + 4) / 9 \text{ truncated}].$$

If CHAR = 3 and BIT = 7,
then C = 2(3) + 1 = 7.

If CHAR = 3 and BIT = 4,
then C = 2(3) + 0 = 6.

A 6-bit character location can be translated in this way:

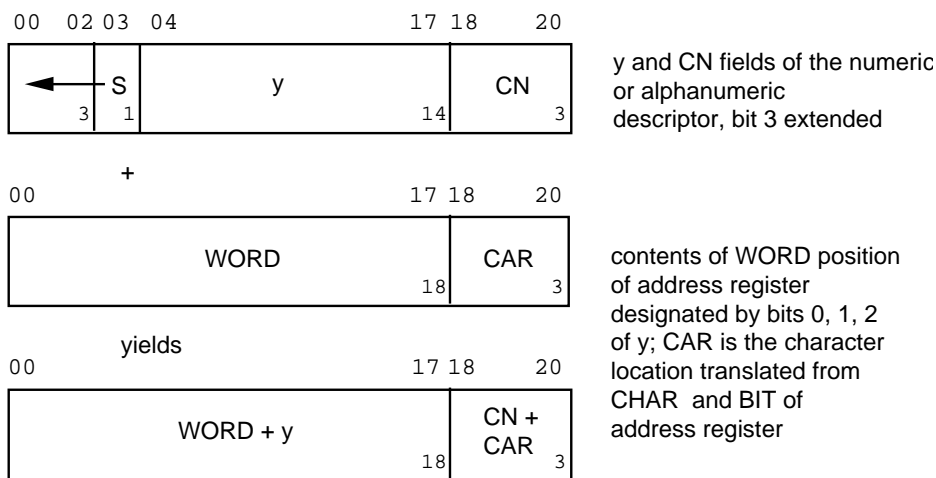
$$C = \frac{9 (\text{CHAR}) + \text{BIT}}{6} \quad (\text{truncated})$$

If CHAR = 3 and BIT = 7,

$$\text{then } C = \frac{9 (3) + 7}{6} = 5$$

The remainder of 4 (which represents the bit position within character position 5) is ignored, forcing the address register to point to the next lower character boundary.

The address modification can now take place.

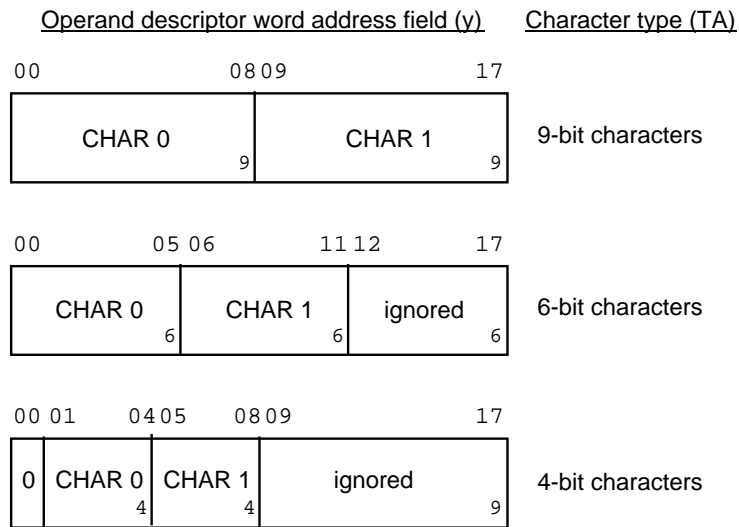


For 4-bit character mode, if CN + CAR is greater than 7, a carry is generated to WORD + y and CN + CAR = (CN + CAR) -8.

For 6-bit character mode, a carry is generated to WORD + y when CN + CAR is greater than 5 and CN + CAR = (CN + CAR) -6.

Address Modification and Development

In the next step of operand descriptor address preparation (item 10 in the flowchart of Figure 5-2), the REG field is checked for a legal code. If DU is specified in the REG field of MF2 in one of the four multiword instructions (SCD, SCDR, SCM, or SCMR) for which DU is legal, the CN field is ignored, and the following arrangement of character or characters within the 18 bits of the word address portion of the operand descriptor results.



In cases where only one character is involved (SCM, SCMR), only character 0 is used.

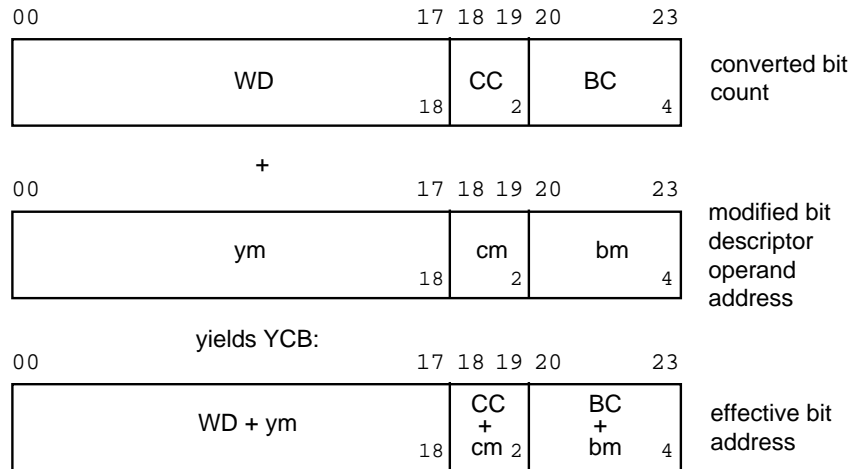
In step 11 of the flow chart (Figure 5-2), the count contained in the register specified by the REG field code is appropriately converted and added to the operand address. The count conversion required depends upon the type of data.

5.2.10.8 Operand Descriptor Address Preparation - Bit Operations

The bit count contained in the register is effectively divided by 36 to give a word count (WD) with a bit remainder (BR). Dividing the bit remainder by 9 gives a character count with a bit remainder. Thus, the original bit count (BC) is converted to a word count, 9-bit character count (CC) and bit remainder, which is in proper form to add to the bit operand address. An example of the effective conversion is shown below.

bit count from register/36 = WD and BR
 BR/9 = CC and BC

expressed as a 24-bit address modifier



Carries may occur from (BC + bm) to (CC + cm) and from (CC + cm) to (WD + ym) as described in step 9.

Address Modification and Development

The formation of WD involves two conditions:

1. If WD is a small number (expressible in less than 18 bits), it is right justified in the 18-bit word area with zero-fill in the most-significant bit positions. Thus, bit counts are always positive, are not two's complement, and have no bit extensions.
2. If the bit count comes from the A- or Q-registers, division by 36 may produce a WD greater than $2^{18}-1$. In this case, the result is interpreted modulo 2^{18} . For example, if the bit count is $(2^{24})-1$:

$$\begin{aligned} & \frac{(2^{24})-1}{36} &= & 466,033 \text{ with BR} = 27 \\ \text{Thus, WD} & &= & 466,033 - 262,144 = 203,889 \\ \text{And, BR/9} & &= & 27/9 = 3 \text{ with 0 remainder} \\ \text{So that, WD} & &= & 203,889 \\ & \text{C} &= & 3 \\ & \text{BC} &= & 0 \end{aligned}$$

No errors occur. The operation is legal and the results are predictable.

5.2.10.9 Operand Descriptor Address Preparation - Character Operations

The character count contained in the register is divided by 4, 6, or 8 (depending upon the data type), which gives a word count with a character remainder. The word and character counts are then appropriately arranged in 21 bits (18-word address and 3 for character position) and added to the modified descriptor operand address. The appropriate carries occur from the character positions to the word when the summed character counts exceed the number of characters in a 36-bit word. When the A- or Q-registers are specified, large counts can cause the result of the division to be greater than $2^{18}-1$, which is interpreted modulo 2^{18} , the same as for bit addressing.

As the final step (see 12 in Figure 5-2), the calculated effective address location is used to retrieve the operand.

EXAMPLES

1	8	16	32

* OPERAND DESCRIPTOR EXAMPLES			
MLR	,,020	,1	move blanks to output record
ADSC6	,,0		
ADSC6	PRTOUT	,0,55+80-31	
MLR			move columns 31-80
ADSC6	RDWRK+5	,0,80-31+1	to print columns 55-104
ADSC6	PRTOUT+9	,0,80-31+1	
LDX7	31-1	,DU	ditto
LDX6	55-1	,DU	
LAR5	=V18	/RDWRK	
LAR4	=V18	/PRTOUT	
MLR	(1,,7)	,(1,,6)	
ADSC6	,,80-31+1	,5	
ADSC6	,,80-31+1	,4	
LAR5	=V18	/RDWRK	ditto
LAR4	=V18	/PRTOUT	
LDX3	80-31+1	,DU	
MLR	(1,1)	,(1,1)	
ADSC6	5,0	,X3,5	
ADSC6	9,0	,X3,4	

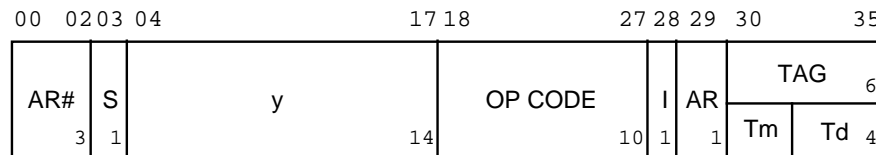
5.3 Address Generation In ES/EI Modes

This subsection discusses the generation of effective addresses only insofar as it differs from the NS mode.

The instruction field and register used in the generation of an effective address are interpreted in a number of ways. See the subsections below.

5.3.1 Instruction Address Field

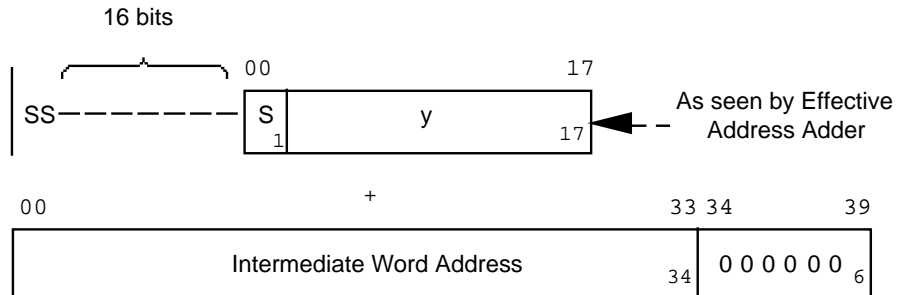
Address preparation for all instructions starts with the address field of an instruction word (or the address field of an indirect word or data descriptor). All instruction words have the same format:



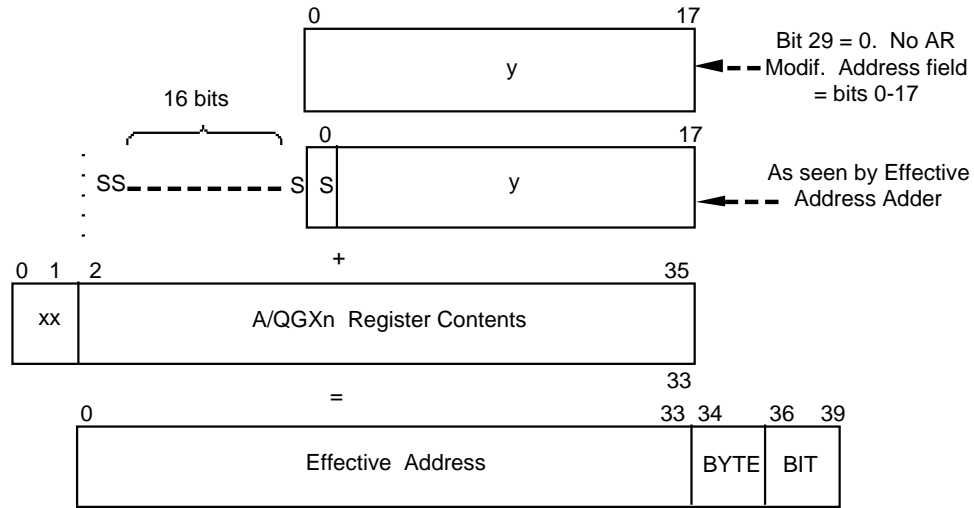
Definitions for the individual fields of this format are found under "Single-Word Address Modification" in this section. The diagrams that follow start with only the address portion of an instruction field (bits 0 - 17).

5.3.2 Address Modification With No AR Indicated

When bit 29 = 0, no AR modification is specified. The sign (S) of (y) is extended 16 bits to the left, starting at bit 0.



When bit 29 = 0, address generation proceeds in the following way.



The y field of an instruction/indirect word/data descriptor is interpreted as given in the two's complement form. Bit 0 is assumed as a sign. To generate the effective address, bit 0 is extended 16 bits to the left. Bit 17 expresses the word location. Up to 128KW-1 can be used to represent addresses in the positive direction. When the A, Q, or a GX_n register is used in the R modification of a basic instruction (single-word), bits 2 through 35 are treated as word address and bits 0 and 1 are ignored. An AL/QL specification in the tag field modification specifies 36-bit A/Q registers. An AU/QU specification results in an IPR fault. Address modification specified by the tag field is performed, resulting in the effective address.

In EI mode only, IC modification causes bits 0-33 of the IC to be used as a word offset.

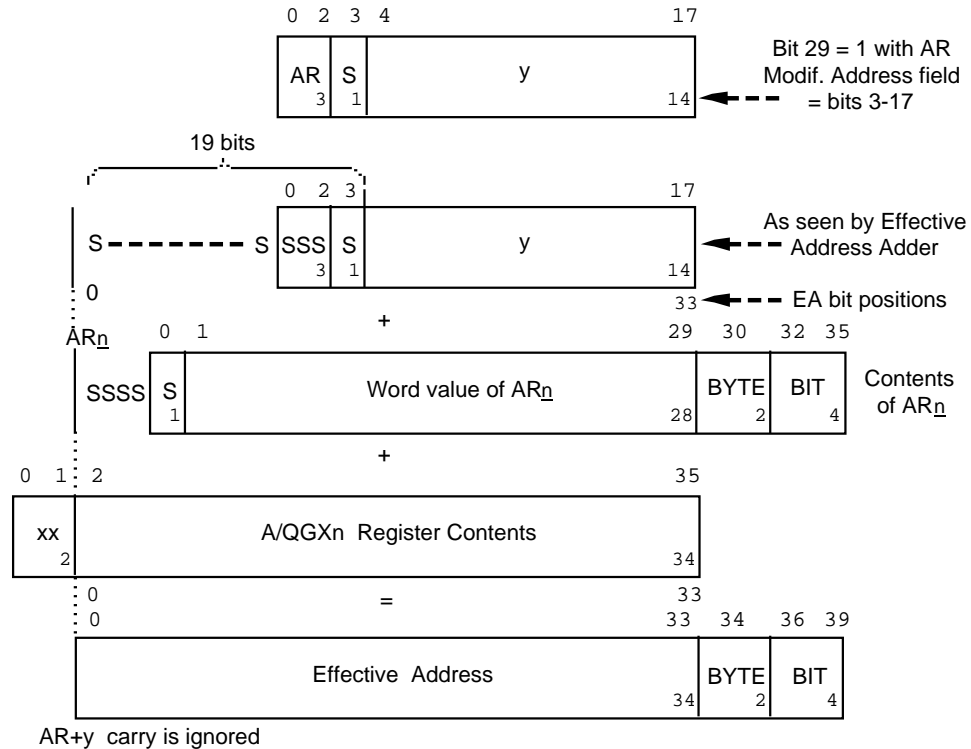
EXAMPLES

	1	8	16	EFFECTIVE ADDRESS
(1)		EAX4	1	Y = B+1
	B	LDA	B, 4	
(2)		LDA	=4, DL	Y = C+4
	C	LDQ	C, AL	
(3)		EAQ	3	Y = B+3
	B	STA	B, QL	

With no AR modification specified, address modification is processed in the same way as address modification in NS mode, with the exception of the AU/QU modification.

5.3.3 Address Modification With AR Indicated

Address register modification is performed when instruction word bit 29 = 1 or when the AR bit of a multiword instruction's MF field is 1.



Bits 3 through 17 of an instruction/indirect word/data descriptor are interpreted as given in a two's complement form. Bit 3 is assumed as a sign. To generate an effective address, bit 3 is extended 19 bits to the left. Bit 17 expresses the word location.

The address register AR_n is interpreted as extended to 36 bits as indicated in the previous format. AR_n is interpreted as given in a two's complement form with bit 0 as a sign bit. In effective address generation, bit 0 is extended 4 bits to the left. Bits 0 through 29 are interpreted as a word address, bits 30 and 31 as a byte address within the word, and bits 32 through 35 as a bit address within the byte. If BIT > 8, BIT = 8 is assumed.

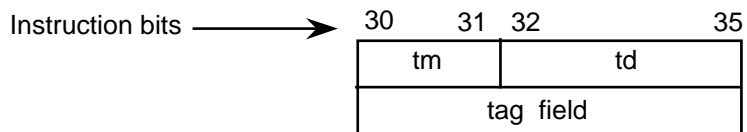
Every specification of an index register (X_n) is interpreted as specifying a 36-bit GX_n. An AL/QL specification in the register modification (R modification, REG modification, N when RL = 1) specifies the 36-bit A/Q registers. Any AU/QU specification results in an IPR fault. When GX_n is used in the R modification of a basic instruction (single-word instruction), bits 2 through 35 are treated as a word address. When GX/A/Q is used in the REG modification of a multiword instruction, bits 0 through 35 are treated as the number of bytes specified by the bit number in the data descriptor.

EXAMPLES

	1	8	16	EFFECTIVE ADDRESS
(1)		EAX2	2	(X2=2)
		AWDX	1,2,3	AR3 = 3/0/0
		STZ	B,2,3	Y = B+5
(2)		EAX3	1	(X3=1)
		AWDX	2,3,1	AR1=3/0/0
		LDA	B,,1	Y=B+3
(3)		AWDX	4,,3	AR=4/0/0
		EAX4	B	X4= address of B
		STA	1,4,3	Y=B+5
(4)		EAX4	B	
		AWDX	0,4,2	AR2= address of B
		STA	2,,2	Y=B+2

5.3.4 Tag Field Modification

In a basic instruction (single-word instruction), a tag field modification is performed after the AR modification (see format below).



The interpretation of a tag field and the accompanying modification method are the same as in the NS mode except that the address modification by the register (A/Q/GX_n/IC is altered as illustrated below. This modification generates the following items:

- an operand address in R modification (tm = 00),
- an indirect word address in RI modification (tm = 01), and
- an operand address in IR modification (tm = 10).

The following should be noted with A/Q/GX_n modification:

1. EA (effective address) may be represented as Y;
2. The GX_n specification code is identical to the X_n specification code;
3. The A/Q specification code is identical to the AL/QL specification code; and
4. An AU/QU specification results in an IPR fault.

EXAMPLES

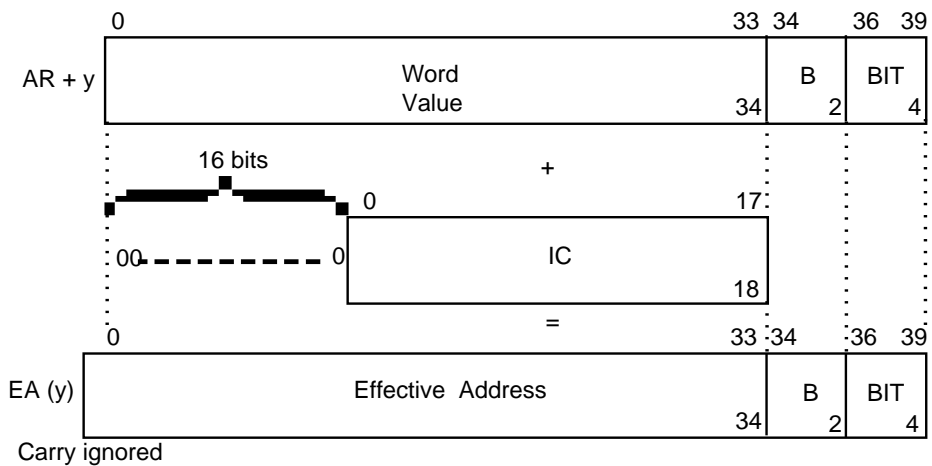
1	8	16	EFFECTIVE ADDRESS
R-Type			
(1)	EAX2 LDA	1 B, 2	Y=B+1
(2)	LDQ LDA	=3, DL B, QL	Y=B+3
RI-Type			
Z	ARG ARG ORG	B A, 2* A+5	
A	ARG ORG . .	B, 5* B+1 . .	
(1)	EAX2 LDA	1 Z, 2*	Y=B+1
(2)	EAX1 STQ	0 Z, 1*	Y=B
(3)	EAX2 STA	3 Z, 2*	Y=A+5
IR-Type			
(1)	LDQ LDA . .	3, DL Z, *QL . .	Y=B+4
Z	ORG	B+1	
(2)	EAX1 LDQ . .	3 X, *1 . .	Y=B+8
X	ORG	B+5	

5.3.5 IC Modification

IC modification is the only case where effective address generation is different for ES and EI modes.

5.3.5.1 ES Mode

When IC modification is specified, effective address development in ES mode is structured in the following way.



The contents of the instruction counter extended on the left with 16 bits zero filled is added to the contents of AR + y.

EXAMPLES

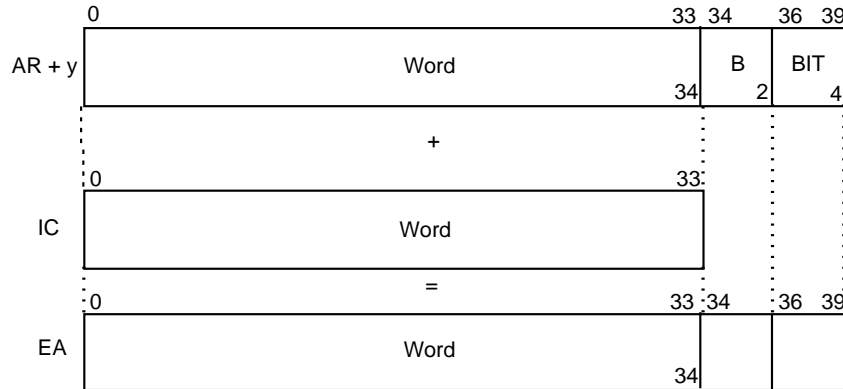
	1	8	16	EFFECTIVE ADDRESS

IC added to AR				
(1)	AWDX	0, QL, 3		
	AWDX	1, QL, 4		
	AWDX	2, QL, 2		
	SZN	TEST		
	TZE	TEST		Y=IC+AR3
	TMI	0, \$, 4		Y=IC+AR4
	TRA	0, \$, 2		Y=IC+AR2
	.			
	.			
(2)	AWDX	1, AL, 2		
	LDA	2, \$, 2		Y=IC+AR2

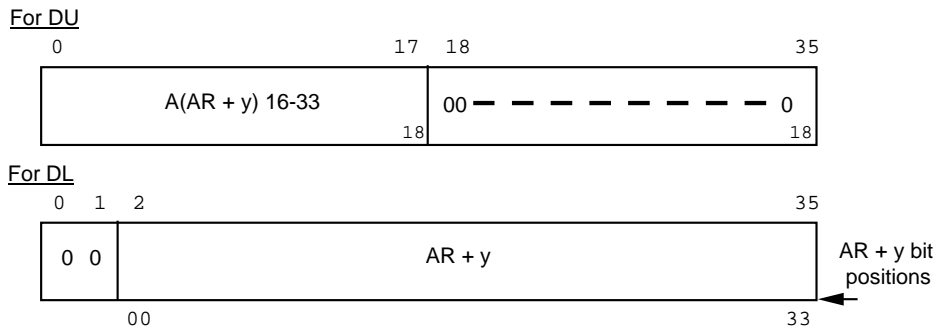
5.3.5.2 EI Mode

A 34-bit IC is added to the contents of AR + y in EI mode. When the instruction TAG specifies IC REG modification, IC(0-33) is used as a word address.

When IC modification is specified, effective address development in EI mode is structured in the following way.



When DU/DL modification is specified, effective address modification interprets the operand data in the following way.



EXAMPLES

	1	8	16	EFFECTIVE ADDRESS

Compare GX1 to AR3				
(1)	EAX1	A		GX1 = address of A
	CMPX	1, DL, 3		
Load AU with contents of AR2				
(2)	EAX3	B		AR2=address of B
	AWDX	0, 3, 2		
	LDA	0, DU, 2		

5.3.6 Operand Descriptor Modification

When REG modification is specified in the MF field of a multiword instruction, it is processed in the following way.

When A/Q/GX_n is specified,

1. the 36 bits of A/Q/GX_n are used as the character number which is the character address and
2. an AU/QU specification results in an IPR fault.

EXAMPLES

1	8	16	EFFECTIVE ADDRESS

(1)	This code moves the string "SOURCE" to the first six characters of "TO". The contents of X3 act as an offset into the source text.		
	LDX3	=11,DL	
	.		
	.		
	MLR	(,,,3),,040	
	ADSC9	FROM,1,6	
	ADSC9	TO,0,6	
	.		
	.		
FROM	ASCII	9,THIS IS THE SOURCE TEXT	
TO	BSS	2	
(2)	The string "LE " is moved to XB, starting at the third character of XB. The Q register can be used in the same way.		
	LDA	=4,DL	
	.		
	.		
	MLR	(,,,A),(,,,,),040	
	ADSC9	XA,0,3	
	ADSC9	XB,2,3	
	.		
	.		
XA	ASCII	5,SAMPLE TEXT TO MOVE	
XB	BSS	3	

When IC is specified in the REG modification, it is treated as an 18-bit word address. However, when the CPU is in EI mode and IC REG modification is specified in the MF field of a multiword instruction, IC(0-33) is used as a word address.

EXAMPLES

1	8	16	EFFECTIVE ADDRESS

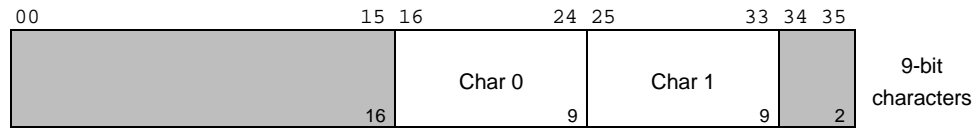
The string "HIS IS" is moved to Y, beginning with the first character.			
	EAX3	Y	
	AWDX	0,3,2	AR2=address of Y
	.		
	MLR	(,,,IC),(1,,,),040	
	ADSC9	3,1,6	
	ADSC9	0,0,6,2	
X	ASCII	4,THIS IS THE TEXT	
Y	BSS	2	

When DU/DL is specified,

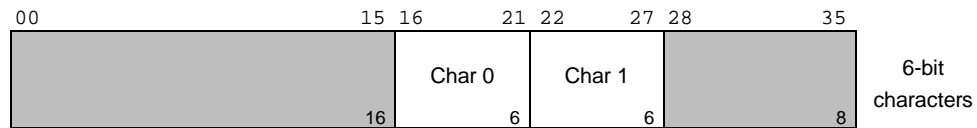
1. DL an IPR fault occurs;
2. DU permitted only in the SCD, SCDR, SCM, and SCMR instructions.

The effective address (EA(y)) generated by the operand descriptor is treated as follows:

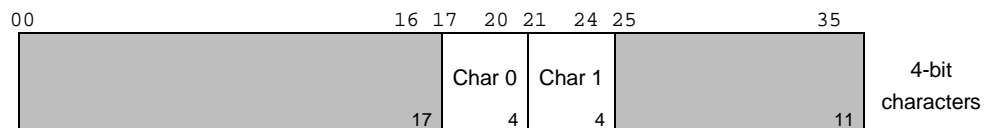
For 9-bit characters, Bits 16 through 33 of the effective address (EA(y)) are interpreted as character data according to its data format (TA or TN field of the descriptor).



For 6-bit characters, Bits 16 through 27 of the effective address (EA(y)) are interpreted as character data according to its data format (TA or TN field of the descriptor).



For 4-bit characters, Bits 17 through 24 of the effective address (EA(y)) are interpreted as character data according to its data format (TA or TN field of the descriptor).



For the SCM or SCMR instructions, only CHAR0 indicated in the diagrams is used. The shaded portions are ignored during effective address generation.

5.4 Address Development

This section discusses the following details of address development: virtual memory addressing, virtual address generation for NS mode, virtual address generation for ES/EI modes; and working space zero.

5.4.1 Virtual Memory Addressing

Virtual memory provides the processor with a virtual memory capability, consisting of a directly addressable virtual space of 2^{43} bytes and the mechanisms for translating this virtual memory address to a real memory address. Memory paging is an integral part of the translation process for this conversion. In both NS and ES modes, the hardware can also enter working space 0 and use page tables prepared by the Maintenance System Operating Supervisor when the system was initialized. In WS0, a total of 2^{28} bytes can be accessed.

To provide for virtual memory management, assignment, and control, the 2^{43} byte virtual memory space is divided into smaller units called working spaces and segments.

Working Spaces (WS)

The 2^{43} bytes of virtual memory space are divided into 512 2^{34} -byte working spaces (WS). WS numbers used to generate a particular virtual memory address are obtained from one of the eight WS registers or a segment descriptor register (Dr). The WS number is the contents of the 9-bit WS registers or a 9-bit WSN field in a segment descriptor register.

Segments

A segment is part of a working space and may be as small as one byte or as large as 2^{32} bytes for an extended segment. (GCOS disallows the use of contiguous working spaces for a single segment.) Thus, unlike the fixed size of a WS, a segment size is variable. Segments are described by a 72-bit data item called a descriptor.

When a virtual address is generated, the descriptor (more commonly referred to as the segment descriptor) is contained in a register such as the instruction segment register (ISR). For operands, the descriptor may be contained in other segment descriptor registers. The area of virtual memory constituting a segment is "framed" by the segment descriptor by defining a base value relative to the base of the WS and a bound value relative to the base of the segment.

Virtual memory affects memory address development for both instructions and operands in Privileged Master, Master and Slave modes of operation.

5.4.1.1 Operand Address Procedure

In the first phase of address generation, the effective address (EA) of the operand is generated as previously described for effective address generation. The EA is that address obtained after all register modification and indirect processing has taken place. It is an 18-bit word, 20-bit byte, or 24-bit bit address in the NS mode, and a 30-bit word, 32-bit byte, or 36-bit bit address in the ES/EI mode.

After the EA has been formed, the processor hardware forms the virtual memory address of the operand using the base, bound, and WS values from 1 of 9 segment descriptors. If bit 29 of the instruction for which the operand address is being prepared is zero, then the operand resides in the instruction segment, and the base, bound, and WS from the instruction segment register (ISR) are used to form the virtual address of the operand. If bit 29 of the instruction is one, then descriptor register \underline{n} (Dr) specified by bits 0, 1, and 2 of the address field of the instruction is used.

NOTE: Specifying $DR_{\underline{n}}$ constitutes specifying $AR_{\underline{n}}$ and vice versa.

When indirect EA development is involved, the following rules apply.

- a) When $DR_{\underline{n}}$ and $AR_{\underline{n}}$ are involved (instruction bit 29 = 1), $AR_{\underline{n}}$ is applied only to the first address in a chain of indirect addresses. However, the base, bound, and WS from $DR_{\underline{n}}$ are applied to each memory reference in the indirect chain.
- b) When no $DR_{\underline{n}}/AR_{\underline{n}}$ is specified (instruction bit 29 = 0), the base, bound, and WS of the ISR are applied to each memory reference in an indirect chain.
- c) A word in an indirect chain cannot specify a $DR_{\underline{n}}$.
- d) An XEC or XED instruction does not constitute an indirect chain. Therefore, the instruction executed may specify a different $DR_{\underline{n}}$ from the XEC/XED instruction, or no $DR_{\underline{n}}$. If the instruction executed by the XEC/XED does not specify a $DR_{\underline{n}}$, the base, bound, and WS from the ISR are used to form the virtual address of the operand.

5.4.1.2 Instruction Address Procedure

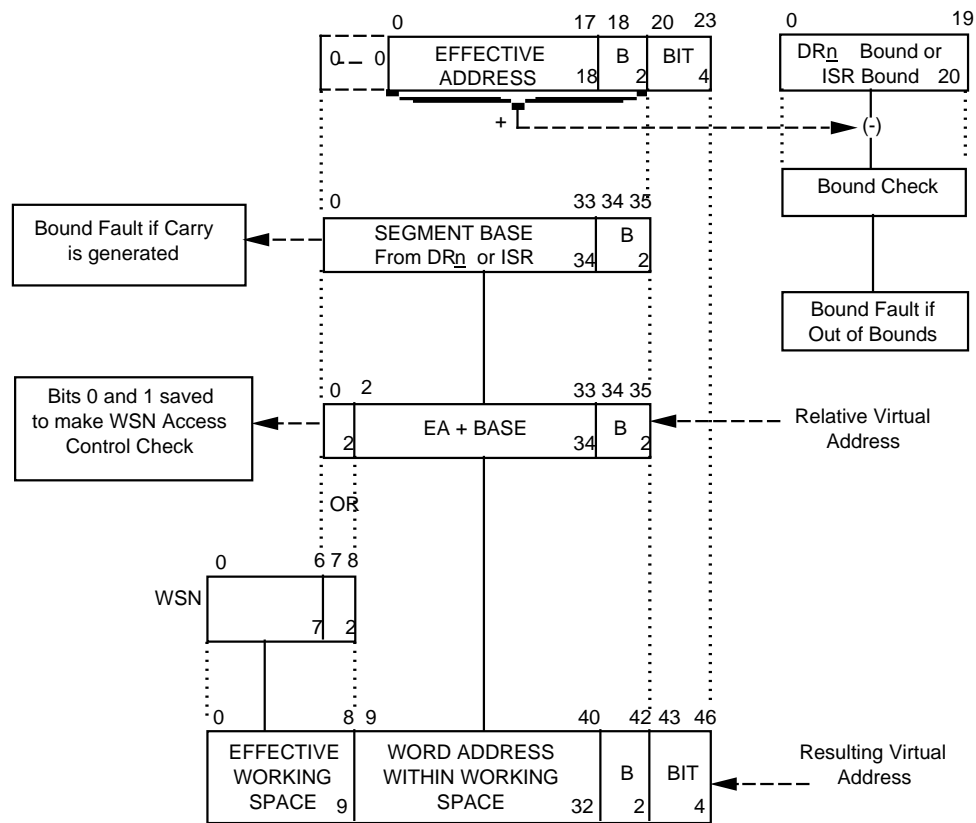
Virtual addresses for instructions are always formed using the value in the instruction counter (IC) and the base, bound, and WS from the ISR.

5.4.2 Virtual Address Generation For NS Mode

For all memory accesses, a virtual address must be generated. The mechanics of generating the virtual memory address depend on whether the involved segment descriptor is a standard descriptor or a super descriptor. Thus, the procedure described below for generating the operand virtual address with a standard descriptor also applies to virtual address generation for accessing the instruction, argument, parameter, and linkage segments (the registers holding the descriptors that define these segments may only contain standard descriptors).

5.4.2.1 Standard Descriptor NS Mode

The method of forming an operand virtual address with a standard descriptor is shown in Figure 5-3. If instruction bit 29=0, the ISR is used. If bit 29=1, then DR_n is used.



where B = page byte
WSN = working space number

Figure 5-3. Virtual Address Generation Using Standard Descriptor (NS Mode)

The bound check is applied to the effective address at the byte level. The bound check is shown for byte or bit instructions. The checks for single-word or multiword instructions require inclusion of the base in upper- and lower bound algorithms.

If a carry is generated when the EA is added to the base, an out-of-bound situation exists, resulting in a Bound fault.

The effective WSN is formed by ORing the low-order two bits of the working space number with bits 0 and 1 of the sum of EA + BASE.

The bit address from the EA becomes the bit address of the virtual address.

5.4.2.2 Super Descriptor NS Mode

The processor does not use the super descriptor directly for address generation. Instead, each time a DR_n is loaded with a super descriptor or the LDEAN instruction is executed, the processor generates a standard descriptor from the super descriptor and holds this generated descriptor in a temporary working register. Then, when a DR_n containing a super descriptor is referenced for address generation, the processor uses the standard descriptor previously generated. This procedure is transparent to software and improves the efficiency of the processor when super descriptors are used. Any software operation (such as copy to another DR or store in memory) with a super descriptor contained in a DR_n is performed using the super descriptor, not the generated standard descriptor.

The following steps describe how the processor generates a standard descriptor from a super descriptor.

1. The base for the standard descriptor is formed as shown in Figure 5-4. If a carry occurs, flag bit 27 of the formed descriptor is forced to zero (bound not valid). Thus, any attempt to generate an address using the formed standard descriptor will result in a Bound fault.

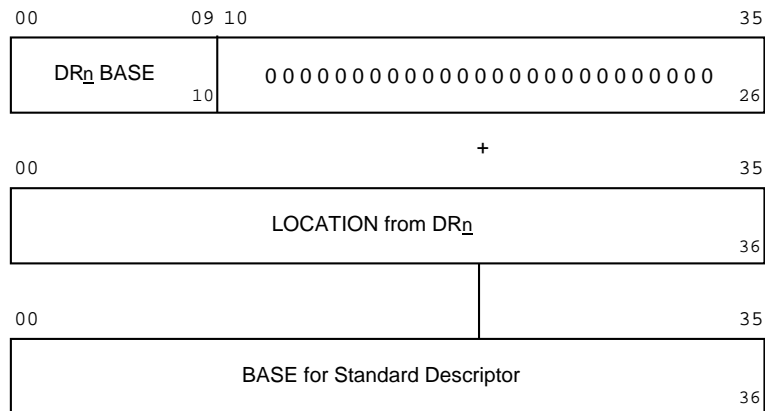


Figure 5-4. BASE For Standard Descriptor

Address Modification and Development

2. The bound for the standard descriptor is formed as shown in Figure 5-5.
 - a) If resulting bits 0-15 are zero, bits 16-35 become the 20-bit bound field.
 - b) If resulting bits 0-15 are not zero, the 20-bit bound field of the standard descriptor is forced to all ones.
 - c) If a borrow occurs in this operation, flag bit 27 of the formed descriptor is forced to zero (bound not valid). Thus, any attempt to access the segment using the formed standard descriptor will result in a Bound fault.

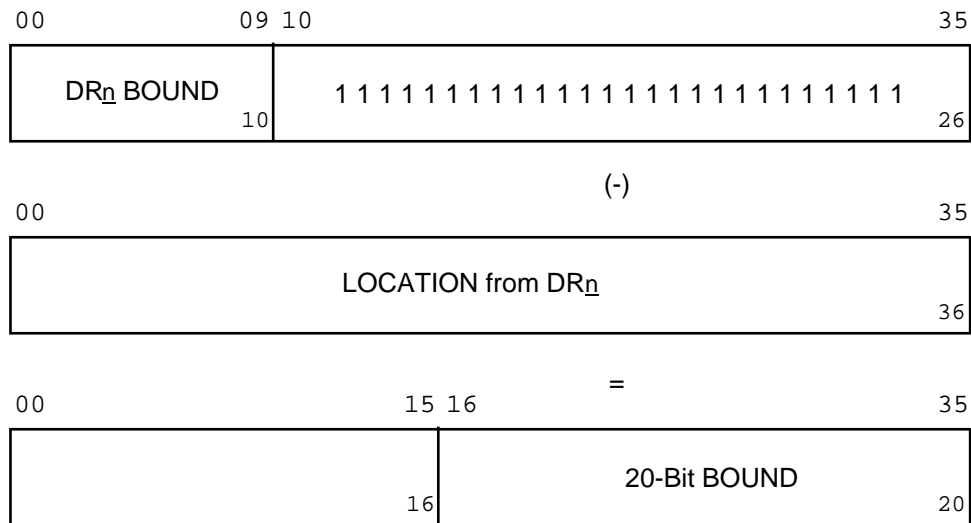


Figure 5-5. Bounds For Standard Descriptor

When a T = 6 descriptor is loaded into a DR_n register, a "standardized" descriptor is formed. If this standardized descriptor is to be marked "bound not valid" (i.e., bit 27 = 0), the instruction loading the DR_n will terminate with a Bound fault. This action is required since T = 2, 3, 6 descriptors are assumed to have bit 27 = 1.

5.4.2.3 Extended Segment Descriptor NS Mode

The method of forming an operand virtual address with an extended segment descriptor is shown in Figure 5-6 and is the same as that with a standard segment descriptor except in the bound check.

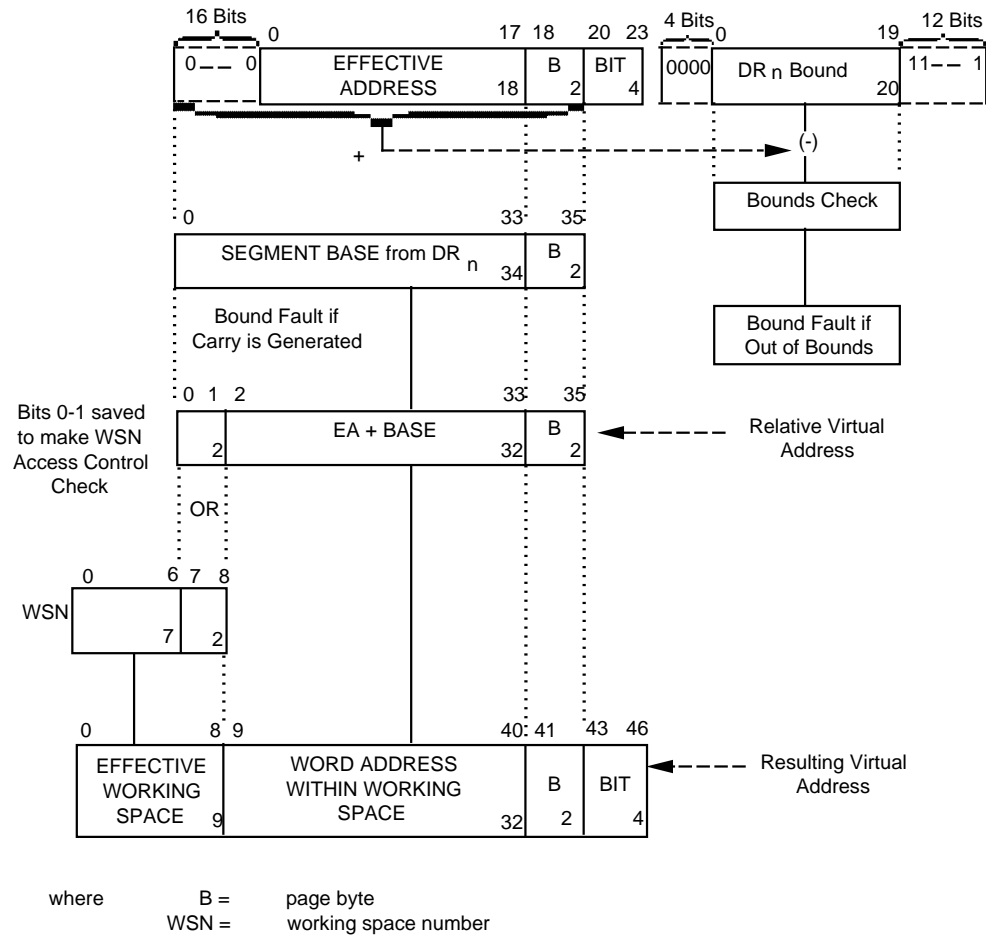


Figure 5-6. Virtual Address Generation Using Extended Segment Descriptor (NS Mode)

5.4.3 Virtual Address Generation For ES/EI Modes

The generation of the virtual address for a standard descriptor in ES/EI mode is similar to that for NS mode. In the ES/EI mode, a 36-bit effective address is added to a segment descriptor to generate a virtual address. The method used for generation of virtual addresses differs depending upon whether the related segment descriptor is a standard segment descriptor or an extended segment descriptor. Super descriptors must not be used for address generation in ES/EI mode as any attempt to do so results in an IPR fault.

5.4.3.1 Standard Descriptor ES Mode

The method of forming an operand virtual address with a standard descriptor in ES mode is shown in Figure 5-7. If instruction bit 29=0, the ISR is used. If bit 29=1, then DR_n is used.

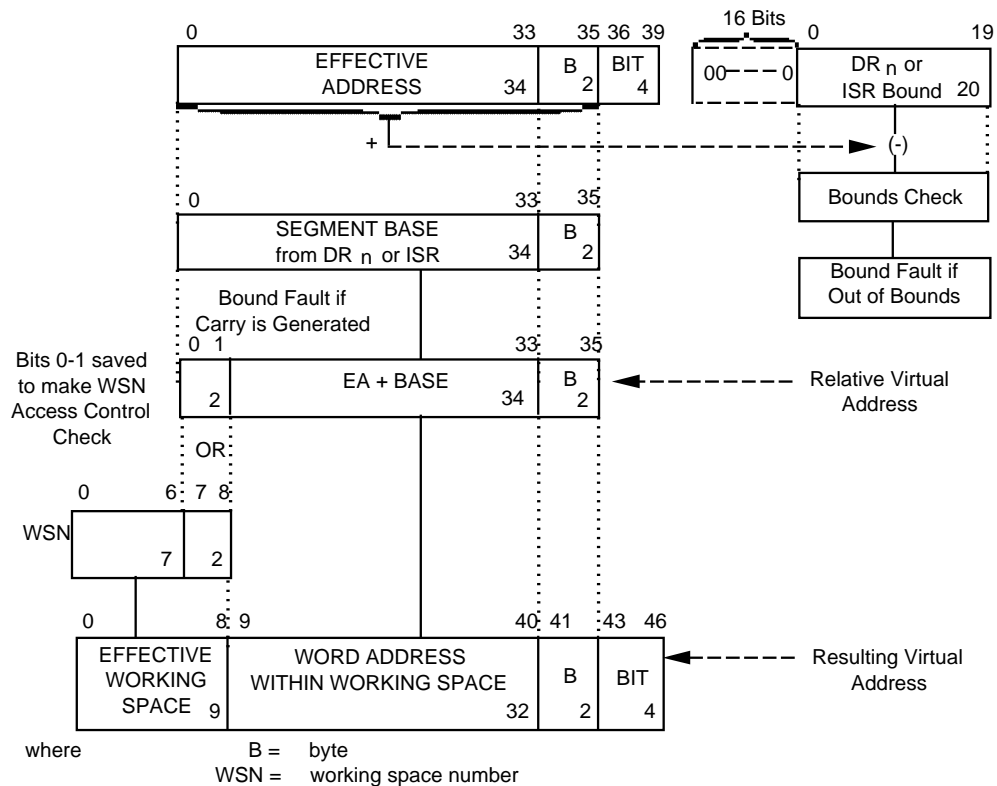


Figure 5-7. Virtual Address Generation Using Standard Descriptor (ES Mode)

5.4.3.2 Extended Segment Descriptor ES Mode

The method of forming an operand virtual address with an extended segment descriptor is shown in Figure 5-8 and is the same as that with a standard segment descriptor except in the bound check.

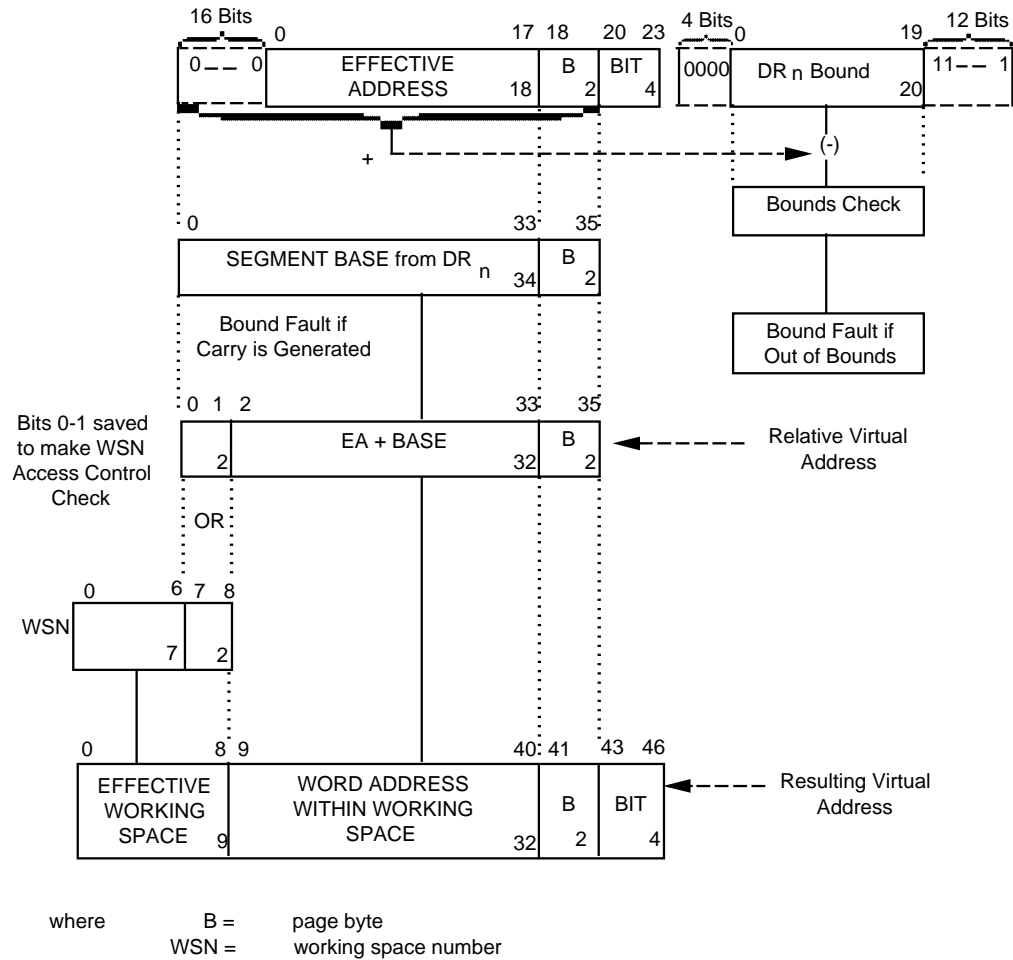


Figure 5-8. Virtual Address Generation Using Extended Segment Descriptor (ES Mode)

5.4.3.3 Virtual Address Generation in EI Mode

The method of forming an operand virtual address with a standard descriptor in EI mode is shown in Figure 5-9. The bounds check is the same as ES mode.

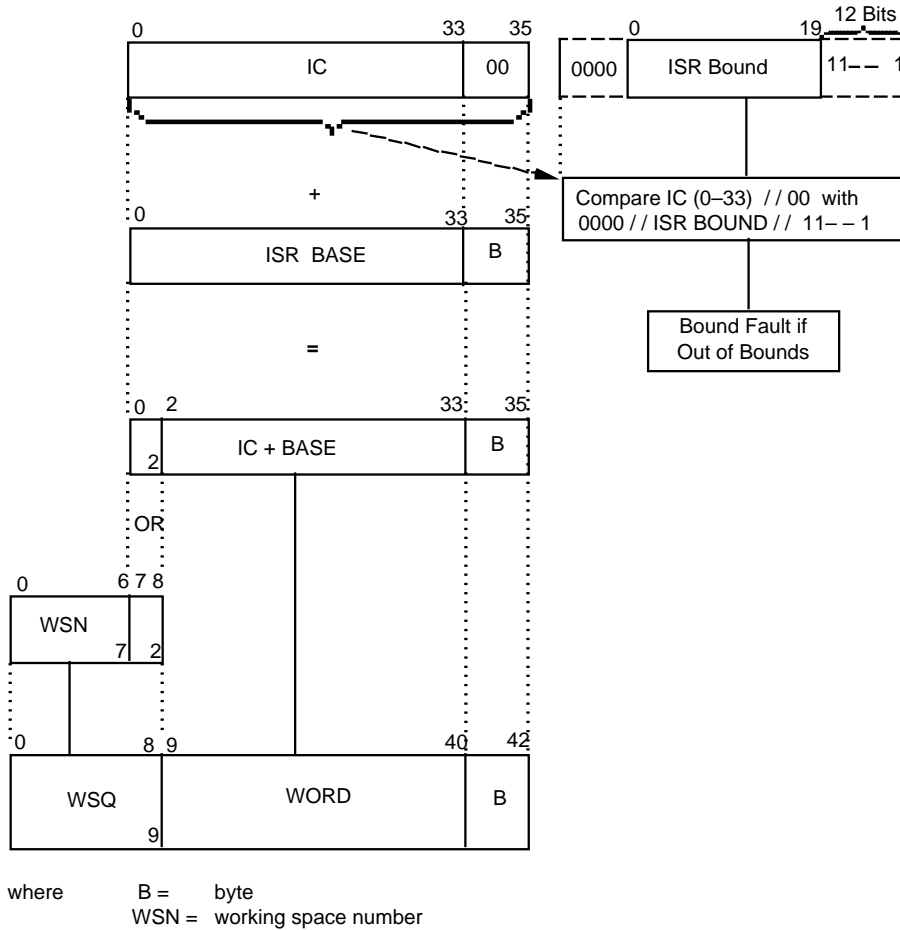


Figure 5-9. Virtual Address Generation Using Standard Descriptor (EI Mode)

5.4.4 Working Space Zero

The CPU supports paging in working space zero in NS, ES, and EI modes. Virtual addresses are generated when the CPU is in the working space zero. However, virtual addresses are a maximum of 2^{28} bytes representing real addresses as seen from the operating system. When the CPU references working space zero for the operating system, the hardware is actually paging, using page tables of working space zero that were prepared by the Service Processor (SP) when the system was initialized.

When the content of a WSR or a particular segment's WSN field is zero, the CPU references working space zero. If WSR1 contained a zero and was referenced by the ISR descriptor, the instruction would be fetched from working space zero. On the other hand, if the instruction specified descriptor register modification, and its associated working space register contained a 1, then the virtual address would be developed in working space one.

To reference working space zero, the CPU must be in Privileged Master Mode with the privileged bit of the Instruction Segment Register (ISR) ON. If these conditions are not satisfied, a Command fault occurs when an attempt is made to reference working space zero.

After the resulting virtual address has been generated and bound checks have been made, the processor performs the checks indicated in Figure 5-10.

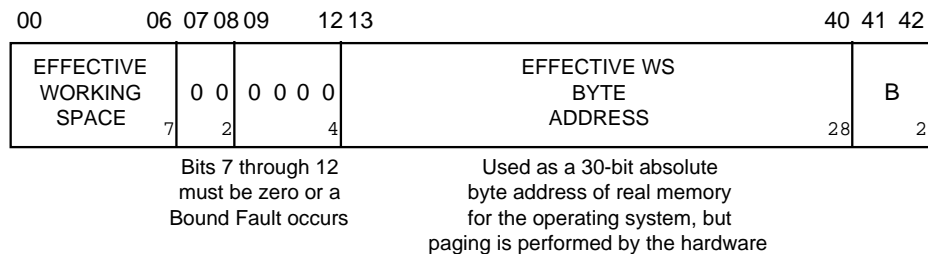


Figure 5-10. Virtual Address Check

5.5 Paging

After generation of a virtual address, an address translation process for mapping a virtual memory address to a real memory address is performed by paging in order to create a real memory address for accessing the real memory.

Paging does not depend upon whether the CPU is in the NS, ES, or EI mode. Each of the 512 working spaces is supported by one page table or one section table. The location of the page table or section table supporting a given WS is indicated by a 9-bit WSN. This WSN indexes the working space page table directory (WSPTD) which is a 512-word table that contains the real memory address of a page table or section table. The section table consists of up to 4K words and includes the real memory address of the page table. The individual words of the section table are called page table base words (PBW). When paging is performed using section tables, PBWs cause the page table to be divided into 1K blocks and distributed throughout memory.

5.5.1 Page Table Directory Word Format – SV Mode

The format of the page table directory word in SV mode is given in Figure 5-11.

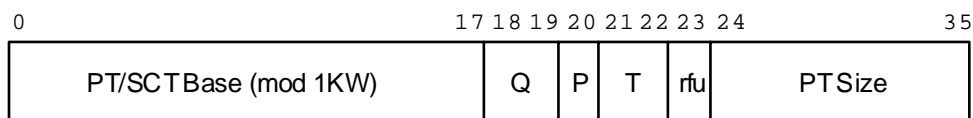


Figure 5-11. Page Table Directory Word (PTDW) Format in SV mode

Bits	Description
0-17	the modulo 1024 base address (real memory address) of a page table (PT) or a section table (SCT)
18,19 (Q)	These bits provide a hardware method to force the isolation of the WS. When one or more WS is allocated to a process, software will record in these bit positions of the associated PTDW the relative WSN within the set of up to four possible numbers. These bits are used to check the WSN at translation from a virtual memory address to a real memory address. A fault occurs if an illegality is detected.
20 (P)	0 The PT/SCT is not present. (A missing working space fault occurs.) 1 The PT/SCT is present.
21,22 (T)	00 The PT/SCT indicated by this word is a dense page table. 10 The PT/SCT indicated by this word is a fragmented page table (not used in GCOS) 01 The PT/SCT indicated by this word is an SCT. 11 A missing working space fault occurs
23	Reserved for future use
24-35	These bits indicates the size of the PT/SCT. <ul style="list-style-type: none"> – For a dense page table, bits 24 to 35 indicate the modulo 64 size of the PT. – For a section table, bits 30 to 35 indicate the modulo 64 size of the SCT. – If bits 30 to 35 are zero, a size of 64 words is assumed. In this case, bits 24 to 29 are ignored.

5.5.2 Page Table Directory Word Format - SVMX Mode

The format of the page table directory word in SVMX mode is given in Figure 5-12.

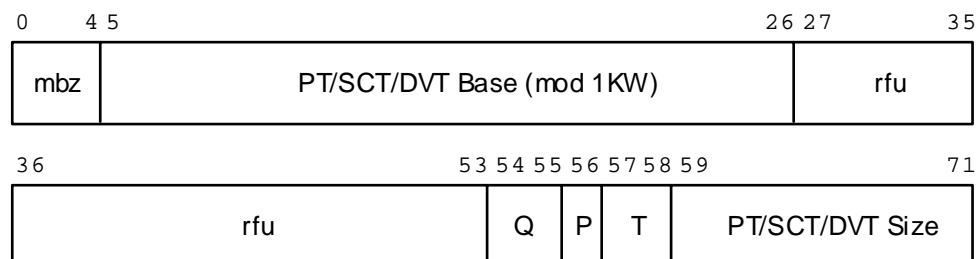


Figure 5-12. Page Table Directory Word (PTDW) Format in SVMX mode

Bits	Description
0-4	Must Be Zero
5-26	The modulo 1024 base address (real memory address) of a PT, SCT, or DVT.
27-35	Reserved for future use
36-53	Reserved for future use
54-55 (Q)	These bits provide a hardware method to force the isolation of the WS. When one or more WS is allocated to a process, software will record in these bit positions of the associated PTDW the relative WSN within the set of up to four possible numbers. These bits are used to check the WSN at translation from a virtual memory address to a real memory address. A fault occurs if an illegality is detected.
56 (P)	0 The PT/SCT/DVT is not present. (A missing working space fault occurs.) 1 The PT/SCT/DVT is present.
57-58 (T)	00 The PT/SCT/DVT indicated by this word is a dense page table (not used by GCOS in SVMX mode). 10 The PT/SCT/DVT indicated by this word is a fragmented page table (not used in GCOS) 01 The PT/SCT/DVT indicated by this word is an SCT. 11 a missing working space fault occurs
59-71	PT/SCT/DVT Size (modulo 64)

5.5.3 Page Table Base Word Format

The format of the page table base word is given in Figure 5-13.

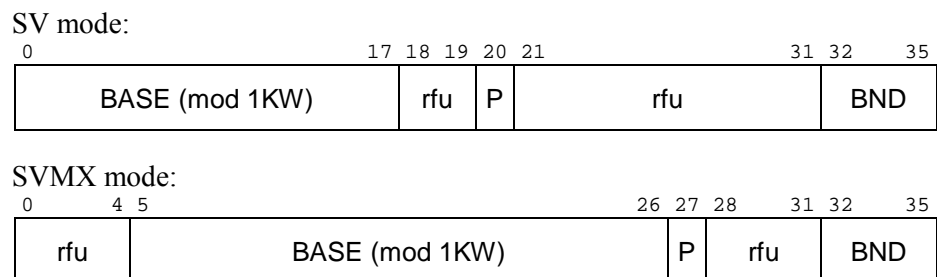


Figure 5-13. Page Table Base Word (PBW) Format

SV mode bit definitions:

Bits	Description
0-17	<p>BASE</p> <p>These bits indicate the modulo 1024 base address (real memory address) of a dense page table. The BASE field with 10 zeroes appended on the right form a 28-bit real word address.</p>
18-19	reserved for future use
20	<p>P</p> <p>0 this bit indicates that the PT is not present. (A missing working space fault occurs.)</p> <p>1 indicates that the PT is present</p>
21-31	reserved for future use
32-35	<p>BND</p> <p>These bits indicate the modulo 64 size of a dense page table. If 0, the size of 64 words is assumed.</p>

SVMX mode bit definitions:

Bits	Description
0-4	reserved for future use
5-26	BASE These bits indicate the modulo 1024 base address (real memory address) of a dense page table.
27	P 0 this bit indicates that the PT is not present. (A missing working space fault occurs.) 1 indicates that the PT is present
28-31	reserved for future use
32-35	BND These bits indicate the modulo 64 size of a dense page table. If 0, the size of 64 words is assumed.

5.5.4 Page Table Word Format

The format of the page table word is given in Figure 5-14.

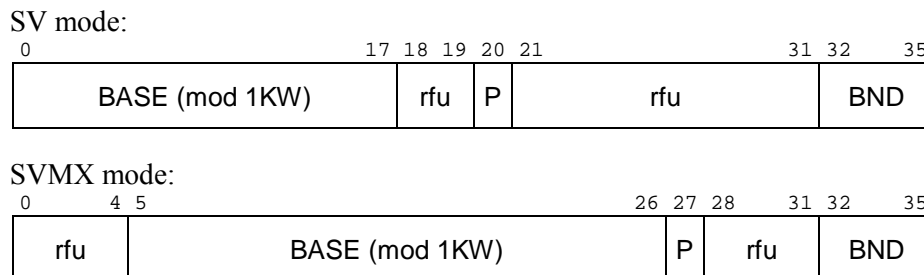


Figure 5-14. Main Memory Page Table Word (PTW) Format

SV Mode bit definitions:

Bits	Description
0-17	<p>MM Page Address This field, appended with 10 zeros on the right, specifies a 28-bit real word address in main memory.</p>
18-28	<p>RFU Software can store information related to the page specified by this PTW. Hardware ignores this field.</p>
29	<p>E = 0 The E bit is used to indicate if the memory page is in real memory or in the EMU. The E bit is set when the page is for an EMU page.</p> <p>NOTE: DPS 9000G2 Systems do not support the EMU.</p>
30-35	<p>Control Field</p> <ul style="list-style-type: none"> 30 MM page present 31 Write permitted 32 Housekeeping bit 33 IO page present 34 Page modified 35 Page accessed

SVMX Mode bit definitions:

Bits	Description
0-4	RFU
5-26	MM Page Address This field, appended with 10 zeroes on the right, specifies a 32-bit real word address in main memory.
27-28	RFU Reserved for future extension.
29	E = 0 The E bit is used to indicate if the memory page is in real memory or in the EMU. The E bit is set when the page is for an EMU page.
	NOTE: DPS 9000G2 Systems do not support the EMU.
30-35	Control Field 30 MM page present 31 Write permitted 32 Housekeeping bit 33 IO page present 34 Page modified 35 Page accessed

Mapping a virtual address to a real address in the VS/XA mode is the same as in the VS/Basic mode in that if a prior memory reference to the same page has already mapped that page to real memory, and if that mapping is still present in the page table associative memory of the processor, the mapping is accomplished by concatenating the word field of the virtual address to the modulo 1024 real address of the page. This process produces the real address for the memory reference. Otherwise, the mapping proceeds by locating and obtaining the PTDW.

The mapping procedure uses the page tables previously described. Bits 0 to 8 of the virtual address are used to access the WSPTD in order to read the PTDW. Bits 21 and 22 that contain the type (T) of this PTDW are then checked, and address translation is executed in accordance with the content of these two bits.

If PTDW.T=00, then the page table is a dense page table.

If PTDW.T=10, then the page table is a fragmented page table (not used in GCOS).

5.5.5 Dense Page Table

When a dense page table is used, the CPU interprets the virtual address as shown in Figure 5-15.

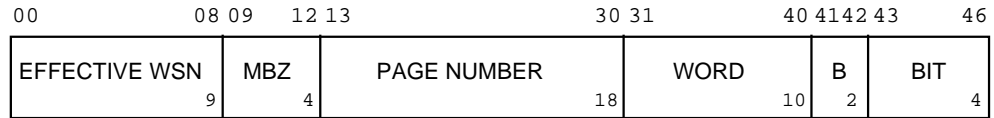


Figure 5-15. Virtual Address

Figures 5-16 5-17 5-18 and 5-19 illustrate virtual to real mapping using a dense page table.

1. The dense page table base address is modulo 1024 words.
2. PTW bits 0 to 17 are the page start address.

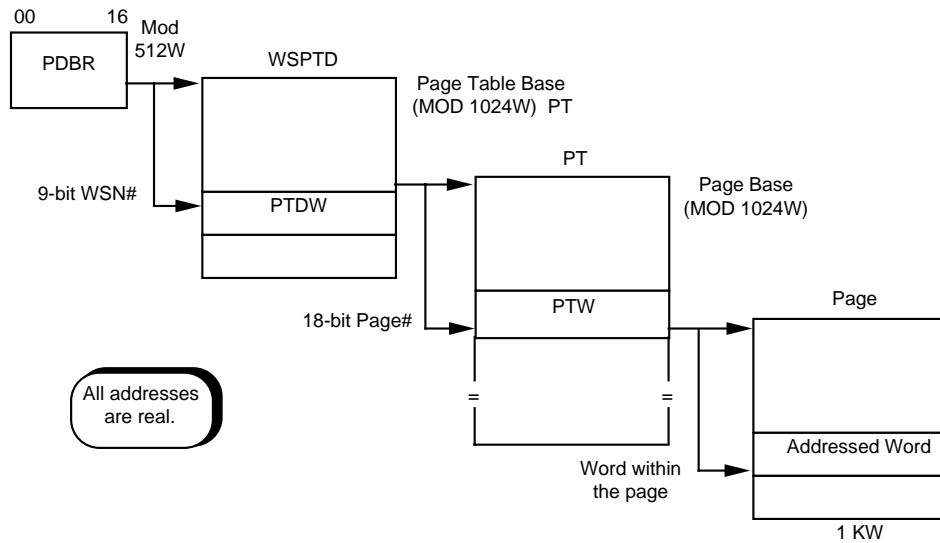


Figure 5-16. Address Mapping Using a Dense Page Table

Address Modification and Development

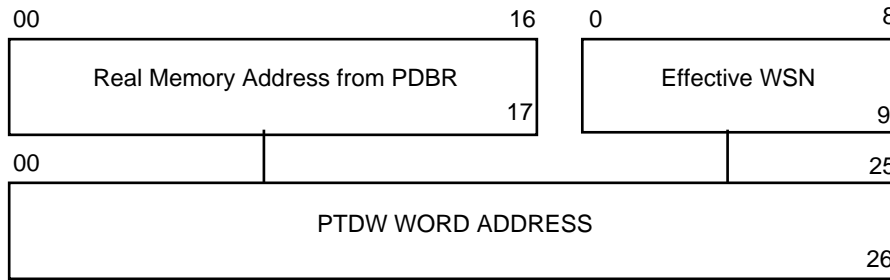


Figure 5-17. PTDW Address

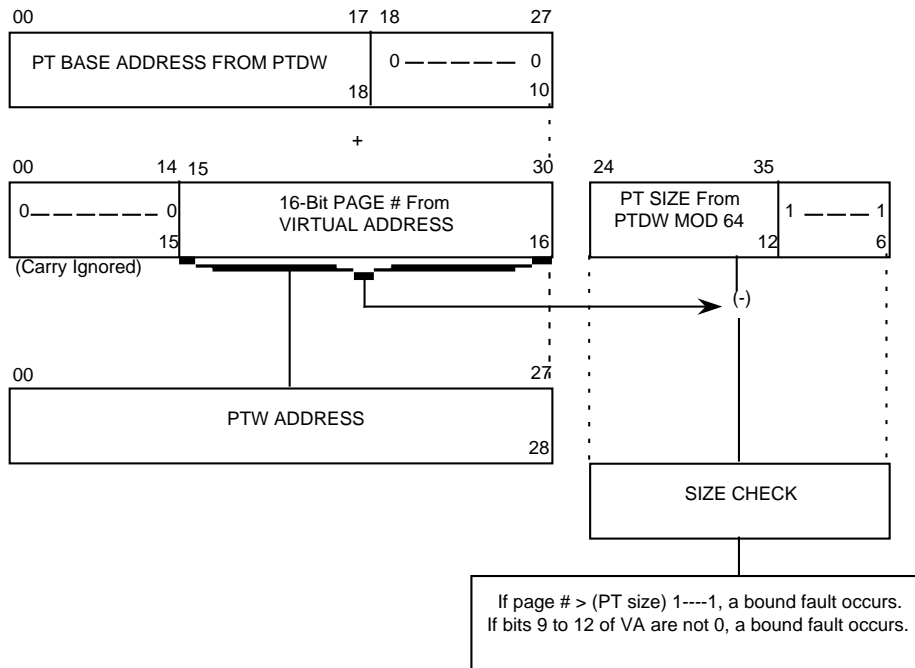


Figure 5-18. PTW Address

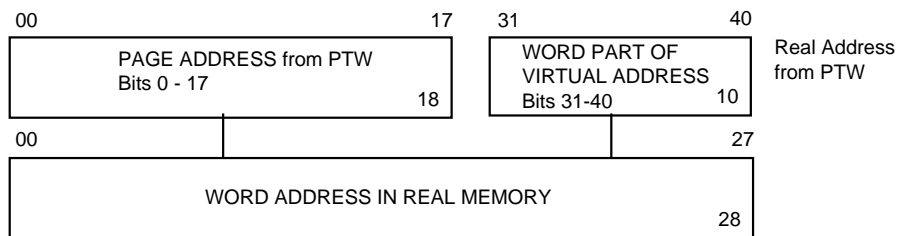


Figure 5-19. Word Address

5.5.6 Section Table

The section table allows the page table for a working space to be fragmented into sections. The PTDW specifies the base of the section table, which contains up to 4K of page table base words (PBW), each of which defines a page table for a section. When a section table (SCT) is specified by the PTDW, the virtual address is interpreted as shown in Figure 5-20

00	08 09	20 21	30 31	40 41	43	46
EFFECTIVE WSN	SECTION NUMBER	PAGE NUMBER	WORD	B	BIT	
9	12	10	10	2	4	

Figure 5-20. Virtual Address

Bits	Description
0-8	working space to be accessed
9-20	section number; an offset of the SCT base for accessing the PBW in the SCT The SC number is a value relative to the SCT base indicated by the PTDW.
21-30	Page number is used as an offset or index into the PT for this WSN, for locating the PTW. The page number is relative to the PT base address (real memory address) indicated by the PBW.
31-40	These bits determine which word within the 1024-word page is being addressed
41-46	byte and bit positions within the word, if applicable

Figure 5-21 illustrates virtual to real mapping when using a section table.

Address Modification and Development

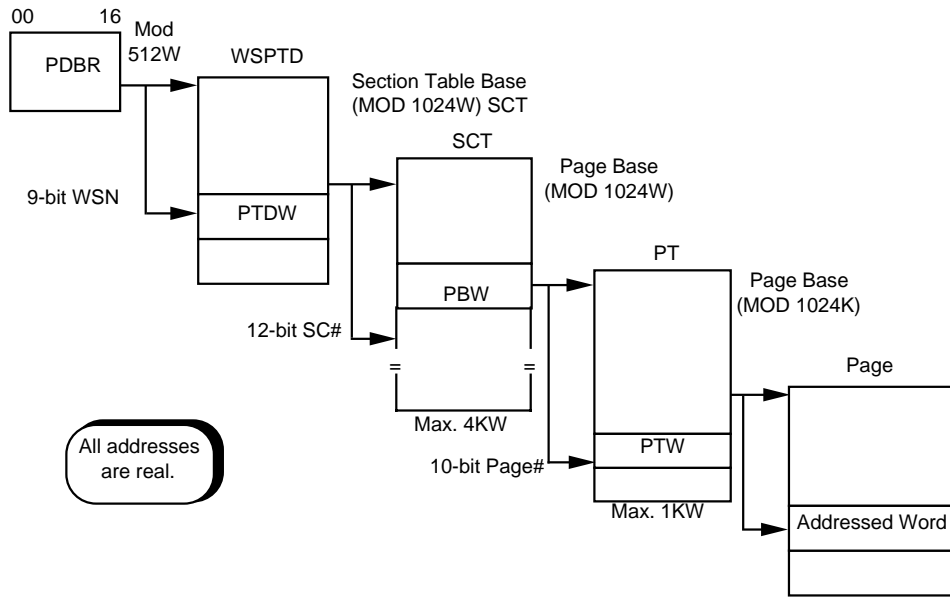


Figure 5-21. Address Mapping Using a Section Table

Figures 5-22 5-23 and 5-24 illustrate the development of a word address from a section table.

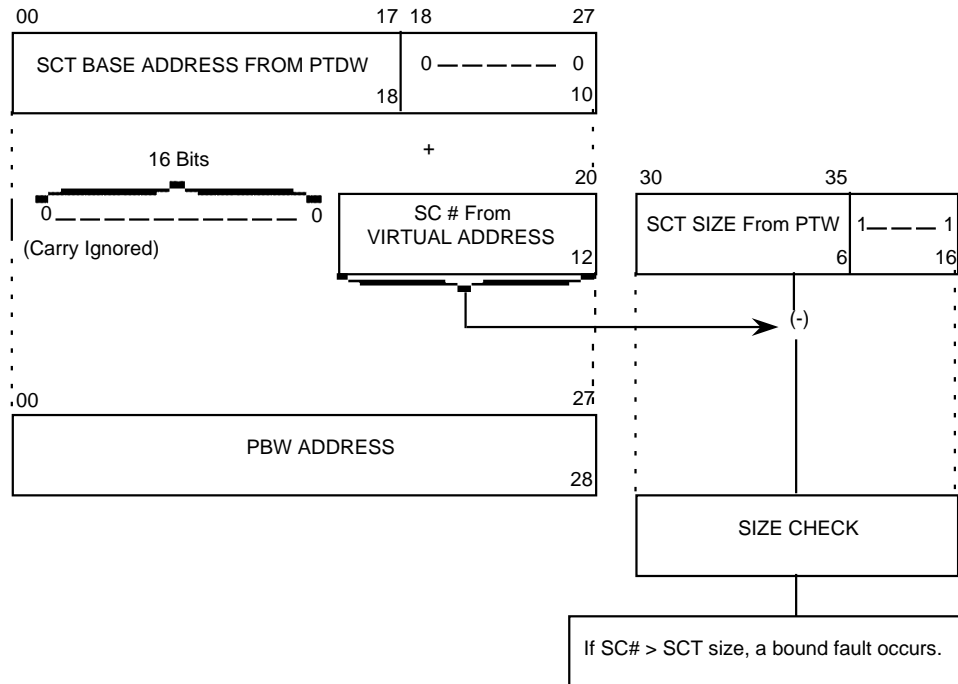


Figure 5-22. PBW Address

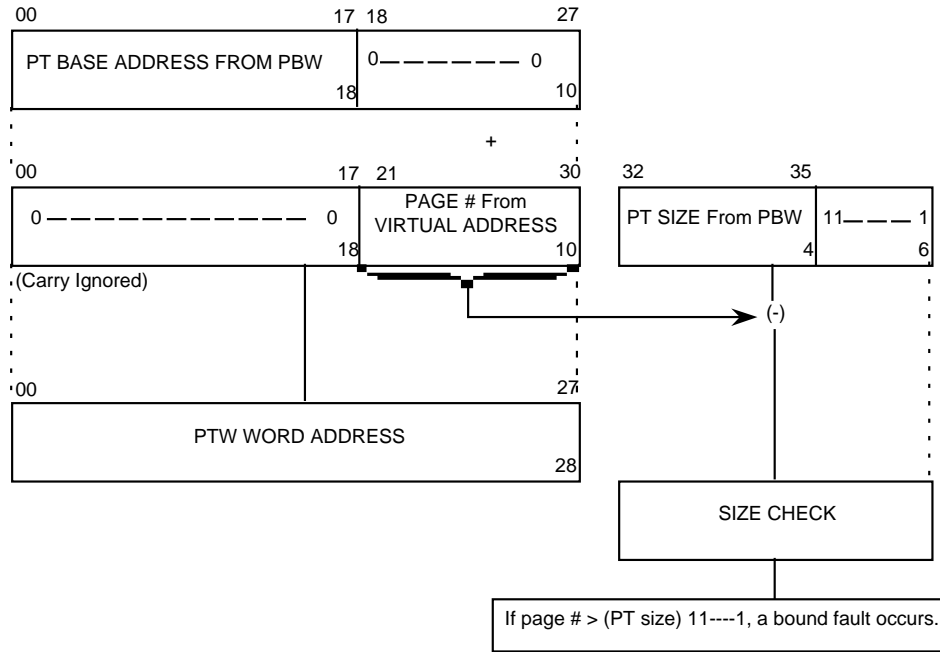


Figure 5-23. PTW Address

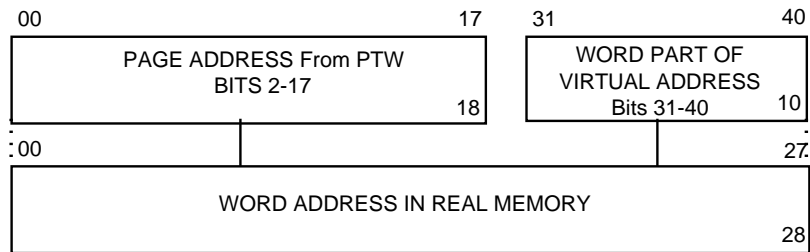


Figure 5-24. Word Address

5.5.7 Address Truncation

The instruction set contains instructions that operate on words, double words, 9-bit bytes, 6-bit characters, 4-bit characters, and bits. Instructions and indirect and tally words that specify 6- or 9-bit characters are considered word instructions. In accessing the operand, the full byte level virtual address is determined. The address is then truncated in accordance with the address type of the instruction, and the access is also in accordance with the type of instruction.

An exception to this procedure applies to the 8-word instructions, such as LREG and SREG. The effective address is truncated to a modulo 8 word address before adding the base. Following the addition of the base, the virtual address is then truncated to a double-word address.

The user is responsible for ascertaining correctness of operation of an instruction as influenced by such address truncation.

5.5.8 Bounds Checking

One of the capabilities provided by virtual memory is that of specifying the base and bound of a segment to the 9-bit byte level, enabling a higher level of security control and more efficient use of main memory. Since the processor interfaces with word-oriented main memories, certain restrictions are also imposed to minimize the impact on performance and hardware complexity. The size of a segment described by a super descriptor is modulo 2^{26} bytes. Therefore, the bounds checking is always the same: BOUND (lower extended with 26 one bits) greater than or equal to LOCATION + EFFECTIVE ADDRESS. The following information applies only to standard descriptors.

5.5.8.1 Word and Double-Word Operations

Word, double word, or a succession of word accesses as in the LREG and SREG instructions are made to real memory word or double word boundaries. Segments that begin or end on byte or word positions and that do not correspond to word or double-word boundaries may be accessed by word or double word instructions. The processor adds the 2-bit byte position held in an address register (if selected) to the byte position of the base before truncating the final virtual address to point to a word or double word. If this truncation results in the virtual address dropping below the base value, a lower bound check will declare an out-of-bounds condition in this case and a Bound fault occurs. Thus, the first word or double word of a segment may be accessed with word-oriented instructions only when the word or double word is entirely within the segment.

Half-word accesses such as the `LXLn` instruction are treated as word accesses in both the lower-and upper-bounds check. If a segment begins in the middle of a word, the `LXLn` and `SXLn` instructions cannot be used to access the lower half-word. If the segment ends in the middle of a word, the `LDXn`, `STXn`, `LXLn`, `ADXn`, etc., instructions cannot be used to access the upper half-word.

The `STCA`, `STCQ`, `STBA`, and `STBQ` instructions store 6-bit or 9-bit characters into character/byte locations within a word. These instructions are considered as word accesses and require the entire word to be within the segment.

Indirect and tally words that specify character/byte locations are considered as addressing words that must be fully contained in the segment. The virtual address is truncated to the next lowest word boundary (i.e., the character position in the base is not added to the character position held in the indirect and tally word).

NOTE: The preceding information is included to provide a warning for operating system and user software. If segments are "shrunk" (see the `LDDn` and `CLIMB` instructions), and the byte portion of the virtual base is changed, a word or double-word access to the new segment may be truncated to a different location within the segment.

All instruction segments must begin at a 0 modulo 8 location and end at a 7 modulo 8 location. Any transfer or `CLIMB` instruction that attempts to load the instruction segment register must specify a segment base whose 5 least-significant bits are 0s, and a segment bound whose five least-significant bits are 1s. This condition allows the processor to access blocks of eight words for `LPL`, `SPL`, `LREG`, `SREG`, `LAREG`, and `SAREG` instructions with the assurance that if the first word is on an assigned page and is within the segment boundary, the other words will also be so located.

All descriptors loaded into the `SSR`, `PSR`, `LSR`, `ASR`, or `DSDR` registers must begin and end on double-word boundaries (the three least-significant bits of the base are 0s and the three least-significant bits of the bound are 1s).

5.5.8.2 Byte Operations

For all 9-bit and 4-bit character operations using multiword instructions, the upper-bound check is made at the 9-bit byte level. A lower-bound check is not required since the effective address is always greater than or equal to zero.

For all 6-bit character operations using multiword instructions, the boundary checking is on a double-word basis, meaning that a double word containing any 6-bit character of the operand must be fully in bounds. If access is attempted to a segment with a base or bound not on a double-word boundary, a Bound fault is generated.

5.5.8.3 Bit Strings and Table of Translate Instruction

Multiword bit string instructions and the index table of the translate instructions (MVT, TCT, and TCTR) have double word bound checking applied. Thus, a double word that includes any part of these operands must be fully in bounds. If access is attempted to a segment that has a base or bound not on a double-word boundary, a Bound fault is generated.

5.5.8.4 Bound Check Equations

The address truncation procedure forces bounds checking to vary, depending upon the type of instruction specified. The resulting three upper bound and lower-bound checks are listed in Table 5-3. A Bound fault is generated if the bound checks are violated.

Table 5-3. Bound Check Equations

Instruction	Bound Check
Double-Word (includes bit string and 6-bit character instructions)	Upper $(\text{BASE}+\text{EA})_{0-32} \parallel 111 \leq \text{BASE}+\text{BOUND}$ Lower $(\text{BASE}+\text{EA})_{0-32} \parallel 000 \geq \text{BASE}$
Single-Word	Upper $(\text{BASE}+\text{EA})_{0-33} \parallel 11 \leq \text{BASE}+\text{BOUND}$ Lower $(\text{BASE}+\text{EA})_{0-33} \parallel 00 \geq \text{BASE}$
Byte (includes 9-bit byte, 4-bit byte)	Upper $(\text{EA})_{0-19} \leq \text{BOUND}$ Lower always satisfied

The base, bound, and effective address (EA) addresses represented in the bound check equations are for 9-bit bytes. For 4-bit byte and bit instructions, the effective address represents the 9-bit byte in which these small quantities are contained. The single- and double word bound check equations include the effect of address truncation. The truncated address is then extended to the largest byte contained therein for the upper-bound check and to the lowest byte for the lower-bound check. The byte checks refer to the byte accessed. In multibyte instructions such as MLR, the access checks are applied to each byte.

Physical accesses, which may be larger than those corresponding to a given instruction (and which therefore may include bytes not contained in the segment) are not bound checked beyond the byte range corresponding to the instruction.

Bound checking is also performed on page table sizes for dense page tables. The page number from the virtual address is bounded by

page number 15-30 ← WSPTD PT BOUND 26-35 ||111111
and page number 9-14 must be zero.

When in working space 0, the virtual address is checked for the 26-bit range of byte address.

Virtual address 9-14 must be zero.

5.5.9 Multiprocessor Memory Management

Virtual memory management permits the base and bound of segments to be located on a byte boundary, both as a virtual address and a real address. Normal software multiprocessor protection does not exist across a segment boundary. Therefore, data may be lost when:

1. two processors simultaneously refer to and change the same double word in memory;
2. the double word contains a segment boundary;
3. one or both processors are executing a multiword instruction, unless the segment boundary is modulo two words.

This condition may occur since the processor always reads a double word from memory, changes the character(s) involved in the operation, and writes the double word back to memory. Thus, between the reading of the double-word for a multiword instruction on one processor and the subsequent double-word store, a second processor could change that part of the double-word not affected by the multiword instruction, and the changed data would be destroyed when the double-word is stored.

6. Faults and Interrupts

This section of the Programmer's Guide describes faults and interrupts, organized as follows:

- Section 6.1, Description of Faults and Interrupts
- Section 6.2, Fault Procedure
- Section 6.3, Faults and Interrupts
- Section 6.4, CPU Cache
- Section 6.5, Interrupt Procedure

Faults and interrupts both result in an interruption of normal sequential processing, but they originate differently. Generally, faults are caused by events or conditions inside the processor. Events or conditions outside the processor cause interrupts. Faults and interrupts enable the processor to respond promptly when conditions occur that require attention to the system.

6.1 Description of Faults and Interrupts

When the processor responds to a fault, interrupt, or special systems entry (PMME), the interdomain ICLIMB version of the CLIMB instruction is executed through an entry descriptor which is obtained from a fixed memory location. The locations in real memory containing the entry descriptors for interrupt, fault, and system entry (PMME) are listed below.

<u>Location (octal)</u>	<u>Type</u>
30-31	Interrupt
32-33	Fault
34-35	Special systems entry

6.2 Fault Procedure

When a fault occurs, the processor generates the appropriate five-bit fault code and executes the ICLIMB version of the CLIMB instruction. During the safe store part of the ICLIMB, the five-bit fault code that has been generated (see Table 6-1) is stored with a flag to indicate that the safe store frame is the result of a fault (bit 11 of word 5 is set to 0).

If the fault occurs during a multiword instruction, the pointer and length storage area is saved in the safe store frame, provided the Stack Control Register (SCR) defines the frame size as 64 or 80 words.

The second word of the "wired-in" ICLIMB instruction is for interrupts and is described later in this section (refer to "Interrupt Procedure").

Three groups of faults are indicated in Table 6-1: ETF, PAF, and RIF, where:

- ETF are Externally activated, Timer-activated Faults;
- PAF are Program-Activated Faults; and
- RIF are Recoverable, Interrupt-like Faults.

These faults are recognized at the same time that an XIP would be recognized, but they have a higher priority than XIPs.

Faults and Interrupts

Table 6-1. Processor Faults By Priority

Fault Code	Fault Name	Priority	Group	Remarks
xxxxxx	Backup fault (BKUP)	1	PAF	Note 1
001100	Startup (STUP)	2	ETF	
001111	Execute (EXF)	2	ETF	
001011	Operation Not Completed (ONC)	3	ETF	
000111	Lockup (LUF)	4	ETF	
001001	Memory System (MEMSYS)	6	PAF	
001110	Divide Check (DVCF)	7	PAF	
001101	Overflow (OFL)	8	PAF	
000101	Command (CMD)	9	PAF	
010010	Missing Segment (MSG)	10	PAF	
000001	Bound (BNDF)	11	PAF	
000010	Master Mode Entry (MME)	12	PAF	
000110	Derail (DRL)	13	PAF	
001010	Illegal Procedure (IPR)	14	PAF	
000011	Fault Tag (FTAG)	15	PAF	
010000	Security fault, Class 1 (SCL1)	16	PAF	
010001	Dynamic Linking (DYN)	17	PAF	
010011	Missing Working Space (MWS)	18	PAF	
010111	Missing Section (MSCT)	20	PAF	
010100	Missing Page (MPF)	21	PAF	
010101	Security fault, Class 2 (SCL2)	22	PAF	
separate	Safe Store Stack fault (SSSF)	27	PAF	Note 2
001000	Connect (CON)	29	RIF	
000100	Timer Runout (TRO)	30	RIF	
010110	Shutdown (SDF)	31	RIF	
011001	MEEP	31	RIF	Note 3
100000	Address Trap (ADT)	34	RIF	

Notes for Table 6-1

1. A Backup fault occurs if a fault or interrupt occurs during the initiation of an ICLIMB instruction or if any fault occurs during the execution of an ICLIMB. On the DPS 9000TA platform, the firmware fault table has been expanded to include a vector pointer to the backup fault vector, which will be initialized by the Service Processor with the offset of the current backup fault vector. A copy of all fault vectors is maintained in Reserved Memory Storage (RMS).

2. The Safe Store Stack Overflow Fault has no fault code. A separate flag is set in a unique 5-bit code that appears in bits 12-16 of word 5 in the safe store stack frame. (See the discussion of Safe Store Stack Fault (SSSF). The Safe Store Stack fault occurs in conjunction with a programmed CLIMB instruction, or in conjunction with the wired-in CLIMB instruction that is the result of a fault or interrupt. The Safestore Stack fault indicates to the operating system that the Safe Store Stack has < 159 words + 3 bytes remaining. (See Safe Store Stack under CLIMB in Section 9 for additional information.)
3. The Service Processor, through the Service Scan Path interface via a Maintenance Processor Service Command, causes MEEP faults. Used primarily for communications between the Service Processor and GCOS 8.

6.3 Fault Processing

When a fault occurs, the processor performs the following steps.

1. The fault entry operation is started when a fault condition is detected with a fetch or execution of the current instruction I(k).
2. The program is stopped. After the fault entry start, these steps are performed.
 - a) Further operation related to I(k) is disabled.
 - b) Further operation of instructions that follow I(k) in the program is disabled.
 - c) The processor completes the preceding I(j), ..., I(k-1) instructions before I(k).
3. When the program has been stopped, fault selection is performed.
 - a) Fault conditions detected simultaneously with the instructions following I(k) are ignored, and detection of other fault conditions with I(k) is disabled.
 - b) While the processor is waiting for completion of all previous instructions, it accumulates fault conditions for those instructions and selects faults in the following way:

Any fault condition detected with a previous instruction is selected, not the fault condition detected with the current instruction. For example, the fault condition detected with I(k-1) or I(k-2), would be selected, not that fault condition detected with the instruction I(k).

When all previous instructions are completed, the processor selects the earliest instruction with a fault, and within that instruction, selects the fault with the highest priority from the priority list in Table 6-1 and generates the fault code.

Selection of Program-Activated Faults (PAF)

In some cases, two or more instructions may be executed simultaneously with processor pipeline processing. Therefore, one or more faults can be detected simultaneously. When multiple faults occur in one or more instructions of the program sequence, the processor selects one fault and notifies the operating system with a fault code.

When PAF is selected, three types of priority are in effect:

a. Program sequence priority

When PAF is detected with instruction I(k), the processor waits until all prior instructions have completed or are stopped because of a fault. A PAF detected with one of these instructions, rather than the I(k) PAF, is reported.

b. Instruction execution sequence priority

As a result of a fault detected with the execution of instruction I(k), further operation in relation to I(k) is stopped, regardless of the number of faults detected simultaneously or the type of detected fault.

If an illegal instruction code, an illegal tag, or a Command fault is detected with instruction I(k), the following operation relating to I(k) is prevented because the following operation is not defined and other faults are meaningless.

When a Bound or Security Class 1 or 2 fault is detected, further memory access is prevented in order to protect memory.

c. Final priority

When a fault is detected with I(k), all prior instructions are completed or their operation is stopped from detection of a fault, and execution of any instructions that follow is stopped. The processor selects one fault with final priority (see Table 6-1.).

Selection of Recoverable, Interrupt-Like Faults (RIF)

As with external activity or timer activity faults (ETF), the signals causing RIF faults may occur at any point in the program sequence. However, they differ from the ETF in that they are processed as an interrupt. After these signals are detected, the processor waits until complete recovery is ensured. The timing with which the processor begins fault processing is the same as that for reception of interrupt.

Faults other than RIF have a higher priority to ensure that when the processor stops instruction I(k) in response to a fault, all previous instructions are completed without encountering a fault.

When the RIF is detected after a part of instruction I(k) is executed, priority is used to ensure that all faults detected before the detection of the RIF are selected in place of the RIF. RIF is saved in the processor internal register until the response for the fault is performed.

Selection of Externally Activated, Timer Activated Faults (ETF)

A signal causing an ETF may occur at any point in the program sequence and will immediately stop the instruction being executed, regardless of the state of the program or occurrence of other faults. ETFs have higher priority than other faults (see Table 6-1) to ensure that they are selected over other faults that occur simultaneously.

4. Notification of Selected Faults

The following operation is performed after selection of the fault with the highest priority.

- a) The 5-bit fault code is generated.
- b) Information is set in the Fault register with Illegal Procedure faults.

5. Instruction Counter Adjustment

The processor adjusts the value in the IC after the program is stopped and a fault is selected.

a) Program-activated faults (PAF)

Except for a Memory System fault caused by a write request, the IC adjustment with fault entry indicates the instruction responsible for the fault condition or the instruction subject to that condition instead of the instruction being executed when the fault condition is detected.

With program stop and fault selection, the first fault caused by the program is selected. IC adjustment ensures that the IC reflects the I(s) location

[I(s) indicates the instruction that caused the finally selected fault during execution or the fetch operation].

When I(s) is reported because of a fault, the IC adjustment ensures that I(s-1) is the last instruction to be completed normally in the program sequence.

b) Recoverable, Interrupt-like faults (RIF)

When a Connect, Timer Runout, or Shutdown fault is selected, the adjusted IC indicates the address of the instruction to which processing is to be returned in the interrupted program. When I(s) is reported because of a fault, the last instruction to be completed normally is I(s-1). This instruction is located immediately before the instruction to which processing is to be returned.

When a RIF occurs with the DIS instruction, the program must be continued from the instruction following the DIS instruction. The DIS instruction address+1 is thus loaded into the IC.

Faults and Interrupts

c) Externally Activated and Timer-Activated Faults (ETF)

These fault conditions are caused by IOP operations, the ONC (Operation Not Complete) timer, or the lock-up timer. The adjusted IC contains the value of the IC at the approximate time when the fault entry was initiated. The reported I(s) does not always indicate that I(s-1) is the last instruction to be completed normally.

6. Loading the Fault Register

With an Illegal Procedure fault, the fault information is loaded into the Fault register after the fault is selected. With other faults, the content of the fault register may be altered, and consequently undefined and not interpreted.

7. Altering Bit 30 of the Indicator Register

The multiword instruction interrupt indicator (IR bit 30) is altered before safe store operation with the wired-in CLIMB in a number of ways.

- a) If the selected fault is PAF, RIF, or ETF occurring with a single-word instruction, IR(30) is set to OFF (0).
- b) If the selected fault is PAF (excluding Missing Page fault) or an ETF occurring with a multiword instruction (excluding CLIMB instructions), IR(30) is set to ON (1).
- c) If the selected fault is RIF or a Missing Page fault occurring during the execution of a multiword instruction (excluding CLIMB instructions) (i.e., when pointers and lengths data is required for resuming the instruction), IR(30) is set to ON (1).
- d) If the selected fault is RIF or a Missing page fault occurring at a point other than during execution of a multiword instruction (excluding CLIMB instructions), IR(30) is reset to OFF (0).
- e) If the selected fault is a PAF, RIF, or ETF occurring with a CLIMB instruction, IR(30) is reset to OFF (0).

8. Termination of Fault Entry Operation

The processor terminates the fault entry operation by executing an inter-domain call with a wired-in CLIMB. The Master Mode indicator (and others) is altered during the wired-in CLIMB processing. (Refer to the discussion of the CLIMB instruction in Section 9 for details.)

6.3.1 Processor Halt Conditions

Under the conditions defined below, the detection of fault conditions does not cause operation of a fault entry but, instead, causes the processor to send an interrupt to the SCU and halts. At this point, the detection of the fault initiates the SCU firmware.

1. Fault on Fault

Any fault condition detected during the fault entry action causes the processor to send an interrupt to the SCU and to halt. The Service Processor is responsible for accessing the situation and proceeding with the appropriate action.

2. Hardware Failure Detection

All hardware failure detection, including Operation not Completed and Memory System fault conditions, cause the processor to send an interrupt to the SCU and to halt. The SCU then initiates the Service Processor which proceeds with test, diagnosis, reconfiguration/identification of repair action, and further dialog with the operating system.

6.3.2 Fault Descriptions

The name of each fault, the fault code, the content of the IC and Fault register and a description of each fault follows.

6.3.2.1 STUP Fault

Mnemonic:	STUP
Name:	Startup
Fault Code:	001100
Description:	This fault is caused by the Service Processor using a high priority form of the MEEP fault mechanism. The fault is used to make an operating system entry after restarting a CPU, which has been stopped for diagnosis, repair, or power off.
IC Value Reported:	Contents of IC or IC+1
Fault Register Content:	Not applicable

6.3.2.2 EXF Fault

Mnemonic:	EXF
Name:	Execute
Fault Code:	001111
Description:	This fault is caused by an external agent.
Fault Register Content:	Not applicable

Faults and Interrupts

6.3.2.3 ONC Fault

Mnemonic: ONC

Name: Operation Not Completed

Fault Code: 001011

Description: This fault is generated if a CPU internal hardware condition has occurred which prohibits communication between the Basic Processing Unit and the Private Cache within the CPU for a period of 4 msec when this interface is not otherwise disabled due to an error alarm or defined halt state. At the time an ONC occurs, the LUF timer is also reset to avoid a lockup fault. This time continues to be reset while the ONC latch is set. Strokes to the ONC timer are also inhibited.

IC Value Reported: The location of the instruction that detected the hardware failure. IC adjustment is normally performed with the fault processing. The last instruction completed successfully is not always the reported value-1.

Fault Register Content: Not applicable

6.3.2.4 LUF Fault

Mnemonic: LUF

Name: Lockup

Fault Code: 000111

Description: This fault occurs when the processor does not respond to an interrupt signal within a specified interval. In the Master and Privileged Master modes, this interval is 32 msec. In the Slave mode, the interval is 16 msec.

For transfer from the Slave to Master or Privileged Master modes, the counter is not reset. However, the interval is extended to 32 ms. In this case, the usable interval in the Master or Privileged Master modes, before the opportunity for interrupt response, is 32 msec minus (the counter value at transfer). The counter is not reset upon transfer from the Master or Privileged mode to the Slave mode, but the interval is reduced from the value at transfer to the value specified with the LUF instruction.

The counter is not reset at the transition of the Slave→Master or Privileged Master→Slave modes. This procedure ensures that when a slave program requests Master or Privileged Master mode service and the processor returns to the Slave mode, a lockup interval is not automatically extended. When processing is transferred to another slave program after transfer from the Slave to Master or Privileged Master modes, the operating system can control the interval by generating an opportunity for response to the interrupt at transfer from the Master or Privileged Master to Slave modes. The 0.5- and 2-ms intervals are used for real-time applications. In such cases, limits are placed on the use of long, multiword instructions, RPT, RPL, and RPD instructions in the slave program.

IC Value Reported: This value is the location of the instruction being executed when a lockup fault was detected. IC adjustment is normally performed with fault processing. The last instruction to be completed successfully is not always the reported value-1.

Fault Register Content: Not applicable

6.3.2.5 MEMSYS Fault

Mnemonic: MEMSYS

Name: Memory System

Fault Code: 001001

Description: The Memory System fault is used to report one of the following types of failures detected by the main memory unit.

- Parity errors in transmission between the SCU and the Main Memory Unit (MMU)
- Uncorrectable data errors in a memory location
- An illegal command received by the MMU
- Non-existent memory

With the exception of non-existent memory, these failures are in hardware. Actual hardware operation is such that the processor sends an interrupt to the SCU and then halts when a MEMSYS fault condition is detected. The SCU then retries execution of the instruction. If the retry is not successful, the SP takes corrective action.

When a fault occurs, bit 8 of the Fault Register is set in the following way:

- 0: Memory system fault is NOT caused by non-existent memory access
- 1: Memory system fault IS caused by non-existent memory access

Fault Register Format

- 0–5 NA
- 6 Illegal Decimal Digit (loaded by IPR)
- 7 NA
- 8 Non-existent Memory (loaded by MEMSYS)
- 9–71 NA

The Fault Register is correctly loaded by the IPR and MEMSYS faults and should only be interpreted after one of these faults. Other faults may alter the contents of the Fault Register to an undefined state.

MEMSYS Fault (cont.)

IC Value Reported: When a MEMSYS fault occurs in response to a read memory request, the IC indicates the instruction that issued the request. These requests occur in the following cases.

- With read requests which fetch the instruction itself
- With explicitly addressed indirect words, pointers, descriptors, and operands
- With implicitly addressed words required to complete execution (e.g., with words in reserved memory space memory space used for execution of the CIOC instruction)

When a MEMSYS fault occurs in response to a write memory request, the IC normally indicates the instruction being executed when the memory fault status is returned instead of the instruction that issued the request.

After the processor issues a write request, the instruction that issued the request is completed and then execution of the program instruction continues. The IC value at the time a fault is reported with a write request may, therefore, indicate the instruction that exists after the program sequence, instead of the instruction that issued the request.

All outstanding memory requests are checked for completion before the segment descriptor registers, IC, and IR are changed, with the execution of an inward or outward CLIMB instruction's interdomain transfer (or in the case of all faults and interrupts, with the execution of a wired-in CLIMB). This check ensures that a MEMSYS fault, which is not reported immediately because of a memory write request, is reported before all Slave, Master, and Privileged Master transitions, and all transitions of programs using inward and outward CLIMB.

Fault Register Content: Not applicable

Faults and Interrupts

6.3.2.6 DVCF Fault

Mnemonic:	DVCF
Name:	Divide Check
Fault Code:	001110
Description:	This fault occurs when division is not completed normally. (Refer to the DIVIDE instruction specifications in Section 9.)
IC Value Reported:	The location of the instruction causing the divide check condition.
Fault Register Content:	Not applicable

6.3.2.7 OFL Fault

Mnemonic: OFL

Name: Overflow

Fault Code: 001101

Description: This fault occurs after completion of an instruction during which the Overflow Mask indicator was OFF and the Overflow or Exponent Overflow indicator was set to ON, or during which the Overflow Mask and Exponent Underflow Mask indicators were both OFF and the Exponent Underflow indicator was set to ON.

An Overflow fault does not occur under any conditions when the Overflow Mask Indicator is ON. If the Exponent Underflow Mask indicator is ON, an Overflow fault does not occur under exponent underflow conditions.

Instructions which alter the Overflow Mask or Exponent Underflow Mask indicators do not cause Overflow faults. An Overflow fault can occur after completion of an instruction in which the Truncation fault enable bit (multiword instruction bit 9) is ON and the Truncation indicator was set to ON.

IC Value Reported: The location of the instruction causing the overflow or truncation condition

Fault Register Content: Not applicable

Faults and Interrupts

6.3.2.8 CMD Fault

Mnemonic:	CMD
Name:	Command
Fault Code:	000101
Description:	This fault is detected and reported when the processor attempts execution of a privileged instruction in a non-privileged mode. Instructions, which may cause Command faults, are noted in the instruction specifications in Sections 8–15.
IC Value Reported:	The location of the instruction causing the command violation.
Fault Register Content:	Not applicable

6.3.2.9 MSG Fault

Mnemonic:	MSG
Name:	Missing Segment
Fault Code:	010010
Description:	This fault occurs when an attempt is made to access memory using a segment descriptor in which bit 28 = 0 (indicating segment not present).
IC Value Reported:	The location of the instruction causing the fault.
Fault Register Content:	Not applicable

Faults and Interrupts

6.3.2.10 BND Fault

Mnemonic: BND

Name: Bound

Fault Code: 000001

Description: This fault is caused by a bound check with a segment descriptor as described in Section 5 under "Virtual Address Generation". It also occurs when the generated virtual address exceeds the PTDW or PBW size, as well as with various checks performed when an instruction is executed. These conditions are included in the individual instruction specifications in Sections 8–15.

IC Value Reported: The location of the instruction on which the Bound fault condition was detected on the fetch or execution of the instruction.

Fault Register Content: Not applicable

6.3.2.11 MME Fault

Mnemonic: MME
Name: Master Mode Entry
Fault Code: 000010

Description: The MME fault is caused by execution of the Master Mode Entry instruction. It is a recoverable trap.

If the safe-store bypass flag in the option register = 1, a safe store frame is generated. The size of this safe store frame is determined by the type of the entry descriptor. The occurrence of the MME fault is indicated in the safe store frame by a code of 00010 in bits 12-16 of word 5.

The wired-in CLIMB instruction functions as though the second word of the CLIMB instruction had the following characteristics:

E	= 0	no parameters
C(18)	= 0	do not load X0
C(19)		no effect, turn Master Mode indicator ON
C(22-23)	= 00	Inward Climb S,D no effect

The entry descriptor specifies a descriptor to be obtained from the linkage segment for loading into the instruction segment register (ISR). The entry descriptor also specifies the value to be loaded into the instruction counter (IC).

IC Value Reported: The processor is placed in Privileged Master mode for the execution of the wired-in CLIMB. Upon completion of the CLIMB, the processor remains in Privileged Master mode if flag bit 26 of the new ISR = 1 (Privileged). Otherwise, the processor changes to Master mode.

Fault Register Content: Not applicable

6.3.2.12 DRL Fault

Mnemonic: DRL

Name: Derail

Fault Code: 000110

Description: The DRL fault is caused by execution of the Derail instruction. It is a recoverable trap.

If the safe-store bypass flag in the option register = 1, a safe store frame is generated. The size of this safe store frame is determined by the type of the entry descriptor. The occurrence of the DRL fault is indicated in the safe store frame by a code of 00110 in bits 12-16 of word 5.

The wired-in CLIMB instruction functions as though the second word of the CLIMB instruction had the following characteristics:

E	= 0	no parameters
C(18)	= 0	do not load X0
C(19)		no effect, turn Master Mode indicator ON
C(22-23)	= 00	Inward Climb S,D no effect

The entry descriptor specifies a descriptor to be obtained from the linkage segment for loading into the instruction segment register (ISR). The entry descriptor also specifies the value to be loaded into the instruction counter (IC).

The processor is placed in Privileged Master mode for the execution of the wired-in CLIMB. Upon completion of the CLIMB, the processor remains in Privileged Master mode if flag bit 26 of the new ISR = 1 (Privileged). Otherwise, the processor changes to Master mode.

IC Value Reported: The location of the DRL instruction.

Fault Register Content: Not applicable

6.3.2.13 IPR Fault

Mnemonic:	IPR						
Name:	Illegal Procedure						
Fault Code:	001010						
Description:	<p>An IPR fault occurs when the processor detects an undefined instruction code or when the processor detects fields or conditions for which further execution of an instruction is undefined.</p> <p>An IPR fault occurs when any of the below conditions are detected. These conditions are included in the discussions of the individual instructions in Sections 8-15.</p> <ul style="list-style-type: none"> – Illegal address modification for the instruction – Illegal use of RPT, RPD, RPL, XEC, and XED instructions <p>Other conditions causing IPR faults are described in the individual instruction specifications. With multiword instructions, the unused fields of the instruction and operand descriptor are generally not detected as IPR faults and have no effect on the operation of instructions.</p>						
IC Value Reported:	The location of the instruction causing the illegal procedure condition						
Fault Register Content:	<p>After the IPR fault occurs, the content of the Fault Register can be stored in the C(y-pair) with the SFR instruction. This information can be used to determine whether or not the cause of the fault was an illegal decimal digit.</p> <p>The Fault Register (FR) content is interpreted in the following way:</p> <p>Bits</p> <table> <tr> <td>0-5</td> <td>not applicable</td> </tr> <tr> <td>6 = 1</td> <td>the IPR is caused by an illegal decimal digit</td> </tr> <tr> <td>7-71</td> <td>not applicable</td> </tr> </table>	0-5	not applicable	6 = 1	the IPR is caused by an illegal decimal digit	7-71	not applicable
0-5	not applicable						
6 = 1	the IPR is caused by an illegal decimal digit						
7-71	not applicable						

Faults and Interrupts

6.3.2.14 FTAG Fault

Mnemonic:	FTAG
Name:	Fault Tag
Fault Code:	000011
Description:	The Fault Tag is a recoverable trap. It occurs when IT modification is specified during execution of an instruction and the corresponding td field is 0000.
IC Value Reported:	The location of the instruction detecting the Fault Tag
Fault Register Content:	Not applicable

6.3.2.15 SCL1 Fault

Mnemonic:	SCL1
Name:	Class 1 Security Fault
Fault Code:	010000
Description:	This fault occurs when execution of an instruction is attempted in an illegal processor mode. For example, if an attempt is made to fetch or modify instructions or operands from a housekeeping page in the Slave mode, or if an attempt is made to access a non-housekeeping page with a segment descriptor of type T = 1 or 3, an SFC1 occurs.
IC Value Reported:	The location of the instruction causing the fault
Fault Register Content:	Not applicable

Faults and Interrupts

6.3.2.16 DYN Fault

Mnemonic:	DYN
Name:	Dynamic Linking
Fault Code:	010001
Description:	This fault occurs when an attempt is made to execute a CLIMB instruction using a segment descriptor of type T = 5.
IC Value Reported:	The location of the CLIMB instruction
Fault Register Content:	Not applicable

6.3.2.17 MWS Fault

Mnemonic:	MWS
Name:	Missing Working Space
Fault Code:	010011
Description:	This fault is generated if: A PT/SCT/DVTBL is not present, OR An undefined Page Table type is specified by a PTDW, OR A PTD is not present in ID paging mode, OR A DVTBL Page Table type is specified by a PTDW in ID paging mode.
IC Value Reported:	The location of the instruction causing the fault
Fault Register Content:	Not applicable

Faults and Interrupts

6.3.2.18 MSCT Fault

Mnemonic: MSCT

Name: Missing Section

Fault Code: 010111

Description: This fault occurs when the Page Table Base Word (PTBW) bit 20 = 0. This fault, like the Missing Page, is recoverable.

6.3.2.19 MPF Fault

Mnemonic:	MPF
Name:	Missing Page
Fault Code:	010100
Description:	This fault occurs when bit 30 of the Page Table Word (PTW) obtained with translation from a virtual to a real memory address is = 0. The processor is able to recover from Missing Page faults for all instructions.
IC Value Reported:	The location of the instruction causing the fault For the RPT, RPD, or RPL instructions, if an MPF is detected with a repeated instruction, the reported value indicates the RPT, RPD, or RPL instruction.
Fault Register Content:	Not applicable
Stack Frame:	Bit 9 of word 5 of the safe store stack frame created for an MPF contains information required by hardware to resume execution of the instruction. Thus, software must not change the status of this bit in the safe store stack frame before returning to the faulted instruction. (Refer to the discussion of the CLIMB instruction in Section 9.)

Faults and Interrupts

6.3.2.20 SCL2 Fault

Mnemonic:	SCL2
Name:	Class 2 Security Fault
Fault Code:	010101
Description:	<p>This fault occurs when conditions specified in a flag field of the segment descriptor or PTW are not satisfied.</p> <ul style="list-style-type: none">– Segment Descriptor Flag bits 20, 21, 22, or 25 are in violation– PTW Flag bit 31 (write control) is in violation– Working Space access control violation during address translation
IC Value Reported:	The location of the instruction causing the fault.
Fault Register Content:	Not applicable

6.3.2.21 SSSF Fault

Mnemonic: SSSF

Name: Safe Store Stack Fault.

Fault Code: Fault code not applicable. Refer to Stack Frame below.

Description: This fault is used to notify software that the safe store stack is full. It is generated and reported in the following procedures:

Before safe store operation with a programmed inward CLIMB (including system entry), the SSR base is increased, and the SSR bound is decreased, in accordance with the value in the SCR. The resulting bound value is then checked and the following checks performed.

If the bound ≥ 159 words + 3 bytes, the status is saved, and the programmed inward CLIMB is completed.

If the bound $< 159 + 3$ bytes, the programmed inward CLIMB is aborted, and the Safe Store Stack fault is generated. The processor status at the beginning of the programmed inward CLIMB is saved in the safe store stack frame. The Safe Store Stack Fault flag (bit 10 of word 5 of the safe store stack frame) is set to ON with Safe Store Stack fault processing. (Refer to Figure 8-7 in Section 8 for the Safe Store Stack Format.)

NOTE: The bound < 159 words + 3 bytes condition is detected again during Safe Store Stack fault processing but Safe Store Stack Fault processing is completed.

Before safe store operation with wired-in CLIMB because of a fault or interrupt, the SSR base and bound are adjusted as with a programmed inward CLIMB. The resulting bound value is checked and the following checks are performed.

If the bound ≥ 159 words + 3 bytes, wired-in CLIMB processing is completed normally.

Faults and Interrupts

SSSF Fault (cont.)

If the bound < 159 words + 3 bytes, the code for fault or interrupt is saved in the safe store stack frame, the Safe Store Stack Fault flag (bit 10 of word 5 of the safe store stack frame) is set to ON, and wired-in CLIMB processing is completed. The relevant software must, therefore, check the Safe Store Stack Fault flag with all faults and interrupts.

In both of the above cases, when the safe store stack is full and the processor must perform the safe store operation, the processor interrupts the SCU and halts with a fault on fault condition. For example, when software is unable to perform suitable processing despite the Safe Store Stack Fault flag being ON, a fault occurs and a further safe store is required.

IC Value Reported:

When this fault occurs with a programmed inward CLIMB, the IC indicates the location of the programmed inward CLIMB instruction. When this fault occurs because of another fault, the IC indicates the location specified with that fault. When this fault occurs together with an interrupt, the IC value + 1 is reported. It indicates the instruction to which processing is returned.

Fault Register Content: Not applicable

Stack Frame: A separate flag (bit 10 of word 5 of the safe store stack frame) is set to ON (1).

6.3.2.22 CON Fault

Mnemonic: CON

Name: Connect

Fault Code: 001000

Description: This fault occurs after the processor receives the connect pulse, when program recovery becomes possible. The processor starts fault processing with the same timing as with the reception of the interrupt. When the processor receives a connect pulse, notification of this fact is retained in the processor until Connect fault processing is executed. If a further connect pulse is received before the Connect fault processing is executed, it is ignored.

IC Value Reported: The location of the instruction to which processing is returned.

Fault Register Content: Not applicable

Faults and Interrupts

6.3.2.23 TRO Fault

Mnemonic: TRO

Name: Timer Runout

Fault Code: 000100

Description: This fault occurs after the Timer Register (TR) in the processor reaches zero when program recovery becomes possible. In VMOS mode a TRO fault occurs when the VM Timer Register equals zero. The processor usually starts fault processing with the same timing. However, when the processor is in the Privileged Master mode, the fault does not occur until it goes to the Slave or Master mode. If bit 28 of the instruction word is = 0 during execution of the DIS instruction, the fault occurs even in the Privileged Master mode.

When the processor detects the timer runout condition, notification of this fact is retained in the processor until the Timer Runout fault processing is executed. The Timer Register is counted at the rate of 512 kHz (i.e., every 1/512 ms.). When the TR reaches 0, it is reset to all ones, and the count begins again.

IC Value Reported: The location of the instruction to which processing is returned.

Fault Register Content: Not applicable

6.3.2.24 SDF Fault

Mnemonic:	SDF
Name:	Shutdown
Fault Code:	010110
Description:	After the CPU receives the shutdown signal, this fault occurs when program recovery becomes possible. The CPU starts fault processing with the same timing as with the reception of an interrupt. When the CPU receives the shutdown signal, notification of this fact is retained in the CPU until Shutdown Fault processing is executed.
IC Value Reported:	The location of the instruction to which processing is returned.
Fault Register Content:	Not applicable

Faults and Interrupts

6.3.2.25 MEEP Fault

Mnemonic: MEEP

Name: MEEP fault

Fault Code: 011001

Description: This fault is generated by information by the Service Processor using the Set Fault MPSC, and is used in conjunction with the DIAG instruction (issued by the operating system). Reserved Memory Space (RMS) is used to pass information. The DIAG instruction and the MEEP fault constitute the OS/SP interface.

6.3.2.26 ADT Fault

Mnemonic: ADT

Name: Address Trap Fault

Fault Code: 100000

Description: An Address Trap fault can occur on either an instruction or operand address and on either Read or Write commands. An Address Trap fault will occur when the following conditions are met:

Read Command

Debug Mode Register (DMR) bit 0 = 1

Instruction is complete

C(AMR) = Virtual Address (lower 3 bits ignored)

OR

Write Command

Debug Mode Register (DMR) bit 1 = 1

Instruction is complete

C(AMR) = Virtual Address (lower 3 bits ignored)

6.3.3 Segment Descriptor Flag Faults

The segment descriptor flag field provides the operating system with the capability of assigning use attributes for the addressed areas specified by the segment descriptors. The attributes are specified within the flag field. When attributes are assigned by software, they are checked by hardware. The purposes of the flag field and the faults which occur with flag violations are described below.

1. Bits 20, 21: Read/Write Permission

The read/write flags apply to operands and indirect words. Fetching instructions from a segment is controlled with the execution flag. When the operand address is generated for instructions that read data from memory (e.g., LDA), hardware checks the read flag to determine whether read from memory is permitted. If it is not permitted, a Class 2 Security Fault is generated. The PTW page accessed bit is not set in this case.

When the operand address is generated for instructions which store data in memory (e.g., STA), hardware checks the write flag to determine whether or not storing data is permitted in that segment. If it is not permitted, a Class 2 Security Fault is generated. In this case, the PTW page modified bit and page accessed bit are not set, and the operand is not stored.

With Read-Alter-Rewrite (RAR) type instructions (e.g., AOS), the write flag is checked together with the read flag when read is performed. If write is not permitted, a Class 2 Security Fault occurs before read, and the indicators remain unchanged.

When indirect words are prepared, read must be permitted for the segment. With indirect then tally modification (IT), both read and write must be permitted for the segment in order to enable the indirect word to be read and stored.

A segment descriptor in the ISR must have execute permission. (See discussion of execute permission flag in Section 3 under Standard Descriptor.)

Because read permission is not required to access instruction segments, reading data from memory when the ISR is used to prepare an operand address is permitted despite the read flag. In the case of multiword instructions, bit 29 of the instruction word = 0, or the AR bit = 0. With the store operation, the write flag is still checked as described above. The execute flag has the meaning of the read flag only when the segment descriptor is in the ISR.

With an XEC or XED instruction, in which bit 29 = 1, the instruction to be executed is accessed as an operand. The specified DR_n segment descriptor must have read permission, but execute permission is not required.

2. Bit 22: Save Permission

Flag bit 22 is checked by hardware when a DR_n segment descriptor is stored in a type T = 1 or 3 segment with the STD_n instruction. A Class 2 Security Fault occurs if an attempt is made to store a DR_n with flag bit 22 (indicates store by STD_n) = 0 in a type T = 1 or 3 segment.

Save permission for entry descriptors is determined by checking bit 18 of the entry descriptor.

3. Bit 23: IGNORED

4. Bit 25: Execute Permission

The execute flag is used to determine execute permission for instructions in the segment. A segment having execute permission does not require read permission to execute instructions (i.e., execution of an instruction includes fetching of the instruction from memory).

When the instruction segment descriptor is loaded into the ISR by the execution of a CLIMB or by a transfer instruction in which bit 29 of the instruction word = 1, the execute flag is checked by the hardware. If an attempt is made to load a segment descriptor in which flag bit 25 = 0, a Class 2 Security Fault occurs.

5. Bit 26: Privileged

The privileged flag is used only for instruction segments. When a segment descriptor in which flag bit 26 = 1 is loaded into the ISR, the Indicator Register Master Mode bit must be ON or a Class 1 Security Fault occurs. When an instruction is executed in the Privileged Master mode, operands and instructions executed with XEC and XED instructions may be in non-privileged segments. When the processor is in the Master or Slave mode, instructions executed with XEC and XED may be in privileged segments (i.e., when the instruction obtains an operand, the privileged bit of the operand segment is not checked by the hardware).

6. Bit 27: Bound Valid

The bound valid flag is used to specify whether the segment is bound valid or contains data. A Bound fault occurs if an attempt is made to access an empty segment (flag bit 27 = 0). A segment descriptor indicating an empty segment cannot be loaded into the ISR. The empty flag is used differently with the ASR in that, when a segment descriptor in the ASR indicates an empty segment, the ASR flag bit 27 is set when a segment descriptor is pushed to the argument segment. (See the specifications for the CLIMB and the SDR_n instructions in Section 9.)

7. Bit 28: Segment Present

The segment present flag is used to indicate whether a segment is in real memory (bit 28 = 1). A Missing Segment fault occurs if an attempt is made to generate a memory address using a segment descriptor in which bit 28 = 0 indicating that a segment is not present. Segment descriptors indicating that a segment is not present cannot be loaded into the ISR.

6.3.4 Page Table Word Control Field Faults

Faults may be generated when the PTW control field is checked by hardware. The various bits in the PTW control field together with their related faults are discussed below. The PTW format is discussed in Section 5.

1. Bit 30 - CPU Page Present/Missing Bit

When a virtual address is translated to a real memory address, bit 30 of the control field is checked each time the PTW is fetched. A Missing Page fault occurs if bit 30 = 0, indicating page missing. Execution of the instruction is continued if bit 30 = 1, indicating page present.

2. Bit 31 - Write Control Bit

The Page Table Word (PTW) control bit 31 is used to control memory write operation at the page level. Write operation at the page level may not be permitted, even though the write operation for the segment including the page is permitted. When data is written into memory, both segment descriptor write permission and PTW write permission are required. If write is not permitted for the segment descriptor but is permitted for the PTW, a Class 2 Security Fault occurs with segment write.

The segment descriptor write flag is checked during preparation of the operand address for storing data in memory. If write is not permitted, the instruction is terminated with a fault, and the PTW write control bit is not checked. The PTW bit 31 is checked if storing in memory proceeds to the point at which the PTW is obtained. If bit 31 = 1 (indicating write is permitted), execution of the instruction is continued. If bit 31 = 0 (indicating write is not permitted), the instruction is terminated with a Class 2 Security Fault.

3. Bit 32 - Housekeeping Bit

The PTW housekeeping bit is used by the operating system to enable allocation in page units of use attributes depending upon the processor mode. (Allocations in the three processor modes is described below.) The hardware checks the PTW housekeeping bit on all instruction fetches, all operand fetches and stores, and all segment descriptor fetches and stores. Instructions and operands must be contained in a segment described with type T = 0, 2, 4, 6, 12, or 14 segment descriptor. The page may be either a housekeeping or nonhousekeeping page. The segment descriptors must be contained in a type T = 1 or 3 segment, and the page must be a housekeeping page.

a) Privileged Master Mode

When the processor is in Privileged Master Mode, all instructions must be fetched from housekeeping pages of type T = 0 segments. An attempt to obtain an instruction from a nonhousekeeping page causes a Class 1 Security Fault. An exception applies for those instructions executed by an XEC or XED. Fetching and storing operands may be performed for both housekeeping and nonhousekeeping pages.

References to a type T = 0, 2, 4, 6, 12, or 14 segment to access or alter data other than instructions may be to either housekeeping or nonhousekeeping pages. The segment descriptors must be contained in a type T = 1 or 3 segment and the page must be a housekeeping page or a Class 1 Security Fault will be generated.

b) Master Mode

When the processor is in Master Mode, instructions may be fetched from housekeeping or nonhousekeeping pages of type T = 0 segments. Operands may be fetched from housekeeping or nonhousekeeping pages of type T = 0, 2, 4, 6, 12, or 14 segments. However, operands may not be stored on housekeeping pages (only Privileged Master Mode instructions may modify these housekeeping pages). Any attempt to modify a housekeeping page in Master Mode causes a Class 1 Security Fault.

Because segment descriptors are not processed as operands, the SDR_n and STD_n instructions may be used to store DR_n content in type T = 1 or 3 segments in housekeeping pages. All segment descriptor segment pages must be housekeeping pages or a Class 1 Security Fault occurs, and the instruction is terminated.

c) Slave Mode

When the processor is in Slave Mode, instructions must be fetched from nonhousekeeping pages of type T = 0 segments. Attempts to obtain an instruction from a housekeeping page result in a Class 1 Security Fault. Operands must be fetched from or stored into nonhousekeeping pages of type T = 0, 2, 4, 6, 12, or 14 segments. Since descriptors in type T = 1 or 3 segments are not treated as operands, they may be stored or fetched from housekeeping pages in Slave Mode. Thus, the SDR_n and STD_n instructions may store the contents of a DR_n in a type T = 1 or 3 segment. In this case, the page must be a housekeeping page, or a Class 1 Security Fault occurs. With the LDD_n, LDP_n, and CLIMB instructions, segment descriptors may be obtained from a type T = 1 or 3 segment. In this case, the page must be a housekeeping page, or a Class 1 Security Fault, occurs.

d) All Modes

Instructions that may refer to type T = 1 or 3 segments (LDP_n, LDD_n, SDR_n, STD_n, and CLIMB) must refer to a housekeeping page when fetching or storing the identified descriptor or safe store data. Otherwise, a Class 1 Security Fault is generated.

Privileged instructions (such as LDSS, LDAS, and STSS) that load descriptors from type T = 0, 2, 4, 6, 12, or 14 segments into registers, or store descriptors from registers into segments, do not require that the housekeeping bit be set ON.

Nonprivileged instructions (such as STAS, STPS, and STD_n) that store descriptors from registers into T = 0, 2, 4, 6, 12, or 14 segments access normal memory areas and do not require the housekeeping bit. The STD_n instruction accesses both normal memory areas and memory areas which contain segment descriptors.

Faults and Interrupts

4. Bit 33 - IO Page Present/Missing Bit

This bit is neither checked nor altered by the processor.

5. Bit 34 - Page Modify Bit

The processor sets this bit to 1 each time data is written into a page in which bit 34 of the PTW = 0 to indicate that the page has been modified.

6. Bit 35 - Page Access Bit

The processor sets this bit to 1 each time a page is accessed via read or write to indicate that the page has been accessed.

6.4 CPU Cache

The DPS 9000G system utilizes two cache levels. The first of these is a 16 K word cache which is an integral part of each CPU. A second level cache of 1 M Word is shared by four CPUs via a common CPU bus. The CPU cache can not be bypassed. Software has no direct control over the operation of the cache.

6.5 Interrupt Procedure

Interrupt procedures use interrupt cells and the Interrupt Mask register. Thirty two interrupt cells exist in the System Control Unit (SCU) which are set with 1 and reset with 0. When the Input/Output Unit (IOU) completes an I/O operation or when particular events need to inform the processor, the IOU sets the corresponding interrupt cell in the SCU and sends an interrupt present (XIP) signal. Each processor possesses an Interrupt Mask register (IMR), as well as the instructions for loading and storing this register. (See the descriptions of LIMR and RIMR.)

6.5.1 Interrupt Timing

The processor receives interrupts (samples the XIP signal) when the following conditions are satisfied. Only the master processor receives an interrupt. (See the specifications for the DIAG instruction in Section 9.)

6.5.1.1 Single-Word Instructions

For a single-word instruction, the processor receives an interrupt after execution of an instruction located at an odd memory address (the IC value is odd) in which the interrupt inhibit bit (bit 28 of the instruction word) is OFF. The processor does not receive an interrupt under the following conditions:

- during execution or after execution of the XEC, XED, RPT, RPD, or RPL instruction;
- after execution of the MME or DRL instructions;
- after execution of an unconditional transfer of control instruction; or
- after execution of a conditional transfer of control instruction with a transfer-go.

NOTE: When the DIS instruction is executed, an interrupt is received, regardless of the interrupt inhibit bit and the memory address (odd or even address) of the instruction.

Interrupts are received only when no fault is present.

6.5.1.2 Multiword Instructions

Except for CLIMB instructions, interrupts are received by multiword instructions under the following conditions:

- after execution of an instruction which is at an odd memory address and in which the interrupt inhibit bit is OFF;
- at the starting point of an instruction in which the interrupt inhibit bit is OFF (regardless of the memory address of the instruction); or
- after processing each appropriate character block of data during execution of one of the interruptible multiword instructions listed below, in which the interrupt inhibit bit is OFF (regardless of the memory address of the instruction).

Interruptible multiword instructions

MLR, MRL, MVT, CMPC, SCD, TCT, CSL, CSR, SZTL, SZTR, CMPB, SCDR, TCTR, SCM, SCMR, CMPCT

Interrupts are not received when the instruction is executed by the XEC or XED instructions, but only when no fault is present.

6.5.1.3 Climb Instruction

An interrupt is received only during execution of data stack clear processing with an outward CLIMB instruction in which the interrupt inhibit bit is OFF.

6.5.2 Interrupt Processing

When the processor receives an interrupt, the following procedures are performed.

1. The processor waits for the completion of all current processing.
2. The processor obtains an interrupt cell from the SCU.
3. It then compares the content of the interrupt cell and the Interrupt Mask register.
4. If the interrupt cell is not set, or is masked, the processor restores its content to the SCU, resets the XIP signal, and completes the interrupt processing.
5. If one or more interrupt cells are set, and the set cell(s) are not masked, the processor selects the cell with the highest priority, i.e., it decreases interrupt priority from 0 to 31. It then resets the selected interrupt cell and restores its content to the SCU.
6. If this interrupt cell is the only one in the interrupt pending status, the processor resets the XIP signal, providing no other cells are set or masked. If another interrupt is pending, the XIP signal is not reset.
7. The processor generates a 5-bit binary number to indicate the location of the selected interrupt cell within the entire group of cells.
8. The processor executes a wired-in CLIMB that is an interdomain call. When the safe store operation is performed, a flag and a 5-bit interrupt cell number are stored in word 5 of the safe store stack (in bit 11 and bits 12-16, respectively). The flag (= 1) indicates that the safe store frame was generated for interrupt. (Refer to the CLIMB instruction specifications in Section 8, including the Safe Store Stack Format in Figure 8-7.)

If the interrupt is received during execution of an interruptible multiword instruction, the processor sets bit 30 of the Indicator register (IR) ON. If the entry descriptor is type T = 11, it saves the contents of the Pointer and Length registers. Bit 30 of the Indicator register is reset to OFF after it is saved as ON. When it returns to the interrupted interruptible multiword instruction (during the execution of an outward CLIMB instruction via a safe store frame), the processor resets bit 30 of the IR restored from the safe store frame to OFF and resumes execution of the instruction.

9. The processor interprets the second word in the wired-in CLIMB in the following way:

E bit = 0 (a parameter is not passed)

C Field

Bit 18 = 0 (X0/GX0 remains unchanged)

Bit 19 = Ignored (IR Master Mode bit is set to ON)

Bits 20, 21 = Ignored

Bits 22, 23 = 00 (Inward Climb is executed)

S, D Fields = Ignored (the entry descriptor is obtained from a fixed memory location)

10. If no entry descriptor appears at the fixed memory location, the processor interrupts the SCU and halts processing.

6.5.3 Instruction Counter Value Stored At Interrupt

The values of the Instruction Counter (IC) stored at interrupt are listed below:

- Single-word Instructions
The address of the instruction to which processing is to be returned is stored. When an interrupt is received with the DIS instruction, the DIS instruction address + 1 is stored.
- Multiword Instructions (excluding CLIMB Instructions)
The address of the instruction to which processing is to be returned is stored. If interrupt occurs during execution of an interruptible multiword instruction, IC + 0 is stored, and the IR bit 30 is stored as a one.
- CLIMB Instruction
Except when the data stack area is cleared with an OCLIMB, interrupt is not received during execution of a CLIMB instruction. If an interrupt occurs during the execution of an OCLIMB, IC + 0 is stored.

7. Machine Instruction Functions

This section of the Programmer's Guide describes Machine Instruction formats, organized as follows:

- Section 7.1, Single-Word Instructions
- Section 7.2, Multiword Instructions
- Section 7.3, Address Register Instructions
- Section 7.4, Boolean Operation Instructions
- Section 7.5, Fixed-Point Instructions
- Section 7.6, Floating-Point Instructions
- Section 7.7, Quadruple-Precision Instructions
- Section 7.8, Multiword Instructions
- Section 7.9, Micro Operations for Edit Instructions MVE, MVNE, MVNEX
- Section 7.10, Virtual Memory Instructions
- Section 7.11, ES And EI Mode Instructions
- Section 7.12, Transfer Instructions
- Section 7.13, Miscellaneous Instructions
- Section 7.14, Coding Limitations
- Section 7.15, NovaScale 9000 Instruction Repertoire

Many of the instructions available in the instruction repertoire are familiar to experienced users of large-scale computers. However, additional instructions have been provided to supply extended capability for character handling, decision-making, and advanced programming techniques involving list processing. In addition, numerous instructions are provided that have capabilities for processing and moving bytes, BCD characters, packed decimal data, and bit strings, and for performing register-to-register operations.

7.1 Single-Word Instructions

Single-word instructions provide for multiple variations by permitting the user to specify not only the type of address modification desired, but also the source and/or destination registers associated with particular operation codes. For example, the operation field for a Transfer and Set Index Register \underline{n} (TSX \underline{n}) instruction specifies the index in the operation field, leaving full address modification capability free for destination calculation.

The processor performs efficient operations on 6-, 9-, 18-, 36-, and 72-bit operands.

The following operations are performed by single-word instructions:

- Address Register Instructions
- Boolean Operations
- Comparison Operations
- Data Movement Instructions
- Data Shifting Instructions
- Effective Address to Register Instructions
- Fixed-Point Arithmetic Instructions
- Floating-Point Arithmetic Instructions
- Quadruple-Precision Instructions
- Master Mode Instructions
- Miscellaneous Instructions
- Random Number Instructions
- ES/EI Mode Instructions
- Special Processor Instructions
- Transfer Instructions

7.1.1 Address Register Instructions

Address register instructions allow for loading and storing of address registers. The number of bits loaded or stored depends upon whether NS, ES, or EI mode is being used. Alter address register instructions are used to replace, increment, and decrement the content of the address register in word, character, or bit. These instructions perform operations between registers and do not refer to memory. Special address register instructions, executable only in the NS mode, use the address registers to manipulate the address portion of numeric and alphanumeric operand descriptors. (Refer to the instruction specifications in Sections 8-15).

7.1.2 Boolean Operations

The logical operations AND, OR, and EXCLUSIVE OR are permitted between storage and the index registers, A- and Q-registers, and the AQ-register.

7.1.3 Comparison Operations

Comparison operations do not alter the contents of storage or the specified register, but merely set or clear the appropriate indicators as the result dictates. The compare instructions enable the user to make many types of program decisions.

Fixed-point compare instructions permit comparison of absolute values (algebraic or characters), provide for tests of word fields, permit searches for identical selectable-word fields, and permit searches for a value within selectable limits.

Floating-point compare instructions are included for single- and double-precision operations on absolute values and algebraic values. All compare instructions are repeatable using the RPT, RPD, or RPL instructions.

7.1.4 Data Movement Instructions

Character handling and manipulation are facilitated by "indirect and tally" (IT) address modification and by instructions for directly storing selected characters of the accumulator or quotient register. Instructions are also included for directly loading the index registers from either memory or the A- and Q-registers, directly storing any register into memory, and loading registers with the two's complement (negative) of the contents of the memory location specified.

7.1.5 Data Shifting Instructions

Shifting is accomplished using an algorithm in which long shifts are executed essentially as fast as short shifts. The A- and Q-registers can be shifted individually or as one unit. The shift commands include right- or left-shift arithmetic, right-shift logical, and left-shift rotate, (right-shift rotate is omitted because the high speed of the left-shift rotate makes the right-shift rotate unnecessary).

7.1.6 Effective Address to Register Instructions

The Effective Address to Register instructions permit the effective address of such an instruction to be placed in any of the index registers, in the A-register, or in the Q-register. Thus, any effective address referenced frequently in a program can be stored in a register and used without lost processing time in repeatedly redeveloping the effective address. Furthermore, the instructions provide the user with the capability of transferring data among any of the index registers and to the A-register and the Q-register.

7.1.7 Fixed-Point Arithmetic Instructions

Instructions for both fractional and integral multiplication and division free the programmer from scaling the results of such operations. Fractional multiplications are performed with the multiplicand in the A-register. The result appears in bit positions 0 through 70 of the AQ-register, automatically scaled with the binary point to the right of position 0. Integral multiplications are performed with the multiplicand in the Q-register. The result appears in bit positions 1 through 71 of the AQ-register, automatically scaled with the binary point to the right of position 71.

Fractional divisions use the full range of the AQ-register for the dividend. The quotient appears in the A-register with the remainder in the Q-register. The binary point is automatically scaled to the right of position 0. Integral divisions have the dividend in the Q-register, with the binary point to the right of position 35. After division, the quotient is in the Q-register with the binary point automatically placed to the right of position 35, and the remainder is in the A-register.

Normally, the integer operations of divide and multiply occur in the Q-register, and the fractional operations of divide and multiply occur in the A-register. This convention permits easy programming of fixed-point arithmetic operations.

Instructions are provided for combining the contents of memory locations directly with the contents of registers and storing the results in the same locations, without recourse to separate store instructions. In all such cases, the programmer can use the 18-bit indexing registers, X0 through X7 in the NS mode, the 36-bit general indexing registers, GX0 through GX7 in the ES mode, and the 36-bit A- and Q-registers. In effect, the Add and Subtract to Storage instructions make arithmetic accumulators of all available memory locations. In all such cases, the register contents are undisturbed.

7.1.8 Floating-Point Arithmetic Instructions

Floating-point operations can be performed on both single- and double-precision data words. Complete sets of data movement, arithmetic, and control instructions are provided for use in both types of operations. Unless otherwise specified by the programmer, the mantissas of all floating-point operation results, except divides, are automatically normalized by the hardware. In additions and subtractions, the operands are automatically aligned.

Operations on floating-point numbers are performed with an extended register composed of a 72-bit AQ-register, which holds the mantissa, and a separate 8-bit exponent register. Operations on the exponent and mantissa are performed by two separate adders. The existence of separate exponent and mantissa registers and adders enables the programmer to efficiently intermix single- and double-precision instructions.

The floating-point instruction repertoire includes two special divide instructions: Floating Divide Inverted (FDI) and Double-Precision Floating Divide Inverted (DFDI). These instructions cause the contents of the memory location to be divided by the contents of the AQ-registers, the reciprocal of other divide instructions in the repertoire. Thus, regardless of whether the contents of the AQ-register must be a dividend or a divisor, the programmer can always perform a division without recourse to wasteful data movement operations.

Floating Negate, Normalize, Add to Exponent, and Single- and Double-Precision Compare instructions further facilitate effective programming. The hexadecimal option may be used in floating-point operations to declare hexadecimal constants, either explicitly or by default. (Refer to Hexadecimal Floating-point Number in Section 2.)

7.1.9 Quadruple-Precision Floating-Point Instructions

Quadruple-precision floating-point instructions provide arithmetic operations for which the exponents are handled as powers of 16. In these operations, the AQ register and the operand register (LOR) handle mantissas and the E register handles exponents. Results of these operations are automatically normalized.

7.1.10 Privileged Master Mode Instructions

The following conditions must be satisfied for execution of these instructions.

1. The Master Mode bit in the Indicator Register is ON.
2. The privileged bit in the Instruction Segment Register (ISR) is ON.
3. The housekeeping bit in the page table word for the instruction is ON. This bit is assumed to be ON in the Working Space 0 Addressing mode.

When these conditions are not met, a Command fault or a Class 1 Security fault occurs. (Refer to the instruction specifications in Sections 8-15.)

7.1.11 Miscellaneous Instructions

This category includes instructions that perform operations such as Binary-to-BCD and gray-to binary conversions, programmed faults, repeat instructions, and no-operation instructions (e.g., NOP).

7.1.12 Random Number Instructions

Random number instructions generate fixed-point and floating-point random numbers.

7.1.13 Special Processor Instructions

Slave mode instructions available to provide the operating system with program gating for multiprocessor configurations include LDAC, LDQC, and SZNC. They provide for clearing the referenced memory cell to zero after the contents are transferred to the processor. The instructions, STAC and STACQ, provide for conditional storage in the referenced memory cell, based on the condition of the C(Y) being zero (STAC) or the comparison of Q with the operand word (STACQ).

The slave mode instructions providing rounded floating-point results include FRD, DFRD, FSTR, and DFSTR.

Four Master Mode instructions provide system information and control: LLUF, SFR, RCCL, and LCCL.

7.2 Multiword Instructions

Multiword instructions fall into five general categories:

1. Alphanumeric instructions,
2. Numeric instructions,
3. Bit string instructions,
4. Conversion instructions,
5. Edited Move Instructions.

7.2.1 Alphanumeric Instructions

Alphanumeric instructions permit moving, transliterating, editing, and comparing alphanumeric data. The operands for these instructions (with the exception of comparisons) can be any combination of alphanumeric types (9-bit, 6-bit, or 4-bit). These operands are translated as part of the instruction execution to permit the different types of character strings to be manipulated in the same instruction.

7.2.2 Numeric Instructions

Numeric instructions include decimal arithmetic functions in addition to moving, comparing, and editing of numeric data. Decimal add, subtract, multiply, and divide operations are permitted. The numeric instructions can be 2- or 3-operand instructions. The operands themselves can be either 9-bit or 4-bit packed decimal. The numbers employed as data can be floating-point with leading sign, scaled fixed-point with trailing sign, leading sign, or no sign. As with alphanumeric instructions, numeric instructions achieve these various characteristics within a single multiword instruction (in conjunction with associated operand descriptors).

7.2.3 Bit String Instructions

Bit string instructions allow two bit strings to be compared on a bit-by-bit basis and Boolean operations to be performed to combine strings and set indicators.

7.2.4 Conversion Instructions

Conversion instructions provide for decimal/binary and binary/decimal conversion.

7.2.5 Edited Move Instructions

Both alphanumeric and numeric edited move instructions (MVE, MVNE, and MVNEX) utilize micro operations (MOPS) to perform editing functions. The sequence of micro-steps to be executed is contained in memory and is referenced by the second operand descriptor of the edited move instructions.

Micro operations provide alphanumeric and numeric edited move instructions with the capability to edit strings on a character-by-character or digit-by-digit basis, or in concatenated series of characters and digits.

Micro operations are not altered by their execution. Therefore, a sequence of micro operations can be set to describe a data field and then can be used repeatedly by the edit instructions. A single instruction can perform a complicated edit function with great speed.

The special edit characters are contained in a hardware edit table. Table entries are modified using micro operations that have been designed for this purpose. (Refer to "Micro Operations for Edit Instructions MVE, MVNE, and MVNEX" later in this section for detailed information.)

7.2.6 Multiword Instruction Capabilities

The capabilities of the multiword instructions are given below.

1. Decimal Arithmetic Capability
 - a) Data types as packed decimal and direct ASCII (may be intermixed)
 - b) Decimal arithmetic operands of 1 to 63 digits in length (including sign)
 - c) Numeric data as fixed-point and/or floating-point (intermixed fixed- and floating-point data is allowed)
 - d) A full set of decimal arithmetic instructions (each is a multiword instruction with either two or three descriptor words) including add, subtract, multiply, and divide
 - e) All numeric instructions with a hardware rounding option
2. Data Manipulation Capability

Five native data modes: ASCII, BCD, packed decimal (numeric only), bit string and EBCDIC
3. Data Movement Capability
 - a) Alphanumeric movement from left or right with character-fill
 - b) Character moves from 9-bit-byte or 8-bit-byte fields
 - c) Numeric move with fill and/or rounding and scale change
 - d) Bit string manipulation using any of 16 different Boolean operations
 - e) Radix conversion and transliteration instructions
4. Data Comparison Capability
 - a) Alphanumeric comparison with fill
 - b) Numeric comparisons between fields of the same or different format and character type
 - c) Bit string comparisons with fill
 - d) String scan for a match of one or two characters
5. Second-Level Indexing Capability

Eight address registers providing for second-level indexing for all instructions (including single-word instructions)

7.3 Address Register Instructions

This set of instructions provides the capability for using address registers to manipulate the address portion of numeric and alphanumeric descriptors. If an address register is to be used in address preparation, this information is specified in the instruction word. All single-word instructions, to which address modification is applicable, have essentially the same machine instruction word format which hardware interprets differently depending on whether the processor is in the NS or the ES mode. (Refer to Section 5.)

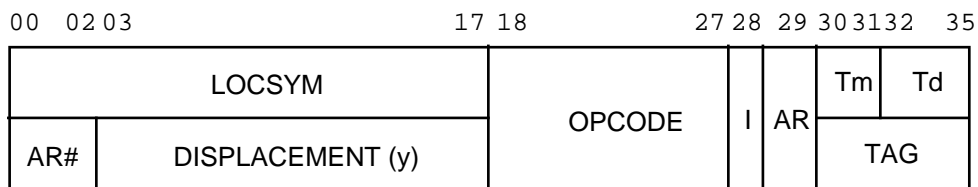


Figure 7-1. Single-word Instruction with Address Modification

AR#	one of eight address registers (0-7)
LOCSYM	represents either the address of operand or displacement from a base
DISPLACEMENT	(y) a 15-bit displacement from the address register address (two's complement: values from -16,384 to +16,383)
OP CODE	a 10-bit operation code field
I	program interrupt inhibit bit
AR	If bit 29 is 1, an address register is to be used and is specified by bits 0, 1, and 2 of the y field. If bit 29 is 0, no address register is used.
TAG	The tag field controls all other address modification. If an address register is used on an instruction with indirect addressing, it is applied only on the fetch of the indirect word.
Tm	tag modifier
Td	tag designator

7.3.1 Address Register Load

LARn	76n (1)	Load Address Register n
LAREG	463 (1)	Load Address Registers

7.3.2 Address Register Store

SARn	74n (1)	Store Address Register n
SAREG	443 (1)	Store Address Registers

7.3.3 Alter Address Register Contents

This set of instructions provides the capability for replacing, incrementing, and decrementing the contents of an address register on either a word, character, or bit address basis. The operation is register-to-register, with no memory fetch involved.

The special instructions have the same instruction format:

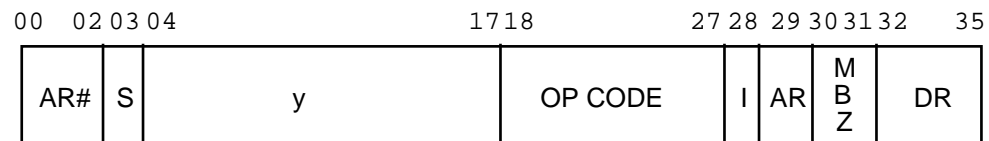


Figure 7-2. Alter Address Register Contents

AR#	selects address register to be altered
S	sign bit (Refer to Section 5 for differences between NS and ES modes.)
y	used as a word displacement (no character or bit position included) along with the contents specified in the DR field to alter the contents of the specified address register. Bit 3 provides negative (two's complement) or positive word displacement.
OP CODE	10-bit operation code field.
I	program interrupt inhibit bit.

AR	address register bit.
	If bit 29 = 1, the sum of the DR (in characters, words, or bits) and the y field (in words) are added to or subtracted from the contents of the AR specified in bits 0-2.
	If bit 29 = 0, the sum of the DR or it's two's complement is loaded into the AR for addition or subtraction, respectively.
	If the mnemonic is coded with X (for example, AWDX), bit 29 is forced to zero.
MBZ	bits 30-31 must be zero
DR	displacement register
	specifies which register contains the displacement value
	The register codes and register lengths are the same as those used in MF fields except that IC modification is illegal. (Refer to Table 5-2) (Refer also to Multiword Modification Field in this section.)

The operations for adding a value to the contents of an address register are like those for effective operand address preparation from an operand descriptor, with the final results being stored in the specified address register.

The subtract operation differs only in that the contents of the register specified by the code in the DR field are first added to the y field. This result is then subtracted from the actual contents of the address register or from the implied zero contents, and the result is placed in the address register. The codes for DU, DL, and IC are illegal for the DR field and cause an IPR fault.

The indicators are unaffected by these instructions.

A4BD(X)	502 (1)	Add 4-Bit Displacement to Address Register
A6BD(X)	501 (1)	Add 6-Bit Displacement to Address Register
A9BD(X)	500 (1)	Add 9-Bit Displacement to Address Register
ABD(X)	503 (1)	Add Bit Displacement to Address Register
AWD(X)	507 (1)	Add Word Displacement to Address Register
S4BD(X)	522 (1)	Subtract 4-Bit Displacement from Address Register
S6BD(X)	521 (1)	Subtract 6-Bit Displacement from Address Register
S9BD(X)	520 (1)	Subtract 9-Bit Displacement from Address Register
SBD(X)	523 (1)	Subtract Bit Displacement from Address Register
SWD(X)	527 (1)	Subtract Word Displacement from Address Register

7.3.4 Special Address Register Instructions

Special instructions provide use of address registers to manipulate the address portion of numeric and alphanumeric operand descriptors. These instructions may be used only in the NS mode. If an attempt is made to execute these instructions in the ES mode, an IPR fault occurs.

These special instructions have the following instruction format:

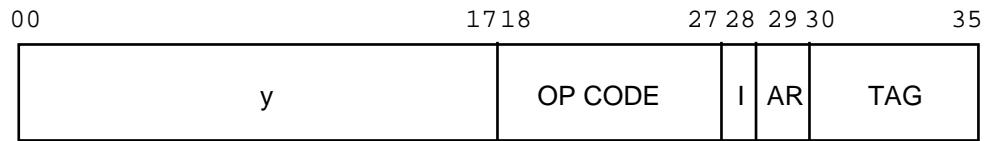


Figure 7-3. Special Address Register Instructions

AARn	56n (1)	Alphanumeric Descriptor to ARn
ARAN	54n (1)	ARn to Alphanumeric Descriptor
ARNn	64n (1)	ARn to Numeric Descriptor
NARn	66n (1)	Numeric Descriptor to Arn

7.4 Boolean Operation Instructions

The logical operations AND, OR, and EXCLUSIVE OR are permitted between storage and the index registers, A- and Q-registers, and the AQ-register.

7.4.1 Boolean Expressions

A Boolean expression is defined similarly to an algebraic expression except that the operators *, /, +, and - are interpreted as Boolean operators. Two types of Boolean expressions are defined below:

1. The expression that appears in the variable field of a BOOL pseudo-operation uses Boolean operators.
2. The expression that appears in the octal subfield of the variable field of a VFD pseudo-operation uses Boolean operators.

7.4.2 Evaluation Of Boolean Expressions

A Boolean expression is evaluated by the same procedure used for an algebraic expression except that the operators are interpreted as Boolean.

In a Boolean expression, the operators +, -, *, and / have Boolean meanings, rather than their normal arithmetic meanings:

Operator	Meaning	Definition
+	OR, inclusive OR, union	$0 + 0 = 0$ $0 + 1 = 1$ $1 + 0 = 1$ $1 + 1 = 1$
-	EXCLUSIVE OR, symmetric difference	$0 - 0 = 0$ $0 - 1 = 1$ $1 - 0 = 1$ $1 - 1 = 0$
*	AND, intersection	$0 * 0 = 0$ $0 * 1 = 0$ $1 * 0 = 0$ $1 * 1 = 1$
/	ones complement, complement, NOT	$/0 = 1$ $/1 = 0$

Machine Instruction Functions

Although / is a unary operation (involving only one term), by convention A/B means A*/B. This notation is not regarded as an error by the assembler. The table for / as a two-term operation is given below.

0/0 = 0
0/1 = 0
1/0 = 1
1/1 = 0

Other conventions are

+A = A+ = A
-A = A- = A
A = A = 0 (possible error, operand missing)
A/ = A/0 = A

7.4.3 Boolean AND

ANA	375 (0)	AND to A-Register
ANAQ	377 (0)	AND to AQ-Register
ANQ	376 (0)	AND to Q-Register
ANSA	355 (0)	AND to Storage from A-Register
ANSQ	356 (0)	AND to Storage from Q-Register
ANSXn	34n (0)	AND to Storage from Index Register n
ANXn	36n (0)	AND to Index Register n

7.4.4 Boolean OR

ORA	275 (0)	OR to A-Register
ORAQ	277 (0)	OR to AQ-Register
ORQ	276 (0)	OR to Q-Register
ORSA	255 (0)	OR to Storage from A-Register
ORSQ	256 (0)	OR to Storage from Q-Register
ORSXn	24n (0)	OR to Storage from Index Register n
ORXn	26n (0)	OR to Index Register n

7.4.5 Boolean EXCLUSIVE OR

ERA	675 (0)	EXCLUSIVE OR to A-Register
ERAQ	677 (0)	EXCLUSIVE OR to AQ-Register
ERQ	676 (0)	EXCLUSIVE OR to Q-Register
ERSA	655 (0)	EXCLUSIVE OR to Storage with A-Register
ERSQ	656 (0)	EXCLUSIVE OR to Storage with Q-Register
ERSXn	64n (0)	EXCLUSIVE OR to Storage with Index Register n
ERXn	66n (0)	EXCLUSIVE OR to Index Register n

7.4.6 Boolean COMPARATIVE AND

CANA	315 (0)	Comparative AND with A-Register
CANAQ	317 (0)	Comparative AND with AQ-Register
CANQ	316 (0)	Comparative AND with Q-Register
CANXn	30n (0)	Comparative AND with Index Register n

7.4.7 Boolean COMPARATIVE NOT AND

CNAA	215 (0)	Comparative NOT AND with A-Register
CNAAQ	217 (0)	Comparative NOT AND with AQ-Register
CNAQ	216 (0)	Comparative NOT AND with Q-Register
CNAXn	20n (0)	Comparative NOT AND with Index Register n

7.5 Fixed-Point Instructions

This section describes Fixed-Point instructions.

7.5.1 Data Movement Load

EAA	635 (0)	Effective Address to A-Register
EAQ	636 (0)	Effective Address to Q-Register
EAXn	62n (0)	Effective Address to Index Register n
LCA	335 (0)	Load Complement into A-Register
LCAQ	337 (0)	Load Complement into AQ-Register
LCQ	336 (0)	Load Complement into Q-Register
LCXn	32n (0)	Load Complement into Index Register n
LDA	235 (0)	Load A-Register
LDAC	034 (0)	Load A-Register and Clear
LDAQ	237 (0)	Load AQ-Register
LDI	634 (0)	Load Indicator Register
LDQ	236 (0)	Load Q-Register
LDQC	032 (0)	Load Q-Register and Clear
LDXn	22n (0)	Load Index Register n from Upper
LREG	073 (0)	Load Registers
LXLn	72n (0)	Load Index Register n from Lower

7.5.2 Data Movement Store

SREG	753 (0)	Store Registers
STA	755 (0)	Store A-Register
STAC	354 (0)	Store A Conditional
STACQ	654 (0)	Store A Conditional on Q
STAQ	757 (0)	Store AQ-Register
STBA	551 (0)	Store 9-bit Bytes of A-Register
STBQ	552 (0)	Store 9-bit Bytes of Q-Register
STC1	554 (0)	Store Instruction Counter Plus 1
STC2	750 (0)	Store Instruction Counter Plus 2
STCA	751 (0)	Store 6-bit Characters of A-Register
STCQ	752 (0)	Store 6-bit Characters of Q-Register
STI	754 (0)	Store Indicator Register
STQ	756 (0)	Store Q-Register
STT	454 (0)	Store Timer Register
STXn	74n (0)	Store Index Register n in Upper
STZ	450 (0)	Store Zero
SXLn	44n (0)	Store Index Register n in Lower

7.5.3 Data Movement Shift

ALR	775 (0)	A-Register Left Rotate
ALS	735 (0)	A-Register Left Shift
ARL	771 (0)	A-Register Right Logical Shift
ARS	731 (0)	A-Register Right Shift
LLR	777 (0)	Long Left Rotate
LLS	737 (0)	Long Left Shift
LRL	773 (0)	Long Right Logical Shift
LRS	733 (0)	Long Right Shift
QLR	776 (0)	Q-Register Left Rotate
QLS	736 (0)	Q-Register Left Shift
QRL	772 (0)	Q-Register Right Logical Shift
QRS	732 (0)	Q-Register Right Shift

7.5.4 Fixed-Point Addition

ADA	075 (0)	Add to A-Register
ADAQ	077 (0)	Add to AQ-Register
ADL	033 (0)	Add Low to AQ-Register
ADLA	035 (0)	Add Logical to A-Register
ADLAQ	037 (0)	Add Logical to AQ-Register
ADLQ	036 (0)	Add Logical to Q-Register
ADLXn	02n (0)	Add Logical to Index Register n
ADQ	076 (0)	Add to Q-Register
ADXn	06n (0)	Add to Index Register n
AOS	054 (0)	Add 1 to Storage
ASA	055 (0)	Add to Storage from A-Register
ASQ	056 (0)	Add to Storage from Q-Register
ASXn	04n (0)	Add to Storage from Index Register n
AWCA	071 (0)	Add With Carry to A-Register
AWCQ	072 (0)	Add With Carry to Q-Register

7.5.5 Fixed-Point Subtraction

SBA	175 (0)	Subtract from A-Register
SBAQ	177 (0)	Subtract from AQ-Register
SBLA	135 (0)	Subtract Logical from A-Register
SBLAQ	137 (0)	Subtract Logical from AQ-Register
SBLQ	136 (0)	Subtract Logical from Q-Register
SBLXn	12n (0)	Subtract Logical from Index Register n
SBQ	176 (0)	Subtract from Q-Register
SBXn	16n (0)	Subtract from Index Register n
SSA	155 (0)	Subtract Stored from A-Register
SSQ	156 (0)	Subtract Stored from Q-Register
SSXn	14n (0)	Subtract Stored from Index Register n
SWCA	171 (0)	Subtract With Carry from A-Register
SWCQ	172 (0)	Subtract With Carry from Q-Register

7.5.6 Fixed-Point Multiplication

MPF	401 (0)	Multiply Fraction
MPY	402 (0)	Multiply Integer

7.5.7 Fixed-Point Division

DIV	506 (0)	Divide Integer
DVF	507 (0)	Divide Fraction

7.5.8 Fixed-Point Comparison

Fixed-point compare instructions permit comparison of absolute values, algebraic values, or characters; provide for test of word fields; permit searches for identical, selectable word fields; and permit searches for a value within selectable limits. Comparison instructions can be repeated by using the RPT, RPD, or RPL instruction.

CMG	405 (0)	Compare Magnitude
CMK	211 (0)	Compare Masked
CMPA	115 (0)	Compare with A-Register
CMPAQ	117 (0)	Compare with AQ-Register
CMPQ	116 (0)	Compare with Q-Register
CMPX _n	10n (0)	Compare with Index Register n
CWL	111 (0)	Compare with Limits
SZN	234 (0)	Set Zero and Negative Indicators from Storage
SZNC	214 (0)	Set Zero and Negative Indicators from Storage and Clear

7.5.9 Fixed-Point Negate

NEG	531 (0)	Negate (A-Register)
NEGL	533 (0)	Negate Long (AQ-Register)

7.6 Floating-Point Instructions

This section described Floating-Point instructions.

7.6.1 Data Movement Load

DFLD	433 (0)	Double-Precision Floating Load
DFLP	532 (0)	Double-Precision Floating Load Positive
FLD	431 (0)	Floating Load
FLP	530 (0)	Floating Load Positive
LDE	411 (0)	Load Exponent Register

7.6.2 Data Movement Store

DFST	457 (0)	Double-Precision Floating Store
DFSTR	472 (0)	Double-Precision Floating Store Rounded
FST	455 (0)	Floating Store
FSTR	470 (0)	Floating Store Rounded
STE	456 (0)	Store Exponent Register

7.6.3 Floating-Point Addition

ADE	415 (0)	Add to Exponent Register
DFAD	477 (0)	Double-Precision Floating Add (Normalized)
DUFA	437 (0)	Double-Precision Floating Add (Unnormalized)
FAD	475 (0)	Floating Add (Normalized)
UFA	435 (0)	Floating Add (Unnormalized)

7.6.4 Floating-Point Subtraction

DFSB	577 (0)	Double-Precision Floating Subtract
DFSBI	467 (0)	Double-Precision Floating Subtract Inverted
DUFS	537 (0)	Double-Precision Unnormalized Floating Subtract
FSB	575 (0)	Floating Subtract
FSBI	465 (0)	Floating Subtract Inverted
UFS	535 (0)	Unnormalized Floating Subtract
UFTR	434 (0)	Unnormalized Floating Truncate Fraction

7.6.5 Floating-Point Multiplication

DFMP	463 (0)	Double-Precision Floating Multiply
DUFM	423 (0)	Double-Precision Unnormalized Floating Multiply
FMP	461 (0)	Floating Multiply
UFM	421 (0)	Unnormalized Floating Multiply

7.6.6 Floating-Point Division

DFDI	527 (0)	Double-Precision Floating Divide Inverted
DFDV	567 (0)	Double-Precision Floating Divide
FDI	525 (0)	Floating Divide Inverted
FDV	565 (0)	Floating Divide

7.6.7 Floating-Point Comparison

Floating-point compare instructions are used for single- and double-precision operations on absolute values and algebraic values. Compare instructions can be repeated by using the RPT, RPD, or RPL instruction.

DFCMG	427 (0)	Double-Precision Floating Compare Magnitude
DFCMP	517 (0)	Double-Precision Floating Compare
FCMG	425 (0)	Floating Compare Magnitude
FCMP	515 (0)	Floating Compare
FSZN	430 (0)	Floating Set Zero and Negative Indicators from Store

7.6.8 Floating-Point Negate

FNEG	513 (0)	Floating Negate
------	---------	-----------------

7.6.9 Floating-Point Normalize

FNO	573 (0)	Floating Normalize
-----	---------	--------------------

7.6.10 Floating-Point Round

DFRD	473 (0)	Double-Precision Floating Round
FRD	471 (0)	Floating Round

7.6.11 Floating-Point Truncate Fraction

FTR	474 (0)	Floating Truncate Fraction
-----	---------	----------------------------

7.7 Quadruple-Precision Instructions

The quadruple-precision instructions permit exponents to be handled as powers of 16. The AQ register and LOR register handle the mantissas, and the E register handles the exponents. The results of these operations are automatically normalized.

QFAD	476 (0)	Quadruple-Precision Floating Add
QFLD	432 (0)	Quadruple-Precision Floating Load
QFMP	462 (0)	Quadruple-Precision Floating Multiply
QFSB	576 (0)	Quadruple-Precision Floating Subtract
QFST	453 (0)	Quadruple-Precision Floating Store
QFSTR	466 (0)	Quadruple-Precision Floating Store Rounded
QSMP	460 (0)	Quadruple-Precision Floating Multiply with Double-Precision Operands

7.8 Multiword Instructions

The format and terms that are common to all multiword instructions are described below.

7.8.1 Multiword Instruction Format

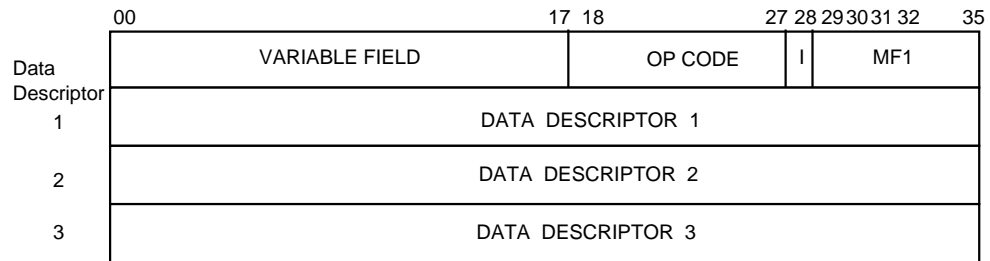


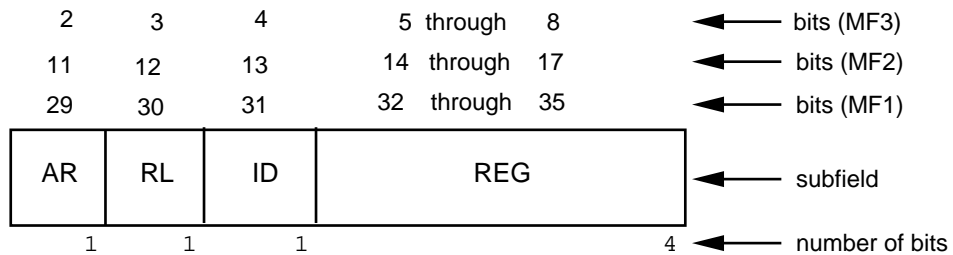
Figure 7-4. Multiword Instruction Format

Bits	Description
0-17	Contain variable information for the executed instruction function The format of this field differs with each instruction. When data descriptors 2 and 3 exist, the corresponding MF2 and MF3 are located in bits 11-17 and 1-8, respectively, of the variable field to describe the address modification executed for the data descriptors. (Refer to the individual instruction specifications in Sections 8-15.)
18-27	10-bit operation code
28	Interrupt inhibit bit
29-35	Modification field 1; describes the address modification executed for data descriptor 1

Data descriptors (2 or 3) follow the basic instruction word. The number of data descriptors is determined by each instruction. Data descriptors consist of the operand descriptor or the indirect word that points to the operand descriptor.

Multiword Modification

Each modification field (MF) contained in a multiword instruction is a 7-bit field specifying the address modification to be performed on the operand descriptors. The modification field is interpreted in the following way:



AR Address Register Specifier

- 0 no address register used
- 1 Bits 0-2 of the operand descriptor address field specify the address register to be used in computing the effective address of the operand. Bits 0-2 also specify the operand descriptor register that defines the segment containing the operand.

RL Register or Length

- 0 Operand length is specified in the N field (bits 32-35) of the operand descriptor.
- 1 The length of operand is contained in the register that is specified by code in the N field (bits 32-35) of the operand descriptor, in the machine format of REG (the coding format is different).

ID Indirect Operand Descriptor

- 0 The operand descriptor follows the instruction word in its sequential memory location.
- 1 The operand descriptor location contains an indirect word that points to the operand descriptor. Only one level of indirection is allowed.

REG Address modification register selection for R-type modification of the operand descriptor address field.

The REG codes are approximately the same as the single-word modifications. In addition, for indirect string length specification (RL = 1), the N field codes are similar to the REG field. A comparison of these codes is shown in Table 5-2.

7.8.2 Operand Descriptors And Indirect Words

The words following a multiword instruction word are either operand descriptors or indirect words to the operand descriptors. The interpretation of the words is performed according to the settings of the control bits in the associated modification field (MF).

Operand Descriptor Indirect Word Format

An indirect pointer to an operand descriptor is interpreted as shown in Figure 7-5 (also see "Indirect Word" in Section 5).

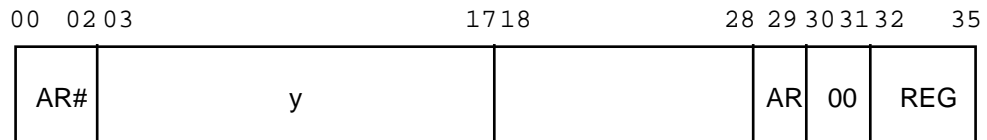


Figure 7-5. Operand Descriptor Indirect Word Format

- AR# a 3-bit pointer register number
- y an 18-bit main memory address or a 15-bit word offset
- AR indirect via bit 29 flag that controls the interpretation of the y field of the indirect pointer
- REG the address modifier for the y field

7.8.3 Alphanumeric Instructions

Alphanumeric instructions permit moving, transliterating, editing, and comparing alphanumeric data.

7.8.3.1 Alphanumeric Operand Descriptor Format

For any operand of a multiword instruction that requires alphanumeric data, the operand descriptor is interpreted as shown in Figure 7-6 (also see "Alphanumeric Operand Descriptors" in Section 5).

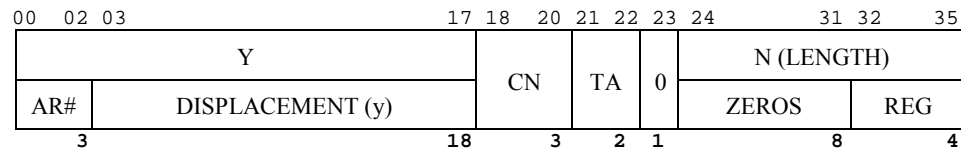


Figure 7-6. Alphanumeric Operand Descriptor Format

- AR# a 3-bit address register number
- Y location or displacement value
- DISPLACEMENT (y) an 18-bit main memory address or a 15-bit word offset relative to the address register's content
- CN character number
This field gives the character position within the word at y of the first operand character. Its interpretation depends on the data type (see TA below) of the operand. Table 7-1 shows the interpretation of the field. A digit in the table indicates the corresponding character position (see Section 2 for data formats). Invalid codes cause IPR faults.

Table 7-1. Alphanumeric Character Number (CN) Codes

C(CN)	Data Type		
	4-bit	6-bit	9-bit
000	0	0	0
001	1	1	IPR
010	2	2	1
011	3	3	IPR
100	4	4	2
101	5	5	IPR
110	6	IPR	3
111	7	IPR	IPR

- TA type alphanumeric
This is the data type code for the operand. The interpretation of the field is shown in Table 7-2. The code shown as Invalid causes an IPR fault.

Table 7-2. Alphanumeric Data Type (TA) Codes

C(TA)	Data Type
00	9-bit
01	6-bit
10	4-bit
11	IPR

N Operand length
 If RL = 0 in the corresponding MF, this field contains the string length of the operand. (Refer to Multiword Modification Field in this section.) If RL = 1, this field contains the code for a register holding the operand string length (See "Register Codes", Table 5-2).

The code for the alphanumeric operand descriptor is given below:

1	8	16	

	{ADSC9}	LOCSYM, CN, N, AM	
	{ADSC6}		(braces indicate a choice)
	{ADSC4}		

where:

- LOCSYM an expression containing either the location of the data or an offset from the base
- CN character number (see above)
- N symbol or decimal value containing either length or a register code
- AM address register containing the base

7.8.3.2 Alphanumeric Compare

CMPC	106 (1)	Compare Alphanumeric Character Strings
CMPCT	166 (1)	Compare Characters and Translate
SCD	120 (1)	Scan Characters Double
SCDR	121 (1)	Scan Characters Double in Reverse
SCM	124 (1)	Scan with Mask
SCMR	125 (1)	Scan with Mask in Reverse
TCT	164 (1)	Test Character and Translate
TCTR	165 (1)	Test Character and Translate in Reverse

7.8.3.3 Alphanumeric Move

MLR	100 (1)	Move Alphanumeric Left to Right
MRL	101 (1)	Move Alphanumeric Right to Left
MVE	020 (1)	Move Alphanumeric Edited
MVT	160 (1)	Move Alphanumeric with Translation

7.8.4 Character Move To/From Register Instructions

Two instructions permit moves of one, two, three, or four 9-bit characters from a memory location to a register or from a register to memory. An indirect word cannot be used for the data descriptor of this instruction.

7.8.4.1 Operand Descriptor for Character Move Instructions

The word following the character move instruction word is the operand descriptor which specifies the origin or destination of the move, indicates the number of characters to be moved, and specifies whether 9-bit characters or 8-bit bytes are to be moved. This word is illustrated in Figure 7-7.

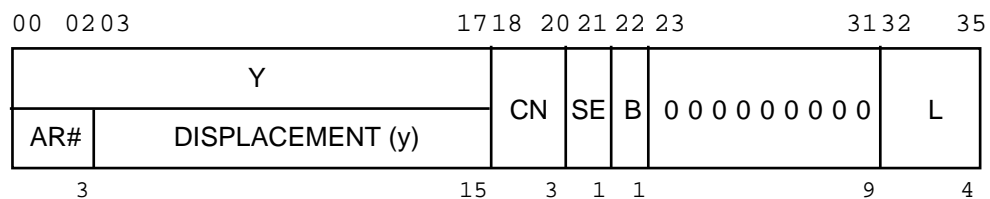


Figure 7-7. Character Move Descriptor Format

Machine Instruction Functions

The character move operand descriptor is created by entering a one-line pseudo operation coded, SDSCn, following an MTR or MTM instruction. This descriptor serves a similar purpose as operand descriptors used with other multiword instructions. SDSCn creates a descriptor word to transfer 9-bit characters or 8-bit bytes for the MTR/MTM instruction, depending upon the specification in n as described below.

1	8	16	

SDSCn		LOCSYM, CN, L, SE, AM	

where:

n		when = 9, B (see descriptor format above) is set to 0 indicating 9-bit characters when = 8, B is set to 1 indicating 8-bit bytes
LOCSYM		address of word containing first character to be moved
CN		character position of left end of operand within a word must be 0-3
L		number of characters to be moved; must be 0-4; defaults to 0
SE		sign extend
AM		optional address register modification (AR#)

NOTE: Refer to the specifications for MTR and MTM in Section 11.

The method of generating a start address for a character move by using the Y field is the same as in other multiword instructions. However, A, Q, X0-X7 or GX0-GX7 must be specified for REG modification.

7.8.4.2 Character Move Instruction Repertoire

MTM	365 (1)	Move to Memory
MTR	361 (1)	Move to Register

7.8.5 Numeric Instructions

The set of numeric instructions deals with sign and magnitude operands. Floating-point decimal zero is represented as $+ 0 * 10^{127}$. If any computation is performed that would result in a zero representation other than this, the hardware forces the zero representation to this format, thus preventing loss of data during decimal point alignment.

All numeric operations are limited to final results not to exceed 63 characters (sign, digits, exponent). If any numeric move, compare, or calculation is specified involving either a number with more than 63 characters or a final product with more than 63 characters, the operation is performed as though 63 characters were specified. No fault occurs unless the specific description of an instruction states that such a fault occurs and/or that operation does not take place.

All characters are carried internally as 4 bits. The upper 5 bits of any 9-bit input character (TN = 0) are truncated. If a 9-bit output is specified, 00011 (ASCII numeric zone) is appended to form the numeric digits. Standard ASCII plus minus characters (octal 053 and 055, respectively) are generated.

7.8.5.1 Numeric Operand Descriptor Format

For any operand of a multiword instruction that requires numeric data, the operand descriptor is interpreted as shown in Figure 7-8 (see also "Numeric Operand Descriptors" in Section 5).

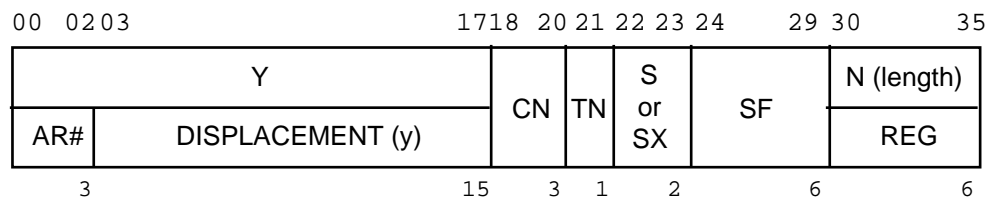


Figure 7-8. Numeric Operand Descriptor Format

- AR# a 3-bit address register number
- Y location or displacement value
- DISPLACEMENT (y) an 18-bit main memory address or a 15-bit word offset relative to the address register's content
- CN character number
This field gives the character position within the word at y of the first operand digit. Its interpretation depends on the data type (see TN below) of the operand.

Machine Instruction Functions

TN	Type Numeric This is the data type code for the operand. The codes are: C(T) Data Type 0 9-bit 1 4-bit
S	Sign and Decimal Type of Data The interpretation of the field is: C(S) Sign and Decimal Type 00 Floating point, leading sign 01 Scaled fixed point, leading sign 10 Scaled fixed point, trailing sign 11 Scaled fixed point, unsigned
SX	Sign and Scaling If TN = 0 (unpacked data) 00 leading sign, overpunched, fixed-point 01 leading sign, separate, fixed-point 10 trailing sign, separate, fixed-point 11 trailing sign, overpunched, fixed-point If TN = 1 (packed data) 00 leading sign, separate, floating-point 01 leading sign, separate, fixed-point 10 trailing sign, separate, fixed-point 11 no sign, fixed-point (Refer to description of overpunched signs under MVNX in Section 8.)
SF	Scaling Factor This field contains the twos complement value of the base 10 scaling factor(i.e., the value of <u>m</u> for numbers represented as <u>n</u> * 10** <u>m</u>). The decimal point is assumed to the right of the least significant digit of <u>n</u> . Negative values of <u>m</u> move the decimal point to the left; positive values, to the right. The range of <u>m</u> is -32 to 31 treated as the powers of 10.
N	Operand Length If RL = 0 in MF, this field contains the operand length in digits. If RL = 1, it contains the REG code for the register holding the operand length and C(REG) is treated as a 0 modulo 64 number.

The numeric operand descriptor is coded in the following way.

1	8	16

	{NDSC9}	LOCSYM, CN, N, S, SF, AM
	{NDSC4}	

where:

LOCSYM	an expression containing either the location of the data or an offset from the base										
CN	character number (see above)										
N	a symbol or decimal value containing either the length for a register code										
S	the sign and decimal type in two bits:										
	<table> <thead> <tr> <th style="text-align: left;"><u>Code</u></th> <th style="text-align: left;"><u>Description</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Floating-point, leading sign</td> </tr> <tr> <td>1</td> <td>Scaled fixed-point, leading sign</td> </tr> <tr> <td>2</td> <td>Scaled fixed-point, trailing sign</td> </tr> <tr> <td>3</td> <td>Scaled fixed-point, unsigned</td> </tr> </tbody> </table>	<u>Code</u>	<u>Description</u>	0	Floating-point, leading sign	1	Scaled fixed-point, leading sign	2	Scaled fixed-point, trailing sign	3	Scaled fixed-point, unsigned
<u>Code</u>	<u>Description</u>										
0	Floating-point, leading sign										
1	Scaled fixed-point, leading sign										
2	Scaled fixed-point, trailing sign										
3	Scaled fixed-point, unsigned										
SX	sign and scaling (see above)										
SF	the scaling factor for scaled decimal numbers; range is -31 to +32 treated as the powers of 10										
AM	address register containing the base (AR#)										

7.8.5.2 Numeric Compare

CMPN	303 (1)	Compare Numeric
CMPNX	343 (1)	Compare Numeric Extended

7.8.5.3 Numeric Move

MVN	300 (1)	Move Numeric
MVNX	340 (1)	Move Numeric Extended
MVNE	024 (1)	Move Numeric Edited
MVNEX	004 (1)	Move Numeric Edited Extended

7.8.6 Bit String Instructions

These instructions provide the capability of performing Boolean operations on bit strings. The Boolean Result (BOLR) control field (bits 5, 6, 7, and 8 of the instruction word) defines one of 16 possible logical operations to be performed. The four bits in this field are associated with the four possible combinations of bits from the two operands. The association rule is given below.

If first operand bit is:	and	second operand bit is:	then result is from bit:
0		0	5
0		1	6
1		0	7
1		1	8

The Boolean operations most commonly used are

Operation	BOLR Field Bits			
	5	6	7	8
MOVE	0	0	1	1
AND	0	0	0	1
OR	0	1	1	1
NAND	1	1	1	0
EXCLUSIVE OR	0	1	1	0
Clear	0	0	0	0
Invert	1	1	0	0

The four bits contained in the Boolean control field are represented in the instruction format by one or two octal digits.

7.8.6.1 Bit String Operand Descriptor Format

For any operand of a multiword instruction that requires bit string data, the operand descriptor is interpreted as shown in Figure 7-9 (see "Bit String Operand Descriptor" in Section 5).

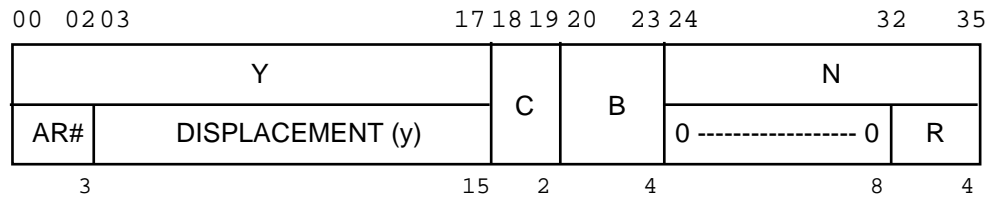
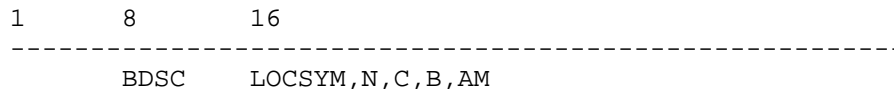


Figure 7-9. Bit String Operand Descriptor Format

- AR# a 3-bit address register number
- Y location or displacement value
- DISPLACEMENT (Y), an 18-bit main memory address or a 15-bit word offset relative to the address register's content
- C the character number of the 9-bit character within the y field containing the first bit of the operand
- B the bit number within the 9-bit character, C, of the first bit of the operand
- N operand length
If RL = 0 in MF, this field contains the string length of the operand. If RL = 1, this field contains the code for a register holding the operand string length.
- R register containing data length

The bit string operand descriptor is coded in the following way.



where:

- LOCSYM an expression containing either the location of the data or an offset from the base
- N symbol or decimal value containing either length or a register code
- C character position (0-3)
- B bit within character (0-8)
- AM address register containing the base (AR#)

Machine Instruction Functions

7.8.6.2 Bit String Combine

CSL	060 (1)	Combine Bit Strings Left
CSR	061 (1)	Combine Bit Strings Right

7.8.6.3 Bit String Compare

CMPB	066 (1)	Compare Bit String
------	---------	--------------------

7.8.6.4 Bit String Set Indicators

SZTL	064 (1)	Set Zero and Truncation Indicators with Bit Strings Left
SZTR	065 (1)	Set Zero and Truncation Indicators with Bit Strings Right

7.8.7 Data Conversion Instructions

Conversion instructions are used for conversions between binary and decimal numbers where the binary number is stored as a character string, starting and ending on 9-bit character boundaries, and the decimal number is stored as a character string.

BTD	301 (1)	Binary-to-Decimal Convert
DTB	305 (1)	Decimal-to-Binary Convert

7.8.8 Arithmetic Instructions

This section describes the Arithmetic instructions.

7.8.8.1 Decimal Addition

AD2D	202 (1)	Add Using Two Decimal Operands
AD2DX	242 (1)	Add Using Two Decimal Operands Extended
AD3D	222 (1)	Add Using Three Decimal Operands
AD3DX	262 (1)	Add Using Three Decimal Operands Extended

7.8.8.2 Decimal Subtraction

SB2D	203 (1)	Subtract Using Two Decimal Operands
SB2DX	243 (1)	Subtract Using Two Decimal Operands Extended
SB3D	223 (1)	Subtract Using Three Decimal Operands
SB3DX	263 (1)	Subtract Using Three Decimal Operands Extended

7.8.8.3 Decimal Multiplication

MP2D	206 (1)	Multiply Using Two Decimal Operands
MP2DX	246 (1)	Multiply Using Two Decimal Operands Extended
MP3D	226 (1)	Multiply Using Three Decimal Operands
MP3DX	266 (1)	Multiply Using Three Decimal Operands Extended

7.8.8.4 Decimal Division

DV2D	207 (1)	Divide Using Two Decimal Operands
DV2DX	247 (1)	Divide Using Two Decimal Operands Extended
DV3D	227 (1)	Divide Using Three Decimal Operands
DV3DX	267 (1)	Divide Using Three Decimal Operands Extended

7.9 Micro Operations for Edit Instructions MVE, MVNE, MVNEX

The Move Alphanumeric Edited (MVE), Move Numeric Edited (MVNE), and Move Numeric Edited Extended (MVNEX) instructions require micro operations to perform the editing functions in an efficient manner. The sequence of micro operation steps to be executed is contained in memory and is referenced by the second operand descriptor of the instruction. Some of the micro operations require special characters for insertion into the string of characters being edited. These special characters are shown in the edit insertion tables in this section.

7.9.1 Micro Operation Sequence

The micro operation string operand descriptor points to a string of 9-bit bytes that specifies the micro operations to be performed during an edited move. Each of the 9-bit bytes defines a micro operation and has the format shown in Figure 7-10.

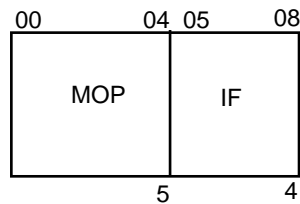


Figure 7-10. Micro Operation (MOP) Character Format

MOP	5-bit code specifying the micro operator (Refer to the Micro Operation Repertoire.)
IF	is an information field containing one of the following items: <ul style="list-style-type: none"> – A sending string character count. A value of 0 is interpreted as 16; – The index of an entry in the edit insertion table to be used; permissible values are 1 through 8; – An interpretation of the "blank-when-zero" operation.

7.9.2 Edit Insertion Tables

While executing an edit instruction, the processor provides a register of eight 9-bit bytes to hold insertion information. This register, called the edit insertion table, is not maintained after execution of an edit instruction. At the start of each edit instruction, the processor initializes the table to the values given in Table 7-3. For MVE and MVNE, the ASCII code is used for each initial value. For MVNEX, the BIT field in the instruction word determines the character set (ASCII, BCD, or EBCDIC) to be used for the initial values. (Refer to the Edit Insertion Table Entries in Table 7-4.)

Table 7-3. Default Edit Insertion Table Characters for MVE and MVNX

Table Entry Number	Character
1	Space
2	*
3	+
4	-
5	\$
6	,
7	.
8	0 (zero)

The relationship between the ASCII character bit positions and the table character positions is given below.

```

0 1 2 3 4 5 6 7 8   Table character bit positions
-----
9 8 7 6 5 4 3 2 1   ASCII character bit positions
    
```

where unused high-order bit positions of the character are zero-filled

One or all of the table entries may be changed by the Load Table Entry (LTE) or the Change Table (CHT) micro operation to provide different insertion characters.

Table 7-4. Edit Insertion Table Entries for MVNEX

Edit Insertion Table		Octal Code		
No.	Character	EBCDIC	BCD	ASCII
1	␣ (space)	100	020	040
2	* (asterisk)	134	054	052
3	+ (plus)	116	060	053
4	- (minus)	140	052	055
5	\$ (dollar sign)	133	053	044
6	, (comma)	153	073	054
7	. (period)	113	033	056
8	0 (zero)	360	000	060

7.9.3 MVNE, MVE, And MVNEX Differences

The processor executes MVNE and MVNEX in a slightly different manner from the way it executes MVE because of the inherent differences in how numeric and alphanumeric data are handled.

7.9.3.1 Numeric Edit (MVNE and MVNEX)

1. Load the entire sending string number (maximum length 63 characters) into the decimal unit input buffer as 4-bit digits (high-order truncating 9-bit data). Strip the sign and exponent characters (if any), put them aside into special holding registers, and decrease the input buffer count accordingly.
2. Test sign and, if required, set the SN flag.
3. Execute micro operation string, starting with the first (4-bit) digit.
4. If an edit insertion table entry or MOP insertion character is to be stored, ANDed, or ORed into a receiving string of 4- or 6-bit characters, high-order truncate the character accordingly.
5. If the receiving string is 9-bit characters, high-order fill the (4-bit) digits from the input buffer with bits 0-4 of character 8 of the edit insertion table. If the receiving string is 6-bit characters, high-order fill the digits with "00".

7.9.3.2 Alphanumeric Edit (MVE)

1. Load the decimal unit input buffer with sending string characters. Data is read from memory in unaligned units (not modulo 8 boundary) of four double-words. The number of characters loaded is the minimum of the remaining sending string count, the remaining receiving string count, and 63.
2. Perform tests for zero on the four least significant bits of each character.
3. Execute micro operation string, starting with the first receiving string character.
4. If an edit insertion table entry or MOP insertion character is to be stored, ANDed, or ORed into a receiving string of 4- or 6-bit characters, use the lower 4 or 6 bits.
5. If the receiving string is 6- or 9-bit characters, the zero-fill is already supplied; do not append bits of any edit insertion table entry as the most significant bits.

7.9.4 Micro Operation Repertoire

MOP	Octal	Binary	Operation
CHT	21	10001	Change Table
ENF	02	00010	End Floating Suppression
IGN	14	01100	Ignore Source Characters
INSA	11	01001	Insert Asterisk on Suppression
INSB	10	01000	Insert Blank on Suppression
INSM	01	00001	Insert Table Entry One Multiple
INSN	12	01010	Insert on Negative
INSP	13	01011	Insert on Positive
LTE	20	10000	Load Table Entry
MFLC	07	00111	Move with Floating Currency Symbol Insertion
MFLS	06	00110	Move with Floating Sign Insertion
MORS	17	01111	Move and OR Sign
MSES	16	01110	Move and Set Sign
MVC	15	01101	Move Source Characters
MVZA	05	00101	Move with Zero Suppression and Asterisk Replacement
MVZB	04	00100	Move with Zero Suppression and Blank Replacement
SES	03	00011	Set End Suppression

7.9.5 Micro Operations Descriptions

The 17 micro operations (MOPs) are described in this subsection. The descriptions are presented in the format shown below.

MOP	Operation	Binary Code
-----	-----------	-------------

EXPLANATION:

Describes how the operation functions

FLAGS:

Describe the setting of the affected flags

NOTES:

Describe any fault conditions

Checks for termination are made during and after each micro operation. All MOPs that make a zero test of a sending-string character test only the four least-significant bits of the character.

Edit Flags

The processor provides the following four edit flags for use by the micro operations.

- ES End Suppression flag; initially OFF, set ON by a micro operation when zero-suppression ends. (This ES should not be confused with ES mode.)
- SN Sign flag; initially set OFF if the sending string has an alphanumeric descriptor or an unsigned numeric descriptor. If the sending string has a signed numeric descriptor, the sign is initially read from the sending string from the digit position defined by the sign and the decimal type field (S or SX); SN is set OFF if positive, ON if negative. If all digits are zero, the data is assumed positive and the SN flag is set OFF, even when the sign is negative.
- Z Zero flag; initially set ON and set OFF whenever a sending string character that is not decimal zero is moved into the receiving string.
- BZ Blank-when-zero flag; initially set OFF and set ON by either the ENF or SES micro operation. If, at the completion of a move (L1 exhausted), both the Z and BZ flags are ON, the receiving string is filled with character 1 of the edit insertion table.

7.9.5.1 CHT Micro Operation

CHT	Change Table	10001
-----	--------------	-------

EXPLANATION:

The edit insertion table is replaced by the string of eight 9-bit characters immediately following the CHT micro operation.

FLAGS:

None affected

NOTE:

C(IF) is not interpreted for this operation.

7.9.5.2 ENF Micro Operation

ENF	End Floating Suppression	00010
-----	--------------------------	-------

EXPLANATION:

Bit 0 of IF, IF(0), specifies the nature of the floating suppression.

Bit 1 of IF, IF(1), specifies if blank when zero option is used.

For IF(0) = 0 (end floating-sign operation),

- If ES is OFF and SN is OFF, then edit insertion table entry 3 is moved to the receiving field and ES is set ON.
- If ES is OFF and SN is ON, then edit insertion table entry 4 is moved to the receiving field and ES is set ON.
- If ES is ON, no action is taken.

For IF(0) = 1 (end floating currency symbol operation),

- If ES is OFF, then edit insertion table entry 5 is moved to the receiving field and ES is set ON.
- If ES is ON, no action is taken.

For IF(1) = 1 (blank when zero): the BZ flag is set ON.

For IF(1) = 0 (no blank when zero): no action is taken.

FLAGS: (Flags not listed are not affected)

ES - If OFF, then set ON

BZ - If bit 1 of C(IF) = 1, then set ON; otherwise, unchanged

7.9.5.3 IGN Micro Operation

IGN	Ignore Source Characters	01100
-----	--------------------------	-------

EXPLANATION:

IF specifies the number of characters to be ignored, where IF = 0 specifies 16 characters.

The next IF characters in the source data field are ignored and the sending tally is reduced accordingly.

FLAGS:

None affected

Machine Instruction Functions

7.9.5.4 INSA Micro Operation

INSA	Insert Asterisk on Suppression	01001
------	--------------------------------	-------

EXPLANATION:

This MOP is the same as INSB except that if ES is OFF, then edit insertion table entry 2 is moved to the receiving field.

FLAGS:

None affected

NOTE:

If C(IF) = 9-15, an IPR fault occurs.

7.9.5.5 INSB Micro Operation

INSB	Insert Blank on Suppression	01000
------	-----------------------------	-------

EXPLANATION:

IF specifies which edit insertion table entry is inserted.

If IF = 0, the 9 bits immediately following the INSB micro operation are treated as a 9-bit character (not a MOP) and are moved or skipped according to ES.

- If ES is OFF, then edit insertion table entry 1 is moved to the receiving field. If IF = 0, then the next 9 bits are also skipped. If IF is not 0, the next 9 bits are treated as a MOP.
- If ES is ON and IF = 0, then the 9-bit character immediately following the INSB micro-instruction is moved to the receiving field.
- If ES is ON and IF > 0, then IF specifies which edit insertion table entry (1-8) is to be moved to the receiving field.

FLAGS:

None affected

NOTE:

If C(IF) = 9-15, an IPR fault occurs.

Machine Instruction Functions

7.9.5.6 INSM Micro Operation

INSM	Insert Table Entry One Multiple	00001
------	---------------------------------	-------

EXPLANATION:

IF specifies the number of receiving characters affected, where IF = 0 specifies 16 characters.

Edit insertion table entry 1 is moved to the next IF (1-16) receiving field characters.

FLAGS:

None affected

7.9.5.7 INSN Micro Operation

INSN	Insert on Negative	01010
------	--------------------	-------

EXPLANATION:

IF specifies which edit insertion table entry is inserted. If IF = 0, the 9 bits immediately following the INSN micro operation are treated as a 9-bit character (not a MOP) and are moved or skipped according to SN.

- If SN is OFF, then edit insertion table entry 1 is moved to the receiving field. If IF = 0, then the next 9 bits are also skipped. If IF is not 0, the next 9 bits are treated as a MOP.
- If SN is ON and IF = 0, then the 9-bit character immediately following the INSN micro-instruction is moved to the receiving field.
- If SN is ON and IF \neq 0, then IF specifies which edit insertion table entry (1-8) is to be moved to the receiving field.

FLAGS:

None affected

NOTE:

If C(IF) = 9-15, an IPR fault occurs.

Machine Instruction Functions

7.9.5.8 INSP Micro Operation

INSP	Insert on Positive	01011
------	--------------------	-------

EXPLANATION:

INSP is the same as INSN except that the responses for the SN values are reversed.

FLAGS:

None affected

NOTE:

If C(IF) = 9-15, an IPR fault occurs.

7.9.5.9 LTE Micro Operation

LTE	Load Table Entry	10000
-----	------------------	-------

EXPLANATION:

IF specifies the edit insertion table entry to be replaced.

The edit insertion table entry specified by IF is replaced by the 9-bit character immediately following the LTE microinstruction.

FLAGS:

None affected

NOTE:

If $C(IF) = 0$ or $C(IF) = 9-15$, an Illegal Procedure fault occurs.

7.9.5.10 MFLC Micro Operation

MFLC	Move with Floating Currency Symbol Insertion	00111
------	--	-------

EXPLANATION:

IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur.

- If ES is OFF and the character is zero, edit insertion table entry 1 is moved to the receiving field in place of the character.
- If ES is OFF and the character is not zero, then edit insertion table entry 5 is moved to the receiving field, the character is also moved to the receiving field, and ES is set ON.
- If ES is ON, the character is moved to the receiving field.

The number of characters placed in the receiving field is data-dependent. If the entire sending field is zero, IF characters are placed in the receiving field. However, if the sending field contains a nonzero character, IF+1 characters (the insertion character plus the characters from the sending field) are placed in the receiving field.

An IPR fault occurs when the sending field is exhausted before the receiving field is filled. In order to provide space in the receiving field for an inserted currency symbol, the receiving field must have a string length one character longer than the sending field. When the sending field is all zeros, no currency symbol is inserted by the MFLC micro operation and the receiving field is not filled when the sending field is exhausted. The user should provide an ENF (ENF,12) micro operation after a MFLC micro operation that has as its character count the number of characters in the sending field. The ENF micro operation is engaged only when the MFLC micro operation fails to fill the receiving field. Then it supplies a currency symbol to fill the receiving field and blanks out the entire field.

FLAGS: (Flags not listed are not affected.)

ES If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, it is unchanged.

NOTE:

Since the number of characters moved to the receiving string is data-dependent, a possible IPR fault may be avoided by ensuring that the Z and BZ flags are ON.

7.9.5.11 MFLS Micro Operation

MFLS	Move with Floating Sign Insertion	00110
------	-----------------------------------	-------

EXPLANATION:

IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur.

- If ES is OFF and the character is zero, edit insertion table entry 1 is moved to the receiving field in place of the character.
- If ES is OFF, the character is not zero, and SN is OFF; then edit insertion table entry 3 is moved to the receiving field; the character is also moved to the receiving field, and ES is set ON.
- If ES is OFF, the character is nonzero, and SN is ON; edit insertion table entry 4 is moved to the receiving field; the character is also moved to the receiving field, and ES is set ON.
- If ES is ON, the character is moved to the receiving field.

The number of characters placed in the receiving field is data-dependent. If the entire sending field is zero, IF characters are placed in the receiving field. However, if the sending field contains a nonzero character, IF+1 characters (the insertion character plus the characters from the sending field) are placed in the receiving field.

An IPR fault occurs when the sending field is exhausted before the receiving field is filled. In order to provide space in the receiving field for an inserted sign, the receiving field must have a string length one character longer than the sending field. When the sending field is all zeros, no sign is inserted by the MFLS micro operation and the receiving field is not filled when the sending field is exhausted. The user should provide an ENF (ENF,4) micro operation after a MFLS micro operation that has as its character count the number of characters in the sending field. The ENF micro operation is engaged only when the MFLS micro operation fails to fill the receiving field; then, it supplies a sign character to fill the receiving field and blanks out the entire field.

Machine Instruction Functions

FLAGS: (Flags not listed are not affected.)

ES If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, it is unchanged.

NOTE:

Since the number of characters moved to the receiving string is data-dependent, a possible Illegal Procedure fault may be avoided by ensuring that the Z and BZ flags are ON.

7.9.5.12 MORS Micro Operation

MORS	Move and OR Sign	01111
------	------------------	-------

EXPLANATION:

IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur.

- If SN is OFF, the next IF characters in the source data field are moved to the receiving data field and, during the move, edit insertion table entry 3 is ORed to each character.
- If SN is ON, the next IF characters in the source data field are moved to the receiving data field and, during the move, edit insertion table entry 4 is ORed to each character.

MORS can be used to generate a negative overpunch for a receiving field to be used later as a sending field.

FLAGS:

None affected

7.9.5.13 MSES Micro Operation

MSES	Move and Set Sign	01110
------	-------------------	-------

EXPLANATION:

IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters. For MVE, starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur.

Starting with the first character during the move, a comparative AND is made first with edit insertion table entry 3. If the result is nonzero, the first character and the rest of the characters are moved without further comparative ANDs. If the result is zero, a comparative AND is made between the character being moved and edit insertion table entry 4. If that result is nonzero, the SN indicator is set ON (indicating negative) and the first character and the rest of the characters are moved without further comparative ANDs. If the result is zero, the second character is treated like the first. This process continues until one of the comparative AND results is nonzero or until all characters are moved.

For MVNE and MVNEX instructions, the sign (SN) flag is already set and IF characters are moved to the destination field (MSES is equivalent to the MVC instruction).

FLAGS: (Flags not listed are not affected.)

SN If edit insertion table entry 4 is found in C(Y-1), then ON; otherwise, it is unchanged.

7.9.5.14 MVC Micro Operation

MVC	Move Source Characters	01101
-----	------------------------	-------

EXPLANATION:

IF specifies the number of characters to be moved, where IF = 0 specifies 16 characters.

The next IF characters in the source data field are moved to the receiving data field.

FLAGS:

None affected

Machine Instruction Functions

7.9.5.15 MVZA Micro Operation

MVZA	Move with Zero Suppression and Asterisk Replacement	00101
------	---	-------

EXPLANATION:

MVZA is the same as MVZB except that if ES is OFF and the character is zero, then edit insertion table entry 2 is moved to the receiving field.

FLAGS: (Flags not listed are not affected.)

ES If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, it is unchanged.

7.9.5.16 MVZB Micro Operation

MVZB	Move with Zero Suppression and Blank Replacement	00100
------	--	-------

EXPLANATION:

IF specifies the number of characters of the sending field upon which the operation is performed, where IF = 0 specifies 16 characters.

Starting with the next available sending field character, the next IF characters are individually fetched and the following conditional actions occur.

- If ES is OFF and the character is zero, then edit insertion table entry 1 is moved to the receiving field in place of the character.
- If ES is OFF and the character is not zero, then the character is moved to the receiving field and ES is set ON.
- If ES is ON, the character is moved to the receiving field.

FLAGS: (Flags not listed are not affected.)

ES If OFF and any of C(Y) is less than decimal zero, then ON; otherwise, it is unchanged.

Machine Instruction Functions

7.9.5.17 SES Micro Operation

SES	Set End Suppression	00011
-----	---------------------	-------

EXPLANATION:

Bit 0 of IF (IF(0)) specifies the setting of the ES switch.

If IF(0) = 0, the ES flag is set OFF.

If IF(0) = 1, the ES flag is set ON.

Bit 1 of IF (IF(1)) specifies the setting of the blank-when-zero option.

If IF(1) = 0, no action is taken.

If IF(1) = 1, the BZ flag is set ON.

FLAGS: (Flags not listed are not affected.)

ES set by this micro operation

BZ If bit 1 of C(IF) = 1, then ON; otherwise, it is unchanged.

7.9.6 Micro Operation Code Assignment Map

Operation code assignments for the micro operations are shown in Table 7-5. Dashes (----) indicate an unassigned code. All unassigned codes cause an Illegal Procedure fault.

Table 7-5. Micro Operation Code Assignment Map

B0	B1	B2 B3 B4							
		000	001	010	011	100	101	110	111
00		----	INSM	ENF	SES	MVZB	MVZA	MFLS	MFLC
01		INSB	INSA	INSN	INSP	IGN	MVC	MSES	MORS
10		LTE	CHT	----	----	----	----	----	----
11		----	----	----	----	----	----	----	----

7.9.7 Terminating Micro Operations

The micro-operation sequence is terminated normally when the receiving string length is exhausted. The micro-operation sequence is terminated abnormally (with an IPR fault) if an attempt is made to move from an exhausted sending string or to use an exhausted MOP string.

Micro Operations Examples

1	8	16	32

	MVNE		
	NDSC4	EPACK, 5, 11, 2	PIC S9(10)
	ADSC9	MOPLST, 0, 9	
	ADSC6	PRTOUT+3, 0, 12	PIC Z(7).999-
	USE	DETOUR	
MOPLST	MICROP	(LTE, 1), 1H, (MVZB, 7), (SES, 8)	
	MICROP	(INSB), 1H., (MVC, 3), (INSN)	
	MICROP	1H-, (LTE, 1), 1H, (MVZB, 2), (MVC, 1)	
	USE		
	MVNE		
	NDSC4	FPAK, 5, 11, 2	PIC S9(10)
	ADSC9	MOPLST, 0, 9	
	ADSC6	PRTOUT+6, 0, 12	PIC Z(7).999-
	MVNE		
	NDSC4	SEQPAK, 5, 3, 3	PIC 999
	ADSC9	MOPLST+2, 1, 4	
	ADSC6	PRTOUT+1, 3, 3	PIC ZZ9

7.10 Virtual Memory Instructions

These instructions support segmentation and paging in the virtual memory environment. Except in the case of the CLIMB instruction, the format of these instructions is the same as the other single-word instructions.

7.10.1 Descriptor Register Instructions

These instructions provide the capabilities of loading or storing a descriptor register (DR_n) with a new descriptor or modifying the descriptor currently contained in DR_n . The LDDn instruction has a direct load option.

LDDn	67n (1)	Load Descriptor Register n
SDRn	11n (1)	Save Descriptor Register n
STDn	05n (1)	Store Descriptor Register n

7.10.2 Domain Transfer (CLIMB)

The CLIMB domain transfer instruction provides the software with a hardware mechanism for transferring control from one software function to another with a high level of software security. This 2-word instruction, described in detail in Section 8, has four versions that perform the functions of call, return, and co-routine invocations for intra- and inter-instruction segments and intra- and inter-domain references.

CLIMB	713 (1)	Domain Transfer
-------	---------	-----------------

7.10.3 Privileged Instructions

Privileged instructions are executed in Privileged Master Mode. Three conditions must be met before the instructions can be executed:

1. the master mode bit in the indicator register must be ON;
2. the privileged bit in the instruction segment register must be ON; and
3. the housekeeping bit in the page table word for the page containing the instruction must be ON.

NOTE: If the processor is in the working space zero addressing mode, this bit is assumed to be ON.

If any of the above conditions does not exist when execution of a privileged instruction is attempted, a Command fault occurs.

7.10.3.1 Clear Associative Memory

CAMP	532 (1)	Clear Associative Memory Pages
------	---------	--------------------------------

7.10.3.2 Diagnostic

DIAG	612 (0)	Communication between GCOS and the Service Processor (SP)
------	---------	---

7.10.3.3 Register Load

LDAS	770 (1)	Load Argument Stack Register
LDDSA	170 (1)	Load Data Stack Address Register
LDDSD	571 (1)	Load Data Stack Descriptor Register
LDPS	771 (1)	Load Parameter Segment Register
LDSS	773 (1)	Load Safe Store Register
LDWS	772 (1)	Load Working Space Registers
LLPNR	364 (1)	Load Logical Processor Number Register
LPDBR	171 (1)	Load Page Table Directory Base Register

7.10.3.4 Register Store

RLPNR	366 (1)	Read Logical Processor Number Register
SFR	452 (0)	Store Fault Register
SICHR	154 (1)	Store IC History Register
SPDBR	151 (1)	Store Page Table Directory Base Register
STAS	750 (1)	Store Argument Stack Register
STDSA	150 (1)	Store Data Stack Address Register
STDSD	551 (1)	Store Data Stack Descriptor Register
STPS	751 (1)	Store Parameter Segment Register
STSS	753 (1)	Store Safe Store Register
STWS	752 (1)	Store Working Space Registers

7.10.3.5 Memory Control

LIMR	553 (0)	Load Interrupt Mask Register
RIMR	233 (0)	Read Interrupt Mask Register

Machine Instruction Functions

7.10.3.6 System Control

CIOC	015 (0)	Connect Input/Output Channel
DIS	616 (0)	Delay Until Interrupt Signal
EPAT	412 (1)	Effective Pointer and Address to Test
LCCL	057 (0)	Load Calendar Clock
LDT	637 (0)	Load Timer Register
LLUF	674 (0)	Load Lockup Fault Register
PAS	176 (1)	Pop Argument Stack
RICHR	156 (1)	Restart IC History Register
RRES	231 (0)	Read Reserve Memory
WRES	232(0)	Write Reserve Memory

7.10.3.7 Pointer Register Instructions

LDPn	47n (1)	Load Pointer Register n
STPn	45n (1)	Store Pointer n
EPPRn	63n (1)	Effective Pointer to Pointer Register n
LDEAn	61n (1)	Load Extended Address n

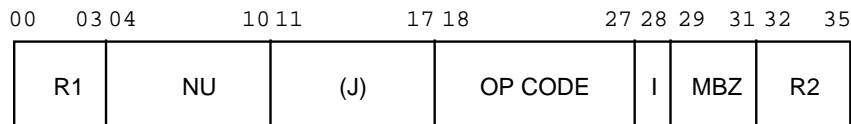
7.11 ES And EI Mode Instructions

ES and EI mode instructions are valid only in the ES/EI modes (ISR bit 24=1). AN IPR fault occurs if an attempt is made to execute these instructions in the NS mode. Except for the AARn, NARn, ARAn, and ARN instructions, all instructions are valid in the ES/EI modes. An IPR fault occurs if an attempt is made to execute these four instructions in the ES/EI modes.

7.11.1 Register to Register Instructions

Register to Register instructions known as "RR" type instructions are valid only in the ES/EI modes. An attempt to execute these instructions in the NS mode results in an IPR fault. RR type instructions permit movement, arithmetic operation, and shift of fixed-point data using the GXn, A and Q registers. An attempt to execute any RR type instruction by the RPT, RPD, or RPL instructions results in an IPR fault.

7.11.1.1 RR Type Instruction



<u>Bits</u>	<u>Field</u>	<u>Description</u>
0 – 3	R1	specifies a code indicating a register to be the destination of the result. The allowable codes are listed below.

<u>Register Code</u>	<u>Result</u>
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A
0110	Q
0111	IPR
1000	GX0
1001	GX1
1010	GX2
1011	GX3
1100	GX4
1101	GX5
1110	GX6
1111	GX7

Machine Instruction Functions

<u>Bits</u>	<u>Field</u>	<u>Description</u>
4 – 10	NU	not used - should be set to 0
11 – 17	J	used only in a shift instruction--specifies the shift number (immediate value)--must be 0 in all but shift instructions
18 – 27	OP CODE	operation code
28	I	interrupt inhibit bit
29 – 31	MBZ	must be zero or an IPR fault occurs
32 – 35	R2	specifies a code that indicates a source register The codes for this register are the same as for R1.

- NOTES:**
1. Specifying a register code of 0000 in a shift instruction does not result in an IPR fault.
 2. If a register pair appears in an instruction specification, the two registers are handled as linked. The list below indicates the register codes to be associated with the register pair.

<u>Register Code</u>	<u>Result</u>
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A, Q
0110	A, Q
0111	IPR
100x	GX0, GX1
101x	GX2, GX3
110x	GX4, GX5
111x	GX6, GX7

where x means this bit is ignored by the hardware

7.11.1.2 Movement and Arithmetic Instructions

ADLR	435 (1)	Add Logical to Register
ADRR	434 (1)	Add Register to Register
ANRR	535 (1)	AND Register to Register
CMRR	534 (1)	Compare Register to Register
DVRR	533 (1)	Divide Register to Register
ERRR	537 (1)	Exclusive OR Register to Register
LDCR	431 (1)	Load Complement to Register
LDDR	433 (1)	Load Double Register to Register
LDPR	432 (1)	Load Positive Register to Register
LDRR	430 (1)	Load Register to Register
MPRR	530 (1)	Multiply Register-Pair to Register
MPRS	531 (1)	Multiply Register-Single to Register
ORRR	536 (1)	OR Register to Register
SBLR	437 (1)	Subtract Logical to Register
SBRR	436 (1)	Subtract Register to Register

7.11.1.3 Shift Instructions

GLLS	466 (1)	GXn Long Left Shift
GLRL	465 (1)	GXn Long Right Logic
GLRS	464 (1)	GXn Long Right Shift
GLS	462 (1)	GXn Left Shift
GRL	461 (1)	GXn Right Logic
GRS	460 (1)	GXn Right Shift

7.11.2 Fixed-Point Instructions

The fixed-point instructions concern movement and arithmetic operations on data in the GXn registers and memory. These instructions are valid only in the ES/EI mode. An attempt to execute these instructions in the NS mode results in an IPR fault.

GLDD	32n (1)	Load Double to GXn (n = 0,2,4,6)
GSTD	14n (1)	Store Double from GXn (n = 0,2,4,6)
MPX	04n (1)	Multiply GXn (n = 0,1,....,7)

7.12 Transfer Instructions

The program transfer instructions permit conditional and unconditional transfers. TSXn also permits the instruction counter to be stored in index registers X0 through X7. Conditional transfers on zero, plus, and carry also have the corollary transfers nonzero, minus, and no carry. The transfers on overflows and underflows are made to maskable fault routines. If the normal fault routine is masked, transfer is optional. The ISR and SEGID(IS) are affected by transfer of control instructions.

7.12.1 Conditional Transfer

TEO	614 (0)	Transfer on Exponent Overflow
TEU	615 (0)	Transfer on Exponent Underflow
TMI	604 (0)	Transfer on Minus
TMOZ	604 (1)	Transfer on Minus or Zero
TNC	602 (0)	Transfer on No Carry
TNZ	601 (0)	Transfer on Nonzero
TOV	617 (0)	Transfer on Overflow
TPL	605 (0)	Transfer on Plus
TPNZ	605 (1)	Transfer on Plus and Nonzero
TRC	603 (0)	Transfer on Carry
TRCTn	54n (0)	Transfer on Count
TRTF	601 (1)	Transfer on Truncation Indicator OFF
TRTN	600 (1)	Transfer on Truncation Indicator ON
TTF	607 (0)	Transfer on Tally Runout Indicator OFF
TTN	606 (1)	Transfer on Tally Runout Indicator ON
TZE	600 (0)	Transfer on Zero

7.12.2 Unconditional Transfer

RET	630 (0)	Return
TRA	710 (0)	Transfer Unconditionally
TSS	715 (0)	Transfer after Setting Slave
TSXn	70n (0)	Transfer and Set Index Register n

7.13 Miscellaneous Instructions

This subsection describes miscellaneous instructions.

7.13.1 Option Register Instructions

LDO	172 (1)	Load Option Register (a privileged instruction)
STO	152 (1)	Store Option Register

7.13.2 Binary-To-BCD Conversion

The Binary to Binary-Coded-Decimal (BCD) instruction converts the magnitude of a 33-bit or smaller binary number to its decimal equivalent in BCD form. The conversion is made automatically, one decimal digit per instruction execution, using previously stored conversion constants. The BCD form of the converted number is readily available for further operations.

BCD	505 (0)	Binary-to-BCD Convert
-----	---------	-----------------------

7.13.3 Execute Instructions

The Execute and Execute Double (XEC and XED) instructions allow remote instructions to be executed singly or in pairs. A program will continue sequentially after the XEC or XED instructions are executed, as long as the referenced instructions do not alter the instruction counter. If a referenced instruction affects the instruction counter, a program transfer occurs.

XEC	716 (0)	Execute
XED	717 (0)	Execute Double

7.13.4 Gray-To-Binary Conversion

The Gray-To-Binary (GTB) instruction converts a 36-bit word containing data in the Gray code (for example, coded analog information from an analog-to-digital input device) to its binary equivalent in only one execution of the instruction. This instruction enhances the use of the information system in real-time applications, such as telemetry.

GTB	774 (0)	Gray-to-Binary Convert
-----	---------	------------------------

7.13.5 Programmed Fault

DRL	002 (0)	Derail
MME	001 (0)	Master Mode Entry

Machine Instruction Functions

7.13.6 No Operation

NOP	011 (0)	No Operation
PULS1	012 (0)	Pulse One
PULS2	013 (0)	Pulse Two

7.13.7 Repeat Instructions

The RPT and RPD instructions permit execution of the next one or two instructions a selected number of times according to program requirements. They are especially useful for operating upon sequential lists in memory. For example, if RPT is used with any of several compare instructions to search a list, termination occurs when a "hit" is made according to conditions specified in the RPT instruction. The "hit" causes transfer to the next sequential instruction.

RPD	560 (0)	Repeat Double
RPL	500 (0)	Repeat Link
RPT	520 (0)	Repeat

7.13.8 Pointer And Length Instructions

LPL	467 (1)	Load Pointer and Length
SPL	447 (1)	Store Pointer and Length

7.13.9 Read Calendar Clock

RCCL	413 (0)	Read Calendar Clock
------	---------	---------------------

7.13.10 Read Processor Number

RPN	367 (1)	Read Processor Number
-----	---------	-----------------------

7.13.11 Random Number

FRAN	362 (1)	Floating-Point Random Number
XRAN	363 (1)	Fixed-Point Random Number

7.14 Coding Limitations

This subsection provides supplementary specification items and notes relating to the software that operates in the DPS 9000.

7.14.1 Operand During IT Modification

In the IT modification of the following types of instruction, if the indirect word points to itself as an operand address, a Lockup fault occurs. For IDC/DIC modification, if the indirect word is defined as an operand at the end of the chain, a Lockup fault occurs.

- Double-word operand read type instructions
- ARAn, ARNn, AARn, NARn instructions
- LDAC, LDQC, SZNC instructions
- RET instruction

7.14.2 DU/DL Modification of Conditional Transfer of Control Instructions

If a DU/DL modification is specified in the conditional transfer of control instruction, an IPR fault occurs when the transfer is executed, but does not occur when the transfer is not executed.

7.14.3 Accepting an Interrupt during Instruction Overlap Processing

The execution of an FLD or FST instruction, when preceded by a floating-point instruction, may be overlapped with the execution of these floating-point instructions. In this case, an interrupt may not be accepted, even if the interrupt is acceptable (interrupt inhibit bit = 0) in the middle of these consecutive instructions.

7.14.4 Shutdown Fault and Safe Store Stack Fault

If a Safe Store Stack fault occurs when safe storing the status into the safe store stack after a Shutdown fault has occurred, the following fault code and flag in the safe store stack will appear:

- Fault code = 0000000
- Safe Store Stack Fault (SSSF) flag = 1
- Bit 9 of word 5 of safe store stack = 1

Machine Instruction Functions

If a Safe Store Stack fault occurs alone, the following fault code and flag in the safe store stack would appear:

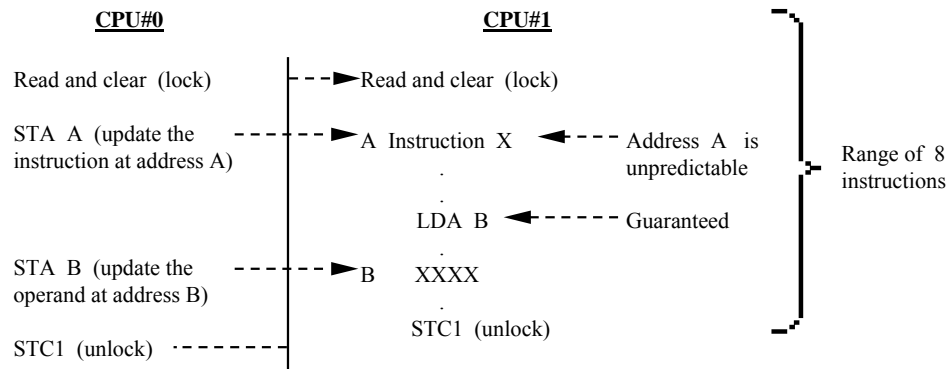
- Fault code = 0000000
- SSSF flag = 1
- Bit 9 of word 5 of safe store stack = 0

If a Shutdown fault occurs alone, the following fault code and flag in the safe store stack would appear:

- Fault code = 0000000
- SSSF flag = 0
- Bit 9 of word 5 of safe store stack = 1

7.14.5 Notes on the Read and Clear Type Instructions

With more than one CPU online, when executing a "read and clear" instruction such as LDAC/LDQC/SZNC, if one of the eight instructions following the read instruction is updated, the updated instruction may not be executed. When the updating involves operand data, the updated data is guaranteed in the operand.



7.14.6 Indirect Modification of Conditional Transfer of Control Instructions

When a conditional transfer of control instruction specifies an indirect modification, execution of the indirect modification is tried even when no transfer is performed. For this reason, if the indirect modification causes a program loop, a Lockup fault can occur even if no transfer takes place.

7.14.7 Operand Store-Compare (alteration of prefetched operand by preceding store instruction)

The store-compare condition is checked by comparing each virtual address of a read operation with the preceding store operation. If one operand in memory can be accessed by two or more virtual addresses, the store-compare condition is not detected correctly. For example, assume that a store the operand by virtual address A occurs and that the very next instruction is a read of the same operand by virtual address B. When both addresses designate the same operand, the store-compare condition is not detected correctly. In this case, more than five instructions must intervene between the store instruction and the read instruction to detect the store-compare condition correctly.

7.14.8 Overlapped Operand of the MLR and MRL Instructions

For performance, the MLR and MRL instructions may prefetch an operand of four double-words.

If the MLR instruction uses overlapped strings (e.g., a case where the starting location of string 1 is less than the starting location of string 2), and if the difference of the starting location between string 1 and string 2 is less than four double-words, the operation is undefined.

7.14.9 Overlapped Operand of the CSL and CSR Instructions

The CSL and CSR instructions may prefetch an operand of three double-words.

If the CSL instruction uses overlapped strings (e.g., a case where the starting location of string 1 is less than the starting location of string 2), and if the difference between the starting location of string 1 and string 2 is less than three double-words, the operation is undefined.

7.14.10 Bound Check of a Multiword Instruction

For performance, the operand of a multiword instruction is prefetched on the pipeline basis. The operand, which will not be used in multiword instruction execution, may be prefetched up to four double-words within the length designated by the instruction. If a prefetch of an operand results in access which exceeds segment bounds, a Bound (BND) fault occurs.

7.14.11 Result of Fault Detection in the MLR/MRL Instruction

When an SFC1/SFC2/BND fault is detected in the MLR/MRL instruction, the last several words (up to four words) preceding the fault may not be stored into memory.

7.14.12 Segment Boundary for Bit String or 6-Bit Character Operand

The operation of a multiword instruction that processes a bit string or a 6-bit character operand is undefined if the segment base of the operand is not on a double-word boundary.

7.14.13 Effective Address (EA) Wraparound Detection

Execution of a multiword instruction that develops addresses at both the upper and lower boundaries of a maximum size segment is not permitted. This restriction is required because of the address wraparound development of the effective address (EA). For each 9-bit byte, 6-bit byte, 4-bit byte or bit string operation byte (each effective address byte), the following checks are made:

- a) for right-to-left instructions (MRL, SCDR, etc.), EA wraparound is detected before any execution;
- b) for left-to-right instructions (MLR, MVT, etc.), EA wraparound is detected at each memory request (which could be up to four double-words ahead of the actual usage because of prefetching).

NOTE: EA wraparound means that the following condition is detected in EA (effective byte address) generation:

NS Mode $(EA + (Length-1))(0-19) = \text{Overflow}$

ES Mode $(EA + (Length-1))(0-35) = \text{Overflow}$

Where, for left-to-right instructions, $(EA + (Length-1))$ means the effective address to be used by each memory request.

If these checks are violated, a Bound fault is generated.

7.14.14 Incorrect Bounds Fault During Multiword Instructions

An incorrect Bounds Fault can occur during Effective Address (EA) generation for the following Multiword, Right-to-Left Instructions when the conditions described below are met:

MRL, SCDR, SCMR, TCTR, CSR, SZTR

Conditions:

The failure occurs only with 6-bit characters or bit string data types when "AR" or "REG" (except IC) modification is used in the Operand Descriptor (the Modifier Field). It does not affect 9-bit Bytes or 4-bit Packed Decimal data types.

IF

1. A negative Effective Address results from the preparation of the word portion of the Effective Address; i.e., before Character or Bit adjustment is applied:

$C(AR\ 0-17) + Y = \text{Negative}$	If AR Modification is used
$C(REG/6) + Y = \text{Negative}$	If 6-bit Character is used
$C(REG/36) + Y = \text{Negative}$	If Bit String and REG

OR

2. The Effective Address would be 0 (zero) after applying C and CN (for 6-bit Characters) or Character/Bit and CC/BC (for Bit String)

An example of such a coded instruction is listed below.

```
LDQ      1,DL
MRL      (,,,Q)
ADSC6    -1,5,1
ADSC6    0,,1
```

This example should reference a One character field starting at Word 0 (zero), Character 0 (zero). However, even though this case should be legitimate, it produces a Bounds Fault.

7.14.15 Prepage Check in a Multiword Instruction

The MVT, TCT, TCTR, and CMPCT instruction have a prepage check. The size of the translate table is determined by the TA1 data type as shown in the table below. Before the instruction is executed, a check is made for allocation in memory for the page for the translate table. If the page is not in memory, a Missing Page fault occurs before execution of the instruction.

TA1	TRANSLATE TABLE SIZE
4-BIT CHARACTER	4 WORDS
6-BIT CHARACTER	16 WORDS
9-BIT CHARACTER	128 WORDS

In some cases (see note below) of the left-to-right 9-bit, 6-bit, 4-bit, or bit string multiword instruction, the page for the virtual address of Base + EA + (Length-1) + 1KW is prepagged. Immediately after this, EA + (Length-1) means the effective address to be used by each memory request.

In other cases (see Note below) of the right-to-left type 9-bit, 6-bit, 4-bit or bit string multiword instruction, the page for the virtual address of Base + EA – 1KW is prepagged. However, any address below the segment base for all right-to-left multiword instruction is not prepagged.

Therefore, software must allocate a dummy page in the next to rightmost page of the segment for each operand segment of the left-to-right type multiword instructions.

NOTE: The following cases generate a correct Missing Page fault:

1. 9-bit character MLR, MRL instructions,
2. Numeric instructions (MVN/X, CMPN/X, AD3D/X, AD2D/X, SB3D/X, SB2D/X, MP3D/X, MP2D/X, DV3D/X, DV2D/X), and
3. 9-bit, 4-bit character and RL=0;

Alphanumeric instructions (MLR, MRL, MVT, CMPC, CMPCT, SCD, SCDR, TCT, TCTR, SCM, SCMR).

The translate table prepagging for the MVT, TCT, TCTR, or CMPCT instructions are limited.

7.14.16 Modification of the "Instruction Stream" Using MLR, MRL, or MTM

If MLR (9 to 9), MRL (9 to 9), or MTM is used to modify a subsequent instruction in the "instruction stream", a minimum of 17 instructions must separate this instruction from the target instruction.

7.15 NovaScale 9000 Instruction Repertoire

Please see Appendix A.

8. Machine Instruction Descriptions (A-B)

This section of the Programmer's Guide provides the Machine Instruction descriptions (A-B), organized as follows:

- Section 8.1, Format of Instruction Descriptions
- Section 8.2, Abbreviations And Symbols
- Section 8.3, Common Attributes Of Instructions
- Section 8.4, Instruction Word Formats
- Section 8.5, Instruction Repertoire
- Section 8.6, Machine Instruction Descriptions (A)
- Section 8.7, Machine Instruction Description (B)

Because of its volume, the catalog of the repertoire of machine instructions for the NovaScale 9000 has been broken into several sections, beginning with this one. Because these instructions are spread over many (consecutive) sections, they have been kept as an alphabetical listing so that individual instructions are easier to find. Their presentation over many sections has been simply a convenience for publication. Section 7 also describes these instructions by functional type, e.g., Floating Point Instructions and Micro Operations.

The generic information about formats, types of instructions, abbreviations, and symbols preceding the catalog of instructions in this section applies to all subsequent sections.

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name, operating code, and page number of the manual.

8.1 Format of Instruction Descriptions

Instructions in the A-B repertoire are described in this section. The descriptions in Sections 8-15 are all presented in the formats shown below.

The format for all instructions is given below.

MNEMONIC	INSTRUCTION NAME	OPCODE
----------	------------------	--------

FORMAT:

<figure or figure reference>

CODING FORMAT:

<text>

PROCESSOR MODE:

<text>

SUMMARY:

<text and/or bit transfer equations>

EXPLANATION:

<text>

ILLEGAL ADDRESS MODIFICATIONS:

<text>

ILLEGAL REPEATS:

<text>

INDICATORS:

<text and/or logic statements>

Machine Instruction Descriptions (A-B)

NOTE(S):

<text>

EXAMPLE(S):

<if applicable>

Line 1: MNEMONIC, INSTRUCTION NAME, OPCODE, [OPE1, OPE2]

MNEMONIC	the mnemonic code for the operation field of the assembler statement. The assembler recognizes this character string value and maps it into the appropriate binary pattern when generating the actual object code.
INSTRUCTION NAME	the name of the machine instruction from which the mnemonic was derived
OPCODE	the octal value of the operation code for the instruction A 0 or a 1 in parentheses following an octal code indicates whether bit 27 (opcode extension bit) of the instruction word is OFF or ON.
OPE1, OPE2	two 2-bit fields, bits 0-1, and 9-10 respectively, that contain a binary extension of the octal op code Contents of OPE1 and OPE2 are generated based on the mnemonic specified for the instruction.

Line 2: FORMAT

The layout and definition of the subfields of the instruction word or words either as a figure or as a reference to a figure

Line 3: CODING FORMAT

The format to be used in coding the instruction

Line 4: OPERATING MODES

The modes in which the processor should be to execute the instruction (Refer to Section 1, Operating Modes.)

Line 5: SUMMARY

The change in the state of the processor affected by the execution of the instruction described in a short, symbolic form

If reference is made to the state of an indicator, it is the state of the indicator before the instruction is executed.

Line 6: EXPLANATION

In instances where more details are needed than supplied in a concise summary, this section describes how the operation functions.

Line 7: ILLEGAL ADDRESS MODIFICATIONS

A list of those modifiers that cannot be used with the instruction

An Illegal Procedure fault occurs when illegal address modification is used.

Line 8: ILLEGAL REPEATS

A list of the repeat instructions that cannot be used with the instruction

Line 9: ILLEGAL EXECUTES

A list of operations or conditions that are prohibited with the instruction

Line 10: INDICATORS

A list of only those indicators whose state can be changed by the execution of the instruction.

In most cases, a condition for setting ON as well as one for setting OFF is stated. If only one of the two is stated, then the indicator remains unchanged if the condition is not met. Unless stated otherwise, the conditions refer to the contents of registers existing after instruction execution.

Line 11: NOTES

Notes regarding specific conditions, faults, and exceptions that affect the operation of the instruction upon the data

Line 12: EXAMPLES

Any coding examples, if required for clarity

8.2 Abbreviations And Symbols

The following abbreviations and symbols are used in the descriptions of the machine operations.

<u>Symbol</u>	<u>Meaning</u>
AND	The Boolean connective AND
AM	Address register modification
AR _n	Address register <u>n</u> specifier in operand descriptor (n = 0, 1,...,7)
b	The original bit position within a 9-bit character
BOLR	Boolean results (4 bits) The BOLR field is used in bit string operations. The bits specify the resultant octal value for four combinations of two input sources.
:(BOLR):	A Boolean operation defined by the BOLR field
c	The original character position within a data word of 9-bit characters
C()	The contents of (). C(string 1) represents the contents of string 1.
C(R)	The complete contents of register R
C(R)(i)	The contents of bit i of register R
C(R)(i-j)	The contents of bits i through j of register R
CN	The original character number within the data word referred to by the original data word address
CS	Character set definition, EBCDIC (0) or ASCII (1)
DR	Displacement register (bits 32-35)
F	Bit value specifier (0 or 1) for bit string fill; used when combining/comparing a short bit string with a long bit string to make the shorter string appear to be the same length as the longer string
FILL	A character used when moving or comparing a short string of characters to a longer string to make the short string appear to be the same length as the longer string (See note under MASK.)
GX _n	General Index Registers 0,1,...7 (ES/EI Mode only)
I	Program interrupt inhibit bit
ID	Indirect operand descriptor indicator

<u>Symbol</u>	<u>Meaning</u>
L	The actual length of the character or bit string, as determined by the register or length (RL) bit in the modification field and by N
LOCSYM	A symbol representing either the address of the operand or the displacement from a base
MASK	Bit pattern used in an instruction word Each 1 bit in the mask causes that bit position in the two characters not to enter into the comparison (coded as octal digits).
NOTE:	FILL and MASK are 9-bit fields. When using 6- or 4-bit characters, the character must be right-justified in the 9-bit field.
MBZ	Must be zero
MF _n	Modification field <u>n</u> describing address modification to be performed in operand descriptor <u>n</u> : MF1 = modification field 1 (bits 29-35) MF2 = modification field 2 (bits 11-17), if operand descriptor 2 is specified MF3 = modification field 3 (bits 2-8), if operand descriptor 3 is specified
N	Either the number of characters or bits in the data string or a 4-bit code (bits 32-35) that specifies a register that contains the number of characters or bits (See L above.)
n	Register designation for those instructions require a register specification to determine operation code
NS	If NS = 0, the operation of the instruction is not affected. If NS = 1, the instruction is not affected unless TN = 0 and SX = 00 or 11, in which case (output is supposed to be overpunched sign) the appropriate overpunched sign character will not be placed in the specified field. Instead, the appropriate numeric (0-9) character will be placed in the specified field, regardless of whether the calculated sign would have been plus or minus. This operation results in a no sign output. For other values of TN and SX the NS bit is ignored. This procedure applies to both EBCDIC and ASCII. This usage of NS is not to be confused with NS used for Normal Segmentation mode.
OPCODE	Operation code field
OPE1, OPE2	Operation code extension fields
OR	The Boolean connective OR (symbol V)

Machine Instruction Descriptions (A-B)

<u>Symbol</u>	<u>Meaning</u>
P	<p>If P = 0, positive signed 4-bit results are stored with octal 14 as the plus sign.</p> <p>If P = 1, positive signed 4-bit results are stored with octal 13 as the plus sign.</p>
R1,R2	General index registers, specified in ES/EI mode only for register to register instructions
R(i)	The ith bit, character, or byte position of R
R(i-j)	Bit, character, or byte positions i through j of R
RD	<p>Rounding numeric indicator flag</p> <p>If RD = 0, no rounding takes place.</p> <p>If RD = 1, rounding takes place as the final operation. The stored result is incremented by 1 at the least significant character if the most significant character of the truncated part is 5 or more.</p>
REG	Address modification register selection for R-type modification of the operand descriptor address field
RL	Register or length indicator
RM	Register modification
S	Sign and decimal type
SF	Scaling factor
SX	Sign and scaling
T	<p>Truncation fault enable indicator:</p> <p>If T = 0, the truncation fault is disabled.</p> <p>If T = 1, the truncation fault is enabled.</p>
TA	A code that defines which type of alphanumeric character is used in the data
TAG	Tag field used to control address modification (bits 30-35)
TN	A code that defines which type of numeric character is used in the data
TR	Timer register
Xn	Index Registers (0,1,...7)
XOR	The Boolean connective EXCLUSIVE OR
y	A 15-bit displacement from the address register address (with bit 29 = 1) or 18-bit address (with bit 29 = 0)
Y	The effective word address (18 bits for NS mode and 34-bits for ES/EI mode) to the word level of the designated instruction

<u>Symbol</u>	<u>Meaning</u>
Y-pair	<p>A symbol denoting that the effective address Y designates a pair of main memory locations (72 bits) with successive addresses, the smaller address being even</p> <p>When Y is even, it designates the pair (Y, Y+1).</p> <p>When Y is odd, it designates the pair (Y-1, Y). The main memory location with the smaller (even) address contains the most significant part of a double-word operand or the first of a pair of instructions.</p>
YC	The effective address for character data
YCB	The effective address for bit string data
Z	The temporary pseudo-result of a nonstore comparison operation
—>	Replace(s)
::	<p>Is compared with</p> <p>Example: C(R) :: C(Y) means C(R) - C((Y)—>C(Z), C(R) and C(Y) unchanged; invisible result C(Z) sets zero, negative and carry indicator as indicated in the instruction descriptions</p>
≠	Not equal
Σ	Sigma sign indicates summary

8.3 Common Attributes Of Instructions

This subsection describes the common attributes of instruction.

8.3.1 Illegal Modification

If an illegal modifier is used with any instruction, an illegal procedure fault with a subcode class of illegal modifier occurs.

8.3.2 Parity Indicator

The parity indicator is turned ON at the end of a main memory access that has incorrect parity.

8.4 Instruction Word Formats

This subsection describes the format of Instruction Words.

8.4.1 Single-Word Instructions

The single-word instruction format is displayed in Figure 8-1.

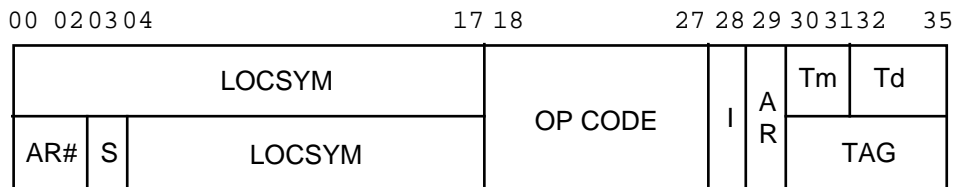
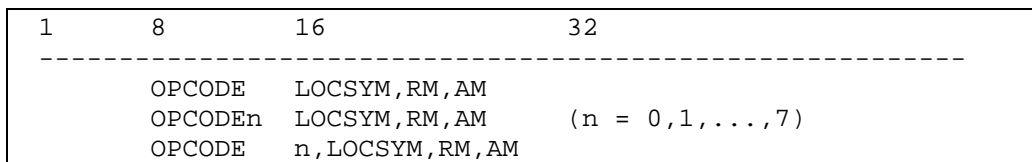


Figure 8-1. Single-Word Instruction Format

CODING FORMATS:



EXAMPLES:

1	8	16	32
*	LDA	AB, X3, AR2	Instruction with no index involved
*	LDX1	AB, X3, AR2	Format 1: instruction with index involved;
*	LDX	1, AB, X3, AR2	Format 2: instruction with index involved.
AB	OCT	0	

where:

AR#	Address register number, if bit 29 = 1
S	Sign bit, if bit 29 = 1
LOCSYM	Address field; bits 0-17 or bits 3-17, depending on the state of bit 29
OP CODE	10-bit operation code field stated as a 3-digit octal number followed by the content of bit 27 (0 or 1) in parentheses
I	Program interrupt inhibit bit
AR	Address register bit <ul style="list-style-type: none"> • If bit 29 = 1, use address register specified in bits 0, 1, and 2 of Y field for address modification. Bit 3 (sign) is then extended to bits 0, 1, and 2. • If bit 29 = 0, no address register modification is performed.
TAG	Tag field; used to control address modification <ul style="list-style-type: none"> • Tm - (Bits 30-31) Type of address modification • Td - (Bits 32-35) Index Register or modification variation designator

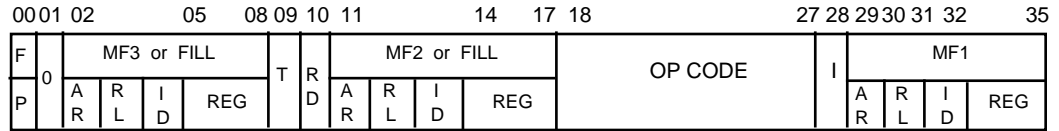
The Repeat (RPT), Repeat Double (RPD), and Repeat Link (RPL) machine instructions and variations of these instructions use special formats and have special tally, terminate, repeat, and other conditions associated with them. The Repeat instructions have no address modifications. Address modifications for the repeated instructions are limited to R and RI with designators specifying X1,...,X7/GX1, ...,GX7. X0/GX0 is used to control terminate conditions and tally. Address Register (AR) modification is also permitted.

The Character Move and Translate instructions (MTR and MTM) use a variation of the single-word instruction format in which two registers are specified.

Indirect words, used for address modification, have the same general format as the instruction words. However, the fields are used in a somewhat different way.

8.4.2 Multiword Instructions

Alphanumeric, numeric, and bit string multiword instructions have the general machine format described in Figure 8-2.



The number of words and fields within the descriptor words will vary by instruction, but use the following general format.

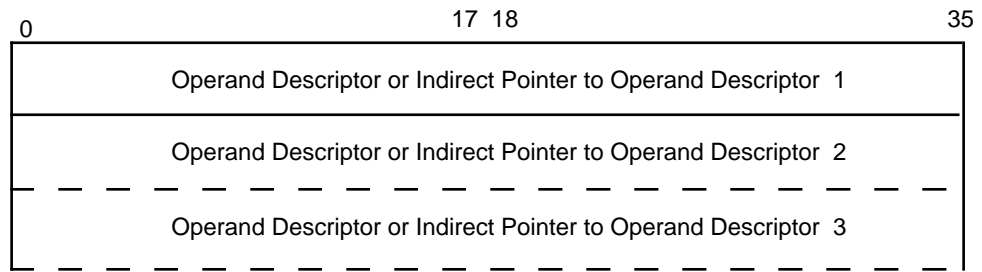


Figure 8-2. Multiword Instruction Format

The fields in the instruction word are defined below. The data fields in the operand descriptor words and the indirect word are discussed in detail in Section 5 under Operand Descriptors, and additional detail, including coding formats, is provided in Section 7 under Multiword Instructions.

where:

F	Bit value specifier for bit string fill
P	Plus sign indicator (octal 13 or 14)
FILL	Fill character specifier
T	Truncation fault enable indicator
RD	Rounding indicator
MF1	Modification field 1 (bits 29-35) denotes address modification to be performed for operand descriptor 1. (See "Multiword Modification Field" in Section 7.)
MF2	Bits 11-17 describe address modification to be performed on this operand for operand descriptor 2
MF3	Bits 2-8 describe address modification to be performed on this operand for operand descriptor 3
OP CODE	10-bit operation code field; octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses
I	Program interrupt inhibit bit
AR	Address register indicator
RL	Register containing length indicator
ID	Indirect operand descriptor indicator
REG	Type of register modification (A, AU, Q, QU, IC, DU, <u>X_n</u> / <u>GX_n</u>)

8.4.3 Address Register Special Arithmetic Instructions

These instructions provide the capability for replacing, adding to, or subtracting from the contents of an address register on either a word, character, or bit address basis. The operation is register-to-register, with no memory fetch involved. The special arithmetic instructions have the format shown in Figure 8-3.

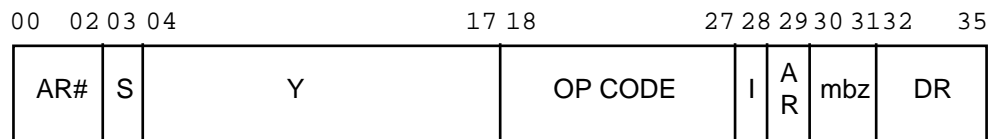


Figure 8-3. Address Register Special Arithmetic Instruction Format

AR#	Selects address register to be altered
S	Sign bit
y	Used as a word displacement (no character or bit position included) along with the contents specified in the DR field to alter the contents of the specified address register. Bit 3 provides negative or positive word displacement.
OP CODE	10-bit operation code field; octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses
I	Program interrupt inhibit bit
AR	Address register bit <ul style="list-style-type: none"> If bit 29 = 1, the sum of the DR (in characters, words, or bits) and the y field (in words) are added to or subtracted from the contents of the AR specified in bits 0-2. If bit 29 = 0, the described sum or its two's complement is loaded into the AR for addition or subtraction, respectively. If the mnemonic is coded with X (for example, AWDX), bit 29 is forced to zero.
mbz	Bits 30-31 must be zero. The operand length is contained in the register specified by DR.
DR	Displacement register; specifies which register contains the displacement value. The register codes and register lengths are the same as those used in MF fields except that IC modification is illegal.

The operations for adding a value to the contents of an address register proceed identically as with effective operand address preparation from an operand descriptor, with the final results being stored in the specified address register. The subtract operation differs only in that the contents of the register specified by the code in the DR field are first added to the y field. This result is then subtracted from the actual contents of the address register or from the implied zero contents and the result is placed in the address register. The codes for DU, DL, and IC are illegal for the DR field and cause an IPR fault.

No indicators are affected by these instructions.

8.4.4 Character Move To/From Register Instructions

Two instructions permit moves of one, two, three, or four 9-bit characters from a memory location to a register or from a register to memory. These instructions have the format shown in Figure 8-4.

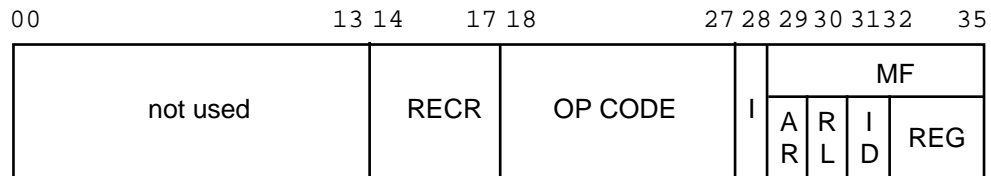


Figure 8-4. Character Move To/From Register Instruction Format

RECR	Specifies the register to which characters are moved (MTR), or from which characters are moved (MTM) (Refer to MTR/MTM instructions.)
OP CODE	10-bit operation code field; octal representation consisting of three octal digits followed by the content of bit 27 (1) in parentheses.
I	Program interrupt inhibit bit.
AR	Address register indicator.
RL	Register containing length indicator.
ID	Indirect operand descriptor indicator.
REG	Type of register modification (A, AU, Q, QU, IC, DU, Xn/GXn).

These instructions move one, two, three, or four 9-bit characters from (MTR) or to (MTM) a memory location to or from a register specified by the RECR field.

8.4.5 Register to Register Instructions

Register to Register instructions known as "RR" type instructions are valid only in the ES/EI mode. An attempt to execute these instructions in the NS mode results in an IPR fault. RR type instructions permit movement, arithmetic operation, and shift of fixed-point data using the GXn, A and Q registers. An attempt to execute any RR type instruction by the RPT, RPD, or RPL instructions results in an IPR fault. The format for register to register instructions is shown in Figure 8-5.

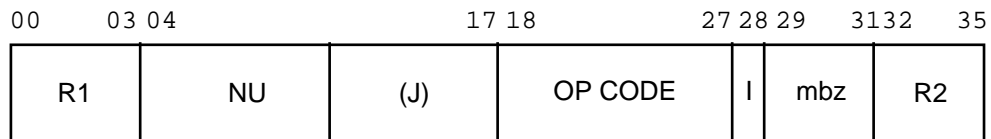


Figure 8-5. Register To Register Instruction Format

<u>Bits</u>	<u>Field</u>	<u>Description</u>																																		
0–3	R1	specifies a code indicating a register to be the destination of the result The allowable codes are listed below.																																		
		<table border="1"> <thead> <tr> <th><u>Register Code</u></th> <th><u>Result</u></th> </tr> </thead> <tbody> <tr><td>0000</td><td>IPR</td></tr> <tr><td>0001</td><td>IPR</td></tr> <tr><td>0010</td><td>IPR</td></tr> <tr><td>0011</td><td>IPR</td></tr> <tr><td>0100</td><td>IPR</td></tr> <tr><td>0101</td><td>A</td></tr> <tr><td>0110</td><td>Q</td></tr> <tr><td>0111</td><td>IPR</td></tr> <tr><td>1000</td><td>GX0</td></tr> <tr><td>1001</td><td>GX1</td></tr> <tr><td>1010</td><td>GX2</td></tr> <tr><td>1011</td><td>GX3</td></tr> <tr><td>1100</td><td>GX4</td></tr> <tr><td>1101</td><td>GX5</td></tr> <tr><td>1110</td><td>GX6</td></tr> <tr><td>1111</td><td>GX7</td></tr> </tbody> </table>	<u>Register Code</u>	<u>Result</u>	0000	IPR	0001	IPR	0010	IPR	0011	IPR	0100	IPR	0101	A	0110	Q	0111	IPR	1000	GX0	1001	GX1	1010	GX2	1011	GX3	1100	GX4	1101	GX5	1110	GX6	1111	GX7
<u>Register Code</u>	<u>Result</u>																																			
0000	IPR																																			
0001	IPR																																			
0010	IPR																																			
0011	IPR																																			
0100	IPR																																			
0101	A																																			
0110	Q																																			
0111	IPR																																			
1000	GX0																																			
1001	GX1																																			
1010	GX2																																			
1011	GX3																																			
1100	GX4																																			
1101	GX5																																			
1110	GX6																																			
1111	GX7																																			
4–10	NU	not used																																		
11–17	J	used only in a shift instruction; specifies the shift number (immediate value); must be 0 in all but shift instructions																																		
18–27	OP	operation code																																		
28	I	interrupt inhibit bit																																		

Machine Instruction Descriptions (A-B)

<u>Bits</u>	<u>Field</u>	<u>Description</u>
29–31	MBZ	must be zero or an IPR fault occurs
32–35	R2	specifies a code indicating the source register The codes for this register are the same as for R1.

- NOTES:**
1. Specifying a register code of 0000 in a shift instruction does not result in an IPR fault.
 2. If a register pair appears in an instruction specification, the two registers are handled as linked. The list below indicates the register codes to be associated with the register pair.

Register Code	Result
0000	IPR
0001	IPR
0010	IPR
0011	IPR
0100	IPR
0101	A, Q
0110	A, Q
0111	IPR
100x	GX0, GX1
101x	GX2, GX3
110x	GX4, GX5
111x	GX6, GX7

where x means this bit is ignored by the hardware

8.5 Instruction Repertoire

The processor interprets a 10-bit field of the instruction word as the operation code. This field size yields 1024 possible instructions codes of which over half are implemented.

A4BD/A4BDX

8.6 Machine Instruction Descriptions (A)

Following is a detailed description of the processor instructions and operation codes beginning with the letter A, followed by the letter B. Codes beginning with the letters C-X can be found in sections 9-15.

8.6.1 A4BD/A4BDX

A4BD A4BDX	Add 4-bit Displacement to Address Register	502(1)
---------------	---	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{A4BD }		
	{A4BDX}	word displacement	,R,AR

When the mnemonic is coded with an "X" (A4BDX), bit 29 is forced to zero.

OPERATING MODES:

Any

A4BD/A4BDX**EXPLANATION:**NS Mode

The count of 4-bit characters contained in the register specified by the DR field is effectively divided by 8, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-17 of the specified AR, with the character count (from the divide) being translated into bit string representation and replacing bits 18-23 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-17 of the specified AR. The CHAR and BIT portions (bits 18-23) of the specified AR are forced to point to a 4-bit character boundary in bit string representation. The resulting character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-23 of the specified AR. With this addition, carry from the CHAR field is transferred to the WORD field.

ES/EI Mode

The count of 4-bit characters contained in the register specified by the DR field is effectively divided by 8, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-29 of the specified AR, with the character count (from the divide) being translated into bit string representation and replacing bits 30-35 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-29 of the specified AR. The CHAR and BIT portions (bits 30-35) of the specified AR are forced to point to a 4-bit character boundary. The resulting character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-35 of the specified AR. With this addition, carry from the CHAR field is transferred to the WORD field.

Effectively, the two bit string representations are added and the result is translated back to a format allowing 2 bits to represent the characters and 4 bits to represent bits. Any overflow of the 2 bits increments the address field and the 4-bit field is handled as mod-9. Any overflow of the 2-bit field increments the character (2-bit) field.

A4BD/A4BDX

ILLEGAL ADDRESS MODIFICATIONS:

When DU, DL, and IC are specified in the DR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

An Illegal Procedure fault occurs if this instruction is executed in SE mode

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

(apply to NS mode only)

1	8	16	

	EAX3	9	
	A4BDX	2,3,5	AR5 octal contents - 00000305
	A4BD	0,3,5	AR5 octal contents - 00000420
	EAX4	6	
	A4BDX	0,4,3	AR3 octal contents - 00000060
	EAX5	9	
	A4BD	4,5,3	AR3 octal contents - 00000565

A6BD/A6BDX**8.6.2 A6BD/A6BDX**

A6BD A6BDX	Add 6-bit Displacement to Address Register	501(1)
---------------	---	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{A6BD}		
	{A6BDX}	word displacement	R,AR

When the mnemonic is coded with an X (A6BDX), bit 29 is forced to zero.

OPERATING MODES:

Any

EXPLANATION:NS Mode

The count of 6-bit characters contained in the register specified by the DR field is effectively divided by 6, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-17 of the specified AR, with the character count (from the divide) being translated into bit string representation and replacing bits 18-23 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-17 of the specified AR. The CHAR and BIT portions (bits 18-23) of the specified AR are forced to point to a 6-bit character boundary. The resulting 6-bit character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-23 of the specified AR. With this addition, carry from the CHAR field (when carry + character count > 5) is transferred to the WORD field.

A6BD/A6BDX

ES/EI Mode

The count of 6-bit characters contained in the register specified by the DR field is effectively divided by 6, producing a word count and a character count. The word count is added to the y field (bit 3 extended).

If bit 29 = 0, this sum replaces bits 0-29 of the specified AR, with the character count (from the divide) being translated into bit string representation and replacing bits 30-35 of AR.

If bit 29 = 1, the sum of the word count (from the divide) and y field is added to bits 0-29 of the specified AR. The CHAR and BIT portions (bits 30-35) of the specified AR are forced to point to a 6-bit character boundary. The resulting 6-bit character count is added to the character count from the divide operation, with the result being translated back into bit string representation. These formed values for the WORD, CHAR, and BIT fields are stored in bits 0-35 of the specified AR. With this addition, carry from the CHAR field (when carry + character count > 5) is transferred to the WORD field.

ILLEGAL ADDRESS MODIFICATIONS:

When DU, DL, or IC are specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

An Illegal Procedure fault occurs if this instruction is executed in SE mode

INDICATORS:

None Affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

A6BD/A6BDX**EXAMPLES:**

(apply to NS mode only)

1	8	16	

	EAX2	8	
	A6BDX	3,2,6	AR6 octal contents - 00000423
	A6BD	2,2,6	AR6 octal contents - 00000746
	EAX4	15	
	A6BDX	0,4,7	AR7 octal contents - 00000240
	A6BD	2,4,7	AR7 octal contents - 00000700

A9BD/A9BDX

8.6.3 A9BD/A9BDX

A9BD A9BDX	Add 9-bit Displacement to Address Register	500(1)
---------------	---	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

{A9BD}			
{A9BDX}		word displacement, R, AR	

When the mnemonic is coded with an X (A9BDX), bit 29 is forced to zero.

OPERATING MODES:

Any

EXPLANATION:

NS Mode

The count of 9-bit characters contained in the register specified by the DR field is effectively divided by 4, producing a word count and a character count. This word count is then added to the y field (bit 3 extended).

If bit 29 = 0, the resulting sum of the word addresses and the character count (from the divide operation) replaces bits 0-19 of the specified AR.

If bit 29 = 1, the resulting sum of the word addresses is added to bits 0-17 of the specified AR and the character count (from the divide operation) is added to bits 18-19 of C(AR). These results are then stored in bits 0-19 of the specified AR. In either case, bits 20-23 of the specified AR are zeroed. Carry is transferred from bit 18 to bit 17 with this addition.

A9BD/A9BDXES/EI Mode

The count of 9-bit characters contained in the register specified by the DR field is effectively divided by 4, producing a word count and a character count. This word count is then added to the y field (bit 3 extended).

If bit 29 = 0, the resulting sum of the word addresses and the character count (from the divide operation) replaces bits 0-31 of the specified AR.

If bit 29 = 1, the resulting sum of the word addresses is added to bits 0-29 of the specified AR and the character count (from the divide operation) is added to bits 30-31 of C(AR). These results are then stored in bits 0-31 of the specified AR. In either case, bits 32-35 of the specified AR are zeroed. Carry is transferred from bit 30 to bit 29 with this addition.

ILLEGAL ADDRESS MODIFICATIONS:

When DU, DL, or IC are specified in the DR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLES:

(apply to NS mode only)

1	8	16	

	EAX1	6	
	A9BDX	2,1,2	AR2 octal contents - 00000340
	A9BD	2,,2	AR2 octal contents - 00000540
	EAX2	15	
	A9BDX	4,2,6	AR6 octal contents - 00000760
	A9BD	0,2,6	AR6 octal contents - 00001340

AAR_n

8.6.4 AAR_n

AAR _{<u>n</u>}	Alphanumeric Descriptor to Address Register <u>n</u>	56 _{<u>n</u>} (1)
-------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

AAR _{<u>n</u>}		LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

For n = 0, 1, ..., or 7 as determined by op code:
 C(Y)(00-17) -> C(AR_n)(0-17);
 C(Y)(18-20) translated C(AR_n)(18-23)

EXPLANATION:

The alphanumeric descriptor is fetched from the computed effective address Y. The TA field, bits 21 and 22, is examined to determine the type of data described. If the TA code indicates 9-bit character data, bits 18 and 19 of the descriptor CN field go to the corresponding bit positions of AR_n and zeros fill bits 20-23 of AR_n. If the TA code indicates 6- or 4-bit character data, the descriptor CN field is appropriately translated into bit string representation and goes to bits 18-23 of AR_n. In all cases, the word portion of the fetched descriptor is placed in the word portion (bits 0-17) of AR_n.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

AAR_n

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

An Illegal Procedure fault occurs if this instruction is executed in ES/EI modes

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used or if the descriptor TA field contains code 11.
2. An IPR fault occurs if descriptor CN field contains xx1 for TA = 00, or 11x for TA = 01.
3. An IPR fault occurs if an attempt is made to execute this instruction in the ES/EI mode.

EXAMPLES:

(apply to NS mode only)

1	8	16	32

	AAR4	DESCR	load data string address into AR4
*	.	.	memory contents in octal
	.	.	
	.	.	
	.	.	
DESCR	ADSC9	FLD1,3,1	001023600001 - descriptor
*			AR4 octal contents - 00102360

ABD/ABDX

8.6.5 ABD/ABDX

ABD ABDX	Add Bit Displacement to Address Register	503(1)
-------------	---	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	32

{ABD}			
{ABDX}		word displacement, RM, AR	

When the mnemonic is coded with an X (ABDX), bit 29 is forced to zero

OPERATING MODES:

Any

EXPLANATION:

NS Mode

The bit string count in the register specified in the DR field is divided by 36. The quotient is taken as the word count and the remainder is taken as the bit count. The word count is added to the y field for which bit 3 of the instruction word is extended and the sum is taken.

If bit 29=0, the sum is loaded into bits 0-17 of the specified AR, and the character portion and the bit portion of the remainder are loaded into bits 18-23 of the specified AR.

If bit 29=1, the sum is added to bits 0-17 of the specified AR. The CHAR and BIT fields (bits 18-23) of the specified AR are added to the character portion and the bit portion of the remainder. WORD, CHAR and BIT fields generated in this manner are loaded into bits 0-23 of the specified AR. With this addition, carry from the BIT field (bit 20) and the CHAR field (bit 18) is transferred (when BIT field >8, CHAR field >3).

ABD/ABDXES/EI Mode

The bit string count in the register specified in the DR field is divided by 36. The quotient is taken as the word count and the remainder is taken as the bit count. The word count is added to the y field for which bit 3 of the instruction word is extended and the sum is taken.

If bit 29=0, the sum is loaded into bits 0-29 of the specified AR, and the character portion and the bit portion of the remainder are loaded into bits 30-35 of the specified AR.

If bit 29=1, the sum is added to the sign extended value of bits 0-29 of the specified AR. The CHAR and BIT fields (bits 30-35) of the specified AR are added to the character portion and the bit portion of the remainder. WORD, CHAR, and BIT fields generated in this manner are loaded into bits 0-35 of the specified AR. With this addition, carry from the BIT field (bit 30) and the CHAR field (bit 32) is transferred (when BIT field >8, CHAR field >3).

ILLEGAL ADDRESS MODIFICATIONS:

When DU, DL, or IC are specified in the DR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

An Illegal Procedure fault occurs if this instruction is executed in SE mode

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ABD/ABDX

EXAMPLES:

(apply to NS mode only)

1	8	16	

	EAX6	85	
	ABDX	7,6,2	AR2 octal contents - 0 0 0 0 1 1 2 4
	ABD	2,6,2	AR2 octal contents - 0 0 0 0 1 5 5 0
	EAX1	74	
	EAX2	30	
	ABDX	4,1,3	AR3 octal contents - 0 0 0 0 0 6 0 2
	ABD	0,2,3	AR3 octal contents - 0 0 0 0 0 6 6 5

8.6.6 ACKS

ACKS	Acknowledge Sync Interrupt	735(1)
------	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

Loads the A-register with zero to simulate a time-out on the ACKS.

ILLEGAL ADDRESS MODIFICATIONS:

IT, RI, IR

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTE:

This instruction is used on Olympus to test shared cache synchronization. Since a commercial processor manages the processor cache on the V9000, the instruction is implemented as if the instruction timed out, i.e., synchronization failed.

AD2D

8.6.7 AD2D

AD2D	Add Using Two Decimal Operands	202(1)
------	--------------------------------	--------

FORMAT:

0001	0809 1011	17 18	27 28 29	35			
P	00000000	T	R D	MF2	202(1)	I	MF1

00 01 02	17 18	20 21 22 23 24	29 30	35		
Y1		CN1	T N 1	S1	SF1	N1
AR#	Y1					

00 01 02	17 18	20 21 22 23 24	29 30	35		
Y2		CN2	T N 2	S2	SF2	N2
AR#	Y2					

CODING FORMAT:

1	8	16

AD2D	(MF1), (MF2), RD, P, T	
NDSCn	LOCSYM, CN, N, S, SF, AM	
NDSCn	LOCSYM, CN, N, S, SF, AM	

(Refer to Section 7 under Multiword Instructions for a description of Multiword Modification Field.)

OPERATING MODES:

Any

AD2D

SUMMARY:

`C(string 2) + C(string 1) -> C(string 2)`

Same as AD3D, except that the sum is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for AD3D

NOTES:

1. All notes for AD3D apply also to AD2D.
2. Illegal Procedure fault same as for MVN
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

AD2D

EXAMPLES:

1	8	16	32

	AD2D	,, ,1	with truncation enable option
	NDSC4	FLD1,0,8,2,-2	FLD1 addend operand descriptor
	NDSC9	FLD2,0,6	FLD2 addend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P123456+	0 1 2 3 4 5 6 +
FLD2	EDEC	6A+1E+2	+ 0 0 0 1 2
	USE		+ 1 3 3 4 0 (Sum)
*			(truncation fault)
*			
	AD2D	,, ,1	with plus sign octal 13 option
	NDSC9	FLD1,0,4	FLD1 addend operand descriptor
	NDSC4	FLD2,1,7,2,-4	FLD2 addend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4A+99.	+ 9 9 0
FLD2	EDEC	8P123456+	0 1 2 3 4 5 6 +
	USE		0 1 1 3 4 5 6 + (Sum)
*			(overflow fault)

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX1	1	load character modifier into X1
	EAX7	7	load FLD1 length into X7
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	AD2D	(1,1,,X1),(, ,1),1,1	rounding, plus sign opts.
	NDSC4	0,,X7,2,-2,4	FLD1 operand descriptor
*			(FLD1,1,7,2,-2)
*	NDSC9	INDSC2	pointer, FLD2 indirect operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P123450-	0 1 2 3 4 5 0
FLD2	EDEC	8A+9876E+2	+ 0 0 9 8 7 6 2
INDSC2	NDSC9	FLD2,0,8	indirect operand descriptor
	USE		+ 9 8 6 3 6 6 0 (Sum)

AD2DX

8.6.8 AD2DX

AD2DX	Add Using Two Decimal Operands Extended	242(1)
-------	--	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
C	N	0000000	T	R	MF2			242(1)			I	MF1
S	S											

00	01	02	17	18	20	21	22	23	24	29	30	35
Y1				CN1	T N 1	SX1	SF1	N1				
AR#	Y1											

00	01	02	17	18	20	21	22	23	24	29	30	35
Y2				CN2	T N 2	SX2	SF2	N2				
AR#	Y2											

CODING FORMAT:

1	8	16	32

AD2DX	(MF1), (MF2), RD, CS, T, NS		
NDSCn	LOCSYM, CN, N, SX, SF, AM		
NDSCn	LOCSYM, CN, N, SX, SF, AM		

(Refer to Section 7 under Multiword Instructions for a description of Multiword Modification Field.)

OPERATING MODES:

Any

AD2DX

SUMMARY:

`C(string 2) + C(string 1) -> C(string 2)`

EXPLANATION:

The decimal numbers of data type TN1, sign and decimal type SX1, and starting location YC1, are added to the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The sum is stored starting in location YC2 as a decimal number of data type TN2 and sign and decimal type SX2.

If SX2 indicates a fixed-point format, the results are stored using scale factor SF2, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.

If SX2 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation.

The character set is defined by CS. Placement of an overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of NS.) If RD is 1, rounding takes place before storage. If strings 1 and 2 are not overlapped, the contents of the decimal number that starts in location YC1 remains unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

AD2DX**INDICATORS:**

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then it is ON. Otherwise (i.e., no least significant digits lost or rounding specified) it is OFF.
Overflow	If data is lost in most significant positions, then ON; otherwise, unchanged
Exponent Overflow	If exponent of floating point result > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating point result < -128, then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.
2. Illegal procedure faults occur when:
 - a) DU or DL modification is in MF1 or MF2,
 - b) the sign and numeric digits contain an unpermitted code,
 - c) though the operand descriptor indicates the presence of a sign or exponent, the value of N₁ or N₂ does not contain the number of characters required for the sign and exponent (when at least one digit is required).
 - d) An illegal repeat is used.
3. Regardless of the data type being used (either packed decimal or 9-bit numeric; floating point or fixed-point), significant digits of the result may be lost if the result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.
4. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.
5. All notes for AD3D apply to AD2DX.
6. Refer to the specifications on MVNX for information on coding of overpunched signs.
7. An Illegal Procedure fault occurs if illegal address modification is used.

AD3D

8.6.9 AD3D

AD3D	Add Using Three Decimal Operands	222(1)
------	----------------------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
P	0	MF3	T	R	D	MF2	222(1)	I	MF1			

00	01	02	17	18	20	21	22	23	24	29	30	35
Y1			CN1	T	N	S1	SF1			N1		
AR#	Y1											

00	01	02	17	18	20	21	22	23	24	29	30	35
Y2			CN2	T	N	S2	SF2			N2		
AR#	Y2											

00	01	02	17	18	20	21	22	23	24	29	30	35
Y3			CN3	T	N	S3	SF3			N3		
AR#	Y3											

CODING FORMAT:

1	8	16		

	AD3D	(MF1) , (MF2) , (MF3) , RD , P , T		
	NDSCn	LOCSYM , CN , N , S , SF , AM		
	NDSCn	LOCSYM , CN , N , S , SF , AM		
	NDSCn	LOCSYM , CN , N , S , SF , AM		

Refer to Section 7 under Multiword Instructions for a description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 2) + C(string 1) -> C(string 3)

EXPLANATION:

The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, are added to the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The sum is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using scale factor SF3, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation.

If P = 1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is 1, rounding takes place before storage.

If strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

The zero indicator is set when the decimal number is zero; it does not indicate the case in which all bits are zero.

If the result is given by a fixed-point, operations are performed by justifying the scaling factors (SF1, SF2, and SF3) of the operands 1, 2, and 3:

```

If SF1 > SF2
SF1 > SF2 >= SF3 -> Justify to SF2
SF3 > SF1 > SF2 -> Justify to SF1
SF1 >= SF3 > SF1 -> Justify to SF3 - 1

```

```

If SF2 > SF1
SF2 > SF1 >= SF3 -> Justify to SF1
SF3 > SF2 > SF1 -> Justify to SF2
SF2 >= SF3 > SF1 -> Justify to SF3 - 1

```

AD3D

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT,RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding is specified), OFF
Exponent Overflow	If exponent of floating-point result is > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128, then ON; otherwise, unchanged
Overflow	If data is lost in most significant positions, then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.
2. Illegal procedure faults occur when
 - a) DU or DL modification is in MF1 or MF2,
 - b) the sign and numeric digits contain an unpermitted code,
 - c) though the operand descriptor indicates the presence of a sign or exponent, the value of N₁ or N₂ does not contain the number of characters required for the sign and exponent (when at least one digit is required).

AD3D

3. Independent of the data type being used (either packed decimal or 9-bit numeric; floating-point or scaled); significant digits in the result may be lost if:
 - The difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal point alignment of source operands, followed by addition.
 - The result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.
4. If an illegal digit or sign is detected, part or all of the receiving field may be changed before the IPR fault occurs.

EXAMPLE:

1	8	16	32

	AD3D	,, ,1,1	w/ rounding, plus sign options
	NDSC9	FLD1,0,4,3,-2	FLD1 addend operand descriptor
	NDSC9	FLD2,0,8,2,-2	FLD2 addend operand descriptor
	NDSC4	FLD3,2,6,1	operand descriptor, sum field
	USE	CONST.	memory contents
FLD1	EDEC	4A1234	1 2 3 4
FLD2	EDEC	8A654321+	0654321+
FLD3	BSS	1	xx+06556 (Sum)
	USE		instruction fault? no

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX2	2	load character modifier into X2
	EAX6	6	load FLD1 length into X6
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	AD3D	(1),(,1,,X2),(,,1),1,1	
	NDSC9	0,0,4,	FLD1 operand descriptor
*			(FLD1,0,4,0)
	NDSC4	FLD2,,X6,3,-2	FLD2 operand descriptor
*			(FLD2,2,6,3,-2)
	ARG	DFLD3	pointer,FLD3 operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4A-12E+2	- 1 2 2
FLD2	EDEC	8P123456	00123456
FLD3	BSS	1	xxx+0346 (Sum)
DFLD3	NDSC4	FLD3,3,5,1,-1	FLD3 sum operand descriptor
	USE		instruction fault? no

AD3DX

8.6.10 AD3DX

AD3DX	Add Using Three Decimal Operands Extended	262(1)
-------	---	--------

FORMAT:

00 01 02	0809 1011	17 18	27 28 29	35			
C S	N S	MF3	T R D	MF2	262(1)	I	MF1

00 01 02	17 18	20 21 22 23 24	29 30	35	
Y1	CN1	T N 1	SX1	SF1	N1
AR#	Y1				

00 01 02	17 18	20 21 22 23 24	29 30	35	
Y2	CN2	T N 2	SX2	SF2	N2
AR#	Y2				

00 01 02	17 18	20 21 22 23 24	29 30	35	
Y3	CN3	T N 3	SX3	SF3	N3
AR#	Y3				

CODING FORMAT:

1	8	16

AD3DX	(MF1) , (MF2) , (MF3) , RD , CS , T , NS	
NDSCh	LOCSYM , CN , N , SX , SF , AM	
NDSCh	LOCSYM , CN , N , SX , SF , AM	
NDSCh	LOCSYM , CN , N , SX , SF , AM	

(Refer to Section 7 under Multiword Instructions for a description of Multiword Modification Field.)

AD3DX**OPERATING MODES:**

Any

SUMMARY:

`C(string 2) + C(string 1) -> C(string 3)`

EXPLANATION:

The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is added to the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The sum is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type SX3.

If SX3 indicates a fixed-point format, the results are stored using scale factor SF3, which causes leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most significant digit overflow or least significant digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation. The character set is defined by CS. Placement of overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of NS.) If RD is 1, rounding takes place prior to storage. If strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2 and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

AD3DX

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding specified), OFF
Overflow	If data is lost in most significant positions, then ON; otherwise, unchanged
Exponent Overflow	If exponent of floating-point result > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result < -128, then ON; otherwise, OFF

NOTES:

1. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.
2. Illegal procedure faults occur when
 - a) DU or DL modification is in MF1 or MF2,
 - b) the sign and numeric digits contain an unpermitted code,
 - c) though the operand descriptor indicates the presence of a sign or exponent, the value of N₁ or N₂ does not contain the number of characters required for the sign and exponent (when at least one digit is required).
3. Independently of the data type being used (either packed decimal or 9-bit numeric, floating-point or scaled) significant digits of the result may be lost if the result field as defined by the result descriptor is not large enough to contain the actual calculated result after it has been aligned.
4. If an illegal digit or sign is detected, part or the entire receiving field may be changed before the IPR fault occurs.
5. For coding of overpunched signs, refer to MVNX.
6. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ADA**8.6.11 ADA**

ADA	Add to A-Register	075(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) + C(Y) \rightarrow C(A)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of bit 0 of $C(A)$ is generated, then ON; otherwise, OFF

ADAQ

8.6.12 ADAQ

ADAQ	Add to AQ-Register	077(0)
------	--------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) + C(Y\text{-pair}) \rightarrow C(AQ)$; $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if an illegal address modification or an illegal repeat is used.

ADE

8.6.13 ADE

ADE	Add to Exponent Register	415(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(E) + C(Y)(0-7) \rightarrow C(E)$

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	Set OFF; set ON if exponent underflow and fault masked.
Negative	Set OFF
Exponent Overflow	If exponent is > +127, then ON
Exponent Underflow	If exponent is < -128, then ON

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. All data is handled as 0 when DL modification is specified in the NS mode.

ADL

8.6.14 ADL

ADL	Add Low to AQ-Register	033(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

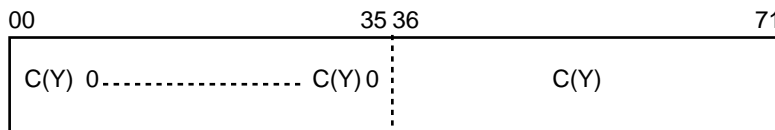
OPERATING MODES:

Any

SUMMARY:

$C(AQ) + C(Y, \text{right-adjusted}) \rightarrow C(AQ)$

This instruction forms the following 72-bit number:



The lower half (bits 36 through 71) is C(Y). The bits in the upper half (bits 0 through 35) are equal to the C(Y) sign bit (C(Y)(0)). This value is added to the AQ. If a carry is generated from Q as a result of this addition, it is passed on to A.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON
Carry	If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ADLA

8.6.15 ADLA

ADLA	Add Logical to A-Register	035(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) + C(Y) \rightarrow C(A)$; $C(Y)$ unchanged

EXPLANATION:

This instruction is identical to ADA with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

Carry If a carry out of bit 0 of $C(A)$ is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of A has been exceeded.

ADLAQ**8.6.16 ADLAQ**

ADLAQ	Add Logical to AQ-Register	037(0)
-------	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) + C(Y\text{-pair}) \rightarrow C(AQ)$; $C(Y\text{-pair})$ unchanged

This instruction is identical to ADAQ with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If $C(AQ)(0) = 1$, then ON; otherwise, OFF

Carry If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of AQ has been exceeded.

ADLAQ

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

ADLQ**8.6.17 ADLQ**

ADLQ	Add Logical to Q-Register	036(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) + C(Y) \rightarrow C(Q)$; $C(Y)$ unchanged

EXPLANATION:

This instruction is identical to ADQ with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

Carry If a carry out of bit 0 of $C(Q)$ is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of Q has been exceeded.

ADLR

8.6.18 ADLR

ADLR	Add Logical Register to Register	435(1)
------	----------------------------------	--------

FORMAT:

00	03 04	17 18		27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2	

CODING FORMAT:

1	8	16	

	ADLR	R1, R2	

OPERATING MODES:

Executes in ES/EI modes only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q;
 C(R1) + C(R2) -> C(R1); C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ADLR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ADLX_n

8.6.19 ADLX_n

ADLX _{<u>n</u>}	Add Logical to Register <u>n</u>	02 _{<u>n</u>} (0)
--------------------------	----------------------------------	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(X_{\underline{n}}) + C(Y)(0-17) \rightarrow C(X_{\underline{n}}); C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(GX_{\underline{n}}) + C(Y) \rightarrow C(GX_{\underline{n}}); C(Y)$ unchanged

EXPLANATION:

This instruction is identical to ADX_n with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of ADLX0

ADLX_n**INDICATORS:**

Zero	If $C(X_n)/(GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n)/(GX_n)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(X_n)/(GX_n)$ is generated, then ON; otherwise, OFF. When the carry indicator is ON, the range of X_n/GX_n been exceeded

NOTES:

1. All data is handled as 0 when DL modification is specified for the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ADQ

8.6.20 ADQ

ADQ	Add to Q-Register	076(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) + C(Y) \rightarrow C(Q)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Q)$ is generated, then ON; otherwise, OFF

ADRR**8.6.21 ADRR**

ADRR	Add Register to Register	434(1)
------	--------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16

ADRR	R1, R2	

OPERATING MODES:

Executes in ES mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q;
 $C(R1) + C(R2) \rightarrow C(R1)$; C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS/EI modes

ADRR

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Overflow	If the range of R1 is exceeded, ON
Carry	If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS/EI modes.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ADTCS**8.6.22 ADTCS**

ADTCS	Add into TCS Clock Register	756(1)
-------	-----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Can not be executed in Slave or Master mode.

SUMMARY:

Implemented as a NOP
No operation takes place. The effective address is always prepared.

EXPLANATION:

No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the generated addresses may cause faults.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words. The Tally Runout indicator may be set ON.

ADTCS

NOTES:

1. An Illegal Procedure fault occurs when an illegal repeat is used.
2. Because address preparation takes place, modification may result in a Bounds fault.

ADX_n**8.6.23 ADX_n**

ADX _n	Add to Index Register <u>n</u>	06 _n (0)
------------------	--------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, \text{or } 7$ as determined by op code:
 $C(X_n) + C(Y)(0-17) \rightarrow C(X_n); C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, \text{or } 7$ as determined by op code:
 $C(GX_n) + C(Y) \rightarrow C(GX_n); C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of ADX0

ADX_n

INDICATORS:

Zero	If $C(X_n)/(GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n)/(GX_n)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of X_n/GX_n is exceeded, then ON
Carry	If a carry out of bit 0 of $C(X_n/GX_n)$ is generated, then ON; otherwise, OFF

NOTES:

1. All data is handled as 0 when DL modification is specified in the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ALR**8.6.24 ALR**

ALR	A-Register Left Rotate	775(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:NS Mode

Rotate C(A) left the number of positions indicated by bits 11-17 of Y (Y modulo 128); enter each bit leaving bit position 0 in bit position 35.

ES/EI Mode

Rotate C(A) left the number of positions indicated by bits 27-33 of Y (Y modulo 128); enter each bit leaving bit position zero in bit position 35.

The rotate count in the instruction must be a decimal number. To "right-rotate" n bits, use ALR 36-n.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(A) = 0, then ON; otherwise, OFF

Negative If C(A)(0)= 1, then ON; otherwise, OFF

ALR

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ALS**8.6.25 ALS**

ALS	A-Register Left Shift	735(0)
-----	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:NS Mode

Shift C(A) left the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros.

ES/EI Mode

Shift C(A) left the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zero.

The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

ALS

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Carry	If $C(A)(0)$ changes during the shift, then ON; otherwise, OFF. When the Carry indicator is ON, the algebraic range of A has been exceeded.

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANA**8.6.26 ANA**

ANA	AND to A-Register	375(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(A)(i) \text{ AND } C(Y)(i) \rightarrow C(A)(i)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

ANAQ

8.6.27 ANAQ

ANAQ	AND to AQ-Register	377(0)
------	--------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 71 :
 $C(AQ)(i) \text{ AND } C(Y\text{-pair})(i) \rightarrow C(AQ)(i)$; $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

ANQ**8.6.28 ANQ**

ANQ	AND to Q-Register	376(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(Q)(i) \text{ AND } C(Y)(i) \rightarrow C(Q)(i)$;

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

ANRR

8.6.29 ANRR

ANRR	AND Register to Register	535(1)
------	--------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

ANRR		R1, R2	

OPERATING MODES:

Executes in ES/EI modes only

SUMMARY:

R1, R2, = 0, 1, 2, 3, 4, 5, 6, 7, A, Q;
 C(R1)(i) AND C(R2)(i) -> C(R1)(I) [i = 0, 1, 2, ..., 35];
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If $C(R1) = 0$, then ON; otherwise, OFF

Negative If $C(R1)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

ANSA

8.6.30 ANSA

ANSA	AND to Storage from A-Register	355(0)
------	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(A)(i) \text{ AND } C(Y)(i) \rightarrow C(Y)(i)$;
 $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or if an illegal repeat is used.

ANSQ**8.6.31 ANSQ**

ANSQ	AND to Storage from Q-Register	356(0)
------	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(Q)(i) \text{ AND } C(Y)(i) \rightarrow C(Y)(i)$;
 $C(Q)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANSX_n

8.6.32 ANSX_n

ANSX _{<u>n</u>}	AND Storage from Index Register <u>n</u>	34 _{<u>n</u>} (0)
--------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0, 1, ..., 7 as determined by op code:
 For i = 0 to 17;
 C(X_n)(i) AND C(Y)(i) -> C(Y)(i);
 C(X_n) and C(Y)(18-35) unchanged

ES/EI Mode

For n = 0, 1, ..., 7 as determined op code:
 For i = 0 to 35;
 C(GX_n)(i) AND C(Y)(i) -> C(Y)(i);
 C(GX_n) is unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL, RPT, or RPD of ANSX0

RPL of any ANSX_n

INDICATORS:

NS Mode

Zero If bits C(Y)(0-17) = 0, then ON; otherwise, OFF

Negative If C(Y)(0) = 1, then ON; otherwise, OFF

ES/EI Mode

Zero If C(Y) = 0, then ON; otherwise, OFF

Negative If C(Y)(0) = 1, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ANX_n

8.6.33 ANX_n

ANX _{<u>n</u>}	AND to Index Register <u>n</u>	36 _{<u>n</u>} (0)
-------------------------	--------------------------------	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0, 1, ..., or 7 as determined by op code:

For i = 0 to 17;

$C(X_{\underline{n}})(i) \text{ AND } C(Y)(i) \rightarrow C(X_{\underline{n}})(i)$

ES/EI Mode

For n = 0, 1, ..., or 7 as determined by op code:

For i = 0 to 35;

$C(GX_{\underline{n}})(i) \text{ AND } C(Y)(i) \rightarrow C(GX)(i)$

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of ANX0

INDICATORS:

NS Mode

Zero If $C(X_n) = 0$, then ON; otherwise, OFF

Negative If $C(X_n)(0) = 1$, then ON; otherwise, OFF

ES/EI Mode

Zero If $C(GX_n) = 0$, then ON; otherwise, OFF

Negative If $C(GX_n)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. All data is handled as 0 when DL modification is specified in the NS mode.

AOS

8.6.34 AOS

AOS	Add One to Storage	054(0)
-----	--------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) + 0\dots01 \rightarrow C(Y)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of bit 0 of C(Y) is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ARAn**8.6.35 ARAn**

<u>ARAn</u>	Address Register <u>n</u> to Alphanumeric Descriptor	54 <u>n</u> (1)
-------------	---	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

<u>ARAn</u>		LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

For $n = 0, 1, \dots, \text{ or } 7$ as determined by op code:
 $C(\text{AR}_n)(0-17) \rightarrow C(Y)(0-17);$
 $C(\text{AR}_n)(18-23) \rightarrow C(Y)(18-20)$ (translated);
 $C(Y)(21-35)$ unchanged

EXPLANATION:

This instruction is the converse of AAR_n. The alphanumeric descriptor is fetched from the computed effective address Y. The TA field code is examined to determine the type of data. Bits 18-23 of AR_n are appropriately translated and replace bits 18-20 of the descriptor, and the word address (0-17) of AR_n replaces bits 0-17. The updated descriptor is then stored back into location Y.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ARAn

ILLEGAL REPEATS:

RPD, RPT, RPL

ILLEGAL EXECUTES:

If this instruction is executed in ES/EI mode

INDICATORS:

None

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used, or if the descriptor TA field contains code 11.
2. AN IPR fault occurs if an attempt is made to execute this instruction in the ES/EI modes.

EXAMPLE:

1	8	16	32

	ARA6	DESCR	AR6 octal contents - 50102407
	.	.	
	.	.	
	.	.	memory contents in octal
DESCR	ADSC9	,,4	501024000004 - DESCR after

ARL**8.6.36 ARL**

ARL	A-Register Right Logical Shift	771(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

Shift C(A) right the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros.

ES/EI Mode

Shift C(A) right the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(A) = 0, then ON; otherwise, OFF

Negative If C(A)(0) = 1, then ON; otherwise, OFF

ARL

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ARN_n**8.6.37 ARN_n**

ARN _n	Address Register <u>n</u> to Numeric Descriptor	64 _n (1)
------------------	--	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

ARN _n		LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

For $n = 0, 1, \dots, 7$ as determined by op code:

$C(ARN_n)(0-17) \rightarrow C(Y)(0-17)$

$C(ARN_n)(18-23) \rightarrow C(Y)(18-20)$ (translated)

bits 21-35 of $C(Y)$ unchanged

EXPLANATION:

This instruction is the converse of NARN_n. The numeric descriptor is fetched from the computed effective address Y and the TN field bit is examined. Bits 0-17 of ARN_n replace the descriptor bits 0-17. Bits 18-23 of ARN_n are appropriately translated and replace bits 18-20 of the descriptor. The updated descriptor is then stored back in location Y.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ARN_n

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

If this instruction is executed in ES/EI mode.

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. An IPR fault occurs if an attempt is made to execute this instruction in ES/EI mode.

ARS**8.6.38 ARS**

ARS	A-Register Right Shift	731(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

Shift C(A) right the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with bit 0 of C(A).

ES/EI Mode

Shift C(A) right the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with bit 0 of C(A).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(A) = 0, then ON; otherwise, OFF

Negative If C(A)(0) = 1, then ON; otherwise, OFF

ARS

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASA**8.6.39 ASA**

ASA	Add to Storage from A-Register	055(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) + C(Y) \rightarrow C(Y)$; $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASQ

8.6.40 ASQ

ASQ	Add to Storage from Q-Register	056(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) + C(Y) \rightarrow C(Y)$; $C(Q)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ASX_n**8.6.41 ASX_n**

ASX _n	Add to Storage from Index Register	04 _n (0)
------------------	------------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(X_n) + C(Y)(0-17) \rightarrow C(Y)(0-17);$
 $C(X_n)$ and $C(Y)(18-35)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(GX_n) + C(Y) \rightarrow C(Y);$
 $C(GX_n)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of ASX0

ASX_n

INDICATORS:

NS Mode

Zero	If $C(Y)(0-17) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of $Y(0-17)$ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

ES/EI Mode

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Y is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

AWCA**8.6.42 AWCA**

AWCA	Add With Carry to A-Register	071(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

If carry indicator is OFF, then:

$C(A) + C(Y) \rightarrow C(A);$

$C(Y)$ unchanged

If carry indicator is ON, then:

$C(A) + C(Y) + 00\dots01 \rightarrow C(A);$

$C(Y)$ unchanged

EXPLANATION:

This instruction operates similarly to the ADA instruction except that if the carry indicator is ON prior to the execution of the instruction, a 1 is added to the least significant position of the A-register.

This instruction is intended for use with multiword precision binary arithmetic and for calculating checksums. The positive 1 added when the carry indicator is ON represents the carry from the next less significant word of the multiword addition.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

AWCA

INDICATORS:

Zero	If C(A) = 0, then ON; otherwise, OFF
Negative	If C(A)(0) = 1, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

EXAMPLE:

(Checksum Calculation)

1	8	16

	LDI	=11324, DL
	LDA	INCARD
	EAX2	INCARD+2
	EAX3	=0
	RPDA	22, 1
	ADLA	0, 2
	AWCA	0, 3
	CMPA	INCARD+1
	TNZ	ERROR
	LDI	=050000, DL

AWCQ**8.6.43 AWCQ**

AWCQ	Add with Carry to Q-Register	072(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

If carry indicator is OFF, then:
 $C(Q) + C(Y) \rightarrow C(Q);$
 $C(Y)$ unchanged

If carry indicator is ON, then:
 $C(Q) + C(Y) + 00\dots01 \rightarrow C(Q);$
 $C(Y)$ unchanged

EXPLANATION:

This instruction operates similarly to the ADQ instruction except that if the carry indicator is ON prior to the execution of the instruction, a 1 is added to the least significant position of the Q-register.

This instruction is intended for use with multiword precision binary arithmetic and for calculating checksums. The positive 1 added when the carry indicator is ON represents the carry from the next less significant word of the multiword addition.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

AWCQ

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of $Q(0)$ is generated, then ON; otherwise, OFF

EXAMPLE:

(Triple-precision Binary Fixed-point Addition)

1	8	16	32
	STI	C	save overflow and overflow mask
	LXL0	C	
	ANX0	=0044000,DU	
	STX0	REST	
	LDA	=1B24,DL	set overflow mask ON
	ORSA	C	
	LDI	C	
	LDQ	A+2	add low-order bits
	ADLQ	B+2	
	STQ	C+2	
	LDQ	A+1	add intermediate bits
	AWCQ	B+1	
	STQ	C+1	
	STI	C	restore overflow/overflow mask
	LDA	=0733777,DL	
	ANA	C	
REST	ORA	** ,DL	
	STA	C	
	LDI	C	
	LDQ	A	add high-order bits
	AWCQ	B	
	STQ	C	

AWD/AWDX**8.6.44 AWD/AWDX**

AWD AWDX	Add Word Displacement to Address Register	507(1)
-------------	--	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{AWD}		
	{AWDX}		word displacement, R, AR

When the mnemonic is coded with X (AWDX), bit 29 is forced to zero.

OPERATING MODES:

Any

SUMMARY:NS Mode

If bit 29 = 0:

$y + C(DR) \rightarrow AR_n(0-17)$

If bit 29 = 1:

$C(AR_n)(0-17) + y + C(DR) \rightarrow AR_n(0-17)$

In either case, 000000 $\rightarrow AR_n(18-23)$

ES/EI Mode

If bit 29 = 0:

$[(se)y + C(DR)](6-35) \rightarrow C(AR)(0-29)$

If bit 29 = 1:

$[(se)C(AR_n) + (se)y + C(DR)](6-35) \rightarrow C(AR)(0-29)$

(se) indicates sign extension

In either case, 000000 $\rightarrow AR_n(30-35)$

AWD/AWDX

EXPLANATION:

NS Mode

The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-17 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is added to bits 0-17 of the specified AR and the resulting sum is stored in bits 0-17 of the specified AR. In either case, bits 18-23 of the specified AR are zeroed.

ES/EI Mode

The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-29 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is added to the sign extended value of the specified AR bits 0-29 and the sum loaded into the specified AR bits 0-29. In either case, bits 30-35 of the specified AR are zeroed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, and IC specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

AWD/AWDX**EXAMPLES:**

(Examples apply to NS mode only)

1	8	16	32

FLD1	BOOL	20100	
	EAX4	FLD1	X4 octal contents - 020100
	AWDX	0,4,7	AR7 octal contents - 02010000
	AWD	2,,7	AR7 octal contents - 02010200
*			
FLD2	BOOL	10000	
	EAX2	FLD2	X2 octal contents - 010000
	EAX3	512	X3 octal contents - 001000
	AWDX	0,2,4	AR4 octal contents - 01000000
	AWD	1,3,4	R4 octal contents - 01100100

AWkD

8.6.45 AWkD

AWkD	Add Word and k(0-3)Byte Displacement to AR _n	00k(1)
------	---	--------

FORMAT:

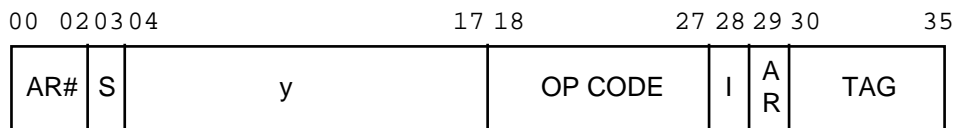


Figure 8-6. AWkD Instruction Format

where:

AR#	Address register number (not affected by value of bit 29)
S	Sign bit
y	Displacement in words; y(4), the sign bit, is extended to the left to form an 18-bit displacement.
OP CODE	10-bit operation code field where bits 25-26 specify a byte displacement value k of 0, 1, 2, or 3 when bits 18-24 = 0000000 and bit 27 = 1.
I	Program interrupt inhibit bit
AR	Address register bit If bit 29 = 1, use address register specified in bits 0-2 in the Effective Address preparation. If bit 29 = 0, address register n is not used in EA preparation.
TAG	Tag field; used to control address modification. Only R modification (except DU, DL, and IC) is permitted. Any other TAG field will cause an IPR fault.

OPERATING MODES:

SE mode only

SUMMARY:

Where $k = 0, 1, 2, 3$

If bit 29 = 0::

$R(\text{word}) : 00 + y(\text{word}) : k(\text{byte}) \rightarrow C(\text{ARn})(0-35)$

If bit 29 = 1::

$\text{ARn}(0-35) + R(\text{word}) : 00 + y(\text{word}) : k(\text{byte}) \rightarrow C(\text{ARn})(0-35)$

Where **R** is the register specified by TAG and **:** denotes concatenation.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, IC, RI, IR, IT, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Indicators are not affected

NOTES:

1. ARn is loaded with the EA + k byte value. EA is formed in normal fashion except that y with sign (S) extended is used as the word displacement regardless of the value of bit 29. Note that this instruction is executable only in SE mode.
2. An Illegal Procedure fault occurs if DU, DL, IC, RI, IR, or IT address modifiers are specified or if the target of a RPT, RPD, or RPL instruction or if execution is attempted in NS, EI, or ES mode.

BCD

8.7 Machine Instruction Description (B)

Following is a detailed description of the processor instructions and operation codes beginning with the letter B.

8.7.1 BCD

BCD	Binary-to-BCD Convert	505(0)
-----	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Shift C(A) left 3 positions;
 $|C(A)| / C(Y) \rightarrow$ 4 bit quotient
 $C(A) - C(Y) * \text{quotient} \rightarrow$ remainder

Shift C(Q) left 6 positions;
 00 \rightarrow C(Q)(30-31)
 4 bit quotient \rightarrow C(Q)(32-35)
 remainder \rightarrow C(A)

EXPLANATION:

The BCD instruction carries out one step of an algorithm for the conversion of a binary number to the equivalent binary-coded decimal, which requires the repeated short division of the binary number or last remainder by a 36-bit constant from memory.

$$c(i) = 8i * 10^{(n-i)} \text{ (for } i = 1, 2, \dots),$$

with n being defined by $10^{n-1} < | \text{ number } | < 10^n$

For base K other than 10:

$$c(i) = 8i * K^{(n-i)}$$

where $K^{n-1} < | \text{ number } | < K^n$

One 6-bit character is produced each time the BCD instruction is executed. The character produced represents a decimal digit from 0 to 9.

The BCD instruction converts the magnitude of the contents of the accumulator to the binary-coded decimal equivalent. The method employed is to effectively divide a number by a constant, place the result in bits 30-35 of the quotient register and leave the remainder in the accumulator. The execution of the BCD instruction allows the user to convert a binary number to BCD, one digit at a time, with each digit coming from the high-order part of the number. The address of the BCD instruction refers to a constant to be used in the division; a different constant is needed for each digit. In the process of the conversion, the number in the accumulator is shifted left three positions. The quotient register is shifted left six positions before the new digit is stored.

The values in the Table 8-1 (on the following pages) are the conversion constants to be used with the binary-to-BCD instruction. Each vertical column represents the set of constants to be used depending on the initial value of the binary number to be converted to its decimal equivalent. The instruction is executed once per digit, using the constant appropriate to the conversion step with each execution.

An alternate use of the table for conversion involves the use of the constants in the row corresponding to conversion step 1. If, after each conversion, the contents of the accumulator are shifted right three positions, the constants in the conversion step 1 row may be used one at a time in order of decreasing value until the conversion is complete.

The largest number that can be converted with this instruction is the number represented in 33 bytes.

BCD

Table 8-1. BINARY-TO-BCD CONVERSION CONSTANTS

	$\frac{-10^{10} + 1}{10^{10} - 1}$	$\frac{-10^9 + 1}{10^9 - 1}$	$\frac{-10^8 + 1}{10^8 - 1}$	$\frac{-10^7 + 1}{10^7 - 1}$	$\frac{-10^6 + 1}{10^6 - 1}$	$\frac{-10^5 + 1}{10^5 - 1}$	$\frac{-10^4 + 1}{10^4 - 1}$	$\frac{-10^3 + 1}{10^3 - 1}$	$\frac{-10^2 + 1}{10^2 - 1}$	$\frac{-10^1 + 1}{10^1 - 1}$
1	$8^1 \times 10^9$	8×10^8	8×10^7	8×10^6	8×10^5	8×10^4	8×10^3	8×10^2	8×10^1	8
2	$8^2 \times 10^8$	$8^2 \times 10^7$	$8^2 \times 10^6$	$8^2 \times 10^5$	$8^2 \times 10^4$	$8^2 \times 10^3$	$8^2 \times 10^2$	$8^2 \times 10^1$	8^2	
3	$8^3 \times 10^7$	$8^3 \times 10^6$	$8^3 \times 10^5$	$8^3 \times 10^4$	$8^3 \times 10^3$	$8^3 \times 10^2$	$8^3 \times 10^1$	8^3		
4	$8^4 \times 10^6$	$8^4 \times 10^5$	$8^4 \times 10^4$	$8^4 \times 10^3$	$8^4 \times 10^2$	$8^4 \times 10^1$	8^4			
5	$8^5 \times 10^5$	$8^5 \times 10^4$	$8^5 \times 10^3$	$8^5 \times 10^2$	$8^5 \times 10^1$	8^5				
6	$8^6 \times 10^4$	$8^6 \times 10^3$	$8^6 \times 10^2$	$8^6 \times 10^1$	8^6					
7	$8^7 \times 10^3$	$8^7 \times 10^2$	$8^7 \times 10^1$	8^7						
8	$8^8 \times 10^2$	$8^8 \times 10^1$	8^8							
9	$8^9 \times 10^1$	8^9								
10	8^{10}									

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPL

BCD**INDICATORS:**

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If before execution bit 0 of $C(A) = 1$, then ON; otherwise, OFF

NOTES:

1. The largest number that can be converted with the BCD instruction is that represented by 33 bits.
2. A 6-bit character is generated in the Q-register each time this instruction is executed.
3. The generated character represents one digit of the values 0-9.
4. One full 36-bit word cannot be directly converted by the BCD instruction.
5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

1	8	16

	LDA	=15, DL
	LDQ	0, DL
	BCD	=80, DL
	BCD	=64, DL

BNCT

8.7.2 BNCT

BNCT	Binary Normalize and Count	410(1)
------	----------------------------	--------

FORMAT:

Register to Register instruction format (see Figure 8-5)

OPERATING MODES:

ES, EI

EXPLANATION:

Binary normalization is performed on the floating-point data (mantissa) stored in C(AQ), and the binary normalization count N is stored in C(R1). The exponent register (ER) is not changed and the hexadecimal mode bit in IR, IR(32), is ignored. If the value of C(AQ) = 0, a value of 72 decimal is stored in C(R1).

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise, OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise, OFF

BNCT

NOTES:

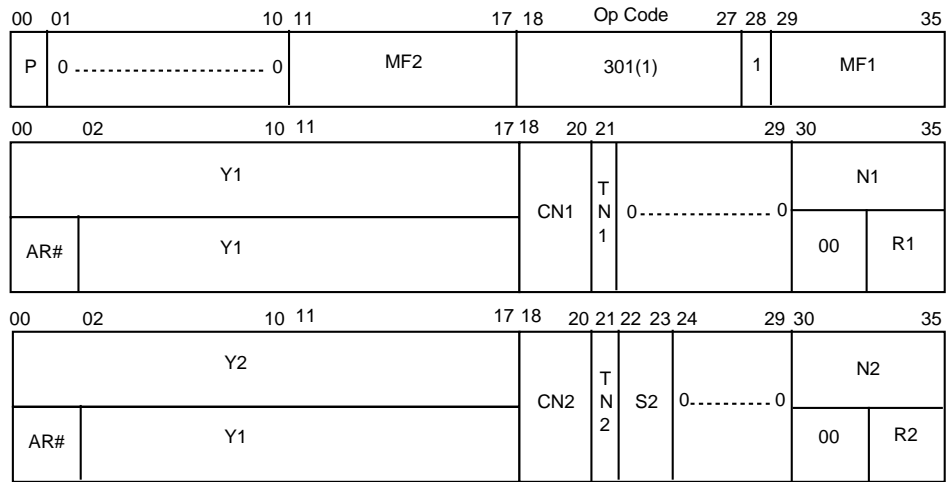
1. If execution is attempted in NS mode, an IPR fault will occur. If this instruction is the target of a RPT, RPD, or RPL instruction, an IPR fault will occur.
2. If R1 designates A or Q, an IPR fault will occur.

BT D

8.7.3 BT D

BT D	Binary-to-Decimal Convert	301(1)
------	---------------------------	--------

FORMAT:



CODING FORMAT:

1	8	16	

		BT D	(MF1) , (MF2) , P
NDSC9			LOCSYM , CN , N , , , AM
NDSC _n			LOCSYM , CN , N , S , , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) -> C(string 2) (converted)

EXPLANATION:

The two's complement binary integer starting at location YC1 is converted into a signed string of decimal characters of data type TN2, sign and decimal type S2 (S2 = 00 is illegal) and scale factor 0; and is stored, right-justified, as a string of length L2 starting at location YC2. If the string generated is longer than L2, the high-order excess is truncated and the overflow indicator is set. If strings 1 and 2 are not overlapped, the contents of string 1 remain unchanged. The length of string 1 (L1) is given as the number of 9-bit segments that make up the string. L1 is equal to or is less than 8. Thus, the binary string to be converted can be 9, 18, 27, 36, 45, 54, 63, or 72 bits long. CN1 designates a 9-bit character boundary. If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If the resultant number generated is zero, then ON; otherwise, OFF
Negative	If the resultant sign is negative, then ON; otherwise, OFF
Overflow	If L2 is less than the length of the string generated, then ON; otherwise, unchanged

BTD

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if an illegal repeat is used.
2. An IPR fault occurs if L1 is less than 1 or greater than 8, if CN1 does not contain a legal code, if S2 = 00, or if N2 is not large enough to specify at least one digit excluding sign.

EXAMPLES:

1	8	16	32

	BTD		
	NDSC9	FLD1,2,2	binary operand descriptor
	NDSC9	FLD2,0,4,1	decimal operand descriptor
	USE	CONST.	memory contents in octal
FLD1	DEC	-512	7 7 7 7 7 7 7 7 7 0 0 0
FLD2	BSS	1	0 5 5 0 6 5 0 6 1 0 6 2
	USE		any indicators set? negative
	BTD		
	NDSC9	FLD1,3,1	binary operand descriptor
	NDSC9	FLD2,1,3,2	decimal operand descriptor
	USE	CONST.	memory contents in octal
FLD1	DEC	255	0 0 0 0 0 0 0 0 0 3 7 7
FLD2	BSS	1	0 0 0 0 6 5 0 6 5 0 5 3
	USE		any indicators set? overflow

9. Machine Instruction Descriptions (C-D)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 9.1, Machine Instruction Descriptions (C)
- Section 9.2, Machine Instruction Descriptions (D)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

9.1 Machine Instruction Descriptions (C)

Following is a detailed description of the processor instructions and operation codes beginning with the letter C.

9.1.1 CAMP

CAMP	Clear Associative Memory Paged	532(1)
------	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

CAMP

EXPLANATION:

This instruction clears (nullifies) the associative memory (AM) in the CPU. The following four operations are specified with the instruction tag bits (bits 30-35).

Normal CAMP

Defined by TAG = 00₈, 01₈, 04₈ + VMOS. All entries of Associate Memory in this CPU are cleared.

Broadcast CAMP

Defined by TAG = 02₈, 03₈, 05₈. + VMOS. All entries of Associate Memory in this CPU and all other active CPUs having the same System Number as the issuing CPU are cleared. The specification of active CPUs is fetched by the CPU Firmware from location 12 (octal) in real Reserve Memory.

Broadcast CAMP T6

Defined by TAG = 06₈. All entries of Associate Memory in this CPU and all other active CPUs having the same System Number as the issuing CPU are cleared. The specification of active CPUs is fetched by the CPU Firmware from location 13 (octal) in real Reserve Memory.

ILLEGAL ADDRESS MODIFICATIONS:

Ignored

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

CAMP

NOTES:

1. Since the execution of a CAMP instruction in the initiating processor may be lengthy, the LUF timer is reset in the initiating processor repeatedly as the time response is monitored.
2. Reserve Memory locations 12 and 13 are maintained by the Service Processor, with bit n of the word in that location defining whether physical CPU#n is or is not active. If bit n = 1, CPU#n is active and a CAMP COMPLETE must be received from that CPU within 32 msec of the issuance of a Broadcast CAMP command on the system bus. If bit n = 0, CPU#n is logically or physically disconnected from the system and a response to a Broadcast Camp is not expected.
3. In NVM or VMM mode, if responses are not received within 32 msec, C(A) bit 0 is set to one before the processor advances. It is software's responsibility to initialize the A register and test it following the CAMP instruction.
4. In VMOS mode, failure to receive all CAMP COMPLETE responses within 32 msec will cause a VM Unsuccessful CAMP fault to occur. In VMOS mode, the C(A) is not changed on the VMUC.
5. If TAGs other than 00₈, 01₈, 02₈, 03₈, 04₈, 05₈, or 06₈ are used, an IPR fault will occur.
6. An IPR fault will occur if the CAMP instruction is used as the target of a RPT, RPD, or RPL instruction.
7. A Command fault will occur if the CAMP instruction is attempted in slave or master mode.

CANA

9.1.2 CANA

CANA	Comparative AND with A-Register	315(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(Z)(i) = C(A)(i) \text{ AND } C(Y)(i)$
 $C(A)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

CANAQ**9.1.3 CANAQ**

CANAQ	Comparative AND with AQ-Register	317(0)
-------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 71 :
 $C(Z)(i) = C(AQ)(i) \text{ AND } C(Y\text{-pair})(i)$
 $C(AQ)$ and $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

CANQ

9.1.4 CANQ

CANQ	Comparative AND with Q-Register	316(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(Z)(i) = C(Q)(i) \text{ AND } C(Y)(i)$
 $C(Q)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

CANX_n**9.1.5 CANX_n**

CANX _n	Comparative AND with Index Register <u>n</u>	30 <u>n</u> (0)
-------------------	---	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code

For $i = 0$ to 17;

$$C(Z)(i) = C(X_n)(i) \text{ AND } C(Y)(i)$$

$C(X_n)$ and $C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code

For $i = 0$ to 35;

$$C(Z)(i) = C(GX_n)(i) \text{ AND } C(Y)(i)$$

$C(GX_n)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of CANX0

CANX_n

INDICATORS:

Zero	If $C(Z) = 0$, then ON; otherwise, OFF
Negative	If $C(Z)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal by the assembler but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CBMX_n**9.1.6 CBMX_n**

CBMX _n	Compare Byte Masked with X _n	07 _n (1)
-------------------	---	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any except NS mode.

SUMMARY:

The most significant byte of GX_n, C(GX_n)(0-8), are masked (AND) with the second most significant byte of the memory word, C(Y)(9-17), to form a test byte, T(0-8), which is arithmetically compared with the most significant byte of the memory word, C(Y)(0-8). Indicators are set as a result of the comparison as defined below.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None except RPT, RPD, RPL for CBMX0

CBMX_n

INDICATORS:

Indicator			Relationship	
Zero IR(18)	Negative IR(19)	Carry IR(20)	Byte Signs	Byte Value
0	0	0	$T(0) = 0, C(Y)(0) = 1$	$T > C(Y)(0-8)$
0	0	1	$T(0) = C(Y)(0)$	$T > C(Y)(0-8)$
0	1	0	$T(0) = C(Y)(0)$	$T < C(Y)(0-8)$
0	1	1	$T(0) = 1, C(Y)(0) = 0$	$T < C(Y)(0-8)$
1	0	0	can not occur	
1	0	1	$T(0) = C(Y)(0)$	$T = C(Y)(0-8)$
1	1	0	can not occur	
1	1	1	can not occur	

NOTE:

An Illegal Procedure fault occurs if CI, SC, or SCR address modification is used or if execution is attempted in NS mode.

CCAC**9.1.7 CCAC**

CCAC	Clear Cache	011(1)
------	-------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

Because the CPU has a cache auto-flush function, no operation is executed.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Illegal repeats cause an IPR fault.
2. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

CIOC

9.1.8 CIOC

CIOC	Connect Input/Output Channel	015(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

A connect word pair entry is sent to a connect queue in reserve memory, and an interrupt is sent to a system bus port. The data is formed from the contents of the CPU A-register and an entry in the Connect Table in reserve memory and the developed absolute address

EXPLANATION:

The CIOC instruction provides three types of connect:

1. CPU to CPU Connect: A CPU-CIOC interrupt is sent on the system bus to the designated CPU.
2. CPU to IO Direct Connect: An IO-CIOC interrupt is sent on the system bus to the designated IO Controller (IOC), and a two-word entry is placed in the IOC connect queue. The entry defines the logical channel number for the connect, the type of connect command, the real address of the mailbox, and the System ID of the OS initiating the connect. See details below.

Prior to the execution of a CIOC instruction, the software must load the A register with the information defined below:

Bit(s)	Field	Description
0-8	CON Cmd	Defines a connect command for IOC firmware
9	LVM	Load VM bases from Reserve Memory table
10-17	RFU	Reserved for Future Use
18-35	CTBL Entry	Offset into connect table

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. When the C(Y)(33-35) code is used for an unconnected CPU or IOC, a command fault occurs.
2. A command fault occurs if the use of this instruction is attempted by a processor in the Slave mode or Master mode, the CTW is not valid for this LC, CTW contains invalid CPU connect code, or CTW contains invalid IO connect code.
3. C(Y)(0-32) should be set to 0 to provide compatibility with future systems.
4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
5. A QOVFL fault occurs if connect was successful but used the last available slot in the IOC connect queue.
6. Execution of this instruction requires CPU firmware accesses to real Reserve Memory Space where the connect queues and pointers are maintained. No bounds check are made on accesses to reserve memory. Limit register contents are ignored.
7. The SP is responsible for loading the real base address of the CTBL and the CPTBL into XRAM.
8. The CTBL is defined to consist of 2080 entries: 32 for CPU and 2048 for logical channels. The hardware will use the entire lower half of the A register to be added to the CTBL base address from the XRAM and could possibly form an address outside the range of the CTBL. It is the responsibility of the OS to ensure that the contents of A(18-35) do not cause reference outside the CTBL.

CIOC

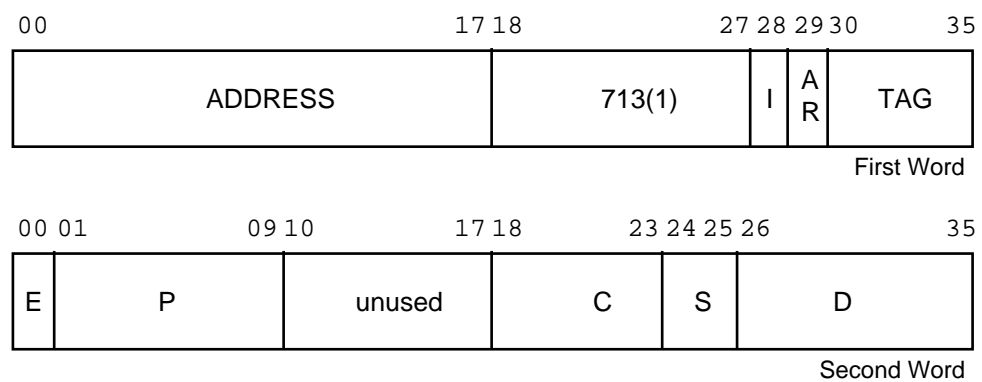
EXAMPLE:

1	8	16	32

IOMCON	STZ	IOMST,3	clear status word
	CIOC	KLCIC,1,P5	connect to IOC
	EAX3	,3	regular or alternate I/O?
	TNZ	RESTA	if alternate, return
	EAX7	IOMST	
	STX7	ISIRG	return status at I/O complete
	STZ	IOSTS	reset important flags if fault
	STZ	INTWS	
	TSX7	STARTA	start alt. I/O if possible
	ARG	TAPE2I	
	TDCW	TPDCW	
	TRA	IODIS	go wait for interrupt

CLIMB**9.1.9 CLIMB**

CLIMB	Domain Transfer	713(1)
-------	-----------------	--------

FORMAT:

The first word has the standard single-word instruction format (see Figure 8-1). The second word of the CLIMB instruction contains four control fields:

1. C(22-23) }
 } the two fields that compose C
2. C(18-19) }
3. E and P
4. S and D.

Field C (1. and 2. above) is composed of *two* fields.

Bits 10-17 and 20-21 are not interpreted.

OPERATING MODES:

Any

CLIMB

EXPLANATION:

This instruction has four variations. It performs functions of call, return, and common routine calls within the same or to a different instruction segment and within the same domain and to a different domain reference.

The instruction word bit 28 (interrupt inhibit bit does not accept interrupt for three of the four functions whether it is set to zero or to one. Bit 28 determines acceptance of interrupt for the other function.

The AR bit (bit 29) specifies whether the address register is to be used to generate effective addresses. The tag field is also for address generation.

Versions of the CLIMB instruction are listed below.

Mnemonic	Meaning
ICLIMB (Inward CLIMB - CALL)	call another procedure that may reside in another domain
OCLIMB (Outward CLIMB - RET)	return to calling domain
GCLIMB (Lateral Transfer - LTRAS)	transfer to another procedure with passed arguments and parameters which may reside in another domain
PCLIMB (Lateral Transfer - LTRAD)	transfer to another procedure which may be in another domain
PMME (System Entry CLIMB)	Privileged Master Mode Entry (This is a form of Inward CLIMB.)

CLIMB

The four control fields of the second word are defined below.

C22,23 Instruction Version

This field determines one of the five versions (including PMME) of the instruction to be executed:

- 00: Inward CLIMB (ICLIMB) Version -
 functions as a CALL, i.e., a procedure invokes another procedure
 to accomplish a task and expects return of control from that other
 procedure .
 Additional descriptors may be passed in a new parameter segment.
 An empty argument segment is created and placed in the argument
 stack. The processor state is saved (safe stored) if the SSF flag of
 the option register = 1. S,D = 0,1760 is the PMME version
 (System Entry). S,D <> 0,1760 is the ICLIMB version.
- 01: Outward CLIMB (OCLIMB) Version (RET) -
 functions as a return to the caller
 The processor state is restored to the last safe store frame.
- 10: Lateral Transfer with same Parameter and Argument Segments
 (LTRAS)
 This version functions as an unconditional transfer, giving the
 callee the same visibility as the caller. The processor state is not
 saved. LTRAS is also called GCLIMB.
- 11: Lateral Transfer with new Parameter and Argument Segments
 (LTRAD)
 This version functions the same as the CALL version, except that
 the processor state is not saved. LTRAD is also called PCLIMB.

The terms inward, outward, and lateral refer to how the stack segments are used. Inward refers to the following procedure: push the safe store frame on the safe store stack (saving the present processor state), frame a new parameter segment (PS), and open a new (empty) argument segment (AS). Outward means pop the safe store frame off the safe store stack (restoring the former processor state) and return PSR, ASR, LSR, ISR, IC, IR, SEGID(IS), DSAR, and if specified, AR0-AR7, SEGID0-SEGID7, DR0-DR7, X0-X7, A, Q, E and the Pointer/Length registers, to their previous settings. Lateral means leave the safe store stack unchanged. The LTRAS version (10) keeps the PSR and ASR unchanged, while the LTRAD version (11) activates new PSR and ASR values in the same manner as an Inward CLIMB.

CLIMB

C(18) X0/GX0 Control

For a CALL, LTRAS, or LTRAD, the C(18) bit allows the caller to load the effective address of the CLIMB instruction into X0/GX0 if C(18) = 1 and if an entry descriptor is referenced during execution of the CLIMB. For a RET, only the condition C(18) = 1 is required to cause X0/GX0 to be loaded with the effective address of the CLIMB. If C(18) = 0, X0/GX0 is not loaded, regardless of CLIMB version.

C(19), Slave Mode

For a CALL, LTRAS, or LTRAD, the C(19) bit allows Slave mode to be set. For an RET, C(19) is ignored. If the CLIMB is the result of a fault interrupt, or invokes the System Entry (PMME), the C(19) bit is overridden, and the Master Mode indicator is set.

Otherwise for CALL, LTRAS, or LTRAD

```
if C(19) = 0; 0 -> C(IR)(28)
if C(19) = 1; no change to C(IR)(28)
```

If a CALL, LTRAS, or LTRAD attempts to transfer to a privileged segment (flag bit 26 = 1) and C(19) = 0, an SCL1 or Security Fault, class 1 occurs.

E and P Argument Passing

The E and P fields are interpreted only for the ICLIMB (CALL) and PCLIMB (LTRAD) versions of the CLIMB instruction.

If E = 1, P+1 descriptors are passed to the called routine. These descriptors are either prepared (shrunk and pushed onto the argument stack) by the instruction, or they are found in a descriptor segment, depending on the contents preset by the caller in DR0. When DR0 refers to an operand segment, a vector list is interpreted by the instruction to prepare descriptors. When DR0 refers to a descriptor segment, the descriptors are in the segment. In both cases, the PSR is loaded with a type 1 descriptor, framing the P+1 descriptors of parameters (or one, if the P field is zero).

If E = 0, no parameters are passed. The P field is ignored.

In both cases, the ASR is updated in such a way that it locates the next available even-word location of the descriptor stack. The bound field is set to zero. The flag bit 27 is set to zero to indicate an empty segment. Details related to the PSR and the ASR are provided later in the CLIMB discussion.

The E and P fields are not interpreted for the RET and LTRAS versions of the CLIMB instruction.

CLIMBS, D Field

For CALL, LTRAS, or LTRAD, this field indicates the origin (SEGID) of the descriptor that determines the destination of the CLIMB, or that the CLIMB is a System Entry (PMME).

For the outward climb (RET), this field is ignored.

Instruction Variations

The following pages describe the CLIMB variations determined by the settings in bits 22 and 23 of the C field. When the CLIMB instruction is executed, a number of checks must be performed before the CPU state is altered. Inward CLIMB (CALL/ICLIMB) C field bits 22 and 23 = 00

9.1.9.1 Inward CLIMB (CALL/ICLIMB) C field bits 22 and 23 = 00

1. The S and D fields are interpreted in the same manner as the S and D fields of the vector in the LDD_n instruction, except that in this instance the values S = 0 and D = 1760 (octal) define a PMME. If S = 0 and D = 1761 or 1763-1767 (octal), an IPR fault occurs.
 - a) When S = 0, D = 1760 (octal), a special system entry is started at the same level as fault and interrupt. In this case, hardware obtains the segment descriptor (must be an Entry Descriptor) from a fixed memory location. The Master Mode indicator is always set to ON, and the C field bit 19 is ignored. After the entry descriptor is obtained from the fixed memory location, execution of the CLIMB instruction is continued as is the case when a normal entry descriptor is obtained. When no entry descriptor is in the fixed memory location, an IPR fault occurs.
 - b) If the CLIMB is a result of a fault or interrupt, an interdomain transfer requiring an entry descriptor is obtained from locations in the operating system:
 - Interrupt: 30-31 octal
 - Fault: 32-33 octal
 - PMME: 34-35 octal.

CLIMB

2. The CLIMB instruction S and D fields are used to access the specified segment descriptor segment or register and to obtain the segment descriptor. The referenced descriptor must be one of the following types in order to continue execution of the CLIMB instruction:
 - a) Standard Descriptor (T = 0)
 - b) Extended Descriptor (T = 12)
 - c) Descriptor Segment Descriptor (T = 1 or 3)
 - d) Entry Descriptor (T = 8, 9, or 11)

If the CLIMB instruction has not yet been linked to one of the preceding descriptors, the obtained descriptor may be a dynamic linking descriptor (T = 5). In this case, the CLIMB instruction is terminated and a Dynamic Linking fault is generated. All other descriptor types (T = 2, 4, 6, 7, 10, or 12-15) terminate the CLIMB instruction and cause an IPR fault.

Given a descriptor segment descriptor, an entry descriptor, or a standard descriptor, the activity varies in the following ways.

- a) Standard Descriptor (T=0 or 12)

When the descriptor referenced by the S and D fields is either a standard or extended descriptor, the CLIMB instruction is an intradomain transfer, and the linkage segment register is not changed.

NOTE: When a segment descriptor of type 12 is loaded into the ISR, an IPR fault occurs if the segment descriptor bit 24 is **not equal** to 1.

The obtained descriptor becomes the new instruction segment descriptor. Flag bits 25, 27, and 28 are checked and must be 1. Otherwise, an appropriate fault occurs. The base and bound are checked for modulo 32 bytes. If the test fails, an IPR fault occurs.

- b) Descriptor Segment Descriptor (T = 1 or 3)

When a type 1 or 3 descriptor is referenced by the S and D fields of the CLIMB instruction, the base of the type 1 or 3 descriptor is used as a pointer to an entry descriptor. Flag bits 20, 27, and 28 must be 1 and the bound field must be ≥ 7 bytes. Otherwise, a Bound fault occurs. If the obtained descriptor is not an entry descriptor nor a dynamic linking descriptor, an IPR fault occurs.

If a dynamic linking descriptor is obtained, a Dynamic Linking fault occurs.

CLIMB

c) Entry Descriptor (T = 7, 8, 9, or 11)

When an entry descriptor is referenced by the S and D fields of the CLIMB instruction (either directly or indirectly), the CLIMB instruction is an interdomain transfer. In this case, entry descriptors may be of type T = 7, 8, 9, or 11. The type of entry descriptor determines how much data (register contents) will be safe stored, and how the renewal of the pointer register will be processed.

Using the entry descriptor, the new instruction segment descriptor is obtained from the new linkage segment described by the entry descriptor. The new linkage segment is assumed to be present in real memory, because the entry descriptor does not have a flags field to indicate this, and the hardware attempts to obtain the new instruction segment descriptor.

When the special entry descriptor of type T = 7 is designated, the ISR must be loaded with the segment descriptor of type T = 12 and with bit 24 = 1. Otherwise, an IPR fault occurs.

The segment descriptor type designated by the DSEG NO. field must be a type T = 12. Otherwise, an IPR fault occurs. This segment descriptor base field indicates the entry location in the instruction segment. The other fields are not used.

When the entry descriptor type is 8, 9, or 11, the obtained instruction segment descriptor must be a standard descriptor with T = 0 and flag bits 25, 27, and 28 must be 1. If flag bit 25 is 0, a Security Fault, Class 2 occurs. If flag bit 28 = 0, a Missing Segment fault occurs. If flag bit 27 = 0, an STR fault occurs. The hardware also checks the base and bound of the new instruction segment descriptor for modulo 32 bytes. If the test fails, the instruction terminates in an IPR fault. If T is not 0, an IPR fault occurs.

CLIMB

3. A new parameter segment is prepared as described below.

The E bit of the second word of the CLIMB instruction is checked. If the E bit = 0, the segment descriptor is not passed (no parameter segment is prepared), and the operation proceeds to the safe store.

If the E bit = 1, the segment descriptor is passed. In this case, the following operation depends on the types of the segment descriptor in DR0. An IPR fault occurs if the type for this segment is 3, 5, 7-11, 13 or 15.

- a) Descriptor Type in DR0 = 1

If the descriptor type contained in DR0 is 1, the descriptors to be passed as parameters have already been prepared and are the last P+1 descriptors in this descriptor segment. Thus, the hardware does not prepare any descriptors but frames these last P+1 descriptors with the parameter segment register. In this case, hardware performs a bound check, and if $P + 111 > DR0$, a bound fault occurs.

- b) Descriptor Type in DR0 = 0, 2, 4, 6, 12, or 14

If the descriptor type contained in DR0 is 0, 2, 4, 6, 12, or 14, the hardware prepares descriptors. The vector list is located by pointer register zero (i.e., AR0 and DR0 combined). The descriptor identified by the S and D fields of each vector is obtained, prepared exactly as described in the definition of the LDD_n instruction, and placed in the next available location in the argument segment as described in the definition of the SDR_n instruction. This procedure is continued until all P+1 descriptors have been prepared and placed in the argument segment. Various faults may occur during this operation as described in the definitions of the LDD_n and SDR_n instructions. A vector with an S and D field of S = 0, D = 1761 (octal) causes an IPR fault. S and D field values of S = 0, D = 1763 or 1764 (octal) require that the processor be in Privileged Master Mode (as described in LDD_n), which in this case refers to the processor mode at the beginning of the CLIMB instruction.

If a vector specifies that a data stack descriptor is to be formed and the associated bit in the option register specifies that the stack space is to be cleared, the CLIMB instruction performs the clear function.

If several data stack shrinks are specified, the second and subsequent data stack shrink operations are performed using the previously changed new value of the data stack address register (DSAR).

CLIMB

4. Safe Store Operation

The safe store operation differs depending on the type of the segment descriptor referenced with the ICLIMB S and D fields. The size of the generated safe store frame and the stored data is determined by the referenced segment descriptor. The SSR base indicates the starting address of the last frame of the stored data before this CLIMB. The size of the last frame must therefore be added to the SSR base before the new frame is stored. In relation to the SSR, a 2-bit hardware control register called the stack control register (SCR) is used. The SCR contains a code indicating the size of the last frame placed in the safe store stack (the SCR is initialized to 11 or 10 (binary) when the LDSS instruction is executed). (Refer to details for the LDSS instruction.) The following table displays the flow of the safe store operation. When the safe store bypass flag (option register bit 19) is ON (zero), safe store is bypassed and processing proceeds to change the register contents as described under Loading the Registers.

- a) The SSR base is increased, and the bound decreased, based on the SCR content.

<u>SCR</u>	<u>SSR Base</u>	<u>SSR Bound</u>
00 ₂	+16 words	-16 words
01 ₂	+24 words	-24 words
10 ₂	+80 words	-80 words
11 ₂	+64 words	-64 words

The SSR base indicates the start of the newly generated safe store frame as a result of this operation.

If the SSR bound < 159 words + 3 bytes as a result of the SSR adjustment, a safe store stack fault occurs.

NOTE: When hardware adds the SSR base, no check is performed to check for carry. Software must ensure that the base value initially loaded into the SSR is not at the end of the working space.

- b) The SCR content is saved in the new safe store frame.

CLIMB

- c) The new SCR value is determined in the following way, with the lower two bits of the type field (T) of the first word of the last segment descriptor referenced by the CLIMB instruction S and D fields, and the value of bit 24 of the ISR before the start of the CLIMB instruction.

<u>T Field</u>	<u>ISR bit 24</u>	<u>SCR</u>
0, 8, or 12	0 or 1	00 ₂
9	0 or 1	01 ₂
11	0	11 ₂
11	1	10 ₂

- d) The amount of stored data (register content) is determined by the SCR value at this time (as described in item c above). The value of the SCR at this time is determined by the type of segment descriptor referenced by this CLIMB instruction and the ISR bit 24). As illustrated in Figure 9-1, 16, 24, 64, or 80 words are stored in accordance with the SCR content.

Machine Instruction Descriptions (C-D)

CLIMB

10 11 01 00 SCR		1 0 0/1 0/1 ISR(bit 24)			Octal	
				High Water Mark Register [20]	Steering data on ISCF, else unused	0
				Micro instruction fault data [36]		1
				unused [36]		2
		1		IC (EI/SE mode) [36]		3
		6		IC (NS/ES mode) [18]	Indicator Register [18]	4
				Fault Flags & Code [18]	n LPNR [SCR] SEGID(IS) [12]	5
				DSAR [17]	0 EWSQ#U(XV)/unused EWSQ#L(XV)/EWSQ#	6
		2		Relative Virtual Address or EMU Real Page Address (byte) [36]		7
		4		ISR [72]		10
				ASR [72]		11
				LSR [72]		12
		6		PSR [72]		13
		4		ARn (NS mode); n=0-7 [24]		14
				SEGIDn; n=0-7 [12]		15
				DRn; n=0-7 [72]		16
		8		X0 [18]	X1 [18]	17
		0		X2 [18]	X3 [18]	20
				X4 [18]	X5 [18]	27
				X6 [18]	X7 [18]	30
				AQ [72]		...
				E [8]	unused [28]	47
				TR or VMTR(if VMOS) [27]	unused [9]	50
				Pointer/Length data [36]		51
				LOR [72]		52
				GXn; n=0-7 [36]		53
				ARn (ES, EI, SE modes) [36]		54
				unused		55
						56
						57
						60
						65
						66
						67
						70
						77
						100
						107
						110
						117

Figure 9-1. Safe Store Stack Format

Some of the fields shown in Figure 9-1 are stored only with a CLIMB instruction executed by hardware in response to faults or interrupts, and are meaningless when using the programmed CLIMB instruction.

CLIMB

The following discussion explains the contents of the safe store stack as illustrated in Figure 9-1.

Word 0:	
Bits 0-19	Content of HWMR when CLIMB begins execution
Bits 20-35	Steering data on Inter-System Connect Fault, else unused.
Word 1:	Micro-instruction Fault Data
Word 2:	unused
Word 3:	Instruction Counter value, in EI mode
Word 4:	
bits 0-17	Instruction counter (IC) value, in NS or ES mode
bits 18-35	Indicator register (IR) contents
Word 5:	
bits 0-17	Fault Flags and Code:
	<u>bit</u> <u>flag</u>
	0 must be zero
	1 must be zero
	2 ignored (Relief fault)
	3 reserved for hardware use
	4 ignored (SP fault)
	5 IT, RP, EX Fault Flag
	6 ignored (ICP mode)
	7 ignored (Firmware Intercept)
	8 Interrupt/Fault
	9 Recoverable Flag
	10 Safestore Stack Overflow Fault
	11-17 Fault Code
bit 18	unused
bits 19-21	Logical Processor Number Register
bits 22, 23	Stack Control Register
bits 24-35	SEGID(IS)

CLIMB

Word 6:	
bits 0-16	The value stored here is the DSAR content when the CLIMB instruction started.
bit 17	0
bits 18-26	unused
bits 27-35	Effective Working Space Quarter number
Word 7:	Relative Virtual Address (byte) or EMU Real Page Address (DPS 9000G systems only).
Words 8-9: (10-11 octal)	Instruction Segment Register
Words 10-11: (12-13 octal)	Argument Stack Register
Words 12-13: (14-15 octal)	Linkage Segment Register
Words 14-15: (16-17 octal)	Parameter Segment Register
Words 16-23: (20-27 octal)	
bits 0-23	AR0-AR7 - Address Registers(NS mode)
bits 24-35	SEGID0-SEGID7
Words 24-39: (30-47 octal)	DR0-DR7 - Segment Descriptor Registers
Words 40-43: (50-53 octal)	
bits 0-17	X0, X2, X4, X6 - Index Registers
bits 18-35	X1, X3, X5, X7 - Index Registers
Words 44-45: (54-55 octal)	AQ register
Word 46: (56 octal)	
bits 0-7	Exponent Register
bits 8-35	unused

CLIMB

Word 47: (57 octal)	
bits 0-26	Timer Register
bits 27-35	unused
Words 48-53: (60-65 octal)	Pointer Length data Hardware stores information for restart of instruction execution only in response to faults and interrupts. The information stored in this area is normally the content of the pointers and lengths register when a fault or interrupt occurs during execution of an interruptible multiword instruction (when saved with the IR bit 30 set to ON). Even when the IR bit 30 is not set to ON, information is stored in this area, for example, for a missing page fault. The content of this area must not be changed by software.
Words 54-55: (66-67 octal)	Low Operand Register
Words 56-63: (70-77 octal)	GX0-GX7 - General Index Registers
Word 64-71: (100-107 octal)	AR0-AR7 (ES, EI modes) - Address Registers
Words 72-79: (110-117 octal)	unused

5. Loading the Registers

After the state is saved in the safe store stack, the registers are changed as described below.

a) Loading the Instruction Segment Register (ISR)

For an intradomain transfer, the standard descriptor referenced by the S and D fields of the instruction is placed in the ISR. If the S and D fields referenced a DR_n (177n), then $SEGID_n \rightarrow SEGID(IS)$. Otherwise, S and D $\rightarrow SEGID(IS)$.

For an interdomain transfer, the descriptor pointed to by the ISEGNO field of the entry descriptor is loaded into the ISR. $SEGID(IS)$ is set to S = 3, D = ISEGNO.

b) Loading the Instruction Counter (IC)

For an intradomain transfer, an effective address is formed by using the address field of the CLIMB instruction and then applying the indicated AR and/or tag field modification. Then the Instruction Counter is loaded in one of the following ways:

from NS/ES to NS/ES mode

$Y(0-17) \rightarrow C(IC)(0-17)$

from NS to EI mode

$00\dots0 \rightarrow C(IC)(0-15)$

$Y(0-17) \rightarrow C(IC)(16-33)$

from ES/EI to EI mode

$Y(0-33) \rightarrow C(IC)(0-33)$

from EI to ES/NS mode

$Y(16-33) \rightarrow C(IC)(0-17)$

c) Loading the Linkage Segment Register (LSR)

For an intradomain transfer, the linkage segment does not change.

For an interdomain transfer, a standard descriptor from the entry descriptor is placed in the LSR:

Base = Linkage base (LBASE) with zeros in the 10 most significant bit positions

Size = Linkage bound (LBOUND) extended with three 1 bits on the right and with zeros in the 7 most significant bit positions

WSR = WSR (working space register)

T = 1

CLIMB

Flags:

Bits 20, 22, 23, 27, and 28 = 1
 Bits 21, 24, 25, and 26 = 0

For an interdomain transfer, the Instruction Counter is loaded in one of the following ways:

from NS/ES to NS/ES using T = 8, 9, 11

entry loc of entry descriptor -> C(IC)(0-17)

from EI to NS/ES using T = 8, 9, 11

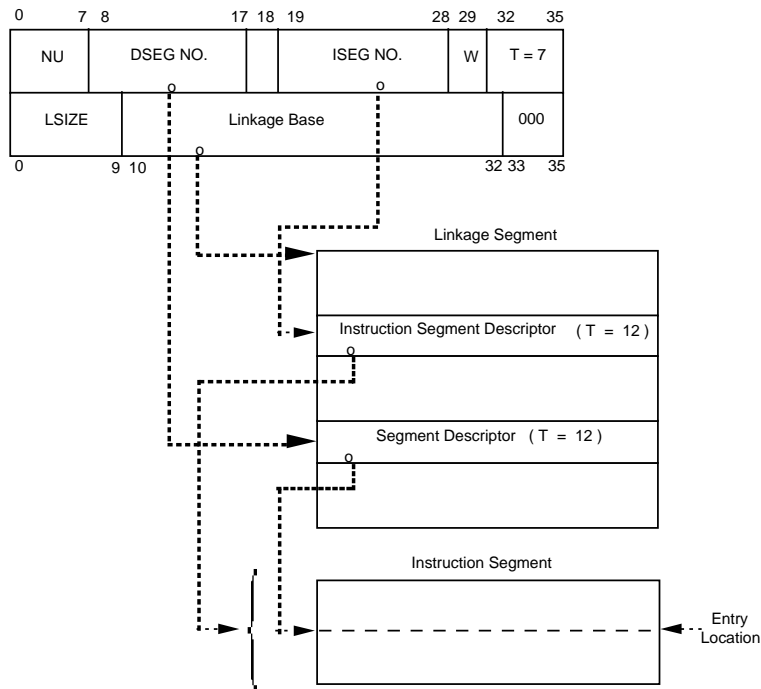
entry loc of entry descriptor -> C(IC)(0-17)

from EI to EI using T = 8, 9, 11

entry loc of entry descriptor -> C(IC)(16-33)
 00...0 -> C(IC)(0-15)

NS/ES/EI to EI using T = 7

base field bits 0-33 of the segment descriptor designated by DSEG NO. of the entry descriptor -> C(IC)(0-33)



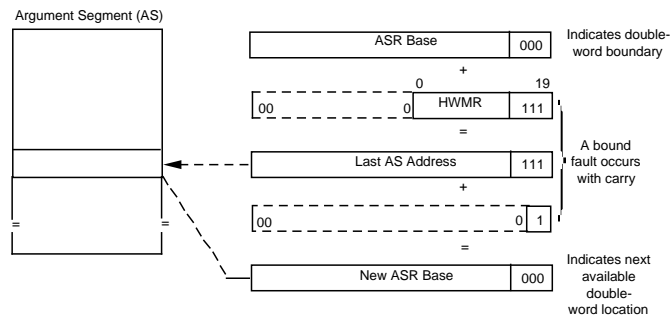
- d) Adjust the argument stack register (ASR) and the parameter segment register (PSR) in the following way:

if E bit = 0 (pass no parameters),

set PSR flag bit 27 to 0 to indicate bound not valid;

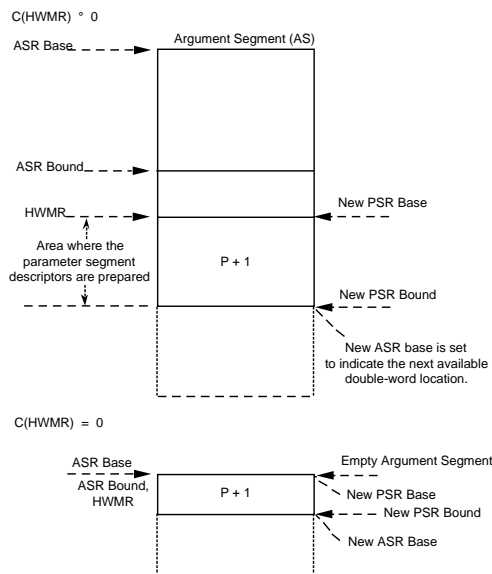
if C(HWMR) = 0, the ASR base does not change;

if C(HWMR) <> 0, set the ASR base as shown below.



The new ASR bound and flag bit 27 are set to 0 for both conditions of C(HWMR).

If E bit = 1 (pass parameters) and DR0 type = 0, 2, 4, 6, 12 or 14, the parameter segment is prepared by using vectors from the segment defined by DRO. These parameters are placed on the argument stack, beginning at an address defined by C(HWMR) rather than by ASR Bound (as in the SDRn instruction). (See the illustration below.)



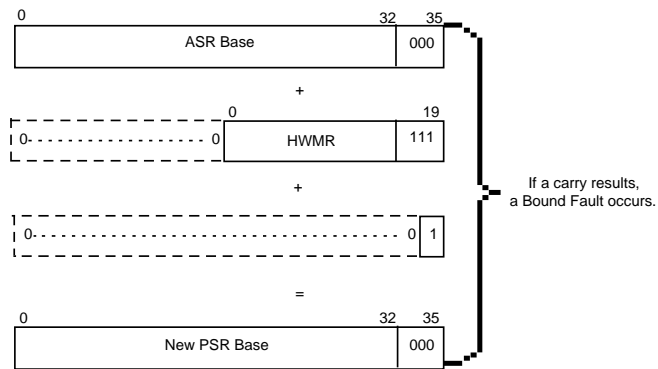
CLIMB

The new PSR Base is generated in the following manner.

When $C(HWMR) = 0$

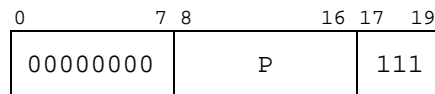
ASR Base \rightarrow New PSR Base

When $C(HWMR) \neq 0$



The new PSR base is used as the starting address in the area where segment descriptors are prepared as parameters.

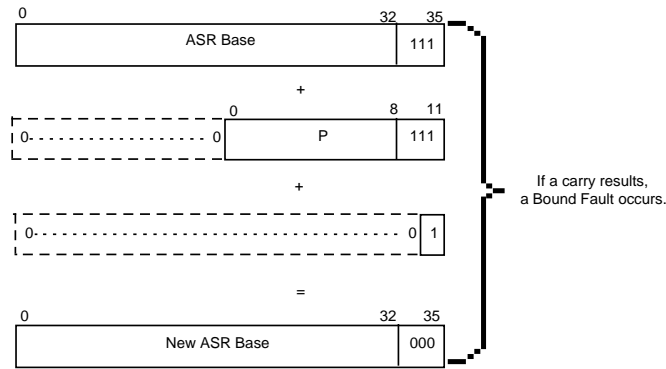
The new PSR bound is generated in the following way for both conditions of $C(HWMR)$.



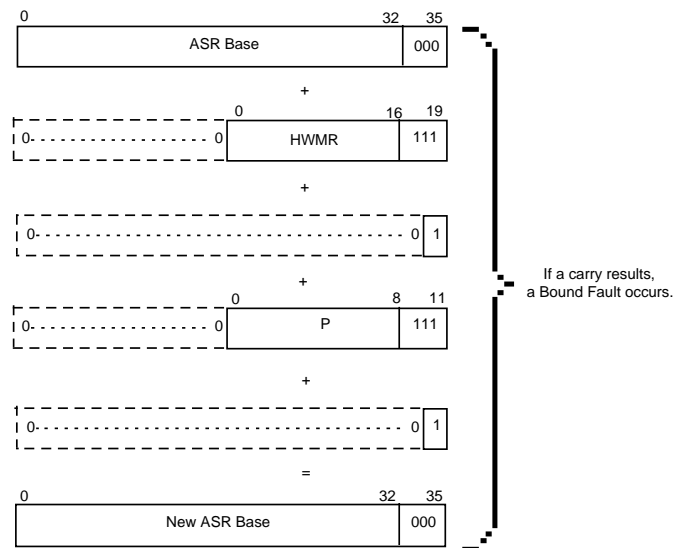
The ASR flag field (with the exception of bit 27) is copied into the PSR flag field. (This copy subordinates the PSR to the ASR, i.e., the PSR frames a portion of the space that was framed by the ASR and should not be allowed to grant additional privileges to the space control.) This process also sets bit 27 of the PSR to 1 to indicate that it is not empty.

The new ASR base is generated in the following way.

When $C(HWMR) = 0$



When $C(HWMR) \neq 0$



The new ASR bound and flag bit 27 are set to 0 for both conditions of $C(HWMR)$.

The ASR flag bit 27 is set to zero to indicate that this segment is empty, and the ASR bound field is set to zero.

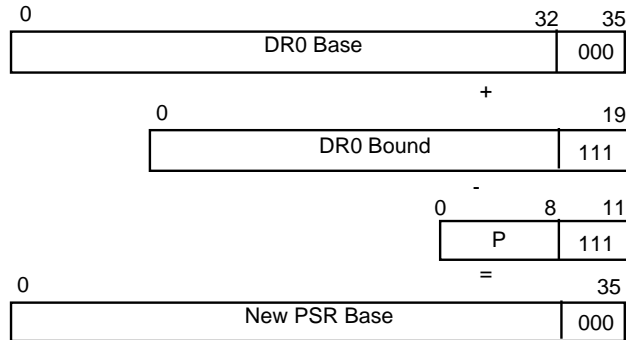
If $E \text{ bit} = 1$ and $DR0 \text{ type} = 1$,

The descriptors to be framed by the PSR are the last $P+1$ descriptors in the descriptor segment pointed to by $DR0$;

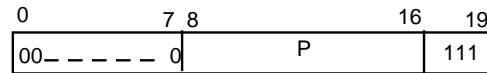
CLIMB

The ASR base and bound are adjusted exactly as described for the case when the E bit is 0. ASR flag bit 27 is also set to zero.

The new PSR base is set to the value DR0 base + DR0 bound - P as shown below.



The new PSR bound is generated in this way.



The new base and bound values formed are loaded into the PSR, framing the last P+1 descriptors of the segment. Bits 20-35 of the first word of DR0 (flags field, WSR or WSN field, and T field) are copied to the corresponding bit positions of the PSR.

e) Loading the Pointer Registers

If type 11 entry descriptor was referenced by the S and D fields of the ICLIMB instruction, all pointer registers are set to the value of the target IS.

ISR	->	DR0 through DR7
SEGID (IS)	->	SEGID0 through SEGID7
00....0	->	AR0 through AR7

NOTE: When the entry descriptor type is not T = 11, the pointer register content remains unchanged. However, unless the ISR is copied into the DR_n with the ICLIMB instruction altering the ISR bit 24, the content of AR_n, and SEGID_n is undefined.

CLIMB

f) Loading X0/GX0

If bit 18 of the C field of a CLIMB instruction is 1 and the operation is an interdomain transfer, the load is generated in the following way.

Old ISR Bit 24	New ISR Bit 24	X0	GX0
0	0	* $C(X0) \leftarrow Y_{00-17}$	Undefined (meaningless)
0	1	* Undefined (meaningless)	$C(GX0)_{00-17} \leftarrow 0$ $C(GX0)_{18-35} \leftarrow C(Y)_{0-17}$
1	0	$C(X0) \leftarrow Y_{16-33}$	** Undefined (meaningless)
1	1	Undefined (meaningless)	** $C(GX0)_{0-1} \leftarrow 0$ $C(GX0)_{2-35} \leftarrow C(Y)_{0-33}$

If X0 is to be stored in the safe store stack, the content of X0 at the start of a CLIMB instruction is stored.

If GX0 is to be stored in the safe store stack, the content of GX0 at the start of a CLIMB instruction is stored.

If bits 18 of the C field of a CLIMB instruction is 0, or the operation is not an interdomain transfer, the load is as shown below.

Old ISR Bit 24	New ISR Bit 24	X0	GX0
0	0	Unchanged	Unchanged (meaningless)
0	1	Unchanged (meaningless)	$C(GX0)_{00-17} \leftarrow 0$ $C(GX0)_{18-35} \leftarrow C(X0)$
1	0	$C(X0) \leftarrow C(GX0)_{18-35}$	Unchanged (meaningless)
1	1	Unchanged (meaningless)	Unchanged

This table also applies to the fault/interrupt CLIMB.

NOTE: When the CLIMB instruction alters bit 24 of the ISR, the content of X1-X7/GX1-GX7 is undefined.

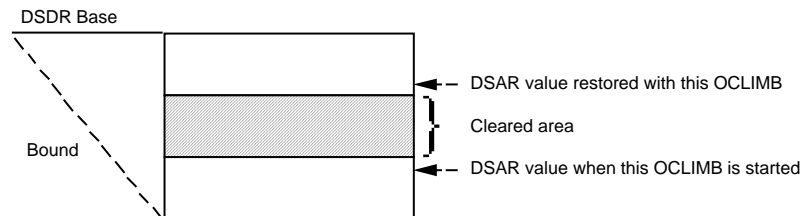
CLIMB

6. Setting Mode Indicators for System Entry CLIMB

When the CLIMB is a system entry (PMME) where $S = 0$ and $D = 1760$ (octal), the Master mode indicator is set ON. If it is not a system entry and bit 19 of the C field equals 0, the processor is set to Slave mode and the Master mode indicator is set OFF. If it is neither, the mode remains unchanged. When this CLIMB is executed as a response to a fault or an interrupt, the Master mode indicator is always set ON.

9.1.9.2 Outward CLIMB (RET/OCLIMB) C Field Bits 22 and 23 = 01

1. In the OCLIMB version of the CLIMB instruction, a return occurs according to the last frame stored in the safe store stack.
2. The E, P, S, and D fields, and bits 19, 20, and 21 of the C field are ignored.
3. The data stack clear flag (DSCF) of the option register is checked. When $DSCF = 1$, the data stack area used with the procedure executing the outward CLIMB is cleared. The cleared area is represented by the shaded in the diagram below.



In this case, a security fault, class 1 occurs if the DSAR at the start of the CLIMB is less than the restored DSAR.

If a missing page fault occurs while the data stack is being cleared, the hardware saves the state at the time the fault occurred. When the operating system loads this missing page and returns to the executing procedure, the clearing of the data stack area is re-executed correctly.

4. When an OCLIMB starts, the SCR value determines the number of registers allowed. Registers are restored with the SCR content indicated in the list below.

An IPR fault occurs if the option register safe store bypass flag (SSBF) is ON at the time.

When the SCR = 00 (binary), the following registers are restored:

- Instruction Counter (IC)
- Indicator Register (IR)
- Stack Control Register (SCR)
- Instruction Segment Identity Register - SEGID(IS)
- Data Stack Address Register (DSAR)
- Instruction Segment Register (ISR)
- Linkage Segment Register (LSR)
- Argument Stack Register (ASR)
- Parameter Segment Register (PSR)

When SCR = 01 (binary), all the registers that meet the checks for SCR = 00 (binary) are restored, plus AR 0-7 and SEGID 0-7.

When SCR = 10 or 11 (binary), the registers for SCR = 01 (binary), the DR0-DR7, X0-X7/GX0-GX7, A, Q, E, and LOR are restored. When word 5, bit 9 of the safe store stack is 1, the pointers and lengths register and the fault recovery information are restored.

In all cases, the processor number and the timer register are not restored.

5. The base and bound values of the safe store register (SSR) are adjusted according to the new values placed in the SCR from the safe store stack:

<u>SCR (bin..)</u>	<u>Base</u>	<u>SSR Bound</u>
00	-16 words	+16 words
01	-24 words	+24 words
10	-80 words	+80 words
11	-64 words	+64 words

6. Loading DR0-DR7

When an OCLIMB uses 16 or 24 words for the safe store stack (i.e., the old SCR value = 00 or 01) and then transfers to Slave mode, the new ISR value is loaded into DR0-DR7.

CLIMB

7. Loading X0/GX0

When the OCLIMB instruction C field bit 18 = 1, the effective address specified with the instruction, in accordance with bit 24 of the ISR restored from the safe store stack, is loaded into X0/GX0. (Refer to chart for loading X0/GX0 when bit 18 = 1 under ICLIMB.)

When the OCLIMB instruction C field bit 18 = 0, with a 64-word or 80-word safe store stack, the safe store stack content is restored into X0/GX0. With other than a 64-word or 80-word safe store stack, the content of X0/GX0 is determined as shown in the chart for loading X0/GX0 when bit 18 = 0 under the ICLIMB discussion.

NOTE: When the contents of X1-X7/GX1-GX7, ARn, and SEGIDn are not restored with the OCLIMB instruction that alters bit 24 of the ISR, those contents are undefined.

8. Control is passed to the instruction indicated by the IC and ISR. The IC is restored from the safe store stack in the following ways:

From NS, ES, or EI to NS or ES mode

Word 4 (0-17) → C(IC)(0-17)

From NS, ES, or EI to EI mode

Word 3 (2-35) → C(IC)(0-33)

The HWMR is restored from Word 0(0-19).

9. When the indicator register is restored (with the value stored in the safe store stack), the Master mode bit may be set to ON.

10. Outward CLIMB is interruptible during execution when the following conditions are satisfied.

The option register data stack clear flag (DSCF) = 1.

The interrupt inhibit bit = 0 (bit 28 of the first word of the instruction).

If the interrupt inhibit bit = 1, interrupt is not permitted for this instruction during execution. Interpretation of bit 28 is only valid at the time of outward CLIMB. With the other three CLIMB variations, interrupt is not accepted during execution and the value of bit 28 is not affected by execution of the instruction.

The procedure executing this outward CLIMB has used the data stack area.

If no area is to be cleared (i.e., if the restored DSAR value is equal to the current DSAR value) despite the above two conditions being satisfied, this OCLIMB is not interruptible during execution.

CLIMB

When the OCLIMB is being executed and the above three conditions are satisfied, the processor samples interrupt at suitable times and responds to any interrupt received to ensure that a Lockup fault does not occur while the data stack is being cleared. At response to the interrupt, the processor saves the current state in the safe store stack and the interrupted OCLIMB is re-executed normally. The clear operation is restarted correctly from the point at which it was interrupted.

9.1.9.3 Lateral Transfer (LTRAS/GCLIMB) C Field Bits 22 and 23 = 10

In the GCLIMB version of the CLIMB instruction, the safe store register and the parameter segment register remain unchanged. The base and bound of the argument stack register also remain unchanged.

1. The bit in the E field is not interpreted, and the SCR remains unchanged.
2. If a system entry (S = 0, D = 1760 (octal)) is specified, an IPR fault occurs.
3. The GCLIMB may be an inter- or intradomain transfer that is determined by the descriptor referenced in the S and D fields. This version functions as the ICLIMB, except as indicated. Since the state of the processor is not saved, control cannot return to an instruction executing the GCLIMB.
4. Because the processor state is not saved, the procedure executing the GCLIMB cannot return correctly with an OCLIMB.
5. If the descriptor referenced by the S and D fields of the GCLIMB instruction is a type 11 descriptor, the pointer registers are set to the state of the target instruction segment. When the type is not 11, the pointer register remains unchanged. If T is not 11 when the GCLIMB instruction is altering bit 24 of the ISR, the pointer registers are undefined.

CLIMB

9.1.9.4 Lateral Transfer (PCLIMB/LTRAD) C Field Bits 22 and 23 = 11

The execution of the PCLIMB version is identical with that of ICLIMB, except for the following conditions:

1. The CPU state is not saved in the safe store stack.
2. The SCR remains unchanged.
3. When a system entry (S=0, D=1760 (octal)) is specified, an IPR fault occurs.

If the descriptor referenced by the S and D fields of the GCLIMB instruction is a type 11 descriptor, the pointer registers are set to the state of the target instruction segment. When the type is not 11, the pointer register remains unchanged. If T is not 11 when the GCLIMB instruction is altering bit 24 of the ISR, the pointer registers are undefined.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

XEC or XED

INDICATORS:

Master Mode See notes below and discussion of "C19, Slave Mode" in earlier pages of the CLIMB explanation.

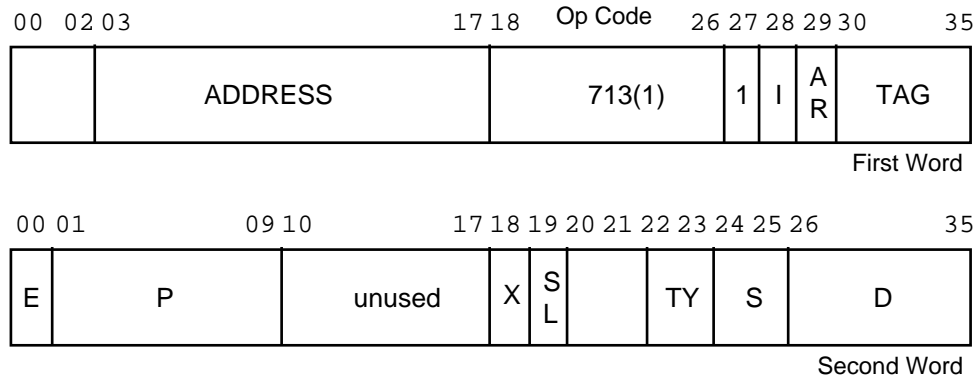
NOTES:

1. Any of the following conditions causes an IPR fault:
 - illegal repeats or executes precede modifications
 - illegal address modification is used
 - the base and bound fields of the instruction segment descriptor are not modulo 32 bytes
 - the S and D fields are $S = 0$ and $D = 1760$ (octal), and the descriptor from the system entry location is not an entry descriptor
 - the descriptor referenced in the S and D fields is not a standard, entry, or dynamic linking descriptor ($T = 0, 5, 8, 9, \text{ or } 11$)
 - the type of the descriptor referenced with the S and D fields is $T = 1$ or 3 , and the segment descriptor obtained from this descriptor is not an entry or dynamic linking descriptor
 - the S and D fields of the vector or the CLIMB instruction are $S = 0$ and $D = 1761$ (octal)
 - the transfer destination ISR type T is not 0 or 12
 - a normal or extended shrink is specified for a segment descriptor placed in the address segment and the pushed segment descriptor type is illegal ($T = 5, 7 \text{ to } 11, 13, 15$)
 - the S and D fields of the vector are $S = 0$ and $D = 1760$ (octal)
 - a system entry ($S = 0, D = 1760$ (octal)) is specified with a lateral transfer (LTRAS/LTRAD)
2. A Command fault occurs if the S and D fields of the vector are $S = 0$ and $D = 1763$ or 1764 (octal) and the processor is not in Privileged Master mode.
3. Any of the following conditions causes a bound fault:
 - in the ICLIMB version of the instruction, field $E = 1$, DR0 type = 1, and $(P + 1)$ is greater than the DR0 bound
 - the transfer destination ISD flag (bit 27) of the instruction segment descriptor is 0 (empty segment)
 - a carry occurs in forming a new argument stack register (ASR) or parameter segment register (PSR) base
4. A Security Fault, Class 2 occurs if flag bit 25 of the instruction segment descriptor is 0 (no execute permission).

CLIMB

5. Missing Segment and Missing Page faults may also occur.

SUMMARY OF CLIMB INSTRUCTION FORMAT:



The control fields are defined below.

E = 0	No parameters are passed
E = 1	Pass P+1 parameters (ICLIMB, PCLIMB only)
P = N-1	Number (minus 1) of descriptions or vectors to pass if E = 1
X = 0	Climb will not affect X0/GX0.
X = 1	If entry descriptor (T = 8, 9, or 11) is referenced or OCLIMB is executed, X0/GX0 is loaded with the effective address designated by the address tag and AR fields of the CLIMB instruction.
SL = 0	Set Slave mode.
SL = 1	Do not change Master mode indicator.
TY = 00	ICLIMB (or PMME)
TY = 01	OCLIMB
TY = 10	GCLIMB (LTRAS) Transfer with the same ASR and PSR. Do not save processor state.
TY = 11	PCLIMB (LTRAD) Transfer with the new ASR and PSR. Do not save processor state.
S,D	Target SEGID

CODING FORMAT:

Coding of a CLIMB varies with the version of the CLIMB instruction being executed.

CLIMB

The following list contains each of the five versions of the CLIMB instruction with their respective fields, which are defined below. The underlined fields are required. All others are optional.

ICLIMB	<u>entry</u> , count, effective address, flags
PCLIMB	<u>entry</u> , count, effective address, flags
GCLIMB	<u>entry</u> , effective address, flags
OCLIMB	effective address
PMME	<u>effective address</u> , count, flags

The fields in the CLIMB instruction are described below.

<u>Entry</u>	Name of an entry or a 12-bit number (SEGID) that identifies a descriptor specifying a new linkage segment and instruction segment or the same linkage segment and an instruction segment
<u>Count</u>	Decimal expression representing a value in the range $0 \leq \text{count} \leq 512$. This value indicates the number of parameters or descriptors (one for each argument) pointed to by PR0. The first of these is at the location indicated by pointer register zero. A value of zero means that no arguments, and consequently no vectors or descriptors, are present. If no value is given, zero is assumed.
<u>effective address</u>	The effective address may include a tag pointer designation. When this occurs, the field must be enclosed by parentheses, e.g., (address, tag) or (address, tag, pointer). The effective address is used to establish the next instruction location, but only when the entry identifies a descriptor that does not specify a linkage segment. The effective address is a requirement only for the PMME version to designate the Master mode entry.

CLIMB

effective address
(cont.)

If the entry identifies a descriptor that specifies a linkage segment (entry descriptor), index register 0 may be loaded with the effective address. If the entry identifies a descriptor that does not specify a linkage segment (standard descriptor), this address is added to the base of the instruction segment (described in the descriptor) to establish the next instruction location and may be loaded in index register 0. If bit 18 of field C is zero or this address is omitted, the content of the effective address field is not loaded in index register 0.

NOTE: An explicit zero is required to load index register 0 with a zero, since a null field prevents register loading.

flags:
EAX0

Sets bit 18 of the second word

The keyword EAX0 indicates that the effective address field is to be loaded in index register 0 or general index register 0.

NEAX0

Clears bit 18 of the second word

SLAVE

Clears bit 19 of the second word (for PMME, bit 18 of the second word is forced on, bit 19 is ignored by the hardware)

The keyword SLAVE indicates that the processor will enter Slave mode upon change of domain. If this field is omitted, the mode is not changed, except for the PMME version that is always set to Privileged Master mode.

If both keywords are needed, the field must be enclosed by parentheses with a comma separating the keywords: (e.g., EAX0, SLAVE).

MASTER

Sets bit 19 of the second word

No flags are used for the OCLIMB version.

CLIMB

NOTE:

PMME is synonymous with ICLIMB with 1760₈ coded in the entry field.

EXAMPLES:

1	8	16	32

*			ICLIMB
	INHIB	OFF	
ODDF	NULL		
NEPR1	LDD	P0,DSTKS	shrink data stack (64 words)
	SDR	P0	
	LDD	P1,ODRSH	shrink safe store
	SDR	P1	
	LDD	P1,IALPS	ISR,ASR,LSR,PSR
	SDR	P1	
	LDD	P1,ISRS	ISR (R,W)
	MLR	(1),(1.	safe store frame to data stack
	ADSC9	0,0,256,P.SSR	
	ADSC9	0,0,256,P0	
	LDP	P0,.ASR,DL	copy ASR to P0
	ICLIMB	.DR+4,3,,SLAVE	climb exception procedure
*	VFD	18/,09/713,1/1,1/0,1/0,6/M.	
*	VFD	1/1,9/3-1,8/0,1/.N,1/.O,2/0,2/0,12/.DR+4	
.			
.			
*			GCLIMB/ICLIMB
	INHIB	ON	
TRVCEL	NULL		
	TRA	2,IC	
	NOP	,DL	
	EPPR0	1,IC	.TROPN (sys. domain only)
	TRA	.CRTRV+12,,P.CR	
	EPPR0	1,IC	.TROPN none (sys. domain only)
	TRA	2,IC	
	EPPR0	1,IC	.TROPN all (slave domain)
	TRA	.CRTRV+14,,P.CR	
TRVC01	LDP7	** ,DL	.TRPUT (system domain)
	TRA	TPUTSY-..DISP,,P7	
	NOP	,DL *	TROPN all macros removed
	NOP	,DL	
TRVC03	GCLIMB	** ,TOPNG	TROPN extension
*	VFD	18/TOPNG,09/713,1/1,1/0,1/0,6/M.	
*	VFD	1/0,9/0,8/0,1/.N,1/.O,2/0,2/2,12/**	
	LDD6	DP.OTE,,P.SSL	.TROPN all for slave domain ext
	ICLIMB	.DR6	
*	VFD	18/,09/713,1/1,1/0,1/0,6/M.	
*	VFD	1/0,9/0,8/0,1/.N,1/.O,2/0,2/0,12/.DR6	
	TRA	0,,P0	

CMG

9.1.10 CMG

CMG	Compare Magnitude	405(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$|C(A)| \text{ :: } |C(Y)|$; $C(A)$, $C(Y)$ unchanged

EXPLANATION:

This instruction compares the magnitude of signed algebraic numbers. For example, if -1 and +1 are compared, they are considered equal and the zero indicator is set ON.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	Negative	Relationship
----	-----	-----
0	0	C(A) > C(Y)
1	0	C(A) = C(Y)
0	1	C(A) < C(Y)

9.1.11 CMK

CMK	Compare Masked	211(0)
-----	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 ,

$$C(Z)_i = \overline{C(Q)_i} \text{ AND } [C(A)_i \text{ XOR } C(Y)_i]$$

$C(A)$, $C(Q)$, $C(Y)$ unchanged

EXPLANATION:

This instruction compares the corresponding bit positions in $C(A)$ and $C(Y)$ to determine whether they are equal or not. Bits for which the corresponding bit of Q is 1 are masked and not compared.

The zero indicator is set ON if the comparison is successful for all bit positions, i.e.,

if for all $i = 0, 1, \dots, 35$
 either $C(A)(i) = C(Y)(i)$
 or
 $C(Q)(i) = 1$ established.
 otherwise, the zero indicator is set OFF.

The negative indicator is set ON if the comparison is unsuccessful for bit position 0, i.e.,

if for $C(A)(0) \neq C(Y)(0)$
 and
 $C(Q)(0) = 0$
 otherwise, the zero indicator is set OFF.

CMK

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If bit 0 of $C(Z) = 1$, then ON; otherwise, OFF

EXAMPLE:

In the following example, the comparison is equal after execution of CMK, and the TZE exit is taken. Only the 2s in NUMBER and DATA are compared. All other bits are masked by 1s in the Q-register.

1	8	16	32

	LDQ	MASK	
	LDA	NUMBER	
	CMK	DATA	
	TZE	OUT	
MASK	OCT	777777777707	
NUMBER	OCT	300333333326	
DATA	OCT	666666666625	

CMPA**9.1.12 CMPA**

CMPA	Compare with A-Register	115(0)
------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) :: C(Y); C(A)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

CMPA

INDICATORS:

Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
0	0	0	$C(A) > C(Y)$	$C(A)0=0, C(Y)0=1$
0	0	1	$C(A) > C(Y)$	$C(A)0=C(Y)0$
1	0	1	$C(A) = C(Y)$	
0	1	0	$C(A) < C(Y)$	$C(A)0=1, C(Y)0=0$
0	1	1	$C(A) < C(Y)$	

Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
0	1	$C(A) > C(Y)$
1	1	$C(A) = C(Y)$
0	0	$C(A) < C(Y)$

CMPAQ**9.1.13 CMPAQ**

CMPAQ	Compare with AQ-Register	117(0)
-------	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) :: C(Y\text{-pair}); C(AQ)$ and $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

CMPAQ

INDICATORS:

Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
0	0	0	$C(AQ) > C(Y)$	$C(AQ)0=0, C(Y-pr)0=1$
0	0	1	$C(AQ) > C(Y)$	$C(AQ)0=C(Y=pr)0$
1	0	1	$C(AQ) = C(Y)$	
0	1	0	$C(AQ) < C(Y)$	$C(AQ)0=1, C(Y-pr)0=0$
0	1	1	$C(AQ) < C(Y)$	

Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
0	1	$C(AQ) > C(Y-pr)$
1	1	$C(AQ) = C(Y-pr)$
0	0	$C(AQ) < C(Y-pr)$

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

CMPB

9.1.14 CMPB

CMPB	Compare Bit Strings	066(1)
------	---------------------	--------

FORMAT:

0	1	10	11	17	18	27	28	29	35		
F	0000000000						MF2	066(1)		I	MF1

0	2	3	17	18	19	20	23	24	32	35
Y1			C1	B1	N1					
AR#	Y1				00000000		R1			

0	2	3	17	18	19	20	23	24	32	35
Y2			C2	B2	N2					
AR#	Y2				00000000		R2			

CODING FORMAT:

The CMPB instruction code is shown below.

1	8	16

CMPB	(MF1), (MF2), F, d1, d2	
BDSC	LOCSYM, N, C, B, AM	
BDSC	LOCSYM, N, C, B, AM	
ARG	LOCSYM, REG, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

CMPB

SUMMARY:

`C(string 1) :: C(string 2)`

EXPLANATION:

The string of bits starting at location YCB₁ is logically compared with the string of bits starting at location YCB₂ until an inequality is found or until the larger tally (L1 or L2) is exhausted. If L1 is not equal to L2, the fill bit (F) is used to pad the least significant bits of the shorter string. The contents of both strings remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	Carry	Relationship
----	----	-----
0	0	<code>C(string 1) < C(string 2)</code>
1	1	<code>C(string 1) = C(string 2)</code>
0	1	<code>C(string 1) > C(string 2)</code>

NOTES:

1. If L1 or L2 = 0, both the Zero and Carry indicators are turned ON, but no Illegal Procedure fault occurs.
2. An Illegal Procedure fault occurs if DU or DL modifications are used for MF1 or MF2 or if an illegal repeat is used.

CMPB**EXAMPLES:**

1	8	16	32
	CMPB	,,1	fill bit 1 option
	BDSC	FLD1,45,0,0	FLD1 operand descriptor
	BDSC	FLD2,48	FLD2 operand descriptor
	TRC	EQU.GR	FLD1 equal/greater than FLD2
	USE	CONST.	compared (oct. representation)
FLD1	OCT	0,777000000000	0000000000007777
FLD2	OCT	0,777000000000	0000000000007770
	USE		Result - FLD1 > FLD2
	CMPB	no options	
	BDSC	FLD1,36,0,0	FLD1 operand descriptor
	BDSC	FLD2,19,1,3	FLD2 operand descriptor
	TZE	EQUAL	FLD1 = FLD2
	TRC	FLD1GR	FLD1 > FLD2
	TRA	FLD1LS	FLD1 < FLD2
	USE	CONST.	compared (oct. representation)
FLD1	VFD	18/-1	777777000000
FLD2	VFD	12/0,19/-1	777777400000
	USE		Result - FLD1 < FLD2

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32
	EAX2	12	load FLD1 bit modifier into X2
	EAX6	6	load FLD1 length into X6
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	CMPB	(1,1,,X2),,,1.	with modification
	BDSC	0,X6,0,0,4	FLD1 operand descriptor
	ARG	INDSCR	FLD2 indirect descriptor ptr.
	TZE	EQUAL	FLD1 = FLD2
	USE	CONST.	compared memory contents
FLD1	VFD	12/0,6/1	770 000077000000
FLD2	VFD	24/0,6/1	770 000000007700
INDSCR	BDSC	FLD2,9,2,6	indirect operand descriptor
	USE		Result - FLD1 = FLD2

CMPBX

9.1.15 CMPBX

CMPBX	Compare Bit Strings Extended	067(1)
-------	------------------------------	--------

FORMAT:

00 08 09 10 11 17 18 27 28 29 35

F	d	d	MF2	067(1)	I	MF1
	1	2				

00 02 03 17 18 20 21 22 23 24 35

Y1		CN1	T A 1	0	N1
AR#	Y1				

00 02 03 17 18 20 21 23 24 35

Y2		CN2	0 0 0	N2
AR#	Y2			

00 02 03 17 18 28 29 30 31 32 35

Y3		0 0 0 0 0 0 0 0 0 0 0 0	A R	0 0	REG
AR#	Y3				

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

CMPBX**SUMMARY:**

`C(string 1) :: C(string 2)`

EXPLANATION:

The operation of the CMPBX instruction is identical to the CMPB instruction except that the bit position number (0 origin) at which the first bit mismatch is detected as a result of the comparison is stored right-justified in C(Y3). If operand 1 and operand 2 are identical (including the fill), the value stored in C(Y3) is equal to the length of the longer operand.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and REG(Y3)

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	Carry	Relationship
----	----	-----
0	0	<code>C(string 1) < C(string 2)</code>
1	1	<code>C(string 1) = C(string 2)</code>
0	1	<code>C(string 1) > C(string 2)</code>

NOTES:

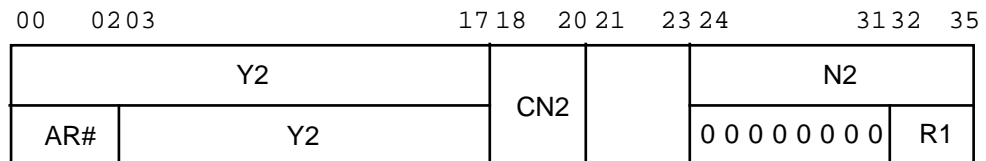
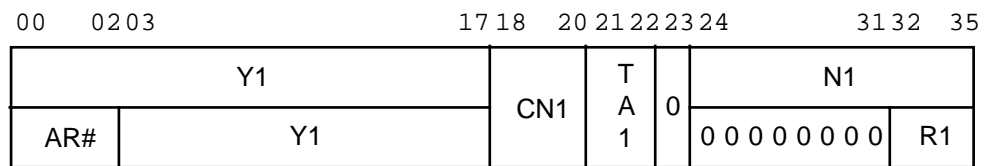
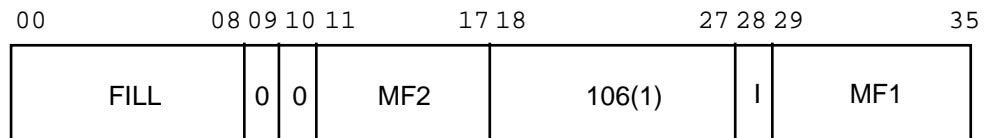
1. If L1 or L2 = 0, both the Zero and Carry indicators are turned ON, but no Illegal Procedure fault occurs.
2. An Illegal Procedure fault occurs if DU or DL modifications are used for MF1 or MF2 or if an illegal repeat is used.
3. In operand descriptor 3, the AR# field (bits 0-2) specifies address register to be used in address modification if AR (bit 29) = 1. Otherwise bits 0-2 are appended to the right of Y3 to form an 18-bit positive word displacement.
4. In operand descriptor 3, if AR (bit 29) = 1, Y3 is a 15-bit twos complement word offset for Y3.

CMPC

9.1.16 CMPC

CMPC	Compare Alphanumeric Character Strings	106(1)
------	--	--------

FORMAT:



CODING FORMAT:

The CMPC instruction code is shown below.

1	8	16

CMPC	(MF1) , (MF2) , FILL	
ADSC _n	LOCSYM , CN , N , AM	
ADSC _n	LOCSYM , CN , N , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

`C(string 1) :: C(string 2)`

EXPLANATION:

Starting at location YC1, the string of alphanumeric characters of type TA1 is logically compared with the string of alphanumeric characters of assumed type TA1 that starts at location YC2 until either an inequality is found or until the larger tally (L1 or L2) is exhausted. If L1 is not equal to L2, the FILL character is used to pad the least significant characters of the shorter string. The contents of both strings remain unchanged. Bits 21-23 of descriptor 2 are not interpreted.

Bits 0-8 are compared for the FILL character to be used to pad the least significant characters of the shorter string. If a character string is a 6- or 4-bit character, zeros are inserted at the left of each to produce 9-bit characters for comparison.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	Carry	Relationship
----	-----	-----
0	0	<code>C(string 1) < C(string 2)</code>
1	1	<code>C(string 1) = C(string 2)</code>
0	1	<code>C(string 1) > C(string 2)</code>

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2.
2. If L₁ or L₂ = 0, the zero and carry indicators are affected as illustrated under Indicators.

CMPC

EXAMPLE:

1	8	16	32

	CMPC	,,020	compare with blank fill
	ADSC6	FLD1,0,6	field 1 operand descriptor
	ADSC6	FLD2,4,4	field 2 operand descriptor
	TZE	EQUAL	both fields equal
	TRC	FLD1GR	field 1 greater
	NULL		field 1 less
	USE	CONST.	characters compared
FLD1	BCI	1,ABCD	ABCD 00
FLD2	BCI	2,XXXXABCDXXXX	ABCD 00
	USE		Result - FLD1 = FLD2

CMPCT

9.1.17 CMPCT

CMPCT	Compare Characters and Translate	166(1)
-------	----------------------------------	--------

FORMAT:

00	08	09	10	11	17	18	Op Code	27	28	29	35
FILL		d 1	d 2	MF2		166(1)		I	MF1		

00	02	17	18	20	21	22	23	24	35
Y1			CN1	T A 1	0	N1			
AR#	Y1								

00	02	17	18	20	21	22	23	24	35
Y2			CN2	0	0	N2			
AR#	Y2								

00	02	17	18	28	29	30	31	32	35
Y3			000000000000			A R	00	REG	
AR#	Y3								

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

CMPCT

EXPLANATION:

Starting at location YC1, the string of alphanumeric characters of type TA1 is logically compared with the string of alphanumeric characters of assumed type TA1 that starts at location YC2, until either an inequality is found or until the larger tally (L1 or L2) is exhausted.

If an inequality is found, the next action depends on d1 and d2. If d1 and d2 = 0, then both characters are transliterated and the resulting characters are compared.

The character from the string starting at YC1 and the character from the string starting at YC2 are each used as an index to a table of 9-bit characters starting at location Y3. The two characters thus taken from the table are compared, the indicators set as indicated below, and the instruction terminates. For the case d1 = d2 = 1, no transliteration takes place. The indicators are set according to the way the two original characters compared. When d1 \diamond d2, one character is translated and the other is not, and then the two characters are compared. For example, if d1 = 1 and d2 = 0 the character from the string starting at YC2 is transliterated (as described above) and compared with the character from the string starting at YC1 and the indicators are set accordingly.

NOTE: A 9-bit compare is always made. For the case where d1 \diamond d2 and the nontranslated character is a 4- or 6-bit character, then the upper bit positions of the character are zero-filled for the 9-bit compare.

If L1 \diamond L2, fill characters are used to fill the low-order character positions of the shorter string. The contents of both strings remain unchanged.

The transliteration table must begin at a word boundary at character position 0. The index, which is expressed by the number of 9-bit characters, is added to the starting word address of the table. The beginning address of the table is calculated in the same manner as is any normal address modification. However, the computed address is used as word address, with character position ignored, and the index is added to this word address as a 9-bit character number.

Refer to the MVT instruction specifications for details on generating the transliteration table address when address register modification is specified.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

CMPCT**INDICATORS:**

Let C1 = C(last char from string 1, translated if d1 = 0)

Let C2 = C(last char from string 2, translated if d2 = 0)

Zero	Carry	Relationship
----	----	-----
0	0	C1 < C2
1	1	C1 = C2
0	1	C1 > C2

NOTES:

1. When L1 or L2 = 0, the zero and carry indicators are still affected as indicated in the above table. If L1=L2=0, both the zero and carry indicators are turned ON.
2. A 9-bit character (zero-filled as appropriate) and/or the full 9 bits of the table entry are used in all comparisons.
3. The CMPCT instruction is intended for comparisons in situations where the character collating sequence is different from the sequence of character codes.
4. If L1 < L2, and type TA1 is 4- or 6-bit, the low-order 4 or 6 bits of the 9-bit FILL character in the instruction are defined as a table index, respectively.
5. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPN

9.1.18 CMPN

CMPN	Compare Numeric	303(1)
------	-----------------	--------

FORMAT:

00	10 11	17 18	Op Code	27 28 29	35
000000000000	MF2	303(1)	1	MF1	

00 02 03	17 18	20 21	22 23 24	29 30	35	
Y1		CN1	T N 1	S1	SF1	N1
AR#	Y1					

00 02 03	17 18	20 21	22 23 24	29 30	35	
Y2		CN2	T N 2	S2	SF2	N2
AR#	Y2					

CODING FORMAT:

The CMPN instruction code is shown below.

1	8	16

CMPN	(MF1), (MF2)	
NDSC _n	LOCSYM, CN, N, S, SF, AM	
NDSC _n	LOCSYM, CN, N, S, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 2) :: C(string 1)

CMPN**EXPLANATION:**

Starting at location YC1, the decimal number of data type TN1 and sign and decimal type S1 is algebraically compared with the decimal number of data type TN2 and sign and decimal type S2 that starts at location YC2. The comparison effectively subtracts number 1 from number 2. Zeros (4 bits - 0000) are used to pad the integral and fractional parts of the shorter field. Both numbers remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	Negative	Relationship
----	-----	-----
0	1	C(number 1) > C(number 2)
1	0	C(number 1) = C(number 2)
0	0	C(number 1) < C(number 2)

Zero	Carry	Relationship
----	-----	-----
0	0	C(number 1) > C(number 2)
1	1	C(number 1) = C(number 2)
0	1	C(number 1) < C(number 2)

NOTES:

1. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
2. An IPR fault occurs if the values for the number of characters (Ni) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPN

EXAMPLES:

1	8	16	32
	CMPN		no modification
	NDSC4	FLD1,0,8,1,-2	FLD1 operand descriptor
	NDSC4	FLD2,0,8,0	FLD2 operand descriptor
	TZE	EQUAL	FLD2 = FLD1
	TMI	LESS	FLD2 < FLD1
	TNC	ABS.LT	FLD2 < FLD1
	USE	CONST.	numbers compared
FLD1	EDEC	8P-12345	-0012345
FLD2	EDEC	8P-123.45	-0012345
	USE		Result - FLD2 = FLD1
	CMPN		no modification
	NDSC9	FLD1,2,2,3	FLD1 operand descriptor
	NDSC4	FLD2,0,8,2,-3	FLD2 operand descriptor
	TZE	EQUAL	FLD2 = FLD1
	TMI	LESS	FLD2 < FLD1
	TRA	GREATER	FLD2 > FLD1
	USE	CONST.	numbers compared
FLD1	EDEC	4A0012	+0012000
FLD2	EDEC	8P12000+	+0012000
	USE		Result - FLD2 = FLD1

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32
	EAX2	2	load character mod. into X2
	EAX6	6	load FLD1 length into X6
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	CMPN	(1,1,,X2),(,,1.	with address modification
	NDSC4	0,0,X6,3,-3,4	FLD1 operand descriptor
*	ARG	FLD2.I	(FLD1,2,6,3,-3) pointer to operand descriptor
	TZE	EQUAL	FLD2 = FLD1
	TPL	MORE	FLD2 > FLD1
	TRA	LESS	FLD2 < FLD1
	USE	CONST.	numbers compared
FLD1	EDEC	8P123456	+00123456
FLD2	EDEC	8P123456+	+01234560
FLD2.I	NDSC4	FLD2,0,8,2,-2	
	USE		Result - FLD2 > FLD1

CMPNX**9.1.19 CMPNX**

CMPNX	Compare Numeric Extended	343(1)
-------	--------------------------	--------

FORMAT:

00	01	02	10	11	17	18	Op Code	27	28	29	35
C	0	0000000000	MF2			343(1)			1	MF1	

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1			CN1	T N 1	SX1	SF1	N1					
AR#	Y1											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2			CN2	T N 2	SX2	SF2	N2					
AR#	Y2											

CODING FORMAT:

1	8	16

CMPNX	(MF1), (MF2), CS	
NDSCn	LOCSYM, CN, N, SX, SF, AM	
NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) :: C(string 2)

CMPNX

EXPLANATION:

Starting at location YC1, the decimal number of data type TN1 and sign and decimal type SX1 is algebraically compared with the decimal number of data type TN2 and sign and decimal type SX2 that starts at location YC2. The comparison effectively subtracts number 1 from number 2. Zeros (4 bits - 0000) are used to pad the integral and fractional parts of the shorter field. Both numbers remain unchanged.

The character set is defined by CS (EBCDIC/ASCII).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	Negative	Relationship
----	-----	-----
0	1	$ C(\text{number } 1) > C(\text{number } 2) $
1	0	$ C(\text{number } 1) = C(\text{number } 2) $
0	0	$ C(\text{number } 1) < C(\text{number } 2) $

Carry	Relationship
----	-----
0	$C(\text{number } 1) > C(\text{number } 2)$
1	$C(\text{number } 1) < C(\text{number } 2)$

NOTES:

1. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or if any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
2. An IPR fault occurs if the values for the number of characters (Ni) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. Refer to the specifications on MVNX for information on coding of overpunched signs.
4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMPQ**9.1.20 CMPQ**

CMPQ	Compare with Q-Register	116(0)
------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) ::= C(Y)$; $C(Q)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

CMPQ

INDICATORS:

Algebraic comparison (Signed Binary Operands)

Zero	Negative	Carry	Relationship	Sign
----	-----	-----	-----	----
0	0	0	$C(Q) > C(Y)$	$C(Q)0=0, C(Y)0=1$
0	0	1	$C(Q) > C(Y)$	$\rightarrow C(Q)0=C(Y)0$
1	0	1	$C(Q) = C(Y)$	
0	1	0	$C(Q) < C(Y)$	$C(Q)0=1, C(Y)0=0$
0	1	1	$C(Q) < C(Y)$	

Logical comparison (Unsigned Positive Binary Operands)

Zero	Carry	Relationship
----	-----	-----
0	1	$C(Q) > C(Y)$
1	1	$C(Q) = C(Y)$
0	0	$C(Q) < C(Y)$

CMPX_n**9.1.21 CMPX_n**

CMPX _n	Compare with Index Register <u>n</u>	10 <u>n</u> (0)
-------------------	--------------------------------------	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(X_n) :: C(Y)(0-17);$
 $C(X_n)$ and $C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, \text{or } 7$ as determined by op code:
 $C(GX_n) :: C(Y);$ $C(GX_n)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of CMPX0

CMPX_n

INDICATORS:

Algebraic (signed binary) comparison:

NS Mode

<u>Zero</u>	<u>Negative</u>	<u>Carry</u>	<u>Relationship</u>	<u>Sign</u>
0	0	0	$C(Xn) > C(Y)_{0-17}$	$C(Xn)_0 = 0, C(Y)_0 = 1$
0	0	1	$C(Xn) > C(Y)_{0-17}$ }	$C(Xn)_0 = C(Y)_0$
1	0	1	$C(Xn) = C(Y)_{0-17}$ }	
0	1	0	$C(Xn) < C(Y)_{0-17}$ }	$C(Xn)_0 = 1, C(Y)_0 = 0$
0	1	1	$C(Xn) < C(Y)_{0-17}$	

ES/EI Mode

<u>Zero</u>	<u>Negative</u>	<u>Carry</u>	<u>Relationship</u>	<u>Sign</u>
0	0	0	$C(GXn) > C(Y)$	$C(GXn)_0 = 0, C(Y)_0 = 1$
0	0	1	$C(GXn) > C(Y)$ }	$C(GXn)_0 = C(Y)_0$
1	0	1	$C(GXn) = C(Y)$ }	
0	1	0	$C(GXn) < C(Y)$ }	$C(GXn)_0 = 1, C(Y)_0 = 0$
0	1	1	$C(GXn) < C(Y)$	

Logical comparison (Unsigned Positive Binary Operands):

NS Mode

<u>Zero</u>	<u>Carry</u>	<u>Relationship</u>
0	1	$C(Xn) > C(Y)_{0-17}$
1	1	$C(Xn) = C(Y)_{0-17}$
0	0	$C(Xn) < C(Y)_{0-17}$

ES/EI Mode

<u>Zero</u>	<u>Carry</u>	<u>Relationship</u>
0	1	$C(GXn) > C(Y)$
1	1	$C(GXn) = C(Y)$
0	0	$C(GXn) < C(Y)$

NOTES:

1. When DL modification is specified in the NS Mode, it is executed with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CMRR**9.1.22 CMRR**

CMRR	Compare Register to Register	534(1)
------	------------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	

	CMRR	R1, R2	

OPERATING MODES:

Executes in ES/EI mode

SUMMARY:

R1, R2, = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) :: C(R2);
C(R1), C(R2) unchanged

EXPLANATION:

C(R1) is compared with C(R2) and the indicators are set as indicated below.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

CMRR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Algebraic (signed fixed-point) Comparison

Zero	Negative	Carry	Relationship	Sign
----	-----	-----	-----	-----
0	0	0	$C(R1) > C(R2)$	$C(R1)0=0, C(R2)0=1$
0	0	1	$C(R1) > C(R2)$	-> $C(R1)0 = C(R2)0$
1	0	1	$C(R1) = C(R2)$	
0	1	0	$C(R1) < C(R2)$	
0	1	1	$C(R1) < C(R2)$	$C(R1)0=1, C(R2)0=0$

Logic (unsigned fixed-point) Comparison

Zero	Carry	Relationship
----	-----	-----
0	0	$C(R1) < C(R2)$
1	1	$C(R1) = C(R2)$
0	1	$C(R1) > C(R2)$

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

CNA**9.1.23 CNA**

CNA	Compare NOT AND with A-Register	215(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0-35$, $C(Z)_i = C(A)_i \text{ AND } \overline{C(Y)_i}$
 $C(A)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

CNAAQ

9.1.24 CNAAQ

CNAAQ	Compare NOT AND with AQ-Register	217(0)
-------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 71 , $C(Z)_i = C(AQ)_i \text{ AND } \overline{C(Y\text{-pair})_i}$
 $C(AQ)$ and $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

CNAQ**9.1.25 CNAQ**

CNAQ	Compare NOT AND with Q-Register	216(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 , $C(Z)_i = C(Q)_i \text{ AND } \overline{C(Y)_i}$
 $C(Q)$ and $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

CNAX_n

9.1.26 CNAX_n

CNAX _{<u>n</u>}	Compare NOT AND with Index Register <u>n</u>	20 _{<u>n</u>} (0)
--------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0,1,...,or 7 as determined by the op code

For i = 0 to 17, C(Z)_i = C(X_n)_i AND $\overline{C(Y)_i}$

C(X_n) and C(Y) unchanged

ES/EI Mode

For n=0,1,...,or 7 as determined by the opcode

For i = 0 to 35, C(Z)_i = C(GX_n)_i AND $\overline{C(Y)_i}$

C(GX_n) and C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of CNAX0

INDICATORS:

Zero	If $C(Z) = 0$, then ON; otherwise, OFF
Negative	If $C(Z)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

CSL

9.1.27 CSL

CSL	Combine Bit Strings Left	060(1)
-----	--------------------------	--------

FORMAT:

00	01	04	05	08	09	10	11	17	18	Op Code	27	28	29	35
F	0000	BOLR	T	0	MF2			060(1)			I	MF1		

00	02	03	17	18	20	21	23	24	32	35
Y1				C1	B1	N1				
AR#	Y1					00000000		R1		

00	02	03	17	18	20	21	23	24	32	35
Y2				C2	B2	N2				
AR#	Y2					00000000		R2		

CODING FORMAT:

The CSL instruction code is shown below.

1	8	16

CSL	(MF1) , (MF2) , BOLR , F , T	
BDSC	LOCSYM , N , C , B , AM	
BDSC	LOCSYM , N , C , B , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) : (BOLR) : C(string 2) -> C(string 2)

EXPLANATION:

The string of bits starting at location YCB1 is evaluated, bit by bit. The string starting at location YCB2 and the appropriate bit from the BOLR control field is placed into each corresponding bit of the string starting at location YCB2. If L1 is greater than L2, the least significant L1-L2 bits of string 1 are truncated and the Truncation indicator is set. If L1 is less than L2, the fill bit (F) is used as the L2-L1 least significant bits of string 1. The contents of string 1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If all the resultant bits generated are zero, then ON if L2=0 and L1 >= 0; otherwise, OFF
Truncation	If L1 is > L2, then ON; otherwise, OFF If L1>0 and L2=0, then ON. If L1=L2=0, then OFF.

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if an illegal repeat is used.
2. An IPR fault does not occur even when L1 = 0 or L2 = 0. In this case, the zero and truncation indicators are affected.

CSL

EXAMPLES:

1	8	16	32

*	REM	BITS 0-17 OF FLD2 FORCED ON	
	CSL	,,07,,1	OR - truncation enable option
	BDSC	FLD1,24,1,3	FLD1 operand descriptor
	BDSC	FLD2,18,0,0	FLD2 operand descriptor
	USE	CONST	memory contents in octal
FLD1	VFD	12/0,18/-1,6/0	000077777700
FLD2	LDA	0,2	000000235012
	USE		777777235012 (Result)
	REM	BITS 18-35 OF FLD2 INVERTED	
	CSL	,,06,1	exclusive OR, fill bit 1 opt
	BDSC	,0	FLD1 operand descriptor
	BDSC	FLD2,18,2,0	FLD2 operand descriptor
	USE	CONST.	memory contents in octal
FLD2	DEC	0	000000000000
	USE		000000777777 (Result)

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX6	12	char/bit address mod to X6
	EAX7	54	load FLD2 length into X7
	EAX4	FLD2	load FLD2 address into X4
	AWDX	0,4,4	put FLD2 address into AR4
	CSL	(,,1),00,(1,1,,6),00	clear operation with
*			address modification ARG 2,4
*			pointer to FLD1 indirect
*			operand descriptor
	BDSC	0,X7,,4	FLD2 operand descriptor
*			(FLD2,54,1,3)
	USE	CONST.	memory contents in octal
FLD2	VFD	36/-1,36/-1	777777777777
	BDSC	,0	FLD1 operand descriptor
*			(control field zeros)
	USE		777700000000 000000000077
*			(Result)

CSR

9.1.28 CSR

CSR	Combine Bit Strings Right	061(1)
-----	---------------------------	--------

FORMAT:

00	01	04	05	08	09	10	11	17	18	Op Code	27	28	29	35
F	0000	BOLR	T	0	MF2			061(1)			I	MF1		

00	02	03	17	18	20	21	23	24	32	35
Y1				C1	B1	N1				
AR#	Y1					00000000		R1		

00	02	03	17	18	20	21	23	24	32	35
Y2				C2	B2	N2				
AR#	Y2					00000000		R2		

CODING FORMAT:

The CSR instruction code is shown below.

1	8	16

CSR	(MF1), (MF2), BOLR, F, T	
BDSC	LOCSYM, N, C, B, AM	
BDSC	LOCSYM, N, C, B, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) : (BOLR) : C(string 2) -> C(string 2)

CSR

EXPLANATION:

This instruction operates the same as CSL except that the starting locations are $YCB1 + (L1-1)$ and $YCB2 + (L2-1)$ and the evaluation is from right to left (least to most significant bits). Any truncation or fill consists of most significant bits.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for CSL

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if an illegal repeat is used.
2. An IPR fault does not occur even when $L1 = 0$ or $L2 = 0$. In this case, the zero and truncation indicators are affected.

EXAMPLES:

1	8	16	32

*	CSR	,,14,,1	invert with truncation fault enable option
	BDSC	FLD1,18,2,0	FLD1 operand descriptor
	BDSC	FLD2,12,0,0	FLD2 operand descriptor
	USE	CONST.	memory contents in octal
FLD1	OCT	444444	000000444444
FLD2	DEC	0	333300000000 (Result)
	USE		truncation
*	CSR	,,17	force ones operation
	BDSC	,0	FLD1 operand descriptor
	BDSC	FLD2,36,0,0	FLD2 operand descriptor
	USE	CONST.	memory contents in octal
FLD2	BSS	1	777777777777 (Result)
	USE		none

CWL

9.1.29 CWL

CWL	Compare with Limits	111(0)
-----	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y)$:: closed (algebraic) interval $[C(A), C(Q)]$ and
 (algebraic comparison) $C(Y) :: C(Q)$

$C(Y)$, $C(A)$, $C(Q)$ are unchanged

EXPLANATION:

This instruction tests the algebraic value of $C(Y)$ to determine if it is within the range of algebraic values bounded by $C(A)$ and $C(Q)$. The indicators are then set to reflect the result. This instruction is not recommended for logical (unsigned) comparisons.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero

If $C(Y)$ is contained in the closed interval $[C(A), C(Q)]$ i.e., either:

$C(A) \leq C(Y) \leq C(Q)$ or

$C(A) \geq C(Y) \geq C(Q)$,

then ON; otherwise, OFF

Negative	Carry	Relationship	Sign
-----	-----	-----	-----
0	0	$C(Q) > C(Y)$	$C(Q)0=0, C(Y)0=1$
0	1	$C(Q) > C(Y)$	$C(Q)0=C(Y)0$
1	0	$C(Q) < C(Y)$	$C(Q)0=C(Y)0$
1	1	$C(Q) < C(Y)$	$C(Q)0=1, C(Y)0=0$

DFAD

9.2 Machine Instruction Descriptions (D)

Following is a detailed description of the processor instructions and operation codes beginning with the letter D.

9.2.1 DFAD

DFAD	Double-Precision Floating Add	477(0)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) + C(Y\text{-pair})]$ normalized $\rightarrow C(EAQ)$;
 $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. The definition of normalization is located under the description of the FNO instruction.
2. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise the floating-point alignment and normalization are binary.
3. An Illegal Procedure fault occurs if illegal address modification is used.

DFCMG

9.2.2 DFCMG

DFCMG	Double-Precision Floating Compare Magnitude	427(0)
-------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$|C(E, AQ_{0-63})| :: |C(Y\text{-pair})|$; magnitude comparison
 $C(EAQ)$, $C(Y\text{-pair})$ unchanged

EXPLANATION:

The DFCMG comparison is executed in the following way:

- a) Compare $C(E) :: C(Y)(0-17)$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.
- b) Compare the absolute values of the mantissas and set the indicators accordingly.

The DFCMG instruction is identical to the DFCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	Negative	Relationship
0	0	$ C(E, AQ_{0-63}) > C(Y\text{-pair}) $
1	0	$ C(E, AQ_{0-63}) = C(Y\text{-pair}) $
0	1	$ C(E, AQ_{0-63}) < C(Y\text{-pair}) $

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFCMP

9.2.3 DFCMP

DFCMP	Double-Precision Floating Compare	517(0)
-------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(E, AQ_{0-63}) :: C(Y\text{-pair});$
 $C(E, AQ), C(Y\text{-pair})$ unchanged

EXPLANATION:

This comparison is executed in the following way.

- a) Compare $C(E) :: C(Y)(0-7)$, select the number with the lower exponent, and shift its mantissa right as many places as the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.
- b) Compare the mantissas and set the indicators accordingly.
 The DFCMP instruction is identical to the FCMP instruction except for the precision of the mantissas actually compared.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFCMP**INDICATORS:**

Zero	Negative	Relationship
0	0	$ C(E, AQ_{0-63}) > C(Y\text{-pair}) $
1	0	$ C(E, AQ_{0-63}) = C(Y\text{-pair}) $
0	1	$ C(E, AQ_{0-63}) < C(Y\text{-pair}) $

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFDI

9.2.4 DFDI

DFDI	Double-Precision Floating Divide Inverted	527(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y\text{-pair}) / C(EAQ) \rightarrow C(EAQ)$; $C(Y\text{-pair})$ unchanged

EXPLANATION:

If $AQ(64-71)$ is not = 0 and $A(0) = 0$, a 1 is added to $AQ(63)$. Zero is moved to $AQ(64-71)$, unconditionally. $AQ(0-63)$ is then used as the divisor mantissa. The 8-bit dividend exponent and 72-bit mantissa are placed in working registers. The dividend mantissa is shifted to the right, and the dividend exponent is increased accordingly until: $|Dividend\ mantissa| < |C(AQ)(0-63)|$. When such a shift occurs, significant bits from the dividend may be lost.

$C(AQ)(0-63)$ is used as the divisor mantissa. 64 bits of quotient mantissa are placed in $AQ(0-63)$. Zeros are placed in $AQ(64-71)$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

	<u>When Division Occurs</u>	<u>When No Division Occurs</u>
Zero	If $C(A) = 0$, then ON; otherwise, OFF	If divisor mantissa = 0, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF	If dividend < 0, then ON; otherwise, OFF
Exponent Overflow	If quotient exponent is > +127, then ON	
Exponent Underflow	If quotient exponent is < -128, then ON	

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. If the divisor mantissa $C(AQ)$ is zero, the division does not take place. Instead, a Divide Check fault occurs and all registers remain unchanged. The dividend and divisor are not normalized by the hardware prior to division.
3. An Illegal Procedure fault occurs if illegal address modification is used.

DFDV

9.2.5 DFDV

DFDV	Double-Precision Floating Divide	567(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ) / C(Y\text{-pair}) \rightarrow C(EAQ)$; $C(Y\text{-pair})$ unchanged

EXPLANATION:

$C(AQ)(0-71)$ are used by this instruction. If the divisor mantissa $C(Y\text{-pair})(8-71)$ is zero, then the division does not take place. Instead, a Divide Check fault occurs. The divisor $C(Y)$ remains unchanged, $C(AQ)$ contains the dividend magnitude in absolute, and the Negative indicator reflects the dividend sign. Dividend and divisor are not normalized by the hardware before division occurs.

The dividend mantissa $C(AQ)$ is shifted right and the dividend exponent is increased accordingly until the following shift occurs:

$$\begin{aligned} &|C(AQ)_{0-63}| < |C(Y\text{-pair})_{8-71}| \text{ with zero fill} | \\ &C(E) - C(Y\text{-pair})_{0-7} \rightarrow C(E) \end{aligned}$$

When such a shift occurs, significant bits from the dividend may be lost. 64 bits of the quotient mantissa are placed in $AQ(0-63)$. Zeros are placed in $AQ(64-71)$.

When the divisor mantissa is 0, division is not executed and a Divide Check fault occurs. The absolute value of the dividend is loaded into AQ , and the Negative indicator is set in accordance with the sign of the dividend.

Refer to the FDV instruction for details of the method of shifting the dividend.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero (division)	If $C(A) = 0$, then ON; otherwise, OFF
Zero (no division)	If divisor mantissa = 0, then ON; otherwise, OFF
Negative (division)	If $C(AQ)(0) = 1$, then ON, otherwise, OFF
Negative (no division)	If dividend < 0 , then ON; otherwise, OFF
Exponent Overflow	If quotient exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFLD

9.2.6 DFLD

DFLD	Double-Precision Floating Load	433(0)
------	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y\text{-pair}), 00\dots0 \rightarrow C(EAQ); C(Y\text{-pair})$ unchanged
 $C(Y)_{00-07} \rightarrow C(E)$
 $C(Y\text{-pair})_{08-71} \rightarrow C(AQ)_{00-63}$
 $00\dots0 \rightarrow C(AQ)_{64-71}$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

DFLP**9.2.7 DFLP**

DFLP	Double-Precision Floating Load Positive	532(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$ C(Y\text{-pair}) $, normalized	→ Z
Z_{00-07}	→ C(E)
Z_{08-71}	→ C(AQ)(00-63)
00...0	→ C(AQ)(64-71)

EXPLANATION:

The memory operand C(Y) is processed as double-precision floating-point data. The absolute value of this data is normalized and its exponent, mantissa (bits 8-71), and 0 are loaded into C(E), C(AQ)(0-63), and C(AQ)(64-71), respectively.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFLP

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -127 , then ON

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

DFMP**9.2.8 DFMP**

DFMP	Double-Precision Floating Multiply	463(0)
------	---------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) * C(Y\text{-pair})]$ normalized $\rightarrow C(EAQ)$;
 $C(Y\text{-pair})$ unchanged

EXPLANATION:

This multiplication is executed in the following way.

$$C(E) + C(Y\text{-pair})_{0-7} \rightarrow C(E)$$

$C(AQ) * C(Y\text{-pair})(8-71)$ results in a 134-bit product plus sign. This sign plus the leading 71 bits are loaded into the AQ. $C(EAQ)$ normalized $\rightarrow C(EAQ)$.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFMP

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -128 , then ON

NOTES:

1. When indicator bit 32=1, floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFRD**9.2.9 DFRD**

DFRD	Double-Precision Floating Round	473(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ)$ rounded to 64 bits and normalized $\rightarrow C(EAQ)$

EXPLANATION:

A true round is performed on $C(EAQ)$ to reduce the mantissa of the floating-point number to 64 bits. The exponent is set to -128 if the rounded mantissa = 0.

This instruction is identical with FRD except that the rounding constant is added to bits 65-71 and the results are rounded to 64 bits of precision. Bits 64-71 of $C(AQ)$ are replaced by zeros.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFRD

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -128 , then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFSB**9.2.10 DFSB**

DFSB	Double-Precision Floating Subtract	577(0)
------	---------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(\text{EAQ}) - C(\text{Y-pair})]$ normalized $\rightarrow C(\text{EAQ})$
 $C(\text{Y-pair})$ unchanged

EXPLANATION:

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFSB

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DFSBI**9.2.11 DFSBI**

DFSBI	Double-Precision Floating Subtract Inverted	467(0)
-------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(Y\text{-pair}) - C(EAQ)]$ normalized $\rightarrow C(EAQ)$
 $C(Y\text{-pair})$ unchanged

EXPLANATION:

The two's complement of the subtrahend is first taken and the smaller value is then right shifted to equalize it. The shifted portion is truncated and the addition is executed. After addition, the sum is normalized and the 72 bits of the mantissa are loaded into AQ.

The order of execution of the operation conforms to that of the DFSB instruction. Normalization is defined under FNO.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DFSBI

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

DFST**9.2.12 DFST**

DFST	Double-Precision Floating Store	457(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(E) -> C(Y-pair)₀₋₇
 C(AQ)₀₋₆₃ -> C(Y-pair)₈₋₇₁
 C(EAQ) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

DFSTR

9.2.13 DFSTR

DFSTR	Double-Precision Floating Store Rounded	472(0)
-------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ)_{0-71}$ rounded, normalized $\rightarrow C(Y\text{-pair})$
 $C(EAQ)$ unchanged

EXPLANATION:

This instruction performs a true round on $C(EAQ)$ to 64 bits of precision in $C(AQ)$. The result is normalized and stored in the Y-pair. $C(EAQ)$ is unchanged. The exponent is stored as -128 if the rounded mantissa = 0. See the FRD instruction for the definition of true round.

Except for precision, this instruction is identical with the FSTR instruction.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

DFSTR**INDICATORS:**

Zero	If C(Y-pair) = floating-point zero, then ON; otherwise, OFF
Negative	If C(Y-pair)(8) = 1, then ON; otherwise, OFF
Exponent Overflow	If exponent > +127, then ON
Exponent Underflow	If exponent < -128, then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

DIAG

9.2.14 DIAG

DIAG	Diagnose	612(0)
------	----------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

EXPLANATION:

The functions executed by this instruction are specified in the TAG field and are summarized in the table below. Because no address generation is performed by this instruction, the y field is not used except for TAG = 01 and no check is performed.

<u>TAG (octal)</u>	<u>Function performed</u>
00	Halt CPU and notify Service Processor
01	Swap and Start
02	Reset & Start Watch Dog Timer (N/A V9000 – IPR)
03	Stop Watch Dog Timer (N/A V9000 – IPR)
04	Set Master CPU
05	NOP
06	rfu (IPR)
07	Read Configuration Register
10	RAR gated OR of DIAG ID word
11	Read and Clear MEEP fault ID word
12	Read gated OR of MEEP fault ID word
13-16	rfu (IPR)
17	Read Time Clock Synchronizer (N/A V9000)
20	rfu (IPR)
21	Retry Test
22-23	rfu (IPR)
24	Load Debug Mode Register
25-26	rfu (IPR)
27	Reserved to Generate IPR
30-77	rfu (IPR)

The functions determined by the values in the TAG field are explained below.

DIAG

TAG = 00 Halt CPU and Notify Service Processor:

The execution of this instruction causes the CPU to halt and report the CPU-down to the Service Processor (SP). The processing after the CPU down is performed by the software on the SP.

TAG = 01 Swap and Start:

The CPU which is executing this instruction (the relief CPU) assume the logical state of another physical CPU (target CPU) as specified by the lower 3 bits of the Y field of the DIAG instruction.

The target CPU state is defined by information contained in a fixed area of RMS that is prepared by the Service Processor or some other unit outside of the relief CPU.

A VMME fault will occur if DIAG TAG=01 execution is attempted in VMOS mode.

TAG = 02 Reset and Start Watch Dog Timer:

Execution of this instruction, if Performance Monitor is OFF, resets the "real" WDT and sets the watchdog timer (WDT) to the value loaded into each CPU by the SP and the WDT operation is started. The WDT is a timer that monitors the operating status of a CPU. When the WDT reaches zero, a Watch Dog Timer Runout fault is generated. In VMOS mode, this DIAG will reset the virtual WDT value in VPCB word 192 to all zero and set the WDT start flag (VPCB word 193, bit 0 = 1).

NOTE: Not implemented on V9000 platform. Execution causes IPR

TAG = 03 Stop Watch Dog Timer:

Execution of this instruction in NVM or VMM mode (if Performance Monitor is OFF) stops the watchdog timer (WDT) operation. In VMOS mode, this DIAG will reset the WDT start flag (VPCB word 193, bit 0) to 0. Subsequent execution of a DIAG instruction with TAG = 2, starts the WDT operation.

NOTE: Not implemented on V9000 platform. Execution causes IPR

DIAG

TAG = 04 Set Master CPU:

Execution of this instruction will load Reserve Memory location .SMCPU, with:

Bits	Description
00-14	Reserved for Hardware Use
15-17	Physical Number of CPU executing DIAG TAG 04
18-35	Reserved for Hardware Use

This is used by the Expanded Memory Adapter to complete an inter-system connect to a System Master CPU. It is also used by the SP to identify the CPU to which certain SP faults are to be directed.

TAG = 07 Read Configuration Register:

Upon execution of this instruction, the content of the Configuration Register is loaded into the A register. The Configuration Register is located in RMS address 10₈ and is loaded by the SP. The indicators are not affected. The format and content of the configuration register loaded into the A register are shown below.

00	CPU 0 connection	=1, if connected and operational
01	CPU 1 connection	=1, if connected and operational
02	CPU 2 connection	=1, if connected and operational
03	CPU 3 connection	=1, if connected and operational
04	CPU 4 connection	=1, if connected and operational
05	CPU 5 connection	=1, if connected and operational
06	CPU 6 connection	=1, if connected and operational
07	CPU 7 connection	=1, if connected and operational
08	RFU	=0
09-17	Reserved for Hardware Use	=00000000
18	IOC 0 connection	=1, if connected and operational
19	IOC 1 connection	=1, if connected and operational
20	IOC 2 connection	=1, if connected and operational
21	IOC 3 connection	=1, if connected and operational
22	IOC 4 connection	=1, if connected and operational
23	IOC 5 connection	=1, if connected and operational
24	IOC 6 connection	=1, if connected and operational
25	IOC 7 connection	=1, if connected and operational
26	RFU	=0
27-35	Reserved for Hardware Use	=00000000

DIAG**TAG = 10 RAR Gated OR of DIAG ID Word:**

This instruction Read Alter Rewrites the contents of A Register into the DIAG ID word in the RMS with the RAR gated OR command and notifies the SP of an interrupt. Then it proceeds to the next instruction.

TAG = 11 Read and Clear MEEP Fault ID Word:

This instruction reads the MEEP Fault ID word in the RMS and loads it into A Register with the Read and Clear command. Then it proceeds to the next instruction.

TAG = 12 Read Gated OR MEEP Fault ID Word:

This instruction stores the contents of A Register into the MEEP Fault ID word in the RMS with the Read Gated OR command and then proceeds to the next instruction.

TAG = 17 Read Time Clock Synchronizer:

This instruction, when executed in NVM or VMM with the Performance Monitor OFF and the TCS ON, will cause a "Read TCS" command to be sent to the Clock Maintenance Unit (CMU). After the command is sent to the CMU, the CPU will proceed to the next instruction. If the CPU is in VMOS mode with the Performance Monitor OFF and the TCS ON, a VMME Fault will occur. If the Performance Monitor is ON or the TCS is OFF when the DIAG Tag 17 is executed, a NOP will be performed.

Upon receipt of the "Read TCS" command, the CMU will capture the year, month, day, hour, minute, and second data; convert this BCD data adding millisecond and microsecond information and save in Reserve Memory locations 360 - 362.

NOTE: Not implemented on V9000 platform.

TAG = 21 Retry Test**TAG = 24 Load Debug Mode Register:**

This instruction loads the Debug Mode Register (DMR) and Address Match Register (AMR) from the AQ Register in NVM or VMM mode. Execution in VMOS mode will result in a NOP.

AQ ₀₀₋₀₅	->	DMR
AQ ₁₂₋₆₃	->	AMR
AQ ₀₆₋₁₁		not used

DIAG

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. A Command fault occurs if this instruction is executed in Slave or Master mode.
2. An IPR fault occurs if used as the target of a RPT, RPD, or RPL instruction or TAG= 06, 13 thru 16, 20, 22, 23, 25, 26, 27 thru 77.

DIS**9.2.15 DIS**

DIS	Delay Until Interrupt Signal	616(0)
-----	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

No operation takes place, and the processor does not continue with the next instruction, but waits for a program interrupt signal.

ILLEGAL ADDRESS MODIFICATIONS:

IT, IR, or RI cause an IPR fault. Other modification specified is performed including the modification of any indirect words specified. However, the effective address has no effect on the operation.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

DIS

NOTES:

1. The inhibit bit in this instruction only affects the recognition of a Timer Runout (TROF) fault.
Inhibit ON delays the recognition of a TROF until the processor enters Slave mode.
Inhibit OFF allows the TROF to interrupt the DIS state.
2. For all other faults and interrupts, the inhibit bit is ignored.
3. The use of this instruction in the Slave or Master mode causes a Command fault.

DIV**9.2.16 DIV**

DIV	Divide Integer	506(0)
-----	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

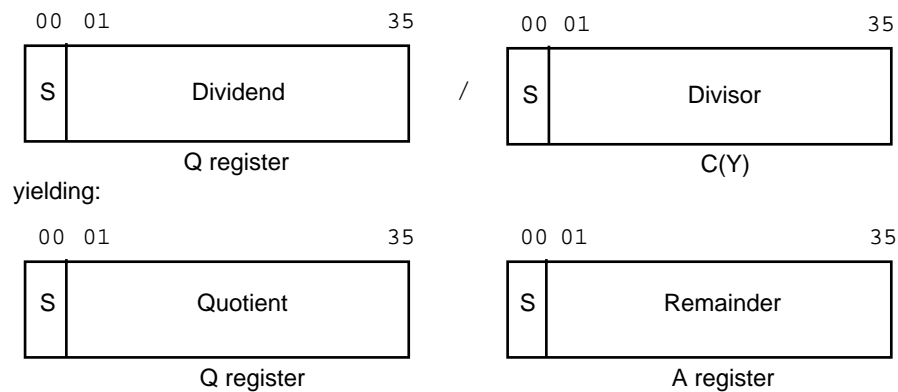
Any

SUMMARY:

$C(Q) / C(Y)$
 integral quotient $\rightarrow C(Q)$, right-adjusted
 integral remainder $\rightarrow C(A)$, right-adjusted
 $C(Y)$ unchanged

EXPLANATION:

$C(Q)$ and $C(Y)$ are considered as 36-bit integers (including sign). The integer quotient of $C(Q)$ divided by $C(Y)$ is loaded into the Q register and the integer remainder is loaded into the A register. The remainder sign is the same as that of the dividend unless the remainder is zero.



DIV

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

	<u>If division takes place</u>	<u>If no division takes place</u>
Zero	If C(Q) = 0, ON; otherwise, OFF	If divisor = 0, ON; otherwise, OFF
Negative	If bit 0 of C(Q) = 1, ON; otherwise, OFF	If dividend < 0, ON; otherwise, OFF

NOTE:

If the dividend = -2^{35} and the divisor = ± 1 , or if the divisor is 0 under any condition, division does not take place. Instead, a Divide Check fault occurs, C(Y) remains unchanged, C(Q) contains the dividend magnitude, and the Negative indicator reflects the dividend sign, and C(A) is set to zero.

DIVN**9.2.17 DIVN**

DIVN	Divide Integer with No Remainder	504(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

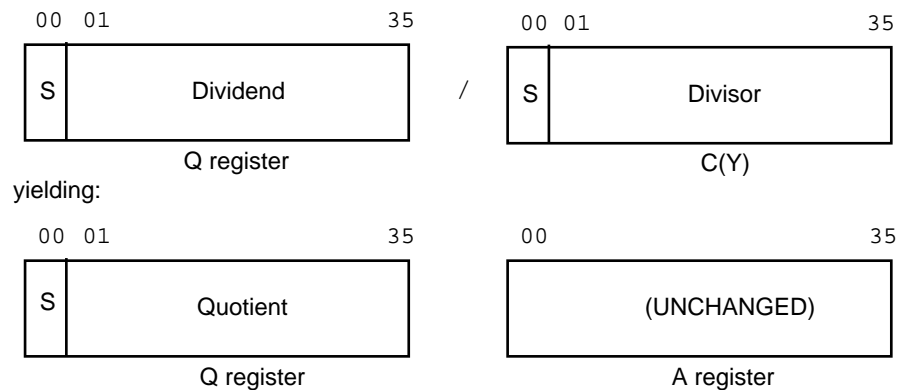
Any

SUMMARY:

$C(Q) / C(Y)$
 integral quotient $\rightarrow C(Q)$, right-adjusted
 C(A) unchanged
 C(Y) unchanged

EXPLANATION:

$C(Q)$ and $C(Y)$ are considered as 36-bit integers (including sign). The integer quotient of $C(Q)$ divided by $C(Y)$ is loaded into the Q register and the A register is unchanged.



DIVN

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

	<u>If division takes place</u>	<u>If no division takes place</u>
Zero	If C(Q) = 0, ON; otherwise, OFF	If divisor = 0, ON; otherwise, OFF
Negative	If bit 0 of C(Q) = 1, ON; otherwise, OFF	If dividend < 0, ON; otherwise, OFF

NOTE:

If the dividend = -2^{35} and the divisor = ± 1 , or if the divisor is 0 under any condition, division does not take place. Instead, a Divide Check fault occurs, C(Y) remains unchanged, C(Q) contains the dividend magnitude, and the Negative indicator reflects the dividend sign.

DLY**9.2.18 DLY**

DLY	Delay EA Process Clock	730(1)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Delays the EA Process Clock.

ILLEGAL ADDRESS MODIFICATIONS:

Ignored

ILLEGAL REPEATS:

None, Ignored.

DRL

9.2.19 DRL

DRL	Derail Fault	002(0)
-----	--------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Generates a DRL fault, which causes the processor to switch to Privileged Master Mode and execute an Inward CLIMB instruction using the entry descriptor obtained from the word pair in memory locations 32 and 33 octal.

ILLEGAL ADDRESS MODIFICATIONS:

Not executed

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTE:

Refer to Section 6, Faults and Interrupts.

DTB

9.2.20 DTB

DTB	Decimal-to-Binary Convert	305(1)
-----	---------------------------	--------

FORMAT:

00	10 11	17 18	Op Code	27 28 29	35
000000000000	MF2	305(1)	I	MF1	

00	02 03	17 18 20 21	22 23 24	29 30	32 35	
Y1		CN1	T N 1	S1	000000	N1
AR#	Y1					00

00	02 03	17 18 20 21	22 23 24	29 30	32 35	
Y2		CN2	T N 2	S2	000000	N2
AR#	Y2					00

CODING FORMAT:

The DTB instruction code is shown below.

1	8	16

DTB	(MF1) , (MF2)	
NDSC _n	LOCSYM, CN, N, S, , AM	
NDSC9	LOCSYM, CN, N, , , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) $\xrightarrow{\text{converted}}$ C(string 2)

DTB

The string of decimal characters of data type TN1, sign and decimal type S1 (S1 = 00 is illegal), and scale factor 0 that starts at YC1 is converted into a two's complement binary integer and stored, right-justified, as a character string of length L2 and starting at location YC2. If the string generated is longer than L2, the high-order excess is truncated and the Overflow indicator is set. CN2 is given in the 9-bit character format with legal codes of 000, 010, 100, and 110.

If string one contains more than 32, when the generated binary string is longer than L2, the upper bits are truncated and the overflow indicator is set.

CN2 specifies the value for the 9-bit character format, the correct codes being 000, 010, 100, or 110. L2 specifies the length of the stored binary value. It is specified in 9-bit units and must be equal to or less than 8. The length of the stored binary value is 9, 18, 27, 36, 45, 54, 63, or 72 bits.

Provided that string 1 and string 2 are not overlapped, the contents of string 1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If all the resultant bits generated are zero, then ON; otherwise, OFF
Negative	If the resultant sign is negative, then ON; otherwise, OFF
Overflow	If L2 is less than the number of 9-bit segments generated, then ON; otherwise, unchanged

NOTES:

1. An Illegal Procedure fault occurs under the following conditions:
 - DU or DL modifications are used for MF1 or MF2
 - L2 is less than 1 or > 8
 - CN2 does not contain a legal code
 - S1 = 00
 - illegal digit or sign is detected in string 1
 - N1 is not large enough to specify the number of characters required for the specified sign and/or exponent, plus at least one digit.
2. An IPR fault occurs if IT, IR or RI address modification is specified or if an illegal repeat is used.
3. The result is stored correctly, and overflow does not occur, even when it is $-2^{(9 \times L2 - 1)}$.
4. The convert operation is executed for all digits even when the number of valid digits in string 1 exceeds 22.
5. If overflow occurs, the result is truncated and stored. In this case, the negative indicator is set according to the most significant bit of the stored data. (If the most significant bit(MSB) = 1, then NEG = 1. If MSB = 0, then NEG = 0.

Example:

```

String 1 = +102310
L2 = 1
Converted Result   = +1 111 111 1112
                    = +1  $\overbrace{111\ 111\ 111}_2$ 
Stored Data       = 111 111 1112
Negative Indicator = 1

```

DTB

EXAMPLES:

1	8	16	32

	DTB		
	NDSC4	FLD1,3,5,2	decimal operand descriptor
	NDSC9	FLD2,0,4	binary operand descriptor
	USE	CONST.	memory contents in octal
FLD1	EDEC	8P1234-	000001043115
FLD2	BSS	1	77777775456 (Result)
	USE		any indicators set? negative
*			
	DTB		
	NDSC9	FLD1,0,22,3	decimal operand descriptor
	NDSC9	FLD2,0,8	binary operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	22A236118324143	4822606847 (max decimal value)
FLD2	BSS	2	37777777777777777777
			(Result)
	USE		any indicators set? none
*			
	DTB		
	NDSC4	FLD1,3,3,3	decimal operand descriptor
	NDSC9	FLD2,2,2	binary operand descriptor
	USE	CONST.	memory contents in octal
FDL1	EDEC	8P51200	000005022000
FLD2	DEC	-1	777777001000
	USE		any indicators set? none
*			
	DTB		
	NDSC9	FLD1,0,4,	decimal operand descriptor
	NDSC9	FLD2,3,1	binary operand descriptor
	USE	CONST.	memory contents in octal
FLD1	EDEC	4A1023	061060062063
FLD2	DEC	0	000000000777
	USE		any indicators set? overflow

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32
	EAX0	0	load FLD character mod into X0
	EAX2	2	load FLD2 length into X4
	EAX7	FLD2	load FLD2 address mod into X7
	AWDX	0,7,4	put FLD2 address mod into AR4
	DTB	(,,1),(1,1,,0)	with modification
	ARG	1,,4	ptr., FLD1 indirect descriptor
	NDSC9	0,,X2,,,4	bin FLD2 descriptor (FLD2,0,2)
	TZE	*+3	zeros was the result
	TMI	*+2	negative result
	TOV	*+1	high-order bit truncated
	USE	CONST.	memory contents in octal
FLD1	EDEC	4PL-512	325022000000
FLD2	OCT	1111111	7770001111111
	NDSC4	FLD1,0,4,1	decimal operand descriptor
	USE		any indicators set? negative

DTRACE

9.2.21 DTRACE

DTRACE	Dump Trace Table	733(1)
--------	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Implemented as NOP. Dumps the hardware trace table to memory on Olympus.

EXPLANATION:

No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the generated address may cause faults.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTES:

1. An Illegal Procedure fault occurs if an illegal repeat is used.
2. Because address preparation takes place, modification may result in a Bounds fault.

DUFA**9.2.22 DUFA**

DUFA	Double-Precision Unnormalized Floating Add	437(0)
------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

[C(EAQ) + C(Y-pair)] not normalized -> C(EAQ)
C(Y-pair) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DUFA

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DUFM**9.2.23 DUFM**

DUFM	Double-Precision Unnormalized Floating Multiply	423(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) * C(Y\text{-pair})]$ not normalized $\rightarrow C(EAQ)$
 $C(Y\text{-pair})$ unchanged

EXPLANATION:

This multiplication is executed like the DFMP instruction, except that the final normalization is performed only in the case of both factor mantissas being = $-1.00\dots 0$.

Except for the precision of the mantissa of the operand from main memory, the DUFM instruction is identical to the UFM instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DUFM

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DUFS**9.2.24 DUFS**

DUFS	Double-Precision Unnormalized Floating Subtract	537(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

[C(EAQ) - C(Y-pair)] not normalized -> C(EAQ)
C(Y-pair) unchanged

EXPLANATION:

The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The portion shifted out is truncated and addition is executed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

DUFS

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

DV2D

9.2.25 DV2D

DV2D	Divide Using Two Decimal Operands	207(1)
------	-----------------------------------	--------

FORMAT:

00 01	09 10 11	17 18	27 28 29	35		
P	000000000	R D	MF2	207(1)	I	MF1

00 0203	17 18	20 21 22 23 24	29 30	35		
Y1		CN1	T N 1	S X 1	SF1	N1
AR#	Y1					

00 0203	17 18	20 21 22 23 24	29 30	35		
Y2		CN2	T N 2	S X 2	SF2	N2
AR#	Y2					

CODING FORMAT:

The DV2D instruction code is shown below.

1	8	16

DV2D	(MF1) , (MF2) , RD , P	
NDSC _n	LOCSYM , CN , N , S , SF , AM	
NDSC _n	LOCSYM , CN , N , S , SF , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

DV2D

OPERATING MODES:

Any

SUMMARY:

`C(string 2) / C(string 1) -> C(string 2)`

EXPLANATION:

Same as for DV3D except that the quotient is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2.

If the denominator is greater than the numerator, the leading zero in the quotient is removed and is not counted as a significant digit.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Exponent Overflow	If exponent of floating-point result is > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128, then ON; otherwise, unchanged
Overflow	If fixed-point integer overflow, then ON; otherwise, unchanged
Truncation	If the least significant digits are truncated without rounding, then ON; otherwise, OFF

NOTES:

1. A Divide Check fault occurs under either of the following three conditions:
 - the divisor is equal to zero; the divisor is the number starting at YC1;
 - S3 specifies that the quotient be stored in scaled format and the calculated length required for the quotient is greater than 63 (refer to length requirements above);
 - a dividend is a floating-point zero with an exponent >63 and the quotient is not a floating-point decimal.

2. An Illegal Procedure fault occurs under these conditions:
 - DU or DL modification is specified for MF1 or MF2;
 - any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor;
 - the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

3. If an illegal digit or sign is detected, the receive field is not changed before the IPR fault occurs.

EXAMPLES:

1	8	16	32

	DV2D		
	NDSC4	FLD1,4,4,2,-4	divisor operand descriptor
	NDSC4	FLD2,0,8,0	dividend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P2+	0002+
FLD2	EDEC	8P+8642E0	+08642 +0
	USE	+43210 +3	(Quotient)
	DV2D	,,1	with rounding option
	NDSC9	FLD1,0,4,1,-3	divisor operand descriptor
	NDSC4	FLD2,0,8,1,-2	dividend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4A+5	+005
FLD2	EDEC	8P+1234	+0001234
	USE	+0246800	(Quotient)
*			indicators on? none

DV2DX

9.2.26 DV2DX

DV2DX	Divide Using Two Decimal Operands Extended	247(1)
-------	--	--------

FORMAT:

00	01	02	09	10	11	17	18	27	28	29	35
C	N	00000000	R	D	MF2	247(1)			I	MF1	
S	S										

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1			CN1			T	S	SF1		N1		
AR#	Y1					N	X					
						1	1					

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2			CN2			T	S	SF2		N2		
AR#	Y2					N	X					
						2	2					

CODING FORMAT:

1	8	16

DV2DX	(MF1), (MF2), RD, CS, NS	
NDSCn	LOCSYM, CN, N, SX, SF, AM	
NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 2) / C(string 1) -> C(string 2)

DV2DX**EXPLANATION:**

Same as for DV3DX except that the quotient is stored using YC2, TN2, SX2 and, if SX2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If fixed-point integer overflow, then ON; otherwise, unchanged
Truncation	If the least significant digits are truncated without rounding, then ON; otherwise, OFF

DV2DX

NOTES:

1. A Divide Check fault occurs under either of the following three conditions:
 - the divisor (the number starting at YC1) is equal to zero;
 - S3 specifies that the quotient be stored in scaled format and the calculated length required for the quotient is greater than 63 (refer to length requirements above);
 - a dividend is a floating-point zero with an exponent > 63 and the quotient is not a floating-point decimal.
2. An Illegal Procedure fault occurs under these conditions:
 - DU or DL modification is specified for MF1 or MF2;
 - any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor;
 - the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. If an illegal digit or sign is detected, the receive field is not changed before the IPR fault occurs.
4. See MVNX for information about coding of overpunched signs.

DV3D

9.2.27 DV3D

DV3D	Divide Using Three Decimal Operands	227(1)
------	-------------------------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
P	0	MF3	0	R	D	MF2	227(1)			I	MF1	

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1				CN1	T N 1	S1	SF1	N1				
AR#	Y1											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2				CN2	T N 2	S2	SF2	N2				
AR#	Y2											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y3				CN3	T N 3	S3	SF3	N3				
AR#	Y3											

DV3D

CODING FORMAT:

The DV3D instruction code is shown below.

1	8	16

	DV3D	(MF1) , (MF2) , (MF3) , RD , P
	NDSC _n	LOCSYM , CN , N , S , SF , AM
	NDSC _n	LOCSYM , CN , N , S , SF , AM
	NDSC _n	LOCSYM , CN , N , S , SF , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

$C(\text{string } 2) / C(\text{string } 1) \rightarrow C(\text{string } 3)$

EXPLANATION:

If the denominator is greater than the numerator, the leading zero in the quotient is removed and is not counted as a significant digit.

The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, is divided into the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The quotient is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the quotient is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If S3 indicates a floating-point format, the quotient is right-justified to preserve the most significant nonzero digits; this may cause least-significant-digit truncation.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign.

If RD is a 1, the quotient is rounded prior to storage.

Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

The divide operation stops when the number of required digits have been formed or, in the case where rounding is specified ($RD = 1$), when the required number of quotient digits plus 1 have been formed.

In fixed-point operations or floating-point operations where the quotient is stored in fixed-point format, the required number of quotient digits is determined in the following way. When the quotient descriptor specifies that the quotient is to be stored in fixed-point format, the necessary number of quotient digits to form is calculated:

$$\#QD = (LD - \#LZD + 1) - (LDR - \#LZR) + (ED - EDR - EQ)$$

where:

#LZD = number of leading zeros in dividend
 #QD = number of quotient digits to form
 LD = length of dividend
 LDR = length of divisor
 #LZR = number of leading zeros in divisor
 ED = exponent of dividend
 EDR = exponent of divisor
 EQ = scale factor for quotient

The hardware performs this calculation prior to beginning the divide operation and, if $\#QD > 63$, the divide operation does not take place. A Divide Check fault occurs. If $\#QD \leq 0$, then zero is stored.

In a floating-point divide operation, the required number of quotient digits is determined as follows. With the divisor greater than the dividend, the algorithm generates a leading zero in the quotient. This characteristic of the algorithm is taken into account along with rounding requirements when determining the required number of digits for the quotient, so that the resulting quotient contains as many significant digits as specified by the quotient operand descriptor.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

DV3D

ILLEGAL REPEATS:

RPT, RPD, RPL

In a floating-point divide operation, the required number of quotient digits is determined as follows. With the divisor greater than the dividend, the algorithm generates a leading zero in the quotient. This characteristic of the algorithm is taken into account along with rounding requirements when determining the required number of digits for the quotient, so that the resulting quotient contains as many significant digits as specified by the quotient operand descriptor.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If fixed-point integer overflow, then ON; otherwise, unchanged
Truncation	If the least significant digits are truncated without rounding, then ON; otherwise, OFF

NOTES:

1. A Divide Check fault occurs under either of the following three conditions:
 - the divisor is equal to zero; the divisor is the number starting at YC1;
 - the quotient is in fixed point format, the dividend equals zero, and the number of digits required to express the quotient is less than or equal to 63; if the quotient length is greater than or equal to 64, a Divide Check fault does not occur and the quotient is set to all zeros;
 - a dividend is a floating-point zero with an exponent > 63 and the quotient is not a floating-point decimal.

2. An Illegal Procedure fault occurs under these conditions:
 - DU or DL modification is specified for MF1 or MF2;
 - any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor;
 - the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.

3. If an illegal digit or sign is detected, the receive field is not changed before the IPR fault occurs.

EXAMPLE:

1	8	16	32
	DV3D	,, ,1,1	rounding, plus sign options
	NDSC9	FLD1,1,3,2,-2	divisor operand descriptor
	NDSC4	FLD2,0,9,0	dividend operand descriptor
	NDSC4	FLD3,2,6,1,-1	quotient operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4A2- 002-	
FLD2	EDEC	9P-876543E-3	-876543-3
FLD3	BSS	1 xx+38272	(Quotient)
	USE		instruction fault? overflow

DV3D

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX2	2	load character mod into X2
	EAX7	8	load FLD2 length into X7
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	DV3D	(1,,2),(,1),(,,1),1,1	with addr mod options
	NDSC9	0,0,2,3,-2,4	divisor operand descriptor
*			(FLD1,2,2,3,-2)
	NDSC9	FLD2,0,X7,0	dividend operand descriptor
*			(FLD2,0,8,0)
	ARG	2,2,4	ptr quotient operand desc
	USE	CONST.	memory contents
FLD1	EDEC	4A2	0002
FLD2	EDEC	8A+876543E-3	+876543-3
FLD3	BSS	1	x+438272
	NDSC4	FLD3,1,7,1,-1	quotient operand descriptor
	USE		instruction fault? none

DV3DX

9.2.28 DV3DX

DV3DX	Divide Using Three Decimal Operands Extended	267(1)
-------	--	--------

FORMAT:

00	01 02	08	09 10 11	17 18	27	28 29	35	
C	N	MF3	0	R	MF2	267(1)	I	MF1
S	S			D				

00	02 03	17 18	20 21 22 23 24	29 30	35	
Y1		CN1	T N 1	S X 1	SF1	N1
AR#	Y1					

00	02 03	17 18	20 21 22 23 24	29 30	35	
Y2		CN2	T N 2	S X 2	SF2	N2
AR#	Y2					

00	02 03	17 18	20 21 22 23 24	29 30	35	
Y3		CN3	T N 3	S X 3	SF3	N3
AR#	Y3					

CODING FORMAT:

1	8	16

DV3DX	(MF1), (MF2), (MF3), RD, CS, NS	
NDSCn	LOCSYM, CN, N, SX, SF, AM	
NDSCn	LOCSYM, CN, N, SX, SF, AM	
NDSCn	LOCSYM, CN, N, SX, SF, AM	

DV3DX

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

$C(\text{string } 2) / C(\text{string } 1) \rightarrow C(\text{string } 3)$

EXPLANATION:

The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is divided into the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The quotient is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type SX3.

If SX3 indicates a fixed-point format, the quotient is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If SX3 indicates a floating-point format, the quotient is right-justified to preserve the most significant nonzero digits. This change may cause least-significant-digit truncation.

The character set is defined by CS (EBCDIC/ASCII). Placement of overpunched sign in the output is controlled by NS. (Refer to the introductory pages of this section for definition of the NS field.) If RD is a 1, the quotient is rounded prior to storage. The contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, or MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If fixed-point integer overflow, then ON; otherwise, unchanged
Truncation	If the least significant digits are truncated without rounding, then ON; otherwise, OFF

NOTES:

1. The explanation of the divide operation in the DV3D description applies.
2. A divide check fault occurs under either of the following two conditions:
 - the divisor (the number starting at YC1) is equal to zero
 - the quotient is in fixed point format, the dividend equals zero, and the number of digits required to express the quotient is less than or equal to 63; if the quotient length is greater than or equal to 64, a Divide Check fault does not occur and the quotient is set to all zeros;
3. Refer to the specifications about MVNX for information about coding of overpunched signs.
4. IPR fault conditions are the same as for DV3D.

DVF

9.2.29 DVF

DVF	Divide Fraction	507(0)
-----	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

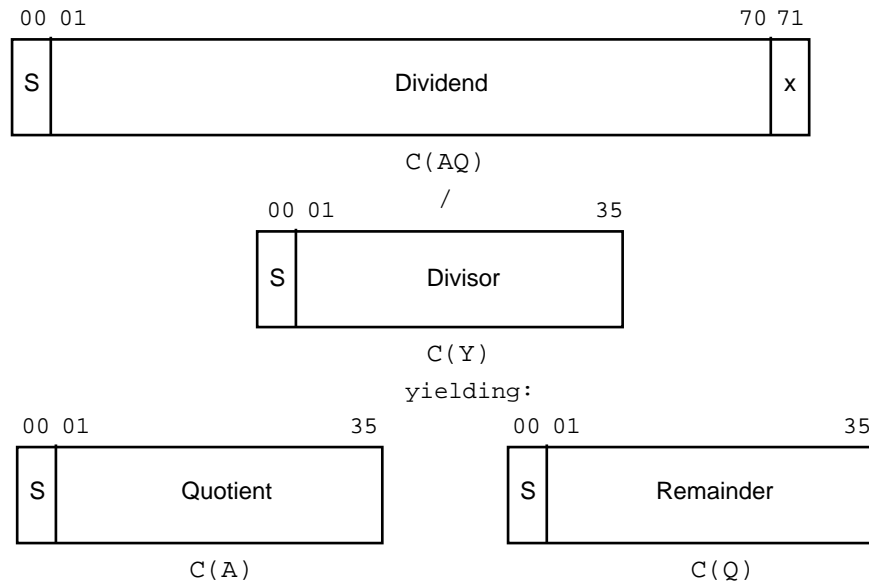
Any

SUMMARY:

$C(AQ) / C(Y)$
 fractional quotient -> C(A), left adjusted
 fractional remainder -> C(Q), left adjusted
 C(Y) unchanged

EXPLANATION:

This instruction divides a 71-bit fractional dividend (including sign) by a 36-bit fractional divisor (including sign) to form a 36-bit fractional quotient (including sign) and a 36-bit fractional remainder (including sign). Bit 35 of the remainder corresponds to bit 70 of the dividend. The remainder sign is equal to the dividend sign unless the remainder is zero. Bit 71 of C(AQ) is not used.



If $|\text{dividend}| \geq |\text{divisor}|$ or if the divisor = 0, division does not take place. Instead, a Divide Check fault occurs, $C(Y)$ remains unchanged, $C(AQ)$ contains the dividend magnitude as an absolute value, and the negative indicator reflects the dividend sign.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

	<u>If division takes place</u>	<u>If no division takes place</u>
Zero	If $C(A) = 0$, ON; otherwise, OFF	If divisor = 0, ON; otherwise, OFF
Negative	If bit 0 of $C(A(0)) = 1$, ON; otherwise, OFF	If dividend < 0, ON; otherwise, OFF

DVRR

9.2.30 DVRR

DVRR	Divide Register by Register	533(1)
------	-----------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	-----
DVRR	R1,R2		

OPERATING MODES:

Executes in ES/EI mode

SUMMARY:

R1 = 0, 2, 4, 6, AQ
 R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

Quotient of C(R1-odd) / C(R2) -> C(R1-odd)
 Remainder of C(R1-odd) / C(R2) -> C(R1-even)
 C(R2) unchanged

EXPLANATION:

A register pair is specified in R1. The content of the odd-numbered register, or Q if AQ is specified, is divided by C(R2). The resulting quotient is loaded into R1-odd and the remainder into R1-even.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero (division)	If $C(R1\text{-odd}) = 0$, then ON; otherwise, OFF
Zero (no division)	If divisor = 0, then ON; otherwise, OFF
Negative (division)	If $C(R1\text{-odd})(0) = 1$, then ON, otherwise, OFF
Negative (no division)	If dividend < 0, then ON; otherwise, OFF

NOTES:

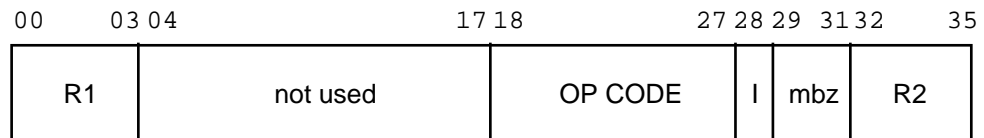
1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.
3. Both the dividend and divisor are regarded as a 36-bit signed integer. The sign of the remainder is the same as that of the dividend unless the remainder is 0.
4. A Divide Check fault occurs in the following cases:
 - Dividend = -2^{35} and divisor = -1
 - Divisor = 0
 - In these cases, the instruction is not executed. C(R2) remains unchanged, C(R1-odd) takes the absolute value of the dividend, and C(R1-even) is 0. If the dividend is -2^{35} , then -2^{35} is loaded into R1-odd.

DVRRN

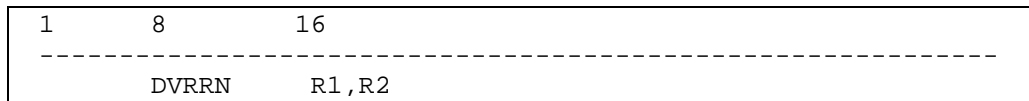
9.2.31 DVRRN

DVRRN	Divide Register by Register with No Remainder	414(1)
-------	--	--------

FORMAT:



CODING FORMAT:



OPERATING MODES:

Executes in ES/EI mode

SUMMARY:

R1 = 0, 2, 4, 6, AQ
 R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

Quotient of C(R1-odd) / C(R2) -> C(R1-odd)
 C(R1-even) unchanged
 C(R2) unchanged

EXPLANATION:

A register pair is specified in R1. The content of the odd-numbered register, or Q if AQ is specified, is divided by C(R2). The resulting quotient is loaded into R1-odd.

DVRRN**ILLEGAL ADDRESS MODIFICATIONS:**

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero (division)	If $C(R1\text{-odd}) = 0$, then ON; otherwise, OFF
Zero (no division)	If divisor = 0, then ON; otherwise, OFF
Negative (division)	If $C(R1\text{-odd}) (0) = 1$, then ON, otherwise, OFF
Negative (no division)	If dividend < 0 , then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.
3. Both the dividend and divisor are regarded as a 36-bit signed integer. The sign of the remainder is the same as that of the dividend unless the remainder is 0.
4. A Divide Check fault occurs in the following cases:
 - Dividend = -2^{**35} and divisor = -1
 - Divisor = 0
 - In these cases, the instruction is not executed. $C(R2)$ remains unchanged, $C(R1\text{-odd})$ takes the absolute value of the dividend, and $C(R1\text{-even})$ is unchanged. If the dividend is -2^{**35} , then -2^{**35} is loaded into R1-odd.

Notes

10. Machine Instruction Descriptions (E-G)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 10.1, Machine Instruction Descriptions (E)
- Section 10.2, Machine Instruction Descriptions (F)
- Section 10.3, Machine Instruction Descriptions (G)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

10.1 Machine Instruction Descriptions (E)

Following is a detailed description of the processor instructions and operation codes beginning with the letter E.

10.1.1 EAA

EAA	Effective Address to A-Register	635(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EAA

SUMMARY:

NS mode

Y -> C(A) (0-17) ;
 0...0 -> C(A) (18-35) ;
 C(Y) unchanged

ES/EI mode

00 -> C(A) (0-1) ;
 Y(0-33) -> C(A) (2-35) ;
 C(Y) unchanged

EXPLANATION:

This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(A) = 0, then ON; otherwise, OFF

Negative If C(A)(0) = 1, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES/EI mode, the negative indicator is always set to OFF.

EAQ**10.1.2 EAQ**

EAQ	Effective Address to Q-Register	636(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

Y → C(Q)(0-17);
00...0 → C(Q)(18-35); C(Y) unchanged

ES/EI Mode

00...0 → C(Q)(0-1)
Y(0-33) → C(Q)(2-35)

EXPLANATION:

This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

EAQ

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES/EI mode, the negative indicator is always set to OFF.

EAX_n**10.1.3 EAX_n**

EAX _{<u>n</u>}	Effective Address to Index Register <u>n</u>	62 _{<u>n</u>} (0)
-------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by opcode:
 $Y(0-17) \rightarrow (X_n)$; $C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by opcode:
 00 $\rightarrow C(GX_n)(0-1)$
 $Y(0-33) \rightarrow C(GX_n)(2-35)$

EXPLANATION:

This instruction permits inter-register data movement. The data source is specified by the address modification and the data destination by the operation code of the instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

EAX_n

ILLEGAL REPEATS:

RPT, RPD, or RPL of EAX0

RPL of any EAX_n

INDICATORS:

Zero If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF

Negative If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. In the ES/EI mode, the negative indicator is always set to OFF.

EPAT**10.1.4 EPAT**

EPAT	Effective Pointer and Address to Test	412(1)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master mode

EXPLANATION:

This instruction tests the virtual address to real memory address mapping function of the hardware. Addresses are generated in the normal sequence and stored in six words in the Reserve Memory Space (RMS) words 64_{10} - 71_{10} (100_8 - 105_8) as described below.

RMS addr.	Mode	bit range	# of bits	Description
100 ₈	SV	00-26	27	Real double word address (mod. 2)
		27-35	9	zeroes
101 ₈	SVMX	00-04	5	zeroes
		05-35	31	Real double word address (mod. 2)
101 ₈	any	00-35	36	Virtual Byte Address (not ORed)
102 ₈	any	00-17	18	zeroes
		18-35	18	EWSQ# (ORed); upper 9 bits zeroes in SV.
103 ₈	any	00-35	36	Virtual Byte Address (not ORed)
104 ₈	any	00-35	36	Segment descriptor upper word
105 ₈	any	00-35	36	Segment descriptor lower word

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

EPAT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.
2. A command fault occurs if this instruction is executed in Slave or Master mode.
3. A-Register will contain the same value as RMS location 100₈.

EPPR_n**10.1.5 EPPR_n**

EPPR _n	Effective Pointer to Pointer Register <u>n</u>	63 _n (1)
-------------------	---	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

These sets of eight instructions generate an effective address (EA) and load it into the pointer register (AR_n, SEGID_n, DR_n.)

NS Mode

Effective address (EA) -> C(AR)(0-23)
 Effective SEGID -> C(SEGID_n)
 Effective DR -> C(DR_n)

ES/EI Mode

Effective address (ES)(4-39) -> C(AR)(0-35)
 Effective SEGID -> C(SEGID_n)
 Effective DR -> C(DR_n)

EXPLANATION:

If the instruction bit 29 = 0, AR is not used for generation of the effective address and the AR_n byte and bit portions are set to zero.

When the instruction bit 29 = 0, the generated operand address is in the instruction segment. The ISR and SEGID(IS) content is loaded into DR_n and SEGID_n respectively.

EPPR_n

If the instruction bit 29 = 1, the Address Register AR_n specified with bits 0, 1, and 2 of the instruction word are used to generate the effective address. Provided that indirect modification is not specified, the AR_n byte and bit portions are preserved during computation of the effective address and loaded into the byte and bit portions of the corresponding AR_n. If indirect modification is specified, zero is loaded into the AR_n byte and bit portions.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

Modifications DU, DL, CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

EXAMPLE:

1	8	16	32

	ADQ	=3HOBI,DC	file codefile
	ORQ	=O400000,DL	read permissions
	EPPR0	ALCPRF	allocate file command block
	PPME	ALPRMF,2	allocate file
*			
ALEPRF	VEC	.ISR,NAME,NAMEX,(R,W,S)	
VEC		.ISR,CBUFF,CBUFFX,(R,W,S)	
.			
NAME	BCI	4	
	.		
	.		
	.		
NAMEX	EQU	* -NAME	
CBUFF	BSS	355	
CBUFFX	EQU	* -CBUFF	

ERA**10.1.6 ERA**

ERA	EXCLUSIVE OR to A-Register	675(0)
-----	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 ,
 $C(A)(i) \text{ XOR } C(Y)(i) \rightarrow C(A)(i)$;
 $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

ERAQ

10.1.7 ERAQ

ERAQ	EXCLUSIVE OR to AQ-Register	677(0)
------	-----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

```
For i = 0 to 71,
  C(AQ)(i) XOR C(Y-pair)(i) -> C(AQ)(i);
  C(Y-pair) unchanged
```

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

ERQ**10.1.8 ERQ**

ERQ	EXCLUSIVE OR to Q-Register	676(0)
-----	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 ,
 $C(Q)(i) \text{ XOR } C(Y)(i) \rightarrow C(Q)(i)$;
 $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

ERRR

10.1.9 ERRR

ERRR	EXCLUSIVE OR Register to Register	537(1)
------	-----------------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
----	-------	-------	----------	-------	----

R1	not used	OP CODE	I	mbz	R2
----	----------	---------	---	-----	----

CODING FORMAT:

1	8	16	32

ERRR		R1, R2	

OPERATING MODES:

Executes in ES/EI mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q;
 where i = 0, 1, 2, ..., 35:
 C(R1)(i) XOR C(R2)(i) -> C(R1)(i);
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ERRR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If $C(R1) = 0$, then ON; otherwise, OFF

Negative If $C(R1)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

ERSA

10.1.10 ERSA

ERSA	EXCLUSIVE OR to Storage with A-Register	655(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 ,
 $C(A)(i) \text{ XOR } C(Y)(i) \rightarrow C(Y)(i)$;
 $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. See Examples under ERA.

ERSQ

10.1.11 ERSQ

ERSQ	EXCLUSIVE OR to Storage with Q-Register	656(0)
------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 ,
 $C(Q)(i) \text{ XOR } C(Y)(i) \rightarrow C(Y)(i)$;
 $C(Q)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

ERSQ

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

1	8	16	32

	LDQ	=1,DL	
	ERSQ	FLAG	
* If bit 35 of FLAG is ON, then set to zero			

ERSX_n

10.1.12 ERSX_n

ERSX _{<u>n</u>}	EXCLUSIVE OR to Storage with Index Register <u>n</u>	64 _{<u>n</u>} (0)
--------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0,1,...,7 as determined by op code:

For i = 0 to 17,

C(X_n)(i) XOR C(Y)(i) -> C(Y)(i);

C(X_n) and C(Y)(18-35) unchanged

ES/EI Mode

For n = 0,1,...,7 as determined by op code:

For i = 0 to 35,

C(GX_n)(i) XOR C(Y)(i) -> C(Y)(i);

C(GX_n) is unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL of ERSX0

RPL of any ERSX_n

INDICATORS:

NS Mode

Zero If $C(Y)(0-17) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

ES/EI Mode

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

ERX_n

10.1.13 ERX_n

ERX _{<u>n</u>}	EXCLUSIVE OR to Index Register <u>n</u>	66 _{<u>n</u>} (0)
-------------------------	--	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0,1,...,or 7 as determined by op code
 For i = 0 to 17,
 C(X_n)(i) XOR C(Y)(i) -> C(X_n)(i);
 C(Y) unchanged

ES/EI Mode

For n = 0,1,...,or 7 as determined by op code
 For i = 0 to 35,
 C(GX_n)(i) XOR C(Y)(i) -> C(GX_n)(i);
 C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of ERX0

INDICATORS:

NS Mode

Zero If $C(X_n) = 0$, then ON; otherwise, OFF

Negative If $C(X_n)(0) = 1$, then ON; otherwise, OFF

ES/EI Mode

Zero If $C(GX_n) = 0$, then ON; otherwise, OFF

Negative If $C(GX_n)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FAD

10.2 Machine Instruction Descriptions (F)

Following is a detailed description of the processor instructions and operation codes beginning with the letter F.

10.2.1 FAD

FAD	Floating Add	475(0)
-----	--------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) + C(Y)]$ normalized $\rightarrow C(EAQ)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. See the FNO instruction for a definition of normalization.
3. An Illegal Procedure fault occurs if illegal address modification is used.

FCMG

10.2.2 FCMG

FCMG	Floating Compare Magnitude	425(0)
------	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$|C(E, AQ(0-27))| :: |C(Y)|$; magnitude comparison;
 $C(EAQ)$, $C(Y)$ unchanged

EXPLANATION:

This comparison is executed in the following procedures.

1. Compare $C(E) :: C(Y)(0-7)$, select the number with the lower exponent, and shift its mantissa right by the number of places (binary or hex) determined by the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.
2. Compare the absolute values of the mantissas and set the indicators accordingly.

The FCMG instruction is identical to the FCMP instruction except that the magnitudes of the mantissas are compared instead of the algebraic values.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

<u>Zero</u>	<u>Negative</u>	<u>Relationship</u>
0	0	$ C(E,AQ(0-27)) > C(Y) $
1	0	$ C(E,AQ(0-27)) = C(Y) $
0	1	$ C(E,AQ(0-27)) < C(Y) $

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

FCMP

10.2.3 FCMP

FCMP	Floating Compare	515(0)
------	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(E, AQ_{0-27}) :: C(Y)$; algebraic comparison

EXPLANATION:

This comparison is executed in the following procedures.

1. Compare $C(E) :: C(Y)(0-7)$, select the number with the lower exponent, and shift its mantissa right by the number of places (binary or hex) determined by the difference of the exponents. If the number of shifts equals or exceeds 72, the number with the lower exponent is defined as zero.
2. Compare the mantissas and set the indicators accordingly.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

FCMP**INDICATORS:**

<u>Zero</u>	<u>Negative</u>	<u>Relationship</u>
0	0	$ C(E,AQ_{0-27}) > C(Y) $
1	0	$ C(E,AQ_{0-27}) = C(Y) $
0	1	$ C(E,AQ_{0-27}) < C(Y) $

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

FDI

10.2.4 FDI

FDI	Floating Divide Inverted	525(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) / C(EAQ) \rightarrow C(EA); 00\dots0 \rightarrow C(Q);$
 $C(Y)$ unchanged

EXPLANATION:

The dividend mantissa is shifted right and the dividend exponent is increased accordingly until:

$$|\text{Dividend mantissa}| < |C(AQ_{0-27})|$$

When such a shift occurs, only zeros from the dividend will be lost.

$C(AQ)(0-27)$ is used as the divisor mantissa.

28 bits of quotient mantissa are placed in A.

If $AQ(28-71)$ is not equal to 0 and $A(0) = 0$, then 1 is added to $AQ(27)$. $0 \rightarrow AQ(28-71)$ unconditionally. $AQ(0-27)$ is then used as the divisor mantissa. The 8-bit dividend exponent and 72-bit mantissa are placed in working registers.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

	<u>If division occurs</u>	<u>If no division occurs</u>
Zero	If $C(A) = 0$, then ON; otherwise, OFF	If divisor mantissa = 0, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF	If dividend < 0 , then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON	
Exponent Underflow	If exponent is < -128 , then ON	

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. If the divisor mantissa $C(AQ)$ is zero, the division does not take place. Instead, a Divide Check fault occurs and all registers remain unchanged. Dividend and divisor are not normalized by the hardware before division.
3. An Illegal Procedure fault occurs if illegal address modification is used.

FDV

10.2.5 FDV

FDV	Floating Divide	565(0)
-----	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ) / C(Y) \rightarrow C(EA);$
 $00...0 \rightarrow C(Q); C(Y)$ unchanged

EXPLANATION:

This division is executed in the following procedures.

The dividend mantissa $C(AQ)$ is shifted right and the dividend exponent $C(E)$ is increased accordingly until:

$$|C(AQ)(0-27)| < |C(Y)(8-35) \text{ with zero fill}|$$

When such a shift occurs, significant bits from the dividend may be lost.

Dividend and divisor are not normalized by the hardware prior to division .

36 bits of quotient mantissa are placed in A.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

	<u>If division occurs</u>	<u>If no division occurs</u>
Zero	If $C(A) = 0$, then ON; otherwise, OFF	If divisor mantissa = 0, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF	If dividend < 0 , then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON	
Exponent Underflow	If exponent is < -128 , then ON	

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. If the divisor mantissa (bits 8-35 of $C(Y)$) is zero, the division does not take place. Instead, a Divide Check fault occurs. The divisor $C(Y)$ remains unchanged, $C(AQ)$ contains the dividend's magnitude as an absolute value, and the negative indicator reflects the dividend's sign.
3. An Illegal Procedure fault occurs if illegal address modification is used.

FLD

10.2.6 FLD

FLD	Floating Load	431(0)
-----	---------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(Y)(0-7) -> C(E)
 C(Y)(8-35) -> C(AQ)(0-27)

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If C(AQ) = 0, then ON; otherwise, OFF
 Negative If C(AQ)(0) = 1, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

FLP**10.2.7 FLP**

FLP	Floating Load Positive	530(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$ C(Y) $, normalized	-> Z
Z(0-7)	-> C(E)
Z(8-35)	-> C(AQ)(0-27)
00...0	-> C(AQ)(28-71)

EXPLANATION:

The memory operand C(Y) is processed as single-precision floating-point data. The absolute value of this data is normalized and its exponent, mantissa (bits 8-35) and 0 are loaded into C(E), C(AQ)(0-27) and C(AQ)(28-71), respectively.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

FLP

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Overflow	If exponent < -127 , then ON

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

FMP**10.2.8 FMP**

FMP	Floating Multiply	461(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) * C(Y)]$ normalized $\rightarrow C(EAQ)$; $C(Y)$ unchanged

EXPLANATION:

This multiplication is executed in the following way:

$C(E) + C(Y)(0-7) \rightarrow C(E)$
 $C(AQ) * C(Y)(8-35)$ results in a 98-bit product plus sign,
the leading 71 bits plus sign of which
 $\rightarrow C(AQ)$.
 $C(EAQ)$ normalized $\rightarrow C(EAQ)$.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

FMP

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

FNEG**10.2.9 FNEG**

FNEG	Floating Negate	513(0)
------	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(\text{EAQ})$ normalized $\rightarrow C(\text{EAQ})$

EXPLANATION:

This instruction changes the number in $C(\text{EAQ})$ to its normalized negative (if $C(\text{AQ}) < 0$). The operation is executed by first forming the two's complement of $C(\text{AQ})$, and then normalizing $C(\text{EAQ})$.

Even if $C(\text{EAQ})$ is already normalized, an exponent overflow can still occur, namely when $C(\text{E}) = +127$ and $C(\text{AQ}) = -100\dots 0$ (the two's complement representation for the decimal value -1.0).

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL

FNEG

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if an illegal repeat is used.

10.2.10 FNO

FNO	Floating Normalize	573(0)
-----	--------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ)$ normalized $\rightarrow C(EAQ)$

EXPLANATION:

The instruction normalizes the number in $C(EAQ)$. If the Overflow indicator is ON, then the number in EAQ is normalized one place to the right; the sign bit 0 of $C(AQ)$ is then inverted to reconstitute the actual sign. The Overflow indicator is set OFF.

A normalized floating binary number is defined as one whose mantissa lies in the interval $(0.5, 1.0)$ such that

$$0.5 \leq |C(AQ)| < 1.0$$

which, in turn, requires that $C(AQ)(0) \neq C(AQ)(1)$

A normalized floating hexadecimal number is defined as one whose mantissa lies in the interval $(0.0625, 1.0)$ such that

$$0.0625 \leq |C(AQ)| < 1.0$$

which, in turn, requires that

if $C(AQ)(0) = 0$, then $C(AQ)(1-4) \neq 0000$, and
 if $C(AQ)(0) = 1$, then $C(AQ)(1-4) \neq 1111$

FNO

Normalization is performed by shifting C(AQ)(1-71) to the left (one place if binary, four places if hex) and reducing C(E) by 1, repeatedly, until the conditions for C(AQ)(0) and C(AQ)(1) or C(AQ)(1-4) are met. Bits shifted out of AQ(1) are lost.

If C(AQ)=0, then C(E) is set to -128 and the zero indicator is set ON.

This instruction can be used to correct overflows that occur with fixed-point numbers:

1	8	16	32

	TOV	1, IC	
	LDAQ	M	
	ADAQ	N	
	LDE	=71B25, DU	
	FNO		

will normalize C(M-pair) + C(N-pair) correctly, whether the addition caused an overflow (assuming overflow masked or successful recovery from Overflow fault).

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise, OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise, OFF
Exponent Overflow	If exponent is > +127, then ON
Exponent Underflow	If exponent is < -128, then ON
Overflow	Set OFF

NOTE:

When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.

FRAN**10.2.11 FRAN**

FRAN	Floating Point Random Number	362(1)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

```

00...0                -> C(Y)(00-16)
C(Y)(71) XOR C(Y)(72),
C(Y)(72) XOR C(Y)(73),
C(Y)(73) XOR C(Y)(74),
...
C(Y)(142) XOR C(Y)(143), -> C(Y)(17-89)
C(Y)(17-70)            -> C(Y)(90-143)

00...0                -> C(E)
0                      -> Z(00)
C(Y)(18-52)            -> Z(01-35)
00...0                -> Z(36-71)

[C(E), Z] normalized   -> C(EAQ)

```

EXPLANATION:

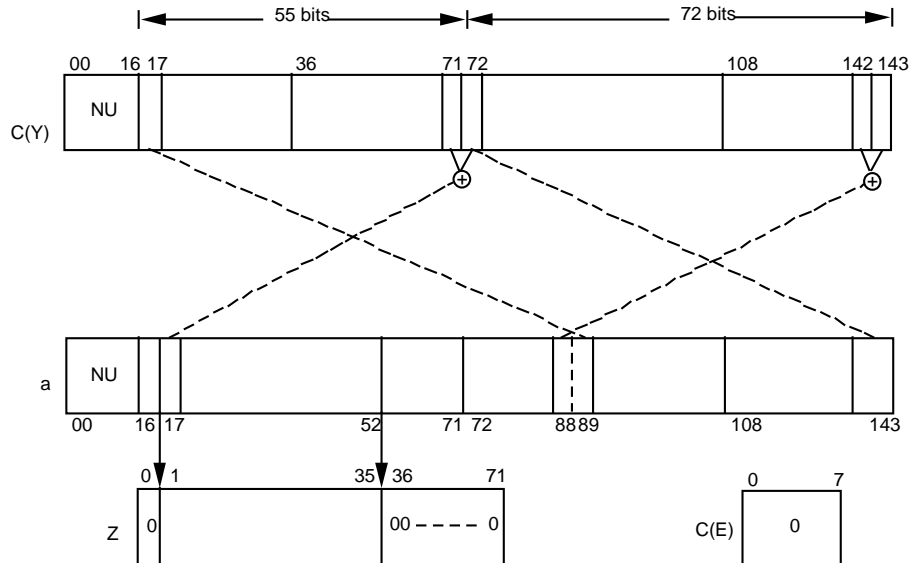
The operation shown in the figure below is performed on the low-order 127 bits of C(Y-4 words). The 127 bits of the result (a) are stored into bits 17 to 143 of the four words starting at memory location Y. Zero is stored into the high-order bits 0 - 16 of memory location Y.

Bits 18-88 of the result (a) are regarded as floating-point data of:

$$y = 2^0 \text{ (or } 16^0) \times 0.a18a19\dots a88$$

FRAN

The data y is normalized and loaded into EAQ. Whether data y is a power of 2 or 16 depends on the Indicator register. The memory location Y must be on a double-word boundary.



[C(E), Z] normalized \rightarrow C(EAQ)

This instruction is not an RAR (Read-Alter-Rewrite) type of instruction. It does not cause a read-lock/write-lock command sequence.

A random number is generated by this instruction using a so-called maximum-length shift register sequence (M-sequence). The primitive polynomial is given by $f(S) = x^{127} = x^{126} + 1$, and the period by $2^{127} - 1 \sim 10^{38}$. Execution of this instruction produces random numbers between 0 and 1.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SD, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

FRAN

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise OFF
Negative	OFF (a positive number is always obtained)

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FRD

10.2.12 FRD

FRD	Floating Round	471(0)
-----	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(EAQ) rounded to 28 mantissa bits and normalized -> C(AQ)

EXPLANATION:

This instruction performs a true round of C(EAQ) to a precision of 28 bits in C(AQ). The result is then normalized and restored to the EAQ registers. A true round means that the same rounding operation applied to a number of the same magnitude and with an opposite sign would result in a sum of the two rounded numbers of exactly zero.

The rounding operation is performed in the following way.

- a) A constant (all 1s) is added to bits 29-71 of the mantissa.
- b) If the number being rounded is positive, a carry is inserted into the least significant bit position of the adder.
- c) If the number being rounded is negative, the carry is not inserted.
- d) Bits 28-71 of C(AQ) are replaced by zeros.

If the mantissa overflows upon rounding, it is shifted right one place and a corresponding correction is made to the exponent.

If the mantissa does not overflow and is nonzero upon rounding, normalization is performed.

If the resultant mantissa is all zeros, the exponent is forced to -128 and the zero indicator is set.

If the exponent resulting from the operation is greater than +127, the exponent Overflow indicator is set.

If the exponent resulting from the operation is less than -128, the exponent Underflow indicator is set.

The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(AQ) = \text{zero}$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if an illegal repeat is used.

FSB

10.2.13 FSB

FSB	Floating Subtract	575(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) - C(Y)]$ normalized $\rightarrow C(EAQ)$; $C(Y)$ unchanged

EXPLANATION:

The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The shifted portion is truncated and the addition is executed. The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

FSBI

10.2.14 FSBI

FSBI	Floating Subtract Inverted	465(0)
------	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(Y) - C(EAQ)]$ normalized $\rightarrow C(EAQ)$; $C(Y)$ unchanged

EXPLANATION:

The two's complement of the subtrahend is first taken and the smaller value is then right-shifted to equalize it. The shifted portion is truncated and the addition is executed. After addition, the sum is normalized and the 72 bits of the mantissa are loaded into AQ.

The order of execution of the operation conforms to that of the FSB instruction. Normalization is defined under FNO.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

FST

10.2.15 FST

FST	Floating Store	455(0)
-----	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(E) -> C(Y)(0-7)
 C(A)(0-27) -> C(Y)(8-35)
 C(E), C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FSTR**10.2.16 FSTR**

FSTR	Floating Store Rounded	470(0)
------	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ)$ rounded and normalized $\rightarrow C(Y)$; $C(EAQ)$ unchanged

EXPLANATION:

This instruction performs a true round of $C(EAQ)$ to a precision of 28 bits in $C(AQ)$. The result is then normalized and stored in Y . A true round means that the same rounding operation applied to a number of the same magnitude and opposite sign would result in a sum of the two rounded numbers of exactly zero.

Upon completion of the rounding and normalization, the exponent and truncated mantissa are stored in the following way.

- a) Exponent in bits 0-7 of $C(Y)$
Bits 0-27 of mantissa in bits 8-35 of $C(Y)$
- b) If the resultant mantissa bits 0-27 are all zero, the exponent is forced to -128 and the zero indicator is set (floating-point zero).

The rounding and normalization operation of this instruction is identical with FRD.

The definition of normalization is located under the description of the FNO instruction.

FSTR

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If C(Y) = floating-point zero, then ON; otherwise, OFF
Negative	If C(Y)(8)= 1, then ON; otherwise, OFF
Exponent Overflow	If exponent is > +127, then ON
Exponent Underflow	If exponent is < -128, then ON

NOTES:

1. When indicator bit 32 = 1, the floating-point alignment and normalization are hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

FSZN**10.2.17 FSZN**

FSZN	Floating Set Zero and Negative Indicators from Storage	430(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Test C(Y); C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

<u>Zero</u>	<u>Negative</u>	<u>Relationship</u>
0	0	Mantissa C(Y)(8-35) > 0
1	0	Mantissa C(Y)(8-35) = 0
0	1	Mantissa C(Y)(8-35) < 0 (bit 8 of C(Y) = 1)

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

FTR

10.2.18 FTR

FTR	Floating Truncate Fraction	474(0)
-----	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(EAQ) fraction-truncated and normalized -> C(EAQ)

EXPLANATION:

This instruction truncates the fraction part of the floating-point data of C(EAQ) to obtain an integer. The result is normalized and stored into C(EAQ). A proper truncation to an integer is such that truncating the fractional parts of two numbers with the same absolute and different sign and adding the results produces 0.

ILLEGAL ADDRESS MODIFICATIONS:

IT, RI, IR

The address modification does not affect instruction operations, but the modification is executed.

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if an illegal repeat is used.

GLDD

10.3 Machine Instruction Descriptions (G)

Following is a detailed description of the processor instructions and operation codes beginning with the letter G.

10.3.1 GLDD

GLDD	Load Double to GX \underline{n}	32 $\underline{n}(0)$
------	-----------------------------------	-----------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

GLDD		n, Y, R, AM	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

$C(Y\text{-pair}) \rightarrow C(GXn\text{-pair})$

EXPLANATION:

$C(Y\text{-pair})$ is loaded into the $GXn\text{-pair}$ specified by bits 24-26 of the op code. The contents of bits 24-26(n) of the op code determines the load destination of the $GXn\text{-pair}$ in the following way.

<u>n (octal)</u>	<u>$GXn\text{-pair}$</u>
0	GX0, GX1
2	GX2, GX3
4	GX4, GX5
6	GX6, GX7

GLDD

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

This instruction can not be executed under the control of any Repeats (RPT, RPL, RPD).

ILLEGAL EXECUTES:

If the instruction is executed in NS mode.

INDICATORS:

Zero If $C(GXn\text{-pair}) = 0$, then ON; otherwise, OFF

Negative If $C(GXn\text{-pair})(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal address modifications or repeats are used or if this instruction is executed in the NS mode.
2. An IPR fault occurs if $n = 1, 3, 5, \text{ or } 7$.

GLLR

10.3.2 GLLR

GLLR	GX _n Long Left Rotate	446(1)
------	----------------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GLLR		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

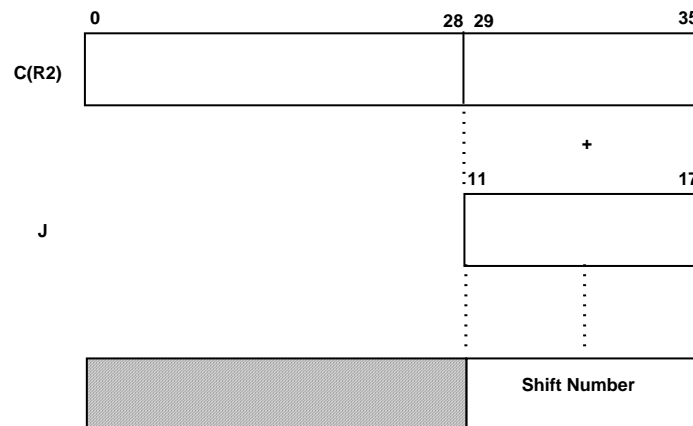
SUMMARY:

R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted left. Each bit shifted out from bit 0 is shifted into bit 71 of the R1-pair.

GLLR**EXPLANATION:**

The number of bits to be shifted can be illustrated in the following way:



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

GLLR

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLLS**10.3.3 GLLS**

GLLS	GX _n Long Left Shift	466(1)
------	---------------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GLLS		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

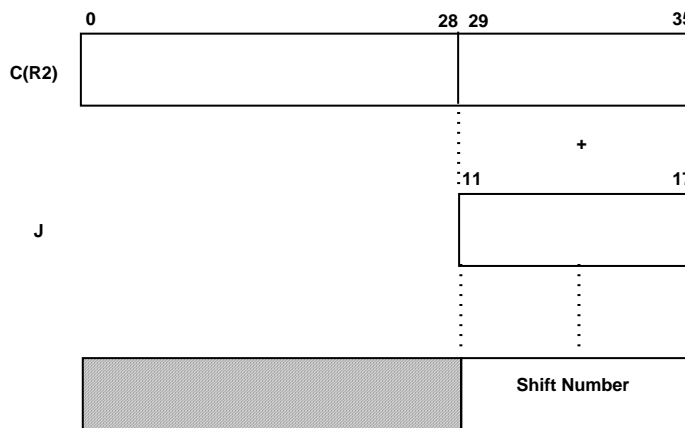
R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted left. Vacated positions in C(R1-pair) are filled with zeros.

GLLS

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

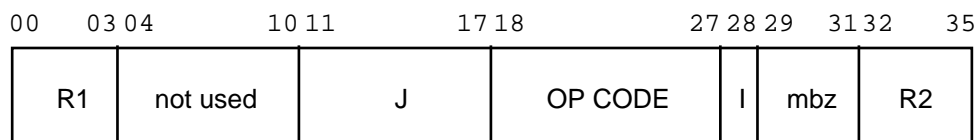
1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLR

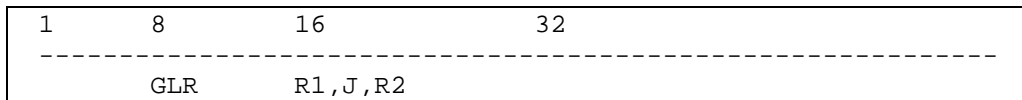
10.3.4 GLR

GLR	GX _n Left Rotate	442(1)
-----	-----------------------------	--------

FORMAT:



CODING FORMAT:



OPERATING MODES:

Executes only in ES/EI mode

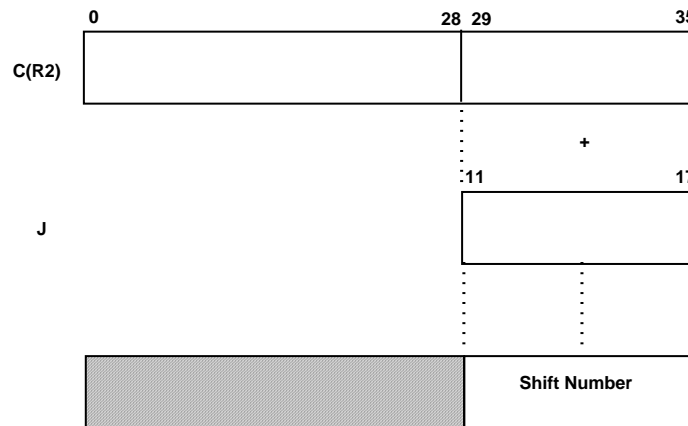
SUMMARY:

R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted left. Each bit shifted out from bit 0 is shifted into bit 35 of R1.

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

GLR

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLRL**10.3.5 GLRL**

GLRL	G <u>X</u> _n Long Right Logic	465(1)
------	--	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GLRL		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

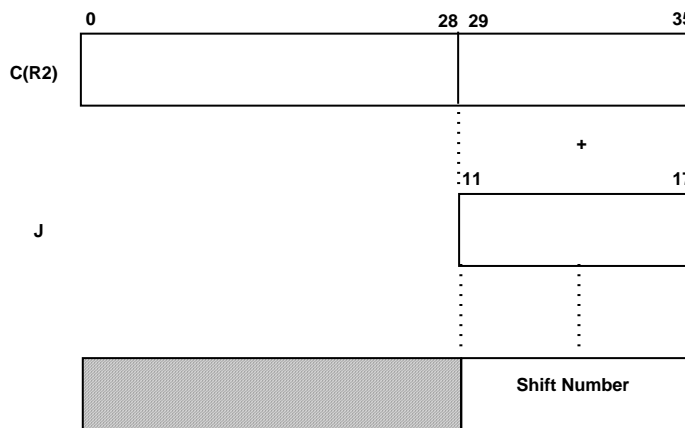
R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted right. Vacated positions in C(R1-pair) are filled with zeros.

GLRL

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1) = 0, then ON; otherwise, OFF

Negative If C(R1)(0) = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLRS

10.3.6 GLRS

GLRS	GX _n Long Right Shift	464(1)
------	----------------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GLRS		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

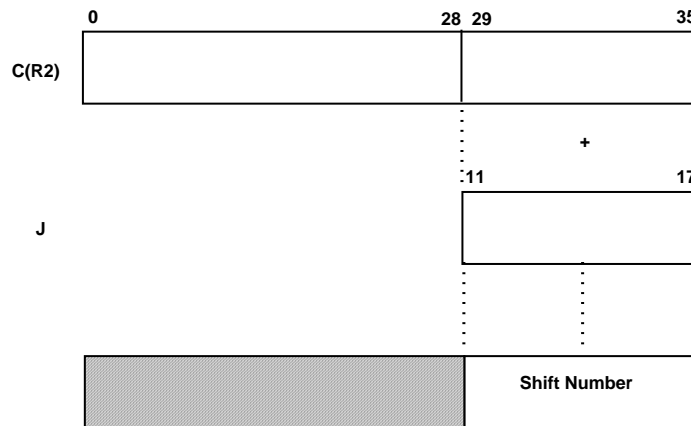
SUMMARY:

R1 = 0, 2, 4, 6, AQ

C(R1-pair) is shifted right. Vacated positions in C(R1-pair) are filled with bits equal to bit 0 of C(R1-pair).

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1) = 0, then ON; otherwise, OFF

Negative If C(R1)(0) = 1, then ON; otherwise, OFF

GLRS

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GLS**10.3.7 GLS**

GLS	GX _n Left Shift	462(1)
-----	----------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GLS		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

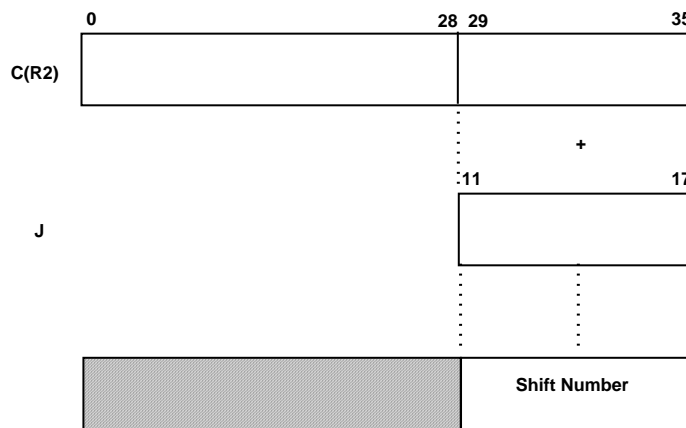
R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted left. Vacated positions in C(R1) are filled with zeros.

GLS

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GRL

10.3.8 GRL

GRL	GX _n Right Logic	461(1)
-----	-----------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GRL	R1, J, R2		

OPERATING MODES:

Executes only in ES/EI mode

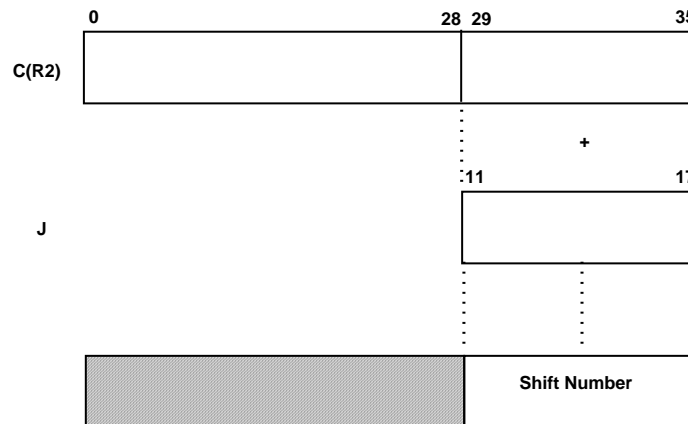
SUMMARY:

R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted right. Vacated positions in C(R1) are filled with zeros.

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1) = 0, then ON; otherwise, OFF

Negative If C(R1)(0) = 1, then ON; otherwise, OFF

GRL

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

GRS**10.3.9 GRS**

GRS	GX _n Right Shift	460(1)
-----	-----------------------------	--------

FORMAT:

00	03 04	10 11	17 18	27 28 29	31 32	35
R1	not used	J	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

GRS		R1, J, R2	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

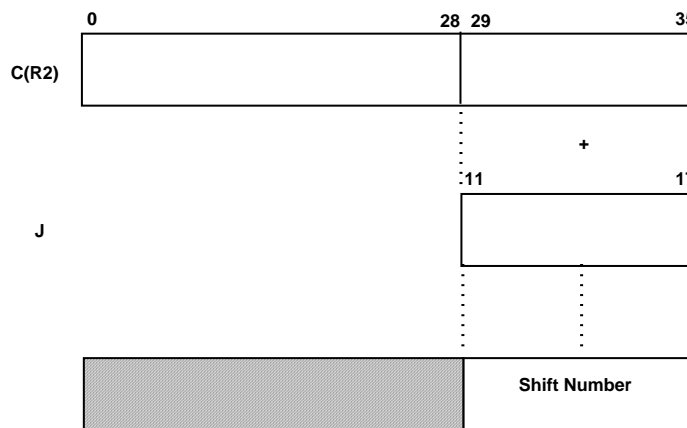
R1 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

C(R1) is shifted right. Vacated positions in C(R1) are filled with bits equal to bit 0 of C(R1).

GRS

EXPLANATION:

The number of bits to be shifted can be illustrated in the following way.



J is added to C(R2)(29-35) and the low-order 7 bits of the sum specify the shift number.

If the R2 field is 0000, the addition of C(R2) and J is not performed and the value of J specifies the shift number.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1) = 0, then ON; otherwise, OFF

Negative If C(R1)(0) = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

GSTD

10.3.10 GSTD

GSTD	Store Double from GX \underline{n}	14 \underline{n} (0)
------	--------------------------------------	------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

GSTD		n, Y, R, AM	

OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

C(GXn-pair) → C(Y-pair)

EXPLANATION:

The content of the GXn-pair specified by bits 24-26 of the op code is stored in the memory location of Y-pair. The GXn-pair whose contents are to be stored is specified in the following way.

<u>n (octal)</u>	<u>GXn-pair</u>
0	GX0, GX1
2	GX2, GX3
4	GX4, GX5
6	GX6, GX7

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

GSTD

ILLEGAL REPEATS:

This instruction can not be executed under the control of RPL
GSTD from GX0 can not be executed under the RPT, RPD

ILLEGAL EXECUTES:

If the instruction is executed in NS mode

INDICATORS:

None affected

NOTE:

An IPR fault occurs if illegal address modifications or repeats are used or if this instruction is executed in the NS mode.

GTB

10.3.11 GTB

GTB	Gray-to-Binary Convert	774(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(A) is converted from Gray code to a 36-bit binary number.

EXPLANATION:

This conversion is defined by the following algorithm in which R and S denote the contents of bit position i of the A-register before and after the conversion.

$$S_0 = R_0$$

$$S_i = (R_0 \text{ AND } S_{i-1}) \text{ OR } (R_i \text{ AND } S_{i-1})$$

where:

$$i = 1, \dots, 35$$

Gray code is a method of transmitting numeric code cyclically, one bit at a time, to eliminate transmission errors. It is defined in the following way.

- a) A positional binary notation for numbers in which any two sequential numbers whose difference is 1 are represented by expressions that are the same except in one place or column, and in that place or column differ by only one unit.
- b) A type of cyclic unit-distance binary code evolved from the 4-word, 2-bit unit distance code (00, 01, 11, 10) according to the following rule:

To construct an (n+1)-bit reflected binary code from an n-bit reflected binary code, write the n-bit code twice in sequence, first in forward and then in reverse sequence of code words. Prefix an extra bit to each word, assigning the value 0 to the forward version and the value 1 to the backward version of the n-bit code.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if an illegal repeat is used.

Notes

11. Machine Instruction Descriptions (L-M)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 11.1, Machine Instruction Descriptions (L)
- Section 11.2, Machine Instruction Descriptions (M)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

11.1 Machine Instruction Descriptions (L)

Following is a detailed description of the processor instructions and operation codes beginning with the letter L.

11.1.1 LAREG

LAREG	Load Address Register	463(1)
-------	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

LAREG		LOCSYM, R, AR	

LAREG

OPERATING MODES:

Any

SUMMARY:

NS Mode

$C(Y, Y+1, \dots, Y+7) (0-23) \rightarrow C(AR0, AR1, \dots, AR7)$

ES/EI Mode

$C(Y, Y+1, \dots, Y+7) \rightarrow C(AR0, AR1, \dots, AR7)$

EXPLANATION:

The formats of ARn in the three segment modes are:

Field:	Position in ARn		Description:
	NS mode:	ES/EI mode:	
WORD	00-17		Unsigned word displacement
		00-29	Twos-complement word displacement
BYTE	18-19	30-31	Byte address within word
BIT	20-23	32-35	Bit address within byte

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LAREG**NOTE:**

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16	32

	LAREG	REGW	load AR0..AR7 from REGW..REGW+7
	.		
	.		
	EIGHT		
REGW	DEC	0,0,0,0,0,0,0,0	
* Result is that all address Registers are cleared.			

LAR_n

11.1.2 LAR_n

LAR _n	Load Address Register <u>n</u>	76 _n (1)
------------------	--------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	32

LAR _n		LOCSYM, RM, AR	

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n=0,1,...,7 as determined by op code:
 C(Y)(0-23) → C(AR_n); C(Y) unchanged

ES/EI Mode

For n=0,1,...,7 as determined by op code:
 C(Y) → C(AR_n); C(Y) unchanged

EXPLANATION:

The hardware assumes the lower 3 bits of address Y are = 000 and the 8 words beginning from the 8-word boundary are accessed. No check is performed to determine whether the lower 3 bits of Y = 000. Location Y must be forced to a multiple of 8 by entering an 8 in column 7 of the statement that defines Y, or by using the EIGHT pseudo-operation.

The formats of AR_n in the three segment modes are:

Field:	Position in AR _n		Description:
	NS mode:	ES/EI mode:	
WORD	00-17		Unsigned word displacement
		00-29	Twos-complement word displacement
BYTE	18-19	30-31	Byte address within word
BIT	20-23	32-35	Bit address within byte

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LAR_n

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16	32	

	LAR7	ADDR		load address bits 0-23 into AR7
	.			
	.			
ADDR	BDSC	512,,8,8	001000700000	memory contents
	*			
	*CONTENTS OF AR7 AFTER: 0 0 1 0 0 0 7 0			

11.1.3 LCA

LCA	Load Complement into A-Register	335(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(Y) \rightarrow C(A); C(Y)$ unchanged

EXPLANATION:

This instruction changes the number to its negative (if $\neq 0$) while moving it from Y to A. The operation is executed by forming the two's complement of the string of 36 bits. An overflow condition exists if $C(Y) = 2^{**}35$.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON

LCAQ

11.1.4 LCAQ

LCAQ	Load Complement into AQ-Register	337(0)
------	----------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(Y\text{-pair}) \rightarrow (AQ); C(Y\text{-pair})$ unchanged

EXPLANATION:

This instruction changes the number to its negative (if $\neq 0$) while moving it from Y-pair to AQ. The operation is executed by forming the two's complement of the string of 72 bits. An overflow condition exists if $C(Y\text{-pair}) = -2^{*}71$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON

NOTE:

An Illegal Procedure fault occurs if illegal address modifications are used.

LCCL

11.1.5 LCCL

LCCL	Load Calendar Clock	057(0)
------	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

C(AQ)(20-71) -> C(Calendar Clock)

EXPLANATION:

V9000 emulated LCCL by saving the difference between the microseconds specified in the LCCL operand and the microseconds from the current LINUX "gettimeofday()" function. The difference calculated by the LCCL instruction is stored in memory that is accessible by all CPU threads. The difference is stored using the CompareExchange instruction, so that the LCCL instruction is serialized in case it is executed in several processors at the same time.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

LCCL

NOTES:

1. Attempted execution of LCCL in the Slave or Master mode results in a Command fault.
2. An Illegal Procedure fault occurs if an illegal repeat is used.
3. See the RCCL instruction for calendar clock.

LCQ

11.1.6 LCQ

LCQ	Load Complement into Q-Register	336(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(Y) \rightarrow C(Q); C(Y)$ unchanged

EXPLANATION:

This instruction changes the number to its negative value (if $\neq 0$) while moving it from Y to Q. The operation is executed by forming the two's complement of the string of 36 bits. An overflow condition exists if $C(Y) = -2^{**35}$.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Q is exceeded, then ON

EXAMPLE:

1	8	16	32

	LCQ	=5,DL	Loads -5 into the Q-register

LCTR

11.1.7 LCTR

LCTR	Load Weighted Instruction Counter	417(0)
------	-----------------------------------	--------

NOTE: The LCTR instruction functions as a NOP in the V9000.

LCX_n**11.1.8 LCX_n**

LCX _n	Load Complement into Index Register <u>n</u>	32 _n (0)
------------------	---	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For n=0,1,...,or 7 as determined by opcode:
-C(Y)(0-17) -> (X_n); C(Y) unchanged

ES/EI Mode

For n=0,1,...,or 7 as determined by opcode:
-C(Y) -> (GX_n); C(Y) unchanged

EXPLANATION:

This instruction changes the number to its negative value (if $\lt 0$) while moving it from bits 0-17 of Y to X_n or from Y to GX_n. The operation is executed by forming the two's complement of the string of 18 bits.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL of LCX0

LCX_n

INDICATORS:

Zero	If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of X_n/GX_n is exceeded, then ON

NOTES:

1. In the NS mode, if DL modification is used, the hardware executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification is used.

LDA**11.1.9 LDA**

LDA	Load A-Register	235(0)
-----	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \rightarrow C(A)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

LDAB

11.1.10 LDAB

LDAB	Load A-Register with Byte	501(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

ES/EI only

SUMMARY:

$C(Y) \text{ BYTE} \rightarrow C(A)(28-35);$
 $C(Y) \text{ BYTE}(0) \rightarrow C(A)(00-27)$

EXPLANATION:

The byte at the byte position specified by C(Y) is stored right-justified in the C(A). Bit 0 of the byte is left-extended into the remainder of the C(A).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, AU, RI, IR, IT

R modification is byte count (not word count) except IC modification is word count.

ILLEGAL REPEATS:

All

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

LDAB

NOTES:

1. An Illegal Procedure fault occurs if the TAG field specifies RI, IR, IT, or R = DU, DL, AU. If R = IC, the contents of IC are treated as a word count.
2. A segment bound check is performed on the whole C(Y) word regardless of the specified byte position.

LDAC

11.1.11 LDAC

LDAC	Load A-Register and Clear	034(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \rightarrow C(A);$
 $00\dots0 \rightarrow C(Y)$

EXPLANATION:

This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

LDAC

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

LDAH

11.1.12 LDAH

LDAH	Load A-Register with Halfword	511(0)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \text{ HALFWORD} \rightarrow C(A)(18-35);$
 $C(\text{HALFWORD})(0) \rightarrow C(A)(00-17)$

EXPLANATION:

The halfword at the halfword position specified by $C(Y)$ is stored right-justified in the $C(A)$. Bit 0 of the halfword is left-extended into the remainder of the $C(A)$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, AU

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

LDAH

NOTES:

1. An Illegal Procedure fault occurs if the TAG field specifies RI, IR, IT, or R = DU, DL, AU. If R = IC, the contents of IC are treated as a word count.
2. An Illegal Procedure fault occurs
3. A segment bound check is performed on the whole C(Y) word regardless of the specified byte position.
4. The lowest order bit of the address is ignored after address modification is completed.

LDAQ

11.1.13 LDAQ

LDAQ	Load AQ-Register	237(0)
------	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y\text{-pair}) \rightarrow C(AQ)$; $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

LDAS**11.1.14 LDAS**

LDAS	Load Argument Stack Register	770(1)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y\text{-pair}) \rightarrow C(\text{ASR}); C(Y\text{-pair})$ unchanged
 $C(Y)(0-19) \rightarrow C(\text{HWMR})$

EXPLANATION:

A descriptor is fetched from even/odd memory locations Y and Y+1 and the following checks are performed on the descriptor:

- a) Type field T = 1
- b) Base and bound are modulo 2 words (the three least significant bits of base must be zeros; the three least significant bits of bound must be ones if flag bit 27 is 1).

If these conditions are met, the descriptor is loaded into the argument stack register (ASR) and the bound is also loaded into the HWMR. During ASR loading, bits 0-6 of the ASR bound field are forced to zero by the processor instead of being loaded from the memory operand. If flag bit 27 of the operand descriptor is zero, the entire bound field is forced to zero, regardless of any value the operand descriptor bound field may contain and the bound check is bypassed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

LDAS

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Any of the following conditions cause an IPR fault:
 - Modifications DU, DL, CI, SC, and SCR;
 - Illegal repeats RPT, RPD, and RPL;
 - Segment descriptor type field T is not 1;
 - If the base and bound limits of the operand descriptor are not modulo 2 words (only when flag bit 27 = 1).
2. If the processor is in Slave or Master mode, the execution of this instruction causes a Command fault.

EXAMPLE:

1	8	16	32

*	ROUTINE TO LOAD REGISTERS - ASR, PSR, DSAR		
*	CALLING TSX Z,RDSPRG		
*			
	POST	LOST P0,Z	
RDSPRG	EQU	*	
	LDP	P0,.SSR,DL	*safe store frame access
	LDP	P0,.CTYP,DL	*change type
	LDDSA	.WDSAR,,P0	*DSAR
	LDAS	.WASR,,P0	*ASR
	LDPS	.WPSR,,P0	*PSR
	TRA	,Z	*OK

LDCR

11.1.15 LDCR

LDCR	Load Complement Register from Register	431(1)
------	--	--------

FORMAT:

00	03	04		17	18		27	28	29	31	32	35
R1	not used			OP CODE			I	mbz		R2		

CODING FORMAT:

1	8	16	32									

LDCR			R1, R2									

OPERATING MODES:

Executes in ES/EI mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

-C(R2) → C(R1)
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

LDCR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Overflow	If the range of R1 is exceeded, ON

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDD_n**11.1.16 LDD_n**

LDD _n	Load Descriptor Register <u>n</u>	67 _n (1)
------------------	-----------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

This set of eight instructions provides the capability of loading a descriptor register (DR_n) with a new descriptor or modifying the descriptor currently contained in DR_n. The segment type referenced by the generated address determines the function to be executed.

In this discussion, DR_n represents the specified descriptor, whereas, DR_m represents the descriptor register indicated by the y field that is used to load a new segment descriptor.

When the instruction word bit 29 = 1 and the descriptor register specified by bits 0, 1, and 2 of the y field includes a type T = 1 or 3 segment descriptor, the segment descriptor is loaded into the DR_n from the segment descriptor segment specified by DR_m.

When the instruction word bit 29 = 1 and the type for the segment descriptor in DR_m is T = 0, 2, 4, 6, 12, or 14, or when the instruction word bit 29 = 0, a vector operation is performed.

Descriptions of the two types of operations follow. An IPR fault occurs when DR_m includes a type T = 7 - 11, 13, or 15 segment descriptor.

LDD_nInstruction Word Bit 29 = 1. DR_m Type T = 1 or 3

The segment descriptor from the segment descriptor segment indicated by DR_m is loaded into DR_n. When the effective address is generated, only R type modification and DU/DL modification are permitted. The effective address is the offset from the segment descriptor segment indicated by DR_m. The segment descriptor from the even/odd location indicated by this address is loaded into DR_n and the same checks are performed as for any normal memory reference.

- A check is made to determine whether a segment is present and whether read is permitted.
- A bound check is made.

In this case, the housekeeping bit for that page must be ON because the segment descriptor segment is referenced. If it is OFF, the instruction execution is terminated and a Security Fault, Class 1 occurs. The housekeeping page access for access of the segment descriptor is not dependent upon the CPU mode; it may also be executed in the Slave mode.

The AR_n and SEGID_n that correspond to the DR_n are affected in the following ways:

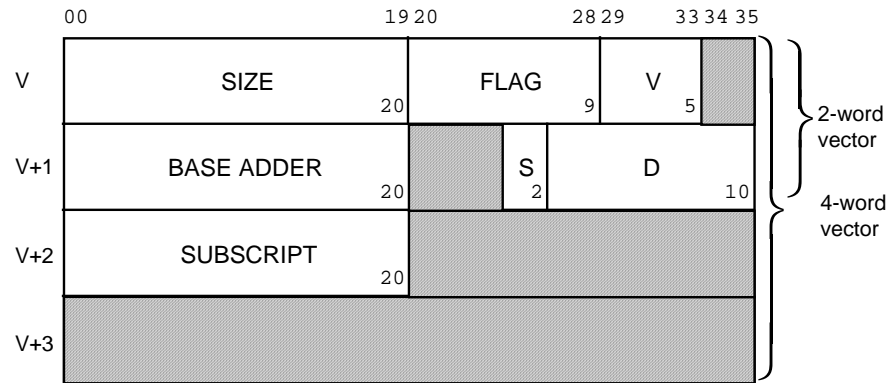
- AR_n is set to zero.
- SEGID_n is set to be self-identifying, i.e., S = 0, D = 177n.

Instruction Word Bit 29 = 0 DR_m Type T = 0, 2, 4, 6, 12, or 14

The memory operand vector, consisting of one or two double words determines the operation to be performed by the instruction. When this vector is obtained from memory, all address modification is permitted except for DU, DL, SC, SCR, and CI.

VECTOR FORMAT

a) Vector for Standard Segment Descriptor/Super Segment Descriptor



The contents of bits 29-33 (the V field) determine the function to be performed (XXX for bits indicates that these bits are ignored):

V = 00XXX Copy: 2-word vector

Copy (load) the selected segment descriptor into DR_n. SEGID_n is set to indicate the location from which the segment descriptor was obtained; AR_n is set to zero.

V = 01XXX Normal Shrink: 2-word vector

Shrink the selected segment descriptor and load it into DR_n. SEGID_n is set to indicate DR_n; AR_n is set to zero.

V = 10000 Extended Shrink: 4-word vector

V = 10001 Special Extended Shrink: 4-word vector

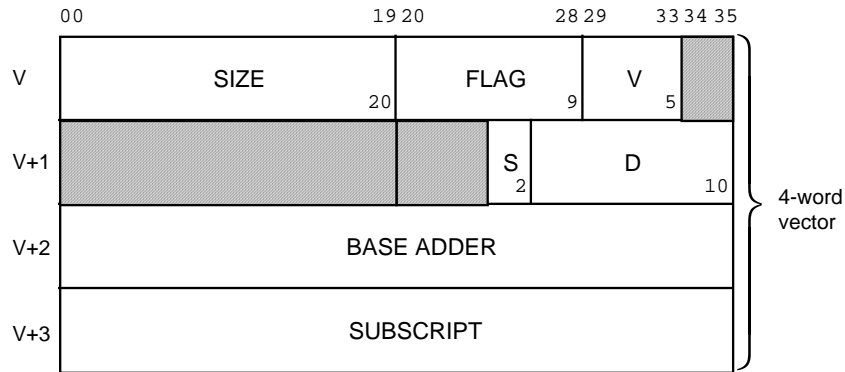
Shrink the selected segment descriptor with the use of the 4-word vector and load it into DR_n. SEGID_n is set to indicate DR_n; AR_n is set to zero. (Refer to details below for difference between Extended Shrink and Special Extended Shrink.)

V = 11XXX Data Stack Shrink: 2-word vector

Use DSDR and DSAR to generate the data stack segment descriptor. Load this segment descriptor into DR_n. DSAR is updated and AR_n is set to zero. SEGID is set to indicate DR_n.

LDD_n

b) Vector for Extended Segment Descriptor



The contents of bits 29 - 33 determine the function to be performed with the format illustrated above as follows:

V = 10100 Normal Shrink with Type Change

Shrink the selected segment descriptor (T = 12 or 14) and change to a Standard Segment Descriptor. SEGID_n is set to indicate DR_n; AR_n is set to zero.

V = 10101 Normal Shrink with No Type Change

Shrink the selected segment descriptor (T = 12 or 14). SEGID_n is set to indicate DR_n; AR_n is set to zero.

V = 10110 Extended Shrink with Type Change

Shrink the selected segment descriptor (T = 12 or 14), by using a subscript, and change to a Standard Segment Descriptor. SEGID_n is set to indicate DR_n; AR_n is set to zero.

V = 10111 Extended Shrink with No Type Change

Shrink the selected segment descriptor (T = 12 or 14) by using a subscript. SEGID_n is set to indicate DR_n; AR_n is set to zero.

V = 10010 Normal Base Shrink with No Type Change

Shrink the base of a selected segment descriptor (T = 0, 2, 12, 14) and reduce the bound by as much as the base shrinkage. The type remains unchanged, SEGID is set to indicate DR_n and AR_n is set to zero.

V = 10011 Extended Base Shrink with No Type Change

The same as the normal base shrink, except that the subscript is used. SEGID_n is set to indicate DR_n and AR_n is set to zero.

SHRINK FOR STANDARD AND SUPER SEGMENT DESCRIPTORS

- a) V = 00XXX Copy (bits indicated by X ignored)

The S and D fields of the vector indicate the location of the segment descriptor to be loaded into DR_n. These two fields are defined below.

For D = 0000 through 1757 (octal), the descriptor is loaded from the parameter segment and D is used as an index to the desired descriptor. The value in D is the number of the descriptor to be loaded and can be treated as a modulo 8 byte index, i.e., D can be converted to a byte address by appending three zeros as the three least-significant bits.

D is bound checked against the PSR (Parameter Segment Register) bound field. If D > PSR bound, a Bound fault occurs. IF D <= PSR bound, D is added to the PSR base and is used as the segment descriptor address. This address is used to obtain the segment descriptor that is then loaded into DR_n.

For D = 1760 through 1777 (octal), the descriptors referenced by S, D are contained in selected registers and copied to the DR_n.

D	=	1760	Undefined, IPR fault
D	=	1761	Change Descriptor Type Field in DR _n
D	=	1762	Instruction Segment Register (ISR)
D	=	1763	Data Stack Descriptor Register (DSDR)
D	=	1764	Safe Store Register (SSR)
D	=	1765	Linkage Segment Register (LSR)
D	=	1766	Argument Stack Register (ASR)
D	=	1767	Parameter Segment Register (PSR)
D	=	1770	DR0, Descriptor Register 0
D	=	1771	DR1, Descriptor Register 1
D	=	1772	DR2, Descriptor Register 2
D	=	1773	DR3, Descriptor Register 3
D	=	1774	DR4, Descriptor Register 4
D	=	1775	DR5, Descriptor Register 5
D	=	1776	DR6, Descriptor Register 6
D	=	1777	DR7, Descriptor Register 7

LDD_n

NOTE: When S = 0: with D = 1761 (octal) and the processor is in the Privileged Master mode, if the descriptor contained in DR_n is type 1 or 3, the type is changed to 0 or 2, respectively. SEGID_n is set to be self-identifying. However, if the descriptor is not type 1 or 3, no fault occurs, and no operation is performed.

When S = 0: with D = 1761, 1763, or 1764 (octal), a command fault occurs unless the CPU is in the Privileged Master Mode.

When S = 2: The D_{th} descriptor of the current argument segment is selected. A relative byte offset is formed by extending the D field by 3 zeros. D is bound checked against the ASR bound field. If D > the ASR bound, a bound fault occurs. If D ≤ the bound, D is added to the ASR base, and the segment descriptor is obtained with this address and then loaded into DR_n.

When S = 1 or 3: The D_n descriptor of the current linkage segment is selected. A relative byte offset is formed by extending the D field by three zeros. D is bound checked against the LSR bound field. If D > bound, a Bound fault occurs. If D ≤ the bound, D is added to the LSR base, and the segment descriptor is obtained with this address and then loaded into DR_n.

For all values of S, the loading of DR_n affects the nth address register (AR_n) and the nth segment identity register (SEGID_n) as described below.

- AR_n is set to zero.
- If DR_n was loaded from another DR or the instruction segment register (ISR), the associated segment identity content is transferred to SEGID_n; otherwise, SEGID_n is set to the S and D value contained in the vector. When S = 0 and D = 1761(octal), SEGID_n is set to be self-identifying.
- If an IPR or a Bound fault occurs, DR_n, AR_n, and SEGID_n are not changed.

b) V = 01XXX Normal Shrink

When bits 29 and 30 of the first word in the vector are 01, the specified segment descriptor is obtained, the shrink operation is performed, and the descriptor is then loaded into DR_n as with copy. When S = 0 and D = 1761 (octal) in the Privileged Master mode, the segment descriptors for type T = 1 or 3 are changed to T = 0 or 2, respectively. The shrink operation is then performed.

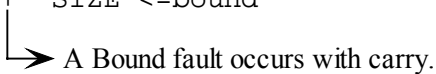
In order to perform the shrink operation, the segment descriptors indicated by S and D must be Standard, Super, or Extended Segment descriptors. An IPR fault occurs if T = 5, 7, 8, 9, 10, 11, 13, or 15. If a fault, such as a Bound fault, occurs during the shrink operation, DR_n, SEGID_n, and AR_n are not changed.

Standard Segment Descriptors

With standard segment descriptors, the shrink operation is performed in the following way.

- The vector BASE ADDER and SIZE fields are the relative values for the selected segment descriptor base and bound fields. The following check is performed for these values.

$$\text{BASE ADDER} + \text{SIZE} \leq \text{bound}$$




A Bound fault occurs when the sum of the BASE ADDER and SIZE exceeds the bound or when carry occurs with this addition. Flag bit 27 is not checked.

LDD_n

- When the check is terminated, a new base and bound are generated.

$$\text{New Base} = \text{old base} + \text{BASE ADDER}$$

$$\text{New bound} = \text{size}$$


 A bound fault occurs with carry.

The new base and bound are loaded into DR_n.

- The vector flag field indicates the attributes given to the segment. It is combined with the flag field of the selected segment descriptor to generate a new flag field. The permission conditions for these new flags are such that they are not increased from the previous conditions (i.e., a bit-by-bit logical AND operation of two flag fields takes place). A fault does not occur even if the vector permission conditions are greater than the segment descriptors permission conditions. The result produced by the combination of these two flag fields is loaded into the DR_n flag field. As the type T = 2 or 3 segment descriptor flag field are three bits in length, the AND operation is performed for these three bits and the corresponding three bits from the vector.

The corresponding AR_n is set to zero.

SEGID_n is set to be self-identifying (DR_n). For example, when this instruction references DR3 (LDD#), SEGID3 is set in the following way:

00	1111111011
2 bits	10 bits(1773 ₈)

Super Segment Descriptors

When shrink operation is performed for a super segment descriptor, a standard segment descriptor is generated. Type T = 4 super segment descriptor becomes type T = 0 standard segment descriptor, and type T = 6 super segment descriptor becomes type T = 2 standard segment descriptor.

The shrink operation is performed this way:

- A check is performed to determine whether the following expression is satisfied.

$$\text{Location} + (\text{BASE ADDER} + \text{SIZE}) \leq \text{bound}$$

\swarrow \searrow
 No fault with carry.
 A bound fault occurs with carry.

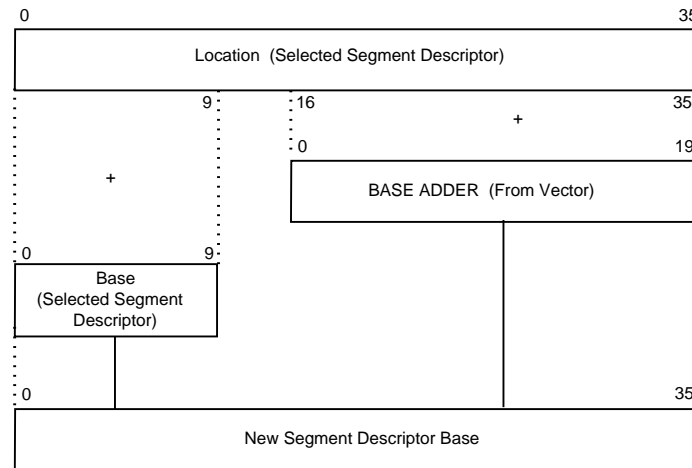
Flag bit 27 is not checked.

If this check is passed, a new base and bound are generated.

$$\text{New base} = \text{base} + (\text{location} + \text{BASE ADDER})$$

\swarrow \searrow
 A Bound fault occurs with carry.
 A Bound fault occurs with carry.

The processing that follows relative to the base and bound fields of the selected descriptor is described below.



The new bound = SIZE. The new base and size field from the vector are loaded in the base and bound field of DR_n.

The new flags field is formed in the same manner as for the standard descriptor. SEGID_n is set as for the standard descriptor shrink and AR_n is zero-filled.

LDD_n

Extended Segment Descriptors

The descriptor is converted to the standard segment descriptor by using "BASE ADDER" (20 bits) and "SIZE" (byte scale) in the two-word vector for the standard segment descriptor.

- The following information is checked.

$$\text{BASE ADDER} + \text{SIZE} \leq \text{bound.11111111111111111111} \\ \text{(12 bits)}$$

If this equation is not satisfied, a bound fault will occur. The left-hand-side of the equation is calculated by a 20 bits byte address. If this calculation causes a carry, a bound fault will occur.

- The new base and bound are generated in the following way.

$$\text{New base} = \text{old base} + \text{BASE ADDER}$$

If the right-hand-side of the equation causes a carry, a bound fault will occur.

$$\text{New bound} = \text{SIZE}$$

- Then, the new type number is set:

$$\text{New T} = 0 \text{ if old T} = 12$$

$$\text{New T} = 2 \text{ if old T} = 14$$

c) V=10000 Extended Shrink

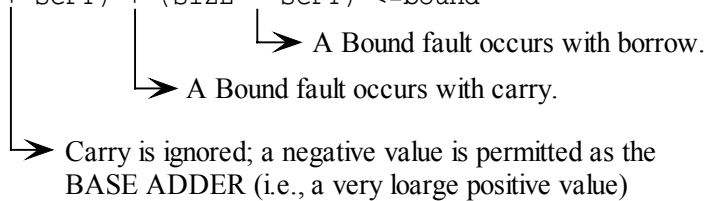
For extended shrink operations, the same conditions that exist for normal shrink operations must be satisfied. If a fault occurs during a shrink operation, DR_n, AR_n and SEGID_n remain unchanged.

Standard Segment Descriptors

A 4-word vector subscript (SCPT) is used when the new segment descriptor base and bound are generated.

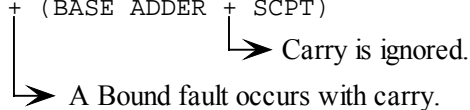
- The following check is performed.

$$(BASE\ ADDER + SCPT) + (SIZE - SCPT) \leq bound$$

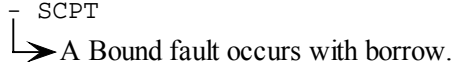


- If this check is passed, a new base and bound are generated.

$$New\ base = base + (BASE\ ADDER + SCPT)$$



$$New\ Bound = SIZE - SCPT$$



The new base and bound are loaded into DR_n.

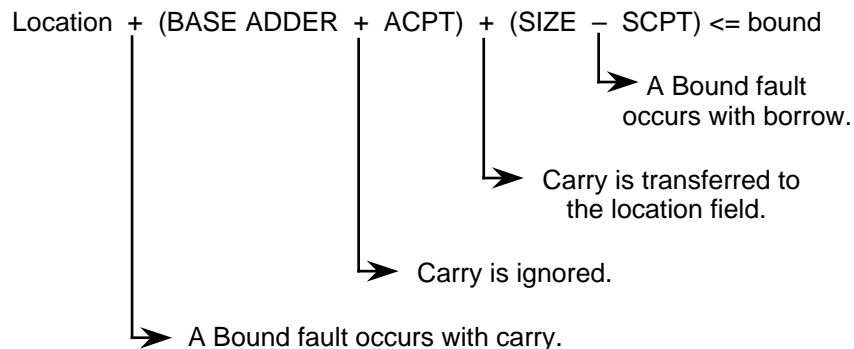
As described in the discussion on normal shrink of a standard segment descriptor, a new flag field is generated. SEGID_n and AR_n are set in the same way.

LDD_n

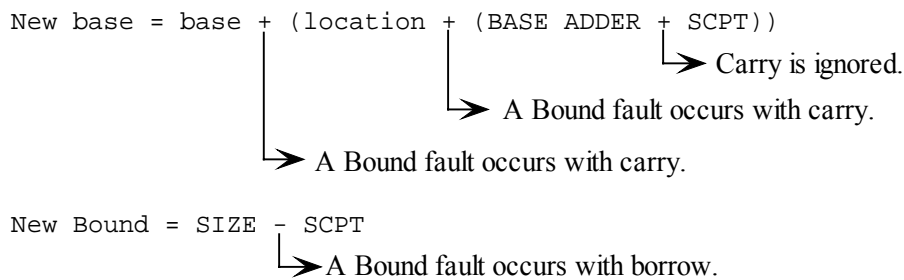
Super Segment Descriptors

The SCPT field is used as described in the discussion on standard segment descriptors.

The following check is performed.



If this check is passed, a new base and bound are generated.



The new base and bound are loaded into DR_n.

- A new flag field is generated as with a standard segment descriptor.
- DR_n type T is set by the following procedures.
 - 1) If old T = 4, then new T = 0.
 - 2) If old T = 6, then new T = 2.
- The corresponding AR_n is set to zero.
- SEGID_n is set to be self-identifying (DR_n). The flag bit 27 of the selected segment descriptor is not checked.

Extended Segment Descriptors

The descriptor is converted to the standard segment descriptor by using "BASE ADDER" (20 bits), "SUBSCRIPT," and "SIZE" (byte scale) in the 4-word vector for the standard segment descriptor.

The following information is checked.

$$(BASE\ ADDER + SCPT)^* + (SIZE - SCPT)** \leq bound.111111111111$$

(12 bits)

* The carry is ignored.

** If it borrows, a bound fault will occur.

If this equation is not satisfied, a bound fault will occur. The left-hand-side of the equation is calculated by a 20 bits byte address. If this calculation causes a carry, a bound fault will occur.

– The new base and bound are generated in the following way.

$$New\ base = old\ base + (BASE\ ADDER + SCPT)^*$$

* The carry is ignored.

If the right-hand-side of the equation causes a carry, a bound fault will occur.

$$New\ bound = SIZE - SCPT$$

If this calculation borrows, a bound fault will occur.

– Then, the new type number is set:

New T = 0 if old T = 12

New T = 2 if old T = 14

LDD_n

d) V = 10001 Special Extended Shrink

The differences between the special extended shrink and the extended shrink (V = 10000) are discussed below.

If the type T of the fetched segment descriptor is not equal to 0, 1, 2, 3, 12, or 14, an IPR fault occurs.

In the case of T = 0, 1, 2, or 3:

The SIZE field (bits 0 - 17) of the vector is ignored, and the following check is made.

$$\text{BASE ADDER} + \text{SCPT} \leq \text{old bound}$$

└─> A carry is ignored.

A new base and bound are created in the following way.

$$\text{New base} = \text{old base} + (\text{BASE ADDER} + \text{SCPT})$$

└─> A carry is ignored.

└─> A Bound fault occurs if a borrow is generated.

$$\text{New bound} = \text{old bound} - (\text{BASE ADDER} + \text{SCPT})$$

└─> A carry is ignored.

└─> A Bound fault occurs if a borrow is generated.

In the case of T = 12 or 14, the descriptor is converted to the standard segment descriptor by using "BASE ADDER" (20 bits), "SUBSCRIPT," and "SIZE" (byte scale) in the 4-word vector for standard segment descriptor.

- The following information is checked.

$$\text{BASE ADDER} + \text{SCPT} \leq \text{bound.111111111111} \\ \text{(12 bits)}$$

If this equation is not satisfied, a bound fault will occur. The left-hand-side of the equation is calculated by a 20 bits byte address. If this calculation causes a carry, a bound fault will occur.

- The new base and bound are generated in the following way.

$$\text{New base} = \text{old base} + (\text{BASE ADDER} + \text{SCPT})^*$$

* The carry is ignored.

If the right-hand-side of the equation causes a carry, a bound fault will occur.

$$\text{New bound} = \text{Min} [\{ \text{(0000.old-bound.111111111111-lower 12 bits} \\ \text{of old base)} - (\text{BASE ADDER} + \text{SCPT}) \}, 2^{**20} - 1] \\ \text{(12 bits)}$$

- Then, the new type number is set:

$$\text{New T} = 0 \text{ if old T} = 12$$

$$\text{New T} = 2 \text{ if old T} = 14$$

LDD_ne) V = 11XXX Data Stack Shrink

When bits 29 and 30 of the first word in the vector are 11, the instruction performs the data stack shrink operation. The second word in the vector is ignored. DSDR, DSAR, and the SIZE and flag field of the first word in the vector are used to generate the new segment descriptor.

- The value in the SIZE field of the vector is checked to determine whether the area between the location currently specified by the DSAR and the value specified by the DSDR bound is equal or greater than the SIZE field. The lower three bits of the vector SIZE field are set to 1 to indicate an even-word boundary (i.e., it is rounded to a double-word expression as the DSAR always specifies an even-word boundary.) $DSAR + SIZE$ (rounded-up) \leq DSDR bound is then checked. If the left portion of this expression exceeds the DSDR bound, or if carry occurs as a result of the addition to the left, a Bound fault is generated. In this case DR_n, AR_n, and SEGID_n are not changed.
- If this check is passed, the DSAR content is added to the DSDR base and a new base is generated. If carry occurs, a bound fault occurs and the register content is not changed.
- The new base (DSAR + DSDR base) is then loaded into the DR_n base field and the vector SIZE (before rounding) is loaded into the DR_n bound field.
- The new flag field values are generated from the vector flag field and the DSDR flag field following the same method as that described for normal shrink of standard segment descriptors.
- The content of the DSDR W and T fields are moved to the DR_n W and T fields.
- The corresponding AR_n is set to zero.
- SEGID_n is set to be self-identifying (DR_n), as with normal shrink.
- The following value is loaded into DSAR.

New DSAR = DSAR + SIZE (rounded-up) + 1 (byte)

As wraparound is not permitted for the DSAR, a bound fault occurs if carry occurs with the above addition.

SHRINK FOR EXTENDED SEGMENT DESCRIPTORSa) V = 10100 Normal Shrink with Type Change

The segment descriptor indicated by the S, D fields of a vector is fetched in the same way as by the copy function. If the type T of the fetched segment descriptor is not 12 or 14, an IPR fault occurs. For a valid segment descriptor, the shrink operation is performed as follows.

- The following check is made.

$$\text{BASE ADDER} + \text{SIZE} \leq \text{bound (111111111111)}$$

12 bits

If the sum of the BASE ADDER and SIZE exceeds the value obtained by extending the bound of the fetched segment descriptor 12 "1" bits to the right, or if the addition produces a carry from the most significant bit, a bound fault occurs.

- After this check, a new base and bound are created.

$$\text{New base} = \text{old base} + \text{BASE ADDER}$$

↳ A Bound fault occurs if a carry is generated.

$$\text{New bound} = \text{SIZE}$$

- A new flag field is created in the same way as for the V = 01XXX normal shrink.
- A new type T is set in the following way.
If old T = 12, then new T = 0.
If old T = 14, then new T = 2.
- SEGID_n and AR_n are set in the same way as for normal shrink.

LDD_n

b) V = 10101 Normal Shrink with No Type Change

The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid descriptor, the shrink operation is performed as shown below:

```

BASE ADDER + (SIZE 000000000000 + base lower-order 12 bits)
              12 bits
<= bound 111111111111
          12 bits
    
```

where the base denotes the value of the base field of the fetched segment descriptor.

First, the sum of the value obtained by extending the SIZE 12 bits to the right and the low-order 12 bits of the base is obtained. If this sum plus the BASE ADDER exceeds the value obtained by extending the bound of the descriptor 12 bits to the right, or if a carry is generated by the addition, a Bound fault occurs.

- After the check, a new base and bound are created.

```

New base = old base + BASE ADDER
                |
                └─> A Bound fault occurs if a carry
                    is generated.
    
```

```

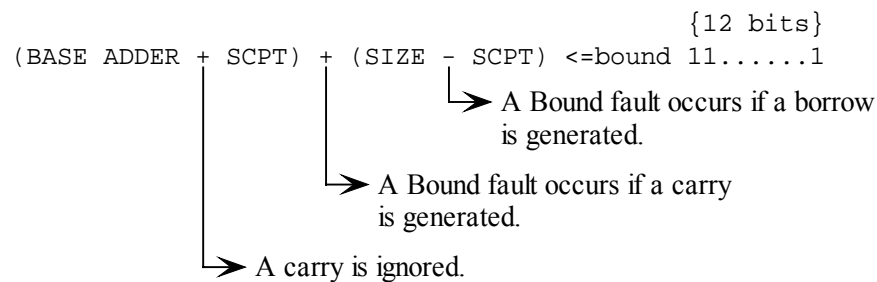
New bound = SIZE
    
```

- SEGID_n and AR_n are set in the same way as for normal shrink.

c) V = 10110 Extended Shrink with Type Change

The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid segment descriptor, the shrink operation is performed as follows.

- The following checks are made on the BASE ADDER and SIZE fields of the vector.



- After the check, a new base and bound are created.

$$\text{New base} = \text{old base} + (BASE\ ADDER + SCPT)$$

→ A carry is ignored.

→ A Bound fault occurs if a carry is generated.

$$\text{New bound} = SIZE - SCPT$$

→ A Bound fault occurs if a borrow is generated.

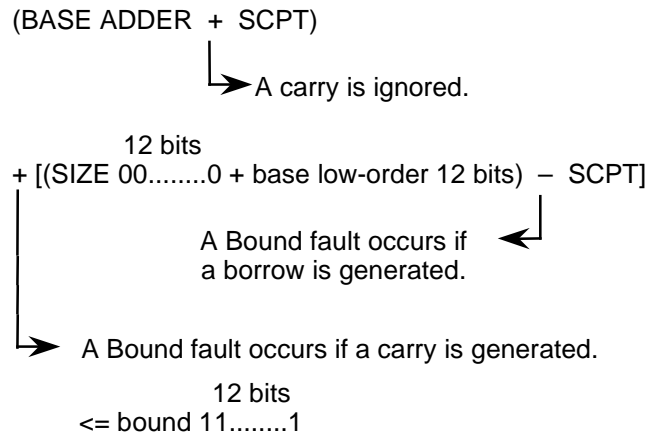
- A new flag field is created in the same way as for a normal shrink (V = 01XXX).
- A new type is set in the following way.
If old T = 12, then new T = 0.
If old T = 14, then new T = 2.
- SEGID_n and AR_n are set in the same way as for a normal shrink.

LDD_n

d) V = 10111 Extended Shrink with No Type Change

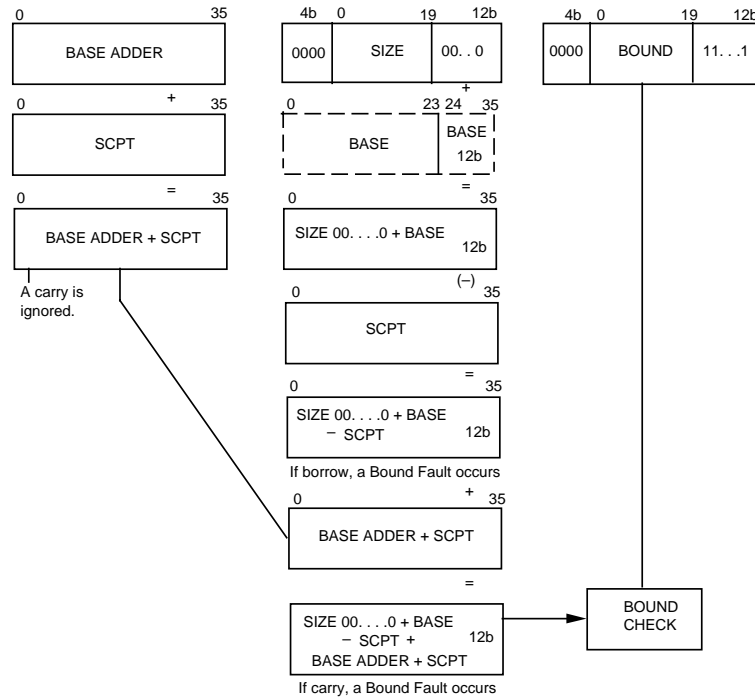
The segment descriptor indicated by the S, D fields of a vector is obtained as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 12 or 14. For a valid descriptor, the shrink operation is performed as described below.

- The following check is made on the BASE ADDER and SIZE fields of the vector.



First, the sum of the value obtained by extending SIZE 12 bits to the right and the low-order 12 bits of the base of the fetched segment descriptor is obtained. The difference between this sum and SCPT is obtained. The difference is added to the sum of the BASE ADDER and SCPT.

Second, this sum is compared to the value obtained by extending the bound of the fetched descriptor 12 bits to the right. This operation is illustrated below.



- After the check, a new base and bound are created.

$$\text{New base} = \text{old base} + (\text{BASE ADDER} + \text{SCPT})$$

A carry is ignored.

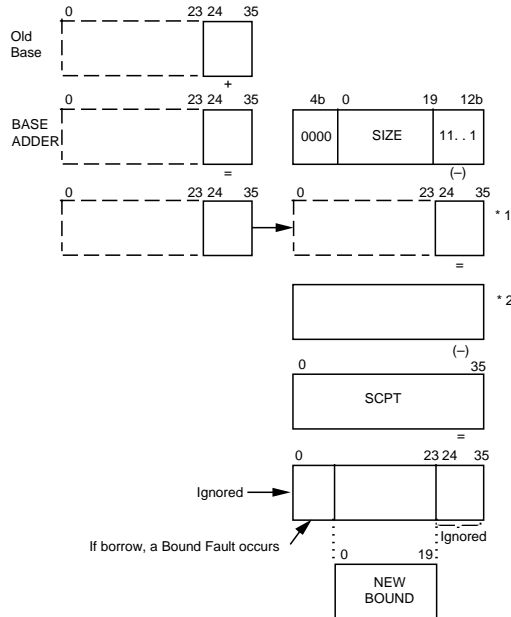
A Bound fault occurs if a carry is generated.

$$\text{New bound} = \{12 \text{ bits}\} [(\text{SIZ } 11 \dots 1 - (\text{old base low-order 12 bits} + \text{BASE ADDER low-order 12 bits} - \text{SCPT})]_{4-23}$$

A Bound fault occurs if a borrow is generated.

LDD_n

The following illustrates locating the new bound.



*1: (Old base + BASE ADDR)_{low-order 12 bits}
{12 bits}

*2: SIZE 111111111111 - (old base + BASE ADDR)_{low 12 bits}
Flag fields are handled as a normal shrink.

- A new type T is the same as the original (old) type T.
- SEGID_n and AR_n are set in the same way as for normal shrink.

e) V = 10010 Normal Base Shrink with No Type Change

The segment descriptor indicated by the S, D fields of a vector is obtained in the same way as for the copy function. An IPR fault occurs if the type T of the fetched segment descriptor is not 0, 2, 12, or 14.

The SIZE field of the vector is ignored in the processing for a valid descriptor illustrated below.

- The following check is made on the BASE ADDER of the vector.

$$\text{BASE ADDER} \leq \underbrace{0000000000000000}_{\{16 \text{ bits}\}} \text{ bound}$$

If the condition in the above check is not met, a Bound fault occurs.

- After the check, a new base and bound are created.

$$\text{New base} = \text{old base} + \text{BASE ADDER}$$

└─> A Bound fault occurs if
a carry is generated.

$$\text{New bound} = \underbrace{[00 \dots 0]}_{\{16 \text{ bits}\}} \text{ bound} - \text{BASE ADDER}]_{16-35}$$

- A new flag field is created the same as for a normal shrink (V = 01XXX).
- A new type T is the same as the original (old) type T.
- SEGID_n and AR_n are set in the same way as for a normal shrink.

For a segment descriptor with T = 12 or 14, the shrink operation is performed in the following way.

- The following check is made on BASE ADDER of the vector.

$$\text{BASE ADDER} + \underbrace{\text{base low-order 12 bits}}_{\{12 \text{ bits}\}} \leq \underbrace{0000}_{\{4 \text{ bits}\}} \underbrace{\text{bound } 11 \dots 1}_{\{12 \text{ bits}\}}$$

└─> A Bound fault occurs if a carry is generated.

where the low-order 12 bits of base are the low-order 12-bits of the base field of the fetched segment descriptor.

If the above condition is not met, a Bound fault occurs.

LDD_n

- After the check, a new base and bound are created.

New base = old base + BASE ADDER

↳ A Bound fault occurs if a carry is generated.

New bound = [({4 bits} old bound {12 bits} 11...1
 - old base low-order 12 bits)
 - BASE ADDER]4-23

- A new flag field is created in the same way as for a normal shrink (V = 01XXX).
- The new type T is the same as the original (old) type T.
- SEGID_n and AR_n are set in the same way as for the normal shrink.

f) V = 10011 Extended Base Shrink with No Type Change

The segment descriptor indicated by the S, D, fields of a vector is located in the same way as for the copy function. An IPR fault occurs if the type T of the fetched descriptor is not 0, 2, 12, or 14.

The SIZE field of the vector is ignored in the processing described below.

For a segment descriptor with T = 0 or 2, the shrink operation is performed in the following way.

- The following check is made on the BASE ADDER and the SCPT of the vector.

$$\text{BASE ADDER} + \overset{\{16 \text{ bits}\}}{\text{SCPT}} \leq 00\dots0 \text{ bound}$$

↳ A carry is ignored.

If these conditions are not met, a Bound fault occurs.

After the check, a new base and bound are created.

$$\text{New base} = \text{old base} + (\text{BASE ADDER} + \text{SCPT})$$

↳ A Bound fault occurs if a carry is generated.

$$\overset{\{16 \text{ bits}\}}{\text{New bound}} = 00\dots0 \text{ bound} - (\text{BASE ADDER} + \text{SCPT})_{16-35}$$

A carry is ignored ←

A new flag field is created in the same way as for a normal shrink (V = 01XXX).

The new type T is the same as the original (old) type T.

SEGID_n and AR_n are set in the same way as for normal shrink.

For a segment descriptor with T = 12 or 14, the shrink operation is performed in the following way.

LDD_n

The following check is made on the BASE ADDER and SUBSCRIPT (SCPT) of the vector.

$$(\text{BASE ADDER} + \text{SCPT}) + \text{base low-order 12 bits}$$

\swarrow A Bound fault occurs if a carry is generated.
 \searrow A carry is ignored.

$\{4 \text{ bits}\}$ $\{12 \text{ bits}\}$
 $\leq 0000 \text{ bound } 11\dots 1$ * (Referred to by NOTE below)

where the base low-order 12 bits are the low-order 12 bits of the base field of the fetched segment descriptor

If this condition is not met, a bound fault occurs.

- After the check, a new base and bound are created.

$$\text{New base} = \text{old base} + (\text{BASE ADDER} + \text{SCPT})$$

\swarrow A Bound fault occurs if a carry is generated.
 \searrow A carry is ignored.

$$\text{New bound} = [(\{4 \text{ bits}\} \text{ old bound } \{12 \text{ bits}\} - \text{old base low-order bits})4-23]$$

\swarrow A Bound fault occurs if a borrow is generated.
 \searrow A carry is ignored.

NOTE: This Bound fault will never occur if the starred (*) check condition above has been met.

- A new flag field is created in the same way as for a normal shrink (V = 01XXX).
- A new type T is the same as the original (old) type T.
- SEGID_n and AR_n are set in the same way as for a normal shrink.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, IR, RI, IT, CI, SC, SCR (See NOTES below for explanation.)

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Illegal Procedure (IPR) Faults can be caused by any of the following conditions:
 - a) Modifications RI, IR, IT, DU, and DL when the DR_m segment descriptor type T = 1 or 3
 - b) Modifications DU, DL, CI, SC, SCR when the DR_m segment descriptor type T = 0, 2, 4, 6, 12, or 14
 - c) Illegal repeats RPT, RPD, and RPL
 - d) Vector fields S = 0 and D = 1760 (octal)
 - e) If vector bits 29 and 30 are 01 or 10 and descriptor obtained is type T=5 or 7-15
 - f) If a carry occurs when a T = 4 or 6 super descriptor is loaded into DR_n, and it is converted by hardware to a standard segment descriptor, an IPR fault will occur. (Refer to the description of Super Descriptors in Section 3.)
 - g) When instruction word bit 29 = 1 and DR_m segment descriptor is type T = 5 or 7-11, 13, 15.
 - h) Specification of a type 10 or type 13 segment descriptor if in SV mode.

LDD_n

2. Command Faults can be caused by any of the following conditions:
 - a) If the CPU is not in Privileged Master mode, when S = 0 and D = 1761, 1763, or 1764 (octal)
 - b) If the CPU is not in Privileged Master mode, when bits 29 and 30 of the first word in the vector do not specify data stack shrink (V = 11XXX) and the vector S and D fields specify DSDR

NOTE: When the CPU is in the Privileged Master mode, the segment descriptor from DSDR is used to execute the specified operation. In this instance, DSDR and DSAR remain unchanged.
3. Bound Faults can be caused by any of the following conditions:
 - a) When S = 0 and D > PSR bound
 - b) When S = 2 and D > ASR bound
 - c) When S = 1 or 3 and D > LSR bound
 - d) When BASE ADDER + vector SIZE > DR_n bound with shrink operation for standard descriptors
 - e) When DR_n location + vector BASE ADDER + vector SIZE > DR_n bound with shrink operation for super descriptors
 - f) When an illegal carry or borrow occurs while a base and bound are generated, while a size check is performed, or while a new DSAR is generated
 - g) In cases other than the above, general fault conditions also apply when segment descriptors and page tables are accessed. These conditions are noted in the individual vector procedures descriptions.
4. Security Fault, Class 1

If the housekeeping bit of the page that includes the selected descriptor is OFF when a descriptor is loaded with the LDD instruction

LDD_n**EXAMPLES:**Direct Load

1	8	16	32

	LDD0	0,,7	Load DR0 from location zero of descriptor segment framed by DR7 1770 -> SEGID0 zeros -> AR0

Copy

1	8	16	32

	LDD0	CPYDR7	Copy DR7 into DR0 1777 -> SEGID0 zeros -> AR0
	.		
	.		
	.		
CRYDR7	CVEC	.DR7	

Normal Shrink

1	8	16	32

	LDD0	BUFVEC	
	.		
	.		
	.		
BUFFER	BSS	320	
BUFLN	EQU	*-BUFFER	
BUFVEC	VEC	.ISR, BUFFER, BUFLN, READ	

LDDR

11.1.17 LDDR

LDDR	Load Double Register to Register Pair	433(1)
------	---------------------------------------	--------

FORMAT:

00	0304	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

LDDR		R1, R2	

OPERATING MODES:

Executes in ES/EI mode only

SUMMARY:

R1, R2 = 0, 2, 4, 6, AQ
 C(R2-pair) -> C(R1-pair)
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

LDDR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1\text{-pair}) = 0$, then ON; otherwise, OFF
Negative	If $C(R1\text{-pair})(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDDSA

11.1.18 LDDSA

LDDSA	Load Data Stack Address Register	170(1)
-------	----------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

Bits 0-16 of C(Y) → C(DSAR)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LDDSA**NOTES:**

1. The DSAR is a 17-bit register that holds an even-word address.
2. Modifications DU, DL, CI, SC, SCR and illegal repeats RPT, RPD, RPL cause an IPR fault.
3. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.

EXAMPLE:

1	8	16

	LDP	P, PSH, SD, PSH, DL
	LDP	P, PSH, .CTYP, DL
	LDDSD	PH, ADS, , P, PSH
	STZ	TEMP, , P, DSR
	LDDSA	TEMP, , P, DSR

LDDSD

11.1.19 LDDSD

LDDSD	Load Data Stack Descriptor Register	571(1)
-------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y\text{-pair}) \rightarrow C(DSDR)$

EXPLANATION:

The double-word memory operand is fetched from even and odd memory locations Y and Y+1. The operand must be in standard descriptor format with a type field of $T = 0$. The lower three bits of the base of this segment descriptor must be zero (i.e., the descriptor in the DSDR specifies the segment beginning from the boundary of an even word). The flag bit 22 must be zero.

When these conditions are met, the obtained descriptor is loaded into the DSDR. If one or more of the above conditions are not met, an IPR fault occurs and the DSDR content remains unchanged.

The lower three bits of the descriptor bound field should all be ones to ensure that the area specified with the DSDR is a multiple of word pairs. Hardware does not check these three bits.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

LDDSD**ILLEGAL REPEATS:**

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Any of the following conditions causes an IPR fault (the DSDR remains unchanged):
 - a) Modification CI, SC, SCR, DU, or DL
 - b) Illegal repeat RPT, RPD, or RPL
 - c) If type field T is not equal to 0
 - d) If the base is not modulo 2 words
 - e) If the descriptor flag bit 22 is not = 0
2. If the processor is Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

1	8	16	32
EXP	LDP	P0,SD.PSH,DL	
	LDD	P0,PH.USL,,P0	
	LDP	P0,.CTYP,DL	
	ADLA	UL.ISR+1,,P0	
	STA	S.ISR+1,QU,P4	
	LDD	P1,S.ISR,QU,P3	P1 = sub-dispatch ISR
	LDAS	S.APR,,P4	load special registers
	LDPS	S.APR,,P4	
	LDDSD	S.DSR,,P4	
	LDDSA	SBDH	
	LDSS	.KLSDS,PN*,P.KL	load SSR for sub-disp by processor number
*	STX6	.KLPRG,7,P.KL	set processor flags,sub-disp
	SXL3	.KLPRG,7,P.KL	
	LDD	P2,S.ENT,QU,P3	P2 = entry descriptor climb
	LCQ	=O204020,DL	
	ANSQ	.QFST,3,P6	clear fault status bits

LDE

11.1.20 LDE

LDE	Load Exponent Register	411(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y)(0-7) \rightarrow C(E)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero Set OFF

Negative Set OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

LDEAn**11.1.21 LDEAn**

LDEAn	Load Extended Address <u>n</u>	61 <u>n</u> (1)
-------	--------------------------------	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(Y) -> location field of Descriptor Register (DRn)

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LDEAn

NOTES:

1. This set of eight instructions enables the loading of the location field of a descriptor register (DR_n) from memory address Y. The DR_n must contain a super descriptor (type field T must be 4 or 6). Otherwise, an IPR fault occurs.
2. If T = 4 or 6, if a carry occurs when creating the base (DR_n base+location field); or, if a borrow occurs when creating the bound (DR_n bound-location field), an IPR fault occurs.
3. Any of the following conditions causes an IPR fault:
 - a) Modification CI, SC, or SCR
 - b) If descriptor type field T of DR_n is not 4 or 6
 - c) Illegal repeat RPT, RPD, or RPL
4. An IPR occurring as the result of an illegal op code sets bit 0 of the Fault Register (FLTR) = 1. An IPR occurring as the result of an illegal decimal digit sets bit 6 of the Fault Register (FLTR) = 1.
5. DU and DL address modification permitted.

EXAMPLE:

1	8	16	32

MSCN7	NULL		
	EAX2	1,2	
	CMPX2	4,DU	is defective memory table full?
	TZE	ESCN	yes
	LDA	.KLMSZ,,KLS	no
	ANA	=0777777,DL	isolate real memory size
	AOS	ADDRS	advance page number
	CMPA	ADDRS	is this page the last?
	TZE	ESCN	yes
	LDEA	RMS,SUPAD	loading loc. - super descriptor
	LDA	1K*4,DL	adjust byte
	ASA	SUPAD	
	TRA	MSCN2	next page scan

LDI**11.1.22 LDI**

LDI	Load Indicator Register	634(0)
-----	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y)(18-33) \rightarrow C(IR)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Master mode (IR bit 28) not affected

All others: If corresponding bit in $C(Y) = 1$, then ON; otherwise, OFF

LDI**EXPLANATION:**

The relation between bit positions of C(Y) and the indicators is shown below.

<u>Bit Position</u>	<u>Indicator (or Mask)</u>
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	UNUSED
27	IGNORED
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Exponent underflow mask
32	Hexadecimal exponent mode
33	Fixed-point overflow mask
34-35	UNDEFINED

NOTES:

1. The Tally Runout indicator reflects bit 25 of C(Y) regardless of what address modification is performed on the LDI instruction for tally operations.
2. Master Mode cannot be changed by the LDI instruction.
3. An Overflow Fault does not occur when the overflow indicator, exponent overflow indicator, or exponent underflow indicator is set ON via the LDI instruction, even if the overflow mask indicator is OFF.
4. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
5. Hexadecimal mode is exclusively controlled by bit 32 of the IR.
6. In SV mode, C(Y)33 is ignored and IR(33) is not changed.
7. Software should not use the LDI instruction to set the indicator bit 30 (Multiword Instruction Interrupt), as unpredictable results may occur on a subsequent multiword restartable EIS instruction.

LDMB**11.1.23 LDMB**

LDMB	Load Performance Monitor Mode	755(1)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

The LDMB instruction sets the operating mode for the Performance Monitor Counters as specified by bits 15-17 of the y field of the instruction (see table below) when the PM option is ON. When the PM is OFF, the LDMB performs no operation (NOP) except for the specified address modification.

<u>y(15-17)</u>	<u>Counters</u>	<u>PM Mode</u>	<u>Use</u>
000	No Change	No Change	Instruction is extended. See "V9000 Extended Functionality" table below.
001	No Change	RUN	Start PM mode run mode with current PMCn values
010	Reset	No Change	DO NOT USE
011	Reset	RUN	Start run mode after resetting all PMCn to zeros.
100	No Change	IDLE, READ	Stop PM counters
101	No Change	RUN	DO NOT USE
110	Reset	IDLE	Stop and reset PM counters
111	Reset	IDLE	DO NOT USE

LDMB

y(0-14)	V9000 Extended Functionality:
00002 octal	Load internal KPX register from X6, load internal SNUMB register from A-register
00001 octal	Post CONNECT_VI message on IOP n message queue. The CONNECT_VI message is intended as a "pollbit notification" or "doorbell" mechanism that notifies the IOP of a new VI message. The actual connects issued for a VI are handled via the 'cioc' instruction. where: n = C(a)(25-27) C(a) → Data word 0 of the message C(q) → Data word 1 of the message
Other	no operation

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LDO**11.1.24 LDO**

LDO	Load Option Register	172(1)
-----	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any. See Explanation below.

EXPLANATION:

When the CPU is in Privileged Master Mode:

Data Stack Clear Flag (DSCF) is loaded from C(Y)18: 0 = do not clear 1 = clear	DSCF controls memory clear operation when data stack shrink is executed with the CLIMB instruction.
Safe Store Bypass Flag (SSBF) is loaded from C(Y)19: 0 = bypass safe store 1 = perform safe store	SSBF controls ICLIMB safe store bypass.
If the CPU is in Master or Slave mode, DSCF and SSBF are unchanged.	

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LDO

NOTES:

1. Although this instruction is legal in all processor modes, the setting of the two flag bits is mode-dependent.
2. Modifications CI, SC, SCR, and illegal repeats RPT, RPD, RPL cause an IPR fault.

LDP_n**11.1.25 LDP_n**

LDP _n	Load Pointer Register <u>n</u>	47 _n (1)
------------------	--------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

This set of eight instructions is similar to the LDD_n instruction with the copy option; however, no vector is required and AR_n may be loaded with a value other than all zeros.

The formats of AR_n in the three segment modes are:

Field:	Position in AR _n		Description:
	NS mode:	ES/EI mode:	
WORD	00-17		Unsigned word displacement
		00-29	Twos-complement word displacement
BYTE	18-19	30-31	Byte address within word
BIT	20-23	32-35	Bit address within byte

LDP_n

NS Mode

If DU or DL modifications are not used:

```
C(Y)(0-23)                -> C(ARn)
C(descriptor spec. - S,D) -> C(DRn)/DRn type field changed.
C(Y)(24-35)                interpreted as S,D field
```

If DU modification is used:

```
C(Y)(0-17)                -> C(ARn)(0-17)
00...0                    -> C(ARn)(18-23)
C(Y)(24-35)                interpreted as S,D field
```

If DL modification is used:

```
00...0                    -> C(ARn)(0-17)
Y(0-5)                    -> C(ARn)(18-23)
Y(6-17)                   interpreted as S,D field
```

ES/EI Mode

If DU or DL modifications are not used:

```
C(Y)(0-35)                -> C(ARn) [double-word aligned]
C(descriptor spec. - S,D) -> C(DRn)/DRn type field changed.
C(Y+1)(0-11)              interpreted as S,D field
C(Y+1)(12-35)            ignored
```

If DU modification is used:

```
Y(16-33)                 -> C(ARn)(0-17)
00...0                   -> C(ARn)(18-35)
00...0                   interpreted as S,D field
```

If DL modification is used:

```
00...0                   -> C(ARn)(0-13)
Y(0-21)                 -> C(AR)(14-35)
Y(22-33)                interpreted as S and D
```

In all modes, interpretation of the S and D fields and the corresponding operation is the same as that for the LDD_n instruction vector S and D fields specified by the copy function (with vector copy function). However, for LDP_n, no vector is required, and AR_n may be loaded with a value other than all zeros. The descriptor is loaded into DR_n. (When S = 0 and D = 1761, the type in DR_n is changed; the value described with the LDD_n instruction copy function is loaded into SEGID_n.)

The S and D fields of the pointer locate the descriptor to be loaded into DR_n in the following way.

When S = 0

For D = 0000 through 1757 (octal) and D ≤ PSR bound, the descriptor is loaded from the parameter segment and D is used as an index to the desired descriptor. The value in D is the number of the descriptor to be loaded and can be treated as a modulo 8 index; that is, D can be converted to a byte address by appending three zeros as the three least significant bits.

For D = 1760 through 1777 (octal), the descriptors referenced by S, D are contained in selected registers and copied to DR_n.

D = 1760	Undefined, IPR fault
D = 1761	Change Descriptor Type Field in DR _n
D = 1762	Instruction Segment Register (ISR)
D = 1763	Data Stack Descriptor Register (DSDR)
D = 1764	Safe Store Register (SSR)
D = 1765	Linkage Segment Register (LSR)
D = 1766	Argument Stack Register (ASR)
D = 1767	Parameter Segment Register (PSR)
D = 1770	DR0, Descriptor Register 0
D = 1771	DR1, Descriptor Register 1
D = 1772	DR2, Descriptor Register 2
D = 1773	DR3, Descriptor Register 3
D = 1774	DR4, Descriptor Register 4
D = 1775	DR5, Descriptor Register 5
D = 1776	DR6, Descriptor Register 6
D = 1777	DR7, Descriptor Register 7

NOTE: When D = 1761 (octal) and the processor is in Privileged Master mode, if the descriptor contained in DR_n is type 1 or 3, the type is changed to 0 or 2, respectively. However, if the descriptor is not type 1 or 3, no change is made and no fault occurs.

When S = 2

The D_n descriptor of the current argument segment is selected. A relative byte offset is formed by extending the D field by 3 zeros.

LDP_n

When S = 1 or 3

The D_n descriptor of the current linkage segment is selected. A relative byte offset is formed by extending the D field by 3 zeros.

For all values of S, loading DR_n affects the nth address register (AR_n) and the nth segment identity register (SEGID_n):

- a) AR_n is set according to the preceding rules for DU/DL modification given for NS and ES modes for the LDP_n EXPLANATION;
- b) If DR_n was loaded from another DR or the instruction segment register (ISR), the associated segment identity content is transferred to SEGID_n; otherwise, SEGID_n is set to the S and D value contained in the pointer;
- c) If an IPR or Bound fault occurs, DR_n, AR_n, and SEGID_n are not changed.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if bit 29=1 and the operand segment is not type T = 0, 2, 4, or 6.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
3. A Command fault occurs as with the LDD_n instruction copy function.
4. Other faults occur as with the LDD_n copy function.

EXAMPLE:

1	8	16	32

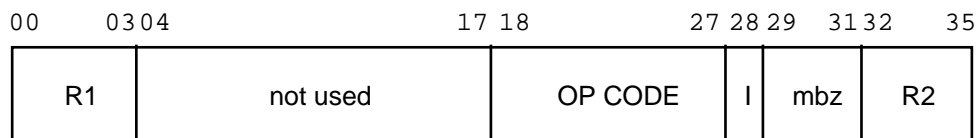
TPUTEX	SZN	TRAPTR	test for trap in use
	TZE	TRAPOK	no trap enabled
	LDP6	TRAPTR	trapping--get location
*			(ensuring that address register
*			has offset and descriptor is
*			type 0) of cell to be monitored
*			in AR via P6; mask for desired
*			pattern, compare with bad value
	SAR6	TRAPCT	
	LDP6	TRAPCT	
	LDA	0,,P6	
	ANA	TRAPMK	
	CMPA	TRAPVL	
	TZE	GOTCHA	trap has sprung
TRAPOK	LDP6	SD.SSA,DL	reload P.SSA (if no/OK trap)
*			TRA monitor if monitor active
	TRA	0,4	exit

LDPR

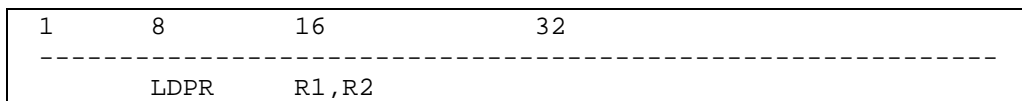
11.1.26 LDPR

LDPR	Load Positive Register to Register	432(1)
------	------------------------------------	--------

FORMAT:



CODING FORMAT:



OPERATING MODES:

Executes only in ES/EI mode

SUMMARY:

R1, R2 : 0, 1, 2, 3, 4, 5, 6, 7, A, Q
 |C(R2)| -> C(R1)
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

LDPR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If C(R1) = 0, then ON; otherwise, OFF
Negative	Set to OFF
Overflow	When C(R2) = 400000000000

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDPS

11.1.27 LDPS

LDPS	Load Parameter Segment Register	771(1)
------	---------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

`C(Y-pair) -> C(PSR); C(Y-pair) unchanged`

EXPLANATION:

The descriptor is fetched from even/odd memory locations Y and Y+1. The hardware performs the following checks on the descriptor.

- Type field must have a value of T = 1.
- Base must be 0 modulo 8 bytes.
- If flag bit 27 = 1 (bound valid), bound must be 7 modulo 8 bytes.

If these conditions are met, the descriptor is loaded into PSR. During PSR load, PSR bound field bits 0-6 are forced to zero by the hardware rather than being loaded from the memory operand. If flag bit 27 of the operand descriptor is equal to zero, the entire bound field of the PSR is forced to zero, independent of any value the operand descriptor bound field may contain, and the bound check is bypassed.

This instruction is identical with LDAS, except that it loads the parameter segment register (PSR) instead of the argument stack register (ASR).

LDPS**ILLEGAL ADDRESS MODIFICATIONS:**

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Any of the following conditions cause an IPR fault:
 - a) Modifications DU, DL, CI, SC, and SCR,
 - b) Illegal repeats RPT, RPD, and RPL,
 - c) Descriptor type field T is not 1,
 - d) If the base and bound limits of the operand descriptor are not modulo 2 words (only when flag bit 27 = 1).
2. If the processor is in Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

1	8	16	32

*	LDP	P.SSR, .SSR, DL	(Load descriptor of fault frame in safe store stack)
	LDP	P.SSR, .CTYP, DL	(Change to type 0)
	LDAS	.WASR, , P.SSR	(Restore ASR from safe store)
	LDPS	.WPSR, , P.SSR	(Restore PSR from safe store)

LDQ

11.1.28 LDQ

LDQ	Load Q-Register	236(0)
-----	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \rightarrow C(Q)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

LDQB**11.1.29 LDQB**

LDQB	Load Q-Register with Byte	502(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \text{ BYTE} \rightarrow C(Q)(28-35);$
 $C(\text{BYTE})(0) \rightarrow C(Q)(00-27)$

EXPLANATION:

The byte at the byte position specified by $C(Y)$ is stored right-justified in the $C(Q)$. Bit 0 of the byte is left-extended into the remainder of the $C(Q)$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, AU

ILLEGAL REPEATS:

All

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

LDQB

NOTE:

1. An Illegal Procedure fault occurs if the TAG field specifies RI, IR, IT, or R = DU, DL, AU. If R = IC, the contents of IC are treated as a word count.
2. An Illegal Procedure fault occurs
3. A segment bound check is performed on the whole C(Y) word regardless of the specified byte position.

LDQC**11.1.30 LDQC**

LDQC	Load Q-Register and Clear	032(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \rightarrow C(Q);$
 $00\dots0 \rightarrow C(Y)$

EXPLANATION:

This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

LDQC

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If bit 0 of $C(Q) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

LDQH**11.1.31 LDQH**

LDQH	Load Q-Register with Halfword	512(0)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y) \text{ HALFWORD} \rightarrow C(Q)(18-35);$
 $C(\text{HALFWORD})(0) \rightarrow C(Q)(00-17)$

EXPLANATION:

The halfword at the halfword position specified by $C(Y)$ is stored right-justified in the $C(Q)$. Bit 0 of the halfword is left-extended into the remainder of the $C(Q)$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, AU

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

LDQH

NOTES:

1. An Illegal Procedure fault occurs if the TAG field specifies RI, IR, IT, or R = DU, DL, AU. If R = IC, the contents of IC are treated as a word count.
2. An Illegal Procedure fault occurs
3. A segment bound check is performed on the whole C(Y) word regardless of the specified byte position.
4. The lowest order bit of the address is ignored after address modification is completed.

LDRR**11.1.32 LDRR**

LDRR	Load Register from Register	430(1)
------	-----------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	32

LDRR		R1, R2	

OPERATING MODES:

Executes in ES/EI mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q
 C(R2) → C(R1)
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

LDRR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If $C(R1) = 0$, then ON; otherwise, OFF

Negative If $C(R1)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

LDSS**11.1.33 LDSS**

LDSS	Load Safe Store Register	773(1)
------	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y)(0-35) \rightarrow C(SSR)(0-35)$
 $C(Y+1)(0-32) \rightarrow C(SSR)(36-68)$
 $000 \rightarrow C(SSR)(69-71)$
 $C(Y+1)(34-35) \rightarrow C(SCR)0-1$

EXPLANATION:

The operand is fetched from even and odd memory locations Y and Y+1. The operand must be a standard descriptor with type T = 1 or 3. The following checks are performed on the descriptor.

- a) For T = 1, flag bits 20, 21, 27, and 28 = 1 and flag bits 25 and 26 = 0.
- b) For T = 3, flag bits 20 and 21 = 1.

If these conditions are met, the descriptor is loaded into the safe store register (SSR). The lower three bits of the SSR base are forcibly set to zero. If one or more of the above conditions is not satisfied, the instruction is terminated and an IPR fault is generated. In this case, the SSR remains unchanged.

Each successful execution of LDSS causes the 2-bit stack control register (SCR) to be loaded in the following way. (The SCR is associated with the SSR and contains a code that denotes the size of the last frame on the stack.)

$C(Y + 1)(34, 35) \rightarrow C(SCR)$

(Refer to Safe Store Stack in discussion of CLIMB instruction.)

The SSR bound is not checked by LDSS.

LDSS

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Any of the following conditions causes an IPR fault:
 - a) Modification DU, DL, CI, SC, or SCR,
 - b) Illegal repeat RPT, RPD, or RPL,
 - c) If T is not equal to 1 nor 3,
 - d) If either the flag bit or the base checks fail.
2. If the processor is not in Master or Slave mode, the execution of this instruction causes a Command fault.

EXAMPLE:

1	8	16	32

FANY	STZ	.SVFLT,,P.SSA	
	LDX0	.ST2CS,,P.SSA	
	TZE	NEPRA	Not type 2 critical
	STSS	.STEMP+6,,P.SSA	
	LDAQ	SSRXX	
	ADLAQ	.STEMP+6,,P.SSA	backup safe store to prior frame
*			
	STAQ	.STEMP+6,,P.SSA	
	LDSS	.STEMP+6,,P.SSA	
	LDP	P0,.SSR,DL	
	LDX0	=O377001,DU	
	STX0	.WREGS,,P0	
	TRA	RETOUT	

LDT**11.1.34 LDT**

LDT	Load Timer Register	637(0)
-----	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y)(0-26) \rightarrow C(TR); C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. The use of this instruction in the Master or Slave mode causes a Command fault.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LDWS

11.1.35 LDWS

LDWS	Load Working Space Registers	772(1)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

SV Mode		
C(Y) bits:	EDW = x0	EDW = x1
00-08	WSR0	WSR4
09-17	WSR1	WSR5
18-26	WSR2	WSR6
27-35	WSR3	WSR7

EXPLANATION:

The contents of memory location Y replace the contents of working space registers (WSRs) 0, 1, 2, and 3 or WSR 4, 5, 6, and 7 based on the value of bit 17 (NS mode) or 33 (ES/EI mode) of the effective address.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

LDWS**INDICATORS:**

None affected

NOTES:

1. Modifications CI, SC, SCR, DU, DL and illegal repeats RPT, RPD, RPL cause an IPR fault.
2. If the processor is not in the Privileged Master mode, the execution of this instruction causes a Command fault.
3. If the LDWS instruction is used to change the contents of the WSR that is currently the WSR for the instruction segment, then the LDWS must be followed immediately by a TRA $*+1$ to ensure that the new contents of the WSR take effect immediately.

EXAMPLE:

1	8	16	32

	EVEN		
WS03	VFD	9/001, 9/001, 9/013, 9/27	
WS47	VFD	9/45, 9/45, 9/63, 9/510	
	.		
	.		
	.		
LDWS	WS03		Load WSR 0-3 from EVEN word
LDWS	WS47		Load WSR 4-7 from Odd word

LDX_n

11.1.36 LDX_n

LDX _n	Load Index Register <u>n</u> from Upper	22 _n (0)
------------------	--	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(Y)(0-17) \rightarrow C(X_n)$; $C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(Y)(0-35) \rightarrow C(GX_n)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL of LDX0

INDICATORS:

Zero If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF

Negative If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF

LDX_n

NOTES:

1. DL modification executes with all zeros for data in the NS mode.
2. An Illegal Procedure fault occurs if illegal address modification is used.

LIMR

11.1.37 LIMR

LIMR	Load Interrupt Mask Register	553(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

C(A)(1,3,5,7) -> C(IPMRp)(0-3)
 C(A)(0,2,4,6) -> unused. Ignored by hardware.
 C(A)(8,10,11) -> ALLFp - See table below
 C(A)(9) -> unused
 C(A)(12-35) -> unused

A(8)	A(10)	A(11)	ALLFp of this CPU	ALLFp of other CPUs on this board	ALLFp of other CPUs on other boards
0	0	x	Unchanged	Unchanged	Unchanged
0	1	x	Reset to 0	Unchanged	Unchanged
1	0	0	Set to 1	Set to 1	Set to 1
1	0	1	Set to 1	Set to 1	Unchanged
1	1	x	Set to 1	Unchanged	Unchanged

EXPLANATION:

The effective address (Y) is not used by the LIMR instruction.

C(A), C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. The use of this instruction in other than Privileged Master Mode causes a command fault.
2. An Illegal Procedure fault occurs if an illegal repeat is used.

LLAR

11.1.38 LLAR

LLAR	Load Limit Address Register	724(1)
------	-----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(Y)(02-17) → C(ULAR)(00-15)
 C(Y)(20-35) → C(LLAR)(00-15)
 C(Y) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL

INDICATORS:

None affected

NOTES:

1. A Command fault occurs if execution attempted in slave or master mode.
2. An Illegal Procedure fault occurs if illegal address modification is used.

LLPNR**11.1.39 LLPNR**

LLPNR	Load Logical Processor Number Register	364(1)
-------	--	--------

FORMAT:

0	17 18	OP CODE	27 28	29 30	35
y	364 (1)	I	AR	TAG	

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

C(Y) (33-35) → C(LP NR)

EXPLANATION:

Load and read the content of the Logical Processor Number Register

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LLPNR

NOTES:

1. A Command fault will occur if execution is attempted in the Slave or Master Mode.

LLR**11.1.40 LLR**

LLR	Long Left Rotate	777(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Rotate C(AQ) left the number of positions indicated by bits 11-17 of Y (Y modulo 128) (NS mode) or bits 27-33 of Y (ES/EI mode); enter each bit leaving bit position 0 of AQ into bit position 71 of AQ.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise, OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise, OFF

LLR

NOTES:

1. The rotate count comes from the value of Y. To "right-rotate" n bits, use LLR 72- n .
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

LLS**11.1.41 LLS**

LLS	Long Left Shift	737(0)
-----	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Shift C(AQ) left by the number of positions indicated by bits 11-17 of Y (Y modulo 128) (NS mode) or Y(27-33) (ES/EI mode); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise, OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise, OFF
Carry	If bit 0 of C(AQ) changes during the shift, then ON; otherwise OFF

LLS

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LLUF

11.1.42 LLUF

LLUF	Load Lockup Fault Register	674(0)
------	----------------------------	--------

NOTE: The LLUF instruction functions as a NOP in the V9000.

LPDBR

11.1.43 LPDBR

LPDBR	Load Page Table Directory Base Register	171(1)
-------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

SV mode:

$C(Y)(0-18) \rightarrow C(PDBR)$

SVMX mode:

$C(Y)(0-35) \rightarrow C(PDBR)$

EXPLANATION:

In SV mode, the contents of bits 0-18 of Y are loaded into the 19-bit PDBR. In SVMX mode, the contents of bits 0-35 of Y are loaded into the 36-bit PDBR. Associative Memory (Translation Lookaside Buffer) is cleared and C(Y) is unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

LPDBR

NOTES:

1. An IPR fault occurs when illegal address modifications or illegal repeats are used.
2. If the processor is in Master or Slave mode, the execution of this instruction causes a Command fault.
3. A Bounds fault occurs if the last workspace directory entry resides outside physical memory.

LPL

11.1.44 LPL

LPL	Load Pointers and Lengths	467(1)
-----	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	LPL	LOCSYM, R, AM

OPERATING MODES:

Any

SUMMARY:

C(Y), C(Y+1) -> C(P&L)0,1
 C(P&L)2,3,4,5 Unchanged
 C(Y+6), C(Y+7) -> C(LOR)

EXPLANATION:

Pointer and length storage (P&L) is used by hardware to store control information to continue execution after an interruptible multiword instruction has been interrupted during execution. The low operand register (LOR) is a register used with quadruple-precision instructions.

The location of Y must be a multiple of 8. A fault does not occur when the lower 3 bits of Y are not 000. For purposes of execution, the hardware assumes that these bits are 000.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used. The contents of the pointer and lengths registers are changed when the illegal execution of RPT, RPD, RPL, XEC, XED, and indirect modification IT occurs.
2. The pointer and length registers (PL 2, 3, 4, and 5) are used to recover from an interrupt or a Missing Page fault. Because the content depends upon hardware, the software must not change the contents of the pointer and lengths registers.

LREG

11.1.45 LREG

LREG	Load Registers	073(0)
------	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

C(Y)(00-17) → C(X0)
 C(Y)(18-35) → C(X1)
 C(Y+1)(00-17) → C(X2)
 C(Y+1)(18-35) → C(X3)
 C(Y+2)(00-17) → C(X4)
 C(Y+2)(18-35) → C(X5)
 C(Y+3)(00-17) → C(X6)
 C(Y+3)(18-35) → C(X7)
 C(Y+4)(00-35) → C(A)
 C(Y+5)(00-35) → C(Q)
 C(Y+6)(00-07) → C(E)

ES/EI Mode

C(Y)(00-35) → C(X0)
 C(Y+1)(00-35) → C(X1)
 C(Y+2)(00-35) → C(X2)
 C(Y+3)(00-35) → C(X3)
 C(Y+4)(00-35) → C(X4)
 C(Y+5)(00-35) → C(X5)
 C(Y+6)(00-35) → C(X6)
 C(Y+7)(00-35) → C(X7)
 C(Y+8)(00-35) → C(A)
 C(Y+9)(00-35) → C(Q)
 C(Y+10)(00-07) → C(E)

LREG

EXPLANATION:

Memory (location Y) is accessed on an eight-word boundary by setting the lower three bits of the effective address Y to zero, adding a base address to it, and truncating the least-significant word address bit.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LRL

11.1.46 LRL

LRL	Long Right Logical Shift	773(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

NS Mode

Shift C(AQ) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with zeros.

ES/EI Mode

Shift C(AQ) right by the number of positions indicated by bits 27-33 of Y (Y modulo 128); fill vacated positions with zeros.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise, OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise, OFF

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LRMB

11.1.47 LRMB

LRMB	Load Reserve Memory Base	712(0)
------	--------------------------	--------

NOTE: No operation (NOP) is performed. Address modification and paging is performed. Indicators are not affected.

LRS**11.1.48 LRS**

LRS	Long Right Shift	733(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

Shift C(AQ) right by the number of positions indicated by bits 11-17 of Y (Y modulo 128); fill vacated positions with the content of bit 0 of C(AQ).

ES/EI Mode

Shift C(AQ) right by the number of positions indicated by bits 27-35 of Y (Y modulo 128); fill vacated positions with the content of bit 0 of (AQ).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(AQ) = 0, then ON; otherwise, OFF

Negative If C(AQ)(0) = 1, then ON; otherwise, OFF

LRS

NOTES:

1. The shift count in the instruction must be a decimal number.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

LVMMR

11.1.49 LVMMR

LVMMR	Load Virtual Machine Mode Register	720(1)
-------	------------------------------------	--------

NOTE: The LVMMR instruction functions as a NOP in the V9000.

LVMTR

11.1.50 LVMTR

LVMTR	Load Virtual Machine Timer Register	722(1)
-------	--	--------

NOTE: The LVMTR instruction is an illegal op. code on the V9000, any attempt to use this instruction will result in an IPR.

LXL_n**11.1.51 LXL_n**

LXL _n	Load Index Register <u>n</u> from Lower	72 <u>n</u> (0)
------------------	--	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For n = 0,1,...,or 7 as determined by op code:
C(Y)(18-35) → C(X_n); C(Y) unchanged

ES/EI Mode

For n = 0,1,...,or 7 as determined by op code:
C(Y)(18-35) with sign extended → C(GX_n); C(Y) unchanged
Bit 18 of C(Y) is extended to bits 0-17 and loaded into GX_n.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL of LXL0

LXL_n

INDICATORS:

Zero	If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. DU modification executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

MLR

11.2 Machine Instruction Descriptions (M)

Following is a detailed description of the processor instructions and operation codes beginning with the letter M.

11.2.1 MLR

MLR	Move Alphanumeric Left to Right	100(1)
-----	---------------------------------	--------

FORMAT:

00	08 09 10 11	17 18	27 28 29	35
FILL	T 0	MF2	100 (1)	I MF1

00 02 03	17 18	20 21 22 23 24	31 32	35
Y1	CN1	TA1 0	N1	
AR# Y1				R1

00 02 03	17 18	20 21 22 23 24	31 32	35
Y2	CN2	TA2 0	N2	
AR# Y2				R2

CODING FORMAT:

The MLR instruction code is shown below.

1	8	16

MLR	(MF1) , (MF2) , FILL , T	
ADSC _n	LOCSYM , CN , N , AM	
ADSC _n	LOCSYM , CN , N , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

MLR

OPERATING MODES:

Any

SUMMARY:

`C(string 1) -> C(string 2)`

EXPLANATION:

Starting at location YC1, the alphanumeric characters of data type TA1 of string 1 replace, from left to right, the alphanumeric characters of data type TA2 of string 2 that starts at location YC2. If TA1 and TA2 differ, each character has high-order truncation or zero-fill, as appropriate.

If L1 is greater than L2, the least significant (L1-L2) characters are not moved and the Truncation indicator is set. If L1 is less than L2, bits 0-8, 3-8, or 5-8 of the FILL character (depending on TA2) are inserted as the least significant (L2-L1) characters. If L1 is less than L2, bit 0 of C(FILL) = 1, TA1 = 01, and TA2 = 10 (6-4 move); the hardware looks for a 6-bit overpunched sign. If a negative overpunch sign is found, a negative sign (octal 15) is inserted as the last FILL character. If a negative overpunch sign is not found, a positive sign (octal 14) is inserted as the last FILL character.

L2 = 0 does not necessarily mean that the instruction functions as a no-op, because the Truncation indicator may be affected.

The contents of string 1 remain unchanged except in cases of string overlap.

MF1 and MF2 (Multiword Modification Fields) are 7-bit fields specifying address modifications to be performed on the operand descriptors. They are broken into four subfields represented as (bit1, bit2, bit3, Index-register) in the instruction. They may be coded in the following way.

bit1	= 0	No address register used
	= 1	The address register is defined in the operand descriptor address field (e.g., ADSC9 ,,AR)

bit2	= 0	Operand length is specified in the N field of the operand descriptor (e.g., ADSC6 „24,)
	= 1	Operand length is contained in the register specified by the code in the N field of the operand descriptor (e.g., ADSC4 „X4,)
bit3	= 0	The operand descriptor follows the instruction word in its memory location.
	= 1	The operand descriptor location following the instruction in memory points to the operand descriptor.
Index-register		The address modification register is defined as 0, 1, 2, 3, 4, 5, 6, 7, AU, QU, A, or Q.

See "Multiword Modification Field" and "Alphanumeric Operand Descriptors" in Section 5, and "Alphanumeric Instructions" under "Multiword Instructions" in Section 7 for additional information.

For speed, the MLR and MRL instructions operate on four double-words at a time. This mode of operation does not cause a problem when moving between either nonoverlapped strings or between any normal combination of any length overlapped strings. (In the latter case, software must choose between MLR and MRL to ensure that the overlapped sending characters are moved before they are moved into because they are also receiving characters.) This mode of operation can cause a problem when MLR or MRL is used to replicate a pattern across a string.

For example, one procedure used to replicate a pattern of K characters across a string of L characters is to 1) store the K characters into character positions 1 through K of the string, and 2) "move" a string of length L - K and starting position 1 to the same length string starting at position K + 1. In this way, the last L - K sending characters are created "on the fly". The mode of operating on four double-words at a time does not allow this creation "on the fly" for K less than four double-words of characters (when K starts on a word boundary or is less than eight double-words of characters and does not start on a word boundary).

To replicate a pattern between two characters and four double-words of characters, additional instructions must be used to initialize the first four double-words of the string of L characters. To replicate a 1-character pattern (most common application), a simple move with fill from a zero-length string can be used. (See examples below.)

MLR

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if illegal repeats are used. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1. A fault does not occur even when L2 = 0. L2 = 0 does not mean NOP; the truncation indicator may be affected.

EXAMPLES:

1	8	16	32

	MLR	,,20	move with blank fill
	ADSC6	FLD1,,12	sending descriptor
	ADSC6	FLD2,4,14	receiving descriptor
	USE	CONST.	memory contents
FLD1	BCI	2,ABCDEFGHIJKL	
FLD2	BSS	3	xxxxABCDEFGHIJKLbb (Result)
	USE		
	MLR	,,400	move with sign captured
	ADSC6	FLD1,3,9	sending descriptor
	ADSC4	FLD2,6,10	receiving descriptor
	USE	CONST.	
FLD1	BCI	2,bb12345678R	
FLD2	BSS	2,	xxx-0012345678 (Result)

1	8	16	32

	MLR	(1,0,0),,,QU)	move 24 words from P.IOQ
*			to A+QU bytes.
	ADSC9	0,0,24,P.IOQ	
	ADSC9	A,,24	

MME**11.2.2 MME**

MME	Master Mode Entry Fault	001(0)
-----	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Generates a MME fault that causes the processor to switch to Privileged Master Mode and to execute an Inward CLIMB using the entry descriptor obtained from the word pair in memory locations 32 and 33 (octal).

ILLEGAL ADDRESS MODIFICATIONS:

Not executed

ILLEGAL REPEATS:

RPT, RPD, RPL cause an Illegal Procedure fault.

NOTES:

1. Refer to Section 6 for the description of faults.
2. An IPR fault occurs if an illegal repeat is used.

MP2D

11.2.3 MP2D

MP2D	Multiply Using Two Decimal Operands	206(1)
------	-------------------------------------	--------

FORMAT:

0001	08 09 10 11	17 18	27 28 29	35			
P	00000000	T	R D	MF2	206(1)	I	MF1

00	02 03	17 18	20 21 22 23 24	29 30	35	
Y1		CN1	T N 1	S1	SF1	N1
AR#	Y1					

00	02 03	17 18	20 21 22 23 24	29 30	35	
Y2		CN2	T N 2	S2	SF2	N2
AR#	Y2					

CODING FORMAT:

The MP2D instruction code is shown below.

1	8	16	
	MP2D	(MF1), (MF2), RD, P, T	
	NDSC _n	LOCSYM, CN, N, S, SF, AM	
	NDSC _n	LOCSYM, CN, N, S, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

MP2D**OPERATING MODES:**

Any

SUMMARY:`C(string 2) * C(string 1) -> C(string 2)`**EXPLANATION:**

Same as for MP3D except that the product is stored using YC2, TN2, S2 and, if S2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding is specified), OFF
Exponent Overflow	If exponent of floating-point result is > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128, then ON; otherwise, unchanged
Overflow	If data is lost in most significant positions then ON; otherwise, unchanged

MP2D

NOTES:

1. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.
2. An Illegal Procedure fault occurs if
 - a) DU or DL modification is specified for MF1 or MF2, or if illegal repeats are used;
 - b) Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.

The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

EXAMPLES:

1	8	16	32

	MP2D	,,1,1	rounding and plus sign options
	NDSC9	FLD1,0,4,2,-3	multiplier operand descriptor
	NDSC4	FLD2,0,8,1,-2	multiplicand operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4A2+	0 0 2 +
FLD2	EDEC	8P+1234567	+1234567
	USE	+0002469	(Product)
*			indicators on? none
	MP2D	,,1	rounding option
	NDSC4	FLD1,0,8,3,-2	multiplier operand descriptor
	NDSC4	FLD2,0,8	multiplicand operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P10	00000010
FLD2	EDEC	8P+123.45	+12345-2
	USE	+12345-3	(Product)
*			indicators on? none

MP2DX

11.2.4 MP2DX

MP2DX	Multiply Using Two Decimal Operands Extended	246(1)
-------	--	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35	
C	N	0000000				T	R	D	MF2	246(1)		I	MF1
S	S												

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1				CN1	T	N	SX1	SF1	N1			
AR#	Y1				1							

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2				CN2	T	N	SX2	SF2	N2			
AR#	Y2				2							

CODING FORMAT:

1	8	16	

	MP2DX	(MF1), (MF2), RD, CS, T, NS	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

MP2DX

SUMMARY:

`C(string 2) * C(string 1) -> C(string 2)`

EXPLANATION:

Same as for MP3DX except that the product is stored using YC2, TN2, SX2 and, if SX2 indicates a scaled format, SF2.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for MP2D

NOTES:

1. See the Notes for the MP3D instruction.
2. See MVNX for information about coding of overpunched signs.

MP3D

11.2.5 MP3D

MP3D	Multiply Using Three Decimal Operands	226(1)
------	---------------------------------------	--------

FORMAT:

00 0102 08 0910 11 17 18 27 28 29 35

P	0	MF3	T	R	D	MF2	226(1)	I	MF1
---	---	-----	---	---	---	-----	--------	---	-----

00 0203 17 18 20 21 22 23 24 29 30 35

Y1				CN1	T N 1	S1	SF1	N1
AR#	Y1							

00 0203 17 18 20 21 22 23 24 29 30 35

Y2				CN2	T N 2	S2	SF2	N2
AR#	Y2							

00 0203 17 18 20 21 22 23 24 29 30 35

Y3				CN3	T N 3	S3	SF3	N3
AR#	Y3							

CODING FORMAT:

The MP3D instruction code is shown below.

1	8	16	32

MP3D	(MF1) , (MF2) , (MF3) , RD , P , T		
NDSC _n	LOCSYM , CN , N , S , SF , AM		
NDSC _n	LOCSYM , CN , N , S , SF , AM		
NDSC _n	LOCSYM , CN , N , S , SF , AM		

MP3D

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

$C(\text{string } 2) * C(\text{string } 1) \rightarrow C(\text{string } 3)$

EXPLANATION:

The decimal number of data type TN2, sign and decimal type S2, and starting location YC2, is multiplied by the decimal number of data type TN1, sign and decimal type S1, and starting location YC1. The product is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least significant truncation. In this case, the most-significant-digit of the mantissa (except for the sign digit) is set to a number digit other than 0.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is a 1, rounding takes place prior to storage.

Provided that string 1, string 2, and string 3 are not overlapped, the contents of strings 1 and 2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If, in the preparation of the final result, one or more least-significant-digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least-significant digits lost or rounding specified), OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If data is lost in most-significant positions, then ON; otherwise, unchanged

NOTES:

1. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.
2. An Illegal Procedure fault occurs if:
 - a) DU or DL modification is specified for MF1, MF2, or MF3, or if illegal repeats are used;
 - b) any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor;
 - c) the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

MP3D

EXAMPLES:

1	8	16	32

	MP3D	, , , 1	with rounding option
	NDSC4	FLD1,6,2,2	multiplier operand descriptor
	NDSC4	FLD2,0,8,1,-3	multiplicand op descriptor
	NDSC9	FLD3,1,7,1,-2	product operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P5+	0000005+
FLD2	EDEC	8P+1234567	+1234567
FLD3	BSS	2	+617284 (Product)
	USE		indicators on? none
	MP3D	, , , , 1	
	NDSC4	FLD1,0,2,3,-2	multiplier operand descriptor
	NDSC4	FLD2,0,8,1,-3	multiplicand op descriptor
	NDSC4	FLD3,1,7	product operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	2PL25	25000000
FLD2	EDEC	8P-1234567	-1234567
FLD3	EDEC	8P+0	+ -3086-1 (Product)
	USE		instruction fault? no
*			indicators on? truncation and negative

MP3DX

11.2.6 MP3DX

MP3DX	Multiply Using Three Decimal Operands Extended	266(1)
-------	--	--------

FORMAT:

00 0102 08 0910 11 17 18 27 28 29 35

C S	N S	MF3	T	R D	MF2	266(1)	I	MF1
--------	--------	-----	---	--------	-----	--------	---	-----

00 0203 17 18 20 21 22 23 24 29 30 35

Y1		CN1	T N 1	SX1	SF1	N1
AR#	Y1					

00 0203 17 18 20 21 22 23 24 29 30 35

Y2		CN2	T N 2	SX2	SF2	N2
AR#	Y2					

00 0203 17 18 20 21 22 23 24 29 30 35

Y3		CN3	T N 3	SX3	SF3	N3
AR#	Y3					

CODING FORMAT:

1	8	16	

	MP3DX	(MF1), (MF2), (MF3), RD, CS, T, NS	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

MP3DX

OPERATING MODES:

Any

SUMMARY:

`C(string 2) * C(string 1) -> C(string 3)`

EXPLANATION:

The decimal number of data type TN2, sign and decimal type S2, and starting location YC2, is multiplied by the decimal number of data type TN1, sign and decimal type S1, and starting location YC1. The product is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If SX3 indicates a fixed-point format, the results are stored using SF3, which may cause leading or trailing zeros (4 bits - 0000,

9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most-significant-nonzero digits even if this causes least-significant truncation. In this case, the most-significant digit of the mantissa (except for the sign digit) is set to a number digit other than 0.

The character set is defined by CS. Placement of over punched sign in the output is controlled by NS. (Refer to introductory pages of this section for definition of NS.) If RD is a 1, rounding takes place before storage.

Provided that string 1, string 2, and string 3 are not overlapped, the contents of strings 1 and 2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for MP3D

MP3DX

NOTES:

1. See the notes for the MP3D instruction.
2. See MVNX for information about coding of overpunched signs.

MPF

11.2.7 MPF

MPF	Multiply Fraction	401(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

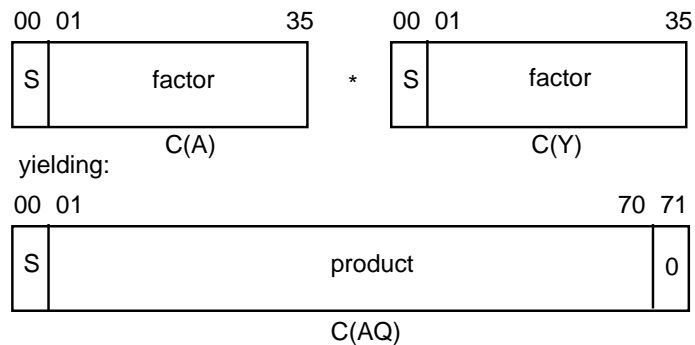
SUMMARY:

$C(A) * C(Y) \rightarrow C(AQ)$, left adjusted; $C(Y)$ unchanged

EXPLANATION:

This instruction multiplies two 36-bit fractional factors (including sign) to form a 71-bit fractional product (including sign). The product is stored in AQ, left-justified. Bit 71 of $C(AQ)$ is filled with a zero bit.

Overflow can occur only when A and Y both = $2^{*}35+1$ and the result exceeds the range of the AQ-register.



ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If bit 0 of $C(AQ) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON

NOTE:

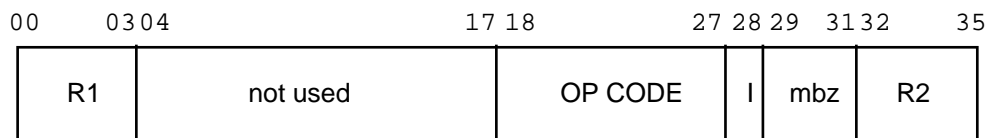
An Illegal Procedure fault occurs if illegal address modification is used.

MPRR

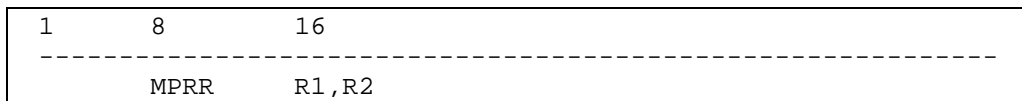
11.2.8 MPRR

MPRR	Multiply Register Pair by Register	530(1)
------	------------------------------------	--------

FORMAT:



CODING FORMAT:



OPERATING MODES:

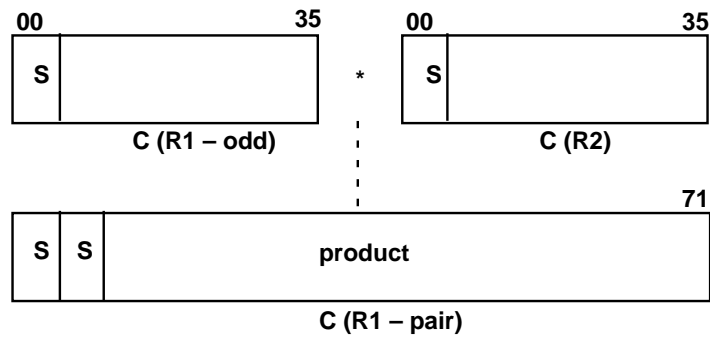
Executes only in ES/EI mode

SUMMARY:

For R1-odd R1: 1, 3, 5, 7, Q;
 For R1-pair R1: 0, 2, 4, 6, A:
 C(R1-odd) * C(R2) -> C(R1-pair)
 C(R2) unchanged

MPRR**EXPLANATION:**

A register pair is specified in R1. The product of the content of the odd-numbered register (Q if A,Q specified) and that of R2 is taken and the result is loaded, right-justified into the R1-pair.

**ILLEGAL ADDRESS MODIFICATIONS:**

None

The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1-pair) = 0, then ON; otherwise, OFF

Negative If C(R1-pair)(0) = 1, then ON; otherwise, OFF

MPRR

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

MPRS**11.2.9 MPRS**

MPRS	Multiply Single Register by Register	531(1)
------	--------------------------------------	--------

FORMAT:

00	0304	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	

	MPRS	R1, R2	

OPERATING MODES:

Executes in ES/EI mode only

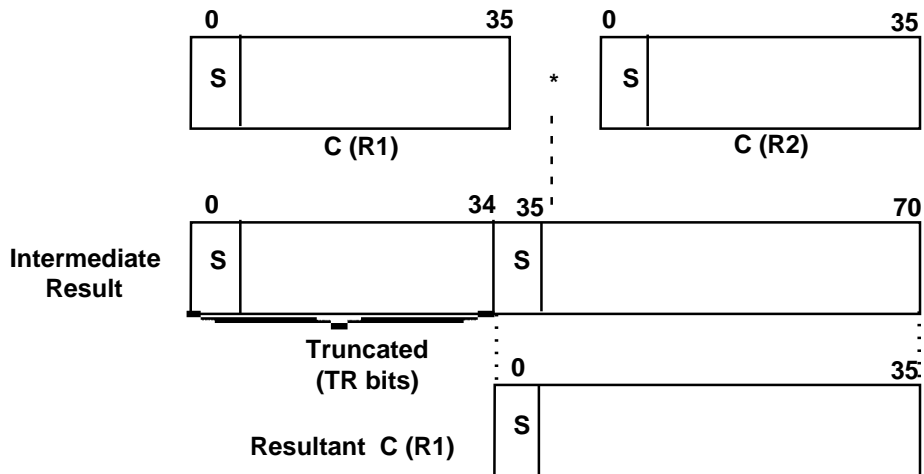
SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q
 $C(R1) * C(R2) \rightarrow C(R1)$
 $C(R2)$ unchanged

MPRS

EXPLANATION:

The product of the content of R1 and that of R2 is taken. The low-order 36 bits of the result are loaded into R1.



The multiplication is performed on the two's complement data to obtain 71-bit two's complement data as an intermediate result. The low-order 36 bits of this intermediate result are loaded into R1.

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If the intermediate result is 0, then ON; otherwise OFF
Negative	If the intermediate result(0) is 1, then ON, otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.
3. No overflow check for the final result is performed; therefore, the Zero and Negative indicators are set by the state of the intermediate result.

MPX

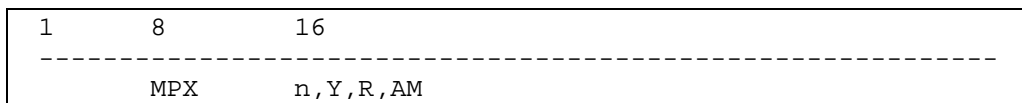
11.2.10 MPX

MPX	Multiply $G\bar{X}_n$	$04\bar{n}(1)$
-----	-----------------------	----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:



OPERATING MODES:

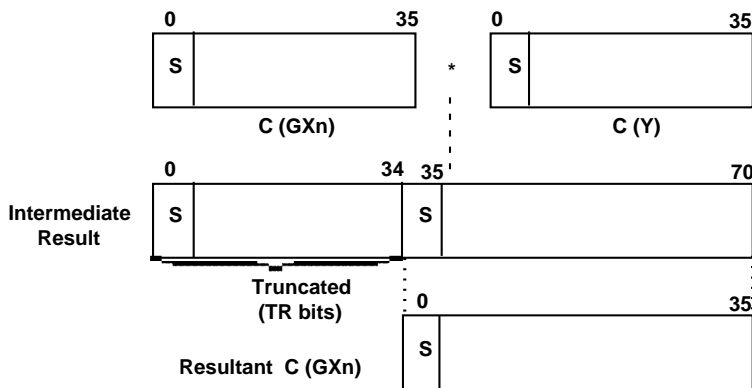
Executes in ES/EI mode only

SUMMARY:

$$C(G\bar{X}_n) * C(Y) \rightarrow G\bar{X}_n$$

EXPLANATION:

The product of the content of $G\bar{X}_n$ and that of the one word at memory location Y is taken. The low-order 36 bits of the result is loaded into $G\bar{X}_n$ as illustrated below.



MPX

The multiplication is performed on the two's complement data to obtain 71-bit two's complement data as an intermediate result. The low-order 36 bits of this intermediate result are loaded into the GXn.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None, except for Multiply GX0

ILLEGAL EXECUTES:

If the instruction is executed in NS mode

INDICATORS:

Zero	If the intermediate result is 0, then ON; otherwise OFF
Negative	If the (intermediate result)(0) is 1, then ON; otherwise OFF.

NOTES:

1. An IPR fault occurs if illegal address modification are used or if the instruction is executed in NS mode.
2. No overflow check for the final result is performed, therefore, the Zero and Negative indicators are set by the state of the intermediate result.

MPY

11.2.11 MPY

MPY	Multiply Integer	402(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

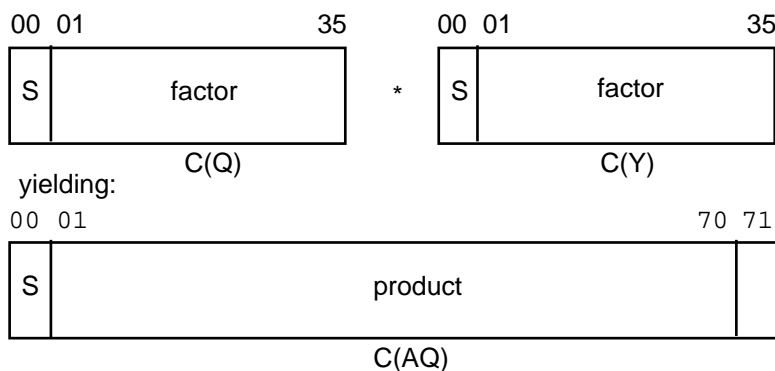
Any

SUMMARY:

$C(Q) * C(Y) \rightarrow C(AQ)$, right adjusted; $C(Y)$ unchanged

EXPLANATION:

This instruction multiplies two 36-bit integral factors (including sign) to form a 71-bit integral product (including sign). The product is stored in AQ, right-justified. Bit 0 of C(AQ) is filled with an "extended sign" bit.



When $(-2^{35}) * (-2^{35}) = +2^{70}$, bit 1 of AQ is used to represent the product rather than the sign and no overflow occurs.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If bit 0 of $C(AQ) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

MRL

11.2.12 MRL

MRL	Move Alphanumeric Right to Left	101(1)
-----	------------------------------------	--------

FORMAT:

00	08 09 10 11	17 18	27 28 29	35
FILL	T 0	MF2	101 (1)	I MF1

00 02 03	17 18	20 21 22 23 24	31 32 35
Y1	CN1	TA1 0	N1
AR#	Y1		R1

00 02 03	17 18	20 21 22 23 24	31 32 35
Y2	CN2	TA2 0	N2
AR#	Y2		R2

CODING FORMAT:

The MRL instruction code is shown below.

1	8	16

MRL	(MF1) , (MF2) , FILL , T	
ADSC _n	LOCSYM , CN , N , AM	
ADSC _n	LOCSYM , CN , N , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) -> C(string 2)

EXPLANATION:

This instruction is identical with MLR except that the starting locations are YC1 + (L1-1) and YC2 + (L2-1) and the movement is from right to left (from least significant character toward most significant character). Consequently, any truncation or fill is of the most significant characters.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Truncation: If L1 is > L2, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if illegal repeats are used.
2. A Truncation fault occurs if the Truncation indicator is set and the truncation fault enable (T) bit is a 1.
3. Refer to Note 3 of the MLR instruction for information on string replication.
4. L2 = 0 does not necessarily mean that the instruction functions as a no-op because the truncation indicator may be affected.

MRL

EXAMPLE:

1	8	16	32
	MRL	,,20	move with blank fill
	ADSC6	FLD1,,12	sending descriptor
	ADSC6	FLD2,4,14	receiving descriptor
	USE	CONST.	memory contents
FLD1	BCI	2,ABCDEFGHIJKL	
FLD2	BSS	3	xxx 00 ABCDEFGHIJKL (Result)
	USE		
	MRL	,,400	move with sign and fill
	ADSC6	FLD1,3,9	sending descriptor
	ADSC4	FLD2, move with	sign and fill
	ADSC6	FLD1,3,9	sending descriptor
	ADSC4	FLD2, ,12	receiving descriptor
	USE	CONST.	memory contents
FLD1	BCI	2, 00 12345678R	
FLD2	BSS	2	xxx-00123456789 (Result)
	USE		

MTM**11.2.13 MTM**

MTM	Move to Memory	365(1)
-----	----------------	--------

FORMAT:

00	13 14	17 18	27 28 29	35
not used	RECR	365(1)	I	MF1

00	02 03	17 18	20 21 22 23	32 33	35
AR	Y	CN	B		L

CODING FORMAT:

The MTM instruction code is shown below.

1	8	16

MTM	(MF1)	, RECR
SDSC	Y, CN, L,	, AM

EXPLANATION:

This instruction moves one, two, three, or four 9-bit characters into memory from the register specified in the RECR field of the instruction. MTM is the inverse of MTR.

The move from the register into memory is done from right to left beginning at YCN + (L-1). (L must be 0-4.)

MTM

The setting of the B field shown in the descriptor diagram above, is determined by the contents of the n in SDSCn. (A 9 in the n field sets B = 0; an 8 sets B = 1.) This setting determines the functions of the move operation:

- if B = 0 The 9-bit characters are fetched at once from the specified register and moved into memory without modification.
- if B = 1 8-bits (1 byte) are fetched from the specified register and 0 is concatenated to the most significant bit position to form a 9-bit character. Then the character is moved to memory. Up to L characters can be moved.

An A, Q, or X0-X7, GX0-GX7 register may be specified in the RECR field.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL specified in MF

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

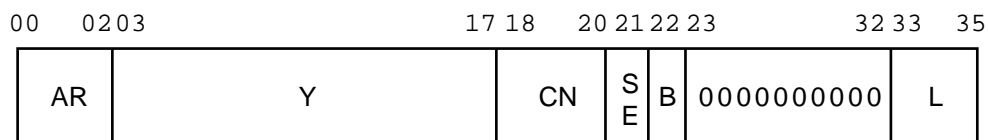
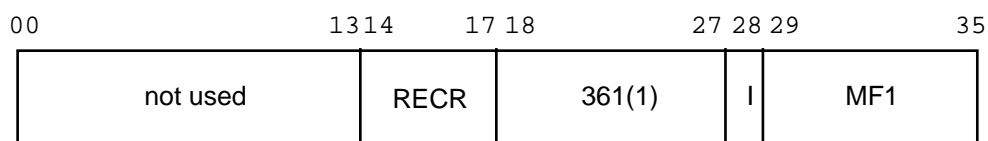
None affected

NOTES:

1. Refer to "Character Move To/From Register Instructions" in Section 7 for a description of the fields in the operand descriptor (SDSC).
2. An IPR fault occurs under the following conditions:
 If RECR specifies X0-X7 and $L > 2$. (X0-X7 can only hold 2 bytes.);
 If RECR specifies A or Q or GX-GX7 and $L > 4$;
 If illegal address modifications or illegal repeats are used.
3. The RL bit of the MF field is ignored. The character length must be specified in the L field of the operand descriptor.
4. When $L = 0$, the MTM instruction functions as a NOP.
5. Refer to Explanation under the MTR instruction for the codes allowed in the RECR field.

MTR**11.2.14 MTR**

MTR	Move to Register	361(1)
-----	------------------	--------

FORMAT:**CODING FORMAT:**

The MTR instruction code is shown below.

1	8	16	32

MTR	(MF1)	RECR	
SDSCn	Y	CN	L, SE, AM

EXPLANATION:

This instruction moves one, two, three, or four 9-bit characters from the memory location beginning at YCN + (L-1) to a register specified by the RECR field (bits 14-17) of the instruction word. MTR is the inverse of MTM.

The moved characters are right-justified in the specified register.

The setting of the B field shown in the descriptor diagram above, is determined by the contents of the n in SDSCn. (SDSC9 sets B = 0; SDSC8 sets B = 1.) The SE field is specified by the user. These settings determine the character positioning functions of the move operation as follows.

MTR

If B = 0 The 9-bit characters from memory are moved to the specified register without modification. If L is less than the character size capacity of the specified register, the vacant high-order character positions of the register are filled in the following way.

SE = 0 The remaining character positions are filled with 0.

SE = 1 Bit 0 of the last character moved is regarded as a sign and the value of this bit is extended to fill the remaining character positions of the register.

If B = 1 Bit 0 of each 9-bit character moved from memory is removed and the resulting 8-bit bytes are moved in a right-justified string into the specified register. The SE field affects the result of the move in the following way.

SE = 0 The remaining bit positions of the specified register are filled with 0.

SE = 1 Bit 0 of the last 8-bit byte moved to the specified register is extended to fill the remaining high-order bits of the register.

An A, Q, or X0-X7, GX0-GX7 register may be specified in the RECR field. The code of these registers is the same as for the register code specified in the REG portion of the MF field. An invalid specification results in an IPR fault.

The RECR codes are displayed below.

RECR Code	Register	
	(NS Mode)	(ES/EI Mode)
0000	<IPR>	<IPR>
0001	A	A
0010	Q	Q
0011	<IPR>	<IPR>
0100	<IPR>	<IPR>
0101	<IPR>	<IPR>
0110	<IPR>	<IPR>
0111	<IPR>	<IPR>
1000	X0	GX0
1001	X1	GX1
1010	X2	GX2
1011	X3	GX3
1100	X4	GX4
1101	X5	GX5
1110	X6	GX6
1111	X7	GX7

The number of characters to be moved is specified in the L field of the operand descriptor.

INDICATORS:

Zero	ON if C(register) = 0; otherwise, OFF
Negative	ON if bit 0 of C(register) = 1; otherwise, OFF

NOTES:

1. Refer to "Character Move To/From Register Instructions" in Section 7 for a description of the fields in the operand descriptor (SDSC).
2. An IPR fault occurs under the following conditions.
If RECR specifies X0-X7 and $L > 2$. (X0-X7 can only hold 2 bytes.)
If RECR specifies A or Q or GX-GX7 and $L > 4$.
If illegal address modifications or illegal repeats are used.
3. The RL bit of the MF field is ignored. The character length must be specified in the L field of the operand descriptor.
4. If $L = 0$, the contents of the receiving register is set to 0, the Zero indicator to ON, and the Negative indicator to OFF.

MVE

11.2.15 MVE

MVE	Move Alphanumeric Edited	020(1)
-----	--------------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
0	0		MF3	T	0		MF2		020(1)	I		MF1

00	02	03	17	18	20	21	22	23	24	29	30	32	35
Y1						CN1	TA1	0	not interpreted			N1	
AR#	Y1											R1	

00	02	03	17	18	20	21	22	23	24	29	30	32	35
Y2						CN2	TA2	0	not interpreted			N2	
AR#	Y2											R2	

00	02	03	17	18	20	21	22	23	24	29	30	32	35
Y3						CN3	TA3	0	not interpreted			N3	
AR#	Y3											R3	

CODING FORMAT:

The MVE instruction code is shown below.

1	8	16

MVE	(MF1), (MF2), (MF3)	
ADSC _n	LOCSYM, CN, N, AM	
ADSC9	LOCSYM, CN, N, AM	
ADSC _n	LOCSYM, CN, N, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) $\xrightarrow{\text{string 2 control}}$ C(string 3)

EXPLANATION:

Starting at location YC1, the string of alphanumeric characters of data type TA1 is moved to the string of alphanumeric characters of data type TA3 starting at location YC3. The move is under control of the micro operation sequence of length L2 and type TA2 = 00 that starts at location YC2. Refer to "Micro Operations" in this section.

Maximum allowable length for L1, L2, and L3 is 63; they are not checked for length greater than 63. Only the rightmost six bits (30-35) are interpreted for length. Likewise, when a register is specified as containing the length, only the rightmost six bits of the register are interpreted.

The operation stops when L3 is exhausted.

The result is unpredictable when strings are overlapped.

The contents of the alphanumeric character string that starts at YC1 and the micro operation sequence that starts at YC2 remain unchanged.

On the processor, L3 = 0 is the normal termination; thus, at the start of the instruction, if L3 = 0 and there are no faults (see Note), no operation is performed and the instruction terminates normally, independently of whether L1 or L2 equals zero, because the hardware does not access these fields when L3 = 0.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

MVE

NOTES:

1. An Illegal Procedure fault occurs under the following conditions:
 - a. If DU or DL modification is used for MF1, MF2, or MF3
 - b. If illegal repeats are used
 - c. If an illegal micro operation is executed. Refer to "micro operations" in this section for additional information
 - d. If TA2 is not = 0
 - e. If an attempt is made to access string 2 when L₂ = 0.
2. Refer to "Micro Operations for Edit Instructions" in Section 7.

EXAMPLES:

1	8	16	32

	MVE		move alphanumeric edited
	ADSC6	FLD1,2,20	
	ADSC9	FLD2,0,35	
	ADSC6	FLD3,0,30	
	USE	CONST.	
FLD1	BCI	4,12	SMITHROGERWILLIAMS25AB
FLD2	MICROP	(CHT,0),8H*,.-	bbbb, (SES,8),(INSB,1),(INSB,5)
	MICROP	(MVC,10),(INSB,2)	(INSB,5),(MVC,7)
	MICROP	(INSB,5),(MVC,1)	(INSB,3),(INSB,5)
	MICROP	(INSB,4),(INSB,5)	(INSB,0),1H#,(MVC,2)
	*		

* The following table explains the above micro-operation sequence:			
*	(CHT,0),8H*,.-	Change edit table to these 8 characters	
*	(SES,8)	- Set End Suppression Flag ON	
*	(INSB,1)	- Insert Edit Table entry #1 (*)	
*	(INSB,5)	- Insert Edit Table entry #5 (b)	
*	(MVC,10)	- Move 10 characters from FLD1 (SMITHROGER)	
*	(INSB,2)	- Insert Edit Table entry #2 (,)	
*	(INSB,5)	- Insert Edit Table entry #5 (b)	
*	(MVC,7)	- Move 7 characters from FLD1 (WILLIAM)	
*	(INSB,5)	- Insert Edit Table entry #5 (b)	
*	(MVC,1)	- Move 1 character from FLD1 (S)	
*	(INSB,3)	- Insert Edit Table entry #3 (.)	
*	(INSB,5)	- Insert Edit Table entry #5 (b)	
*	(INSB,4)	- Insert Edit Table entry #4 (-)	
*	(INSB,5)	- Insert Edit Table entry #5 (b)	
*	(INSB,0),1H#	- Insert specified character (#)	
*	(MVC,2)	- Move 2 characters from FLD1 (25)	
* Memory contents in BCD characters			
FLD	BSS	5	SMITHROGER,WILLIAM\$.#25
	USE		
	MVE		move alphanumeric edited
	ADSC9	FLD1,0,7	sending field operand descriptor
	ADSC9	FLD2,0,6	micro-op string operand descriptor
	ADSC9	FLD3+1,1,7	receiving field operand descriptor
	USE	CONST.	
FLD1	ASCII	2,ERROR-2	
FLD2	MICROP	(LTE,1),1A#,(MVC,5),(INSM,1),(IGN,1),(MVC,1)	
* memory content in ASCII characters			
FLD3	ASCII	3,CODE	codeerror#2 (Result)
	MVE		
	ADSC9	RDWRK,2,6	
	ADSC9	MOPSC,0,11	
	ADSC9	A9,1,7	
	MVT		
	ADSC9	A9,1,7	
	ADSC9	A,1,7	NDSC9 A,1,7,2
	ARG	TABLE-12	
	USE	CONST.	
MOPSC	MICROP	(LTE,3),10000,(LTE,4),10100	
	MICROP	(MSES,6),(LTE,3),1A+,(LTE,4),1A-,(SES),(ENF)	
	OCT	000000000053,000055000000	05X
	OCT	060061062063,064065066067	06X
	OCT	070071000000,000000000000	07X
	OCT	000000000000,000000000000	10X
	OCT	000000061062,063064065066	11X
	OCT	067070071000,000000000000	12X
	OCT	000000000000,000000060000	13X
	OCT	000000000000,000000000000	14X
	OCT	000000061062,063064065066	15X
	OCT	067070071000,000000000000	16X
	OCT	000000000000,000000000000	05X
	USE		

MVN

11.2.16 MVN

MVN	Move Numeric	300(1)
-----	--------------	--------

FORMAT:

00	01	08	09	10	11	17	18	27	28	29	35
P	00000000	T	R D	MF2			300(1)			I	MF1

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1				CN1	T N 1	S1	SF1	N1				
AR#	Y1											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2				CN2	T N 2	S2	SF2	N2				
AR#	Y2											

CODING FORMAT:

The MVN instruction code is shown below.

1	8	16

MVN	(MF1), (MF2), RD, P, T	
NDSC _n	LOCSYM, CN, N, S, SF, AM	
NDSC _n	LOCSYM, CN, N, S, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) -> C(string 2)

EXPLANATION:

Starting at location YC1, the decimal number of data type TN1 and sign and decimal type S1 is moved, properly scaled, to the decimal number of data type TN2 and sign and decimal type S2 that starts at location YC2.

If S2 indicates a fixed-point format, the results are stored as L2 digits using scale factor SF2, and thereby may cause most-significant-digit overflow and/or least-significant-digit truncation.

If P = 1, positive signed 4-bit results are stored using octal 13 as the plus sign. Rounding is legal for both fixed-point and floating-point formats. If P = 0, positive signed 4-bit results are stored using octal 14 as the plus sign.

Provided that string 1 and string 2 are not overlapped, the contents of the decimal number that starts in location YC1 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

MVN

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If least significant truncation without rounding, then ON; otherwise, OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If fixed point integer overflow, then ON; otherwise, unchanged

NOTES:

1. Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is 1.
2. An Illegal Procedure fault occurs if:
 - DU or DL modification is specified for MF1 or MF2, or if illegal repeat is used;
 - Any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor;
 - The values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
3. Refer to Explanation of the MLR instruction for information on string replication.
4. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

MVN**EXAMPLES:**

1	8	16	32
	MVN	,,1	with rounding option
	NDSC4	FLD1,0,8,2,-3	sending field operand descr.
	NDSC4	FLD2,1,7,1,-2	receiving field operand descr.
	USE	CONST.	memory contents
FLD1	EDEC	8P1234567+	1 2 3 4 5 6 7 +
FLD2	EDEC	8P0	0 + 1 2 3 4 5 7 (Result)
	USE		no indicators set ON
	MVN	,,,,1	truncation fault enable option
	NDSC9	FLD1,3,9,2,-2	sending field operand descr.
	NDSC4	FLD2,0,8,0	receiving field operand descr.
	USE	CONST.	memory contents
FLD1	EDEC	12A12345678-	0 0 0 1 2 3 4 5 6 7 8 -
FLD2	BSS	1	- 1 2 3 4 5 + 1 (Result)
	USE		negative and truncation set ON

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32
	EAX1	1	load character address into X1
	EAX2	2	load address modifier into X2
	EAX7	7	load FLD1 length into X7
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	MVN	(1,1,,1),(,,1),1,1	-
*			with rounding and plus sign options
*	NDSC9	0,,X7,2,-2,4	FLD1's operand descriptor (FLD1,1,7,2,-2)
	ARG	FLD2+1	pointer to indirect op. descr.
	USE	CONST.	memory contents
FLD1	EDEC	8A123456+	0 1 2 3 4 5 6 +
FLD2	EDEC	8P0	0 0 0 0 1 2 3 5 (Result)
	NDSC4	FLD2,2,6,3,-2	receiving field indirect operand descriptor
*			operand descriptor
	USE		no indicators set ON

MVNE

11.2.17 MVNE

MVNE	Move Numeric Edited	024(1)
------	---------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
0	0	MF3	0	0	MF2	024 (1)			I	MF1		

00	02	03	17	18	20	21	22	23	29	30	32	35
Y1						CN1	T N 1	S 1	not interpreted		N1	
AR#	Y1								not interpreted		R1	

00	02	03	17	18	20	21	22	23	29	30	32	35
Y2						CN2	T A 2	0	not interpreted		N2	
AR#	Y2								not interpreted		R2	

00	02	03	17	18	20	21	22	23	29	30	32	35
Y3						CN3	T A 3	0	not interpreted		N3	
AR#	Y3								not interpreted		R3	

CODING FORMAT:

The MVNE instruction code is shown below.

1	8	16

MVNE	(MF1) , (MF2) , (MF3)	
NDSC _n	LOCSYM , CN , N , S , , AM	
ADSC9	LOCSYM , CN , N , AM	
ADSC _n	LOCSYM , CN , N , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) $\xrightarrow{\text{string 2 control}}$ (string 3)

EXPLANATION:

Starting at location YC1, the string of numeric characters of data type TN1 is moved to the string of alphanumeric characters of data type TA3 starting at location YC3. The move is under control of the micro-operation sequence of length L2 and type TA2 = 00 that starts at location YC2. Refer to "Micro Operations" in this Section.

Maximum allowable length for L1, L2, and L3 is 63; they are not checked for length greater than 63. Only the rightmost 6 bits (30-35) are interpreted for length. When a register is specified as containing the length, only the rightmost 6 bits of the register are interpreted.

The operation stops when L3 is exhausted.

The results are not guaranteed when strings are overlapped.

The sign and decimal type of the sending field is given by S1. The contents of the numeric character string that starts at YC1 and the micro-operation sequence that starts at YC2 remain unchanged.

On the processor, L3 = 0 is the normal termination; thus, at the start of the instruction, if L3 = 0 and there are no faults (see Note 1), no operation is performed and the instruction terminates normally, independently of whether L1 or L2 equals zero, because the hardware does not access these fields when L3 = 0.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

MVNE

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs under the following conditions:
 - a. If DU or DL modification is used for MF1, MF2, or MF3;
 - b. If illegal repeats are used;
 - c. If an illegal micro operation is executed. Refer to "Micro Operations" in this section for additional information;
 - d. If TA2 is not = 0;
 - e. If an attempt is made to access string 2 when L₂ = 0.
2. Refer to Micro Operations for Edit Instructions in Section 7.

EXAMPLES:

1	8	16	32
	MVNE		with (\$) float and (.) inserted
	NDSC9	FLD1,0,10,2	sending field oper. descriptor
	ADSC9	FLD2,0,14	micro-op string oper. descriptor
	ADSC6	FLD3,0,12	receiving field oper. descriptor
	USE	CONST.	memory contents in ASCII char.
FLD1	EDEC	10A300405-	000300305-00
FLD2	MICROP	(CHT,0),8Hb*+-\$,.0,(MFLC,7),(ENF,8),(INSB,7)	
FLD3	MICROP	(MVC,2),(INSN,4)	memory contents in BCD char.
	BSS	2	000 \$3004.05- (Result)
	USE		
*			
	MVNE		with (*) protection & (.) insertion
	NDSC4	FLD1,0,8,2	sending field oper. descriptor
	ADSC9	FLD2,0,6	micro-op string oper. descr.
	ADSC9	FLD3,0,12	receiving field oper. descr.
	USE	CONST.	memory contents, packed decimal
FLD1	EDEC	8P250509-	025059-
FLD2	MICROP	(MVZA,5),(SES,8),INSA,7),(MVC,2)	
	MICROP	(INSN,4),(INSM,3)	
			memory contents, ASCII char.
FLD3	BSS	3	*2505,09- 000 (Result)
	USE		
*			
	MVNE		+1234 —> 1234
	NDSC4	6PACK,3,5,1	-1234 —> 123M
	ADSC9	MOPS,0,6	
	ADSC6	PRTOUT,0,4	
	MVT		
	ADSC6	PRTOUT,0,4	
	ADSC9	APRINT,0,4	
	ARG	TABLE	
	USE	CONST.	
MOPS	MICROP	(MVC,3),(LTE,3)m10000,(LTE,4),10040,(MORS,1)	
TABLE	ASCII	2,1234567	0X
	VFD	A18/89,18/0,36,0	1X
	OCT	0,0	2X
	OCT	0,0	3X
	UASCI	2,JKLMNQP	4X
	VFD	U18/QR,18/0,36,0	5X
	OCT	0,0	6X
	OCT	0,0	7X
	USE		

MVNEX

11.2.18 MVNEX

MVNEX	Move Numeric Edited Extended	004(1)
-------	------------------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
E	MF3	0	0	MF2	004 (1)			I	MF1			

00	02	03	17	18	20	21	22	23	29	30	32	35
Y1						CN1	T N 1	S 1	not interpreted	N1		
AR#	Y1									R1		

00	02	03	17	18	20	21	22	23	29	30	32	35
Y2						CN2	T A 2	0	not interpreted	N2		
AR#	Y2									R2		

00	02	03	17	18	20	21	22	23	29	30	32	35
Y3						CN3	T A 3	0	not interpreted	N3		
AR#	Y3									R3		

CODING FORMAT:

1 8 16

 MVNEX (MF1) , (MF2) , (MF3) , E
 NDSCn LOCSYM , CN , N , S , , AM
 ADSC9 LOCSYM , CN , N , AM
 ADSCn LOCSYM , CN , N , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

MVNEX**SUMMARY:**

```

C(string 1)  $\xrightarrow{\text{string 2 control}}$  (string 3)

```

EXPLANATION:

The function of this instruction is similar to the MVNE instruction, but with the added capability of initializing an edit insertion table. (See Table 7-2). A 2-bit code entered in field E (bits 0 and 1) specifies the character set associated with the edit insertion table.

<u>E-bits 0,1</u>	<u>Character Set</u>
00	EBCDIC
01	BCD
10	ASCII
11	Illegal, IPR fault

TN1 determines whether the input data is unpacked (0) or packed (1). TA3 determines the character size (9, 6, or 4 bits) of the output data. It is the user's responsibility to make TA3 consistent with bits 0 and 1 of the instruction. S determines the location of the sign of the input data (leading, trailing, unsigned, separate). Refer to the Explanation for MVNE for additional information.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

MVNEX

NOTES:

1. Notes for MVNE apply to MVNEX.
2. An Illegal Procedure fault occurs if DU or DL modifications are specified for MF1, MF2, or MF3, or if illegal repeats are used.
3. Refer to "Micro Operations for Edit Instructions" in Section 7.
4. Only "S" sign types are supported, i.e., no overpunch.

MVNX

11.2.19 MVNX

MVNX	Move Numeric Extended	340(1)
------	-----------------------	--------

FORMAT:

00	01	02	08	09	10	11	17	18	27	28	29	35
C	N	00000000	T	R	D	MF2	340(1)			I	MF1	
S	S											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y1			CN1	T N 1	SX1	SF1	N1					
AR#	Y1											

00	02	03	17	18	20	21	22	23	24	29	30	35
Y2			CN2	T N 2	SX2	SF2	N2					
AR#	Y2											

CODING FORMAT:

1	8	16

MVNX	(MF1), (MF2), RD, CS, T, NS	
NDSCn	LOCSYM, CN, N, SX, SF, AM	
NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

C(string 1) -> C(string 2)

MVNX**EXPLANATION:**

Starting at location YC1, the decimal number of data type TN1 and sign and decimal type SX1 is moved, properly scaled, to the decimal number of data type TN2 and sign and decimal type SX2 that starts at location YC2.

The character set is defined by CS (EBCDIC/ASCII). Placement of an overpunched sign in the output is controlled by NS. (Refer to the definition of the NS field in the beginning of Section 8.)

If SX2 indicates a fixed-point format, the result is stored as L2 digits using scale factor SF2, and thereby may cause most-significant-digit overflow and/or least-significant-digit truncation.

Rounding is legal for both floating and scaled formats. The contents of the decimal number that starts in location YC1 remain unchanged.

The SX field is interpreted in the following way.

TN = 0 : Unpacked data (9 bits)

SX

00: LS*, OVP*, scaled
 01: LS, separate, scaled
 10: TS*, separate, scaled
 11: TS, OVP, scaled

TN = 1 : Packed data (4 bits)

SX

00: LS, separate, floating-point
 01: LS, separate, scaled
 10: TS, separate, scaled
 11 : No sign, scaled

* LS.... Leading sign
 OVP... Overpunched
 TS.... Trailing sign

Bits 0 and 1 of the instruction word are interpreted this way:

Bit 0 of instruction word (CS): specifies the character set

=0: EBCDIC data (but not the strict EBCDIC sign)

=1: ASCII data (but not the strict ASCII sign)

Bit 1 of instruction word (NS): specifies no-sign output

=0: The instruction execution is not affected.

=1: The sign character in the receive field where the result is to be placed is affected this way:

If the operand descriptor of the receive field contains TN = 0 and SX = 00 or 11 (indicating that output is an overpunched sign), the overpunched sign is not placed in the specified field. Instead, an appropriate decimal number (0-9) is placed in the receive field irrespective of whether the sign of the calculated result is positive or negative, which is a no-sign output.

For values of SX and TN, bit 1 is ignored, which applies to both EBCDIC and ASCII.

The hardware recognizes an implied plus sign on input data. For unpacked data (TN=0) with indicated overpunched sign (SX1 = 00 or 11), if the hardware does not find a plus or minus overpunched sign character in the overpunched sign character position, the hardware checks for a numeric digit (0-9). The zone bits are not included in the check; only the lower-order 4 bits are checked. If this check indicates a numeric digit from the appropriate character set, the hardware accepts the digit and assumes the sign to be plus. Otherwise, an IPR fault is generated.

MVNX

The following table shows the character codes for ASCII and EBCDIC overpunched signs.

<u>Card Punch Code</u>	<u>Normal Interp.</u>	<u>Ovrpunch Interp.</u>	<u>ASCII Code</u>	<u>EBCDIC Code</u>
0	0	0	060	360
1	1	1	061	361
2	2	2	062	362
3	3	3	063	363
4	4	4	064	364
5	5	5	065	365
6	6	6	066	366
7	7	7	067	367
8	8	8	070	370
9	9	9	071	371
12	+	+0	053	116
space	space	+0	040	NA
12-0	{	+0	173	300
12-1	A	+1	101	301
12-2	B	+2	102	302
12-3	C	+3	103	303
12-4	D	+4	104	304
12-5	E	+5	105	305
12-6	F	+6	106	306
12-7	G	+7	107	307
12-8	H	+8	110	310
12-9	I	+9	111	311
11	-	-0	055	140
11-0 (GBCD)	^	-0	136	NA
11-0 (ASCII)	}	-0	175	320
11-1	J	-1	112	321
11-2	K	-2	113	322
11-3	L	-3	114	323
11-4	M	-4	115	324
11-5	N	-5	116	325
11-6	O	-6	117	326
11-7	P	-7	120	327
11-8	Q	-8	121	330
11-9	R	-9	122	331

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If result is zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If least-significant truncation without rounding, then ON; otherwise, OFF
Overflow	If fixed-point integer overflow, then ON; otherwise, unchanged
Exponent Overflow	If exponent of floating-point result > 127, then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result < -128, then ON; otherwise, unchanged

NOTES:

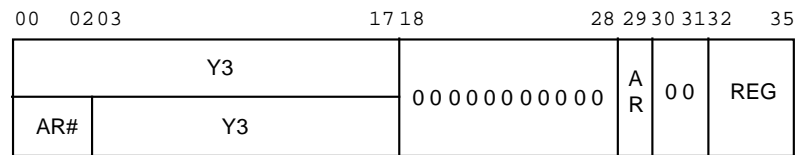
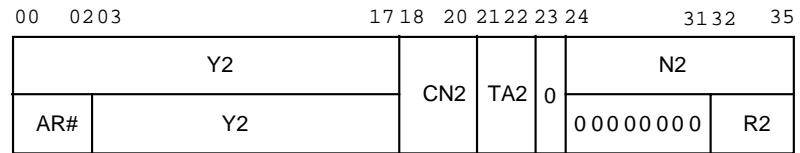
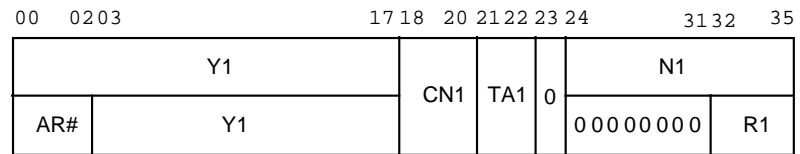
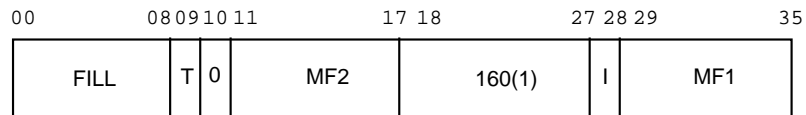
1. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable bit (T) is a 1.
2. An IPR fault occurs if any character (least four bits) other than 0000 - 1001 is detected where digits are defined, or any character (least four bits) other than 1010 - 1111 is detected where the sign is defined by the numeric descriptor.
3. An IPR fault occurs if the values for the number of characters (N1 or N2) of the data descriptors are not large enough to hold the number of characters required for the specified sign and/or exponent, plus at least one digit.
4. An IPR fault occurs if DU or DL modifications are specified for MF1 or MF2, or if illegal repeats are used.
5. Refer to Note 3 of MLR for information on string replication.
6. If an illegal digit or sign is detected, part or all of the receive field may be changed before the IPR fault occurs.

MVT

11.2.20 MVT

MVT	Move Alphanumeric with Translation	160(1)
-----	------------------------------------	--------

FORMAT:



CODING FORMAT:

The MVT instruction code is shown below.

1	8	16

MVT	(MF1) , (MF2) , FILL , T	
ADSC _n	LOCSYM , CN , N , AM	
ADSC _n	LOCSYM , CN , N , AM	
ARG	TABLE , REG , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Starting at location YC1, the alphanumeric characters of data type TA1 are used as an index to a table of contiguous 9-bit characters that start at location Y3 (character position 0). The octal code of the character of string-1 is used as an index to string-3. The indexed 9-bit characters (or right-justified 4- or 6-bit characters) of string-3 replace the contents of string 2, starting at location YC2. If TA1 and TA2 are dissimilar, each character will have high-order truncation. If L1 is less than L2, the FILL character (the entire 9 bits) is used as the index to the table to replace the L2-L1 least significant characters of string 2. The contents of string 1 remain unchanged except in cases of string overlap. When the 9-bit character translate table and the string are overlapped, the result is unpredictable.

$L2 = 0$ does not necessarily mean that the instruction functions as a NOP because the truncation indicator may be affected.

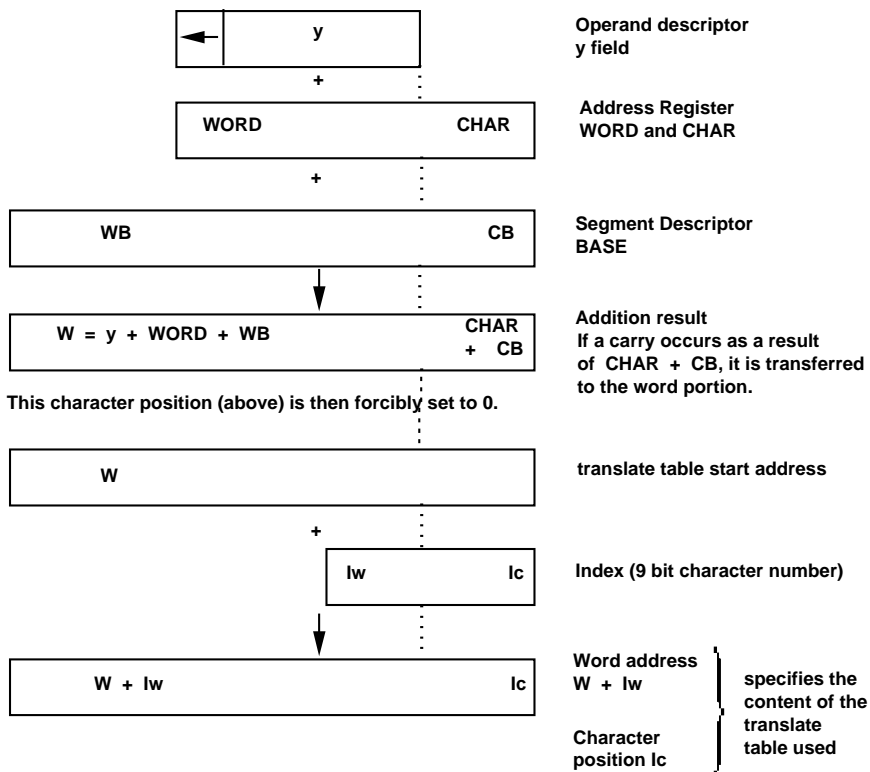
If $L1 < L2$, and type TA1 is 4 or 6-bit, the low-order 4 or 6 bits of the fill character (9-bit) in the instruction word are defined as a table index.

The translation table must begin at a word boundary at character position 0. The index (expressed by the number of 9-bit characters) is added to the starting word address of the table. It is computed in the same way as for normal address modification; however, the computed address is then used as a word address (with character position ignored). The index is added to this word address as a 9-bit number.

The translation table length is determined by the highest possible index character octal value that may be found in the indexing data string. The table is always indexed in 9-bit increments, regardless of the data type being moved. The 9-bit character represented in the table must be the same data type as the receiving field. (See Examples for MVT.)

MVT

When address register modification is specified, the translation table address is generated:



When index register modification is specified, the content of that register is added to the word portion.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and REG field for Y3

ILLEGAL REPEATS:

RPT, RPD, RPL cause an Illegal Procedure fault.

INDICATORS:

Truncation - If L1 is > L2, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1, MF2, or REG fields for Y3 or if illegal repeats are executed.
2. A Truncation fault occurs if the truncation indicator is set and the truncation fault enable (T) bit is a 1.
3. Refer to Explanation of the MLR instruction for information on string replication.

EXAMPLES:

1	8	16	32

	MVT	,,52	with fill index, a minus
	ADSC6	FLD1,4,7	indexing operand descriptor
	ADSC4	FLD2,0,8	receiving operand descriptor
	ARG	TABLE	pointer to 4-bit table
	USE	CONST.	memory contents
FLD1	BCI	2, 000000 123456	2020202001020304050620
FLD2	BSS	1	0123456- (Result)
TABLE	NULL		
	OCT	000001002003,004005006007	0X
	OCT	010011017017,017017017017	1X
	OCT	017017017017,017017017017	2X
	OCT	017017017017,017017017017	3X
	OCT	017017017017,017017017017	4X
	OCT	017017017017,017017017017	5X
	OCT	014017017017,017017017017	6X
	OCT	017017017017,017017017017	7X
	USE		
	MVT		
	ADSC4	FLD3,,8	
	ADSC4	FLD4,,8	
	ARG	TAB	
	USE	CONST.	
FLD3	OCT	022064126317	123456++
FLD4	BSS	1	022064126314 (Result)
TAB	NULL		
	OCT	000001002003,004005006007	
	OCT	010011014014,014015014014	
	USE		

MVT

1	8	16	32
	MVT	0,0	blank fill
	ADSC6	FLD1,0,18	
	ADSC9	FLD2,0,20	
	ARG	TABLE9	pointer to translation table
	USE	CONST.	
FLD1	BCI	3,TTYMESSAGE201	
FLD2	BSS	5	
TABLE9	EDITP	SAVE,ON	
	UASCII	2,01234567	0X
	UASCII	2,89[#@:>?	1X
	UASCII	2,ØABCDEFG	2X
	UASCII	2,HI&.](<\	3X
	UASCII	2,^JKLMN	4X
	UASCII	2,QR-\$(*) ; '	5X
	UASCII	2,/STUVWX	6X
	UASCII	2,YZ_,%="!	7X
	EDITP	RESTORE	
	USE		

NOTE: The translation table length in the above example is determined by the highest octal value for the characters of the indexing string (Field 1). The characters in this translation table are represented in 9-bit ASCII code, the same data type as the receiving field (Field 2). The table is also 64 characters in length, in direct relation to the BCD character set (highest value octal 77).

12. Machine Instruction Descriptions (N-R)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 12.1, Machine Instruction Descriptions (N)
- Section 12.2, Machine Instruction Descriptions (O)
- Section 12.3, Machine Instruction Descriptions (P)
- Section 12.4, Machine Instruction Descriptions (Q)
- Section 12.5, Machine Instruction Descriptions (R)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

12.1 Machine Instruction Descriptions (N)

Following is a detailed description of the processor instructions and operations codes beginning with the letter N.

NAR_n

12.1.1 NAR_n

NAR _n	Numeric Descriptor to Address Register <u>n</u>	66 _n (1)
------------------	--	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

NAR _n		LOCSYM, RM, AM

OPERATING MODES:

NS

SUMMARY:

For n = 0,1,...,or 7 as determined by op code:

C(Y)(0-17) → C(AR_n)(0-17)
 C(Y)(18-20) → C(AR_n)(18-23) [translated]
 C(Y) unchanged

EXPLANATION:

The numeric descriptor is fetched from the computed effective address Y and the TN bit is examined. If TN = 0 (9-bit characters), bits 18 and 19 of the CN field go to the corresponding positions of AR_n and zeros fill bits 20-23 of AR_n. If TN = 1, the 4-bit character contained in the CN field, is converted to bit string representation and placed in bits 18-23 of AR_n. In either case, the descriptor word address field (0-17) goes to bits 0-17 of AR_n.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

NAR_n**ILLEGAL REPEATS:**

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
2. An IPR fault occurs if an attempt is made to execute this instruction in the ES/EI mode.
3. An IPR fault occurs when TN = 0 and bit 21 is set (CN Field).

EXAMPLE:

1	8	16	32

	NAR2	DESCR	load data string address into AR2
	.		
	.		
	.		
DESCR	NDSC4	FLD1,7,8,3,2	032426770210 - descriptor
*			03242665 - result in AR2

NEG

12.1.2 NEG

NEG	Negate (A-Register)	531(0)
-----	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(A) \rightarrow C(A) \text{ if } C(A) \neq 0$

EXPLANATION:

This instruction changes the number in A to its negative (if $\neq 0$). The operation is executed by forming the two's complement of the string of 36 bits.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL causes IPR fault

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON

NOTE:

An Illegal Procedure fault occurs when an illegal repeat is used.

NEGL

12.1.3 NEGL

NEGL	Negate Long (AQ-Register)	533(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$-C(AQ) \rightarrow C(AQ) \text{ if } C(AQ) \neq 0$

EXPLANATION:

This instruction changes the number in AQ to its negative (if $\neq 0$). The operation is executed by forming the two's complement of the string of 72 bits.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPL causes IPR fault

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON

NEGL

NOTE:

An Illegal Procedure fault occurs when an illegal repeat is used.

NOP

12.1.4 NOP

NOP	No Operation	011(0)
-----	--------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

No operation takes place. The effective address is always prepared.

EXPLANATION:

No operation takes place but address preparation is performed according to the specified modifier, if any. If modification other than DU or DL is used, the generated addresses may cause faults.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words. The Tally Runout indicator may be set ON.

NOP

NOTES:

1. An Illegal Procedure fault occurs when an illegal repeat is used.
2. Because address preparation takes place, modification may result in a Bounds fault.

ORA

12.2 Machine Instruction Descriptions (O)

Following is a detailed description of the processor instructions and operation codes beginning with the letter O.

12.2.1 ORA

ORA	OR to A-Register	275(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(A)(i) \text{ OR } C(Y)(i) \rightarrow C(A)(i)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(A) = 0$, then ON; otherwise, OFF

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF

ORAQ**12.2.2 ORAQ**

ORAQ	OR to AQ-Register	277(0)
------	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 71 :
 $C(AQ)(i) \text{ OR } C(Y\text{-pair})(i) \rightarrow C(AQ)(i)$; $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(AQ) = 0$, then ON; otherwise, OFF

Negative If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

ORQ

12.2.3 ORQ

ORQ	OR to Q-Register	276(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(Q)(i) \text{ OR } C(Y)(i) \rightarrow C(Q)(i)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative If $C(Q)(0) = 1$, then ON; otherwise, OFF

ORRR**12.2.4 ORRR**

ORRR	OR Register to Register	536(1)
------	-------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	-----
	ORRR	R1, R2	

OPERATING MODES:

Executes in ES/EI mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q
i = 0, 1, 2, ..., 35
C(R1)(i) OR C(R2)(i) -> C(R1)(i)
C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ORRR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero If C(R1) = 0, then ON; otherwise, OFF

Negative If C(R1)(0) = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

ORSA**12.2.5 ORSA**

ORSA	OR to Storage from A-Register	255(0)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

For $i = 0$ to 35 :
 $C(A)(i) \text{ OR } C(Y)(i) \rightarrow C(Y)(i)$; $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

ORSX_n**12.2.7 ORSX_n**

ORSX _n	OR to Storage from Index Register <u>n</u>	24 _n (0)
-------------------	---	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:NS Mode

For $n = 0, 1, \dots, \text{or } 7$ as determined by op code;

For $i = 0$ to 17:

$C(X_n)(i) \text{ OR } C(Y)(i) \rightarrow C(Y)(i)$;

$C(X_n)$ and $C(Y)(18-35)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, \text{or } 7$ as determined by op code;

For $i = 0$ to 35:

$C(GX_n)(i) \text{ OR } C(Y)(i) \rightarrow C(Y)(i)$;

$C(GX_n)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT or RPD of ORSX0

RPL of any ORSX_n

ORSX_n

INDICATORS:

NS Mode

Zero If $C(Y)(0-17) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then ON; otherwise, OFF

ES/EI Mode

Zero If $C(Y) = 0$, then ON; otherwise, OFF

Negative If $C(Y)(0) = 1$, then OFF; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

ORX_n**12.2.8 ORX_n**

ORX _n	OR to Index Register <u>n</u>	26 <u>n</u> (0)
------------------	-------------------------------	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n=0,1,\dots,$ or 7 as determined by op code;

For $i = 0$ to 17:

$C(X_n)(i) \text{ OR } C(Y)(i) \rightarrow C(X_n)(i);$

$C(Y)$ unchanged

ES/EI Mode

For $n=0,1,\dots,$ or 7 as determined by op code;

For $i = 0$ to 35:

$C(GX_n)(i) \text{ OR } C(Y)(i) \rightarrow C(GX_n)(i);$

$C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL of ORX0

ORX_n

INDICATORS:

Zero	If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(XN/GX_n)(0) = 1$, then ON; otherwise, OFF

NOTES:

1. DL modification is flagged illegal but executes with all zeros for data.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

OWA**12.2.9 OWA**

OWA	OR-Write Memory from A-Register	555(1)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) \text{ OR } C(Y) \rightarrow C(Y)$

EXPLANATION:

The content of the A register is ORed with the content of the memory location specified by Y and the result returned to the specified memory location. Memory access is performed with exclusivity on the C(Y); i.e., other CPUs and IOUs can not access C(Y) while this operation is being performed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

OWA

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

12.2.10 OWQ

OWQ	OR-Write Memory from Q-Register	556(1)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) \text{ OR } C(Y) \rightarrow C(Y)$

EXPLANATION:

The content of the Q register is ORed with the content of the memory location specified by Y and the result returned to the specified memory location. Memory access is performed with exclusivity on the C(Y); i.e., other CPUs and IOUs can not access C(Y) while this operation is being performed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

OWQ

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

OWRES**12.2.11 OWRES**

OWRES	OR-Write Reserve Memory	252(0)
-------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

EXPLANATION:

A word is read from the RMS address, ORed with the content of the A-Register, and the result written back to the same address in RMS. The content of the A-Register is not changed. The RMS access is done as a Read-Alter-Rewrite operation. Memory access is performed with exclusivity on the C(Y); i.e., other CPUs and IOUs can not access C(Y) while this operation is being performed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

PAS

12.3 Machine Instruction Descriptions (P)

Following is a detailed description of the processor instructions and operation codes beginning with the letter P.

12.3.1 PAS

PAS	Pop Argument Stack	176(1)
-----	--------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

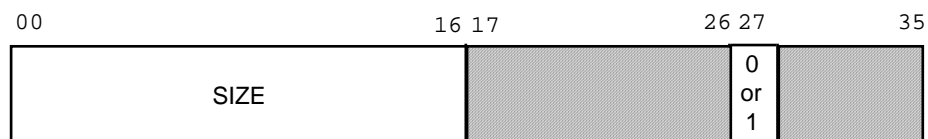
Any

SUMMARY:

Modifies bound field of the argument stack register (ASR)

EXPLANATION:

This instruction provides a means of modifying the bound field of the ASR. The 1-word operand is obtained from memory location Y. The memory operand has the following format:



If ASR flag bit 27 = 0 nothing occurs. The argument segment is empty and the instruction terminates.

If ASR flag bit 27 = 1, the instruction proceeds. The SIZE field is the number of descriptors to be framed, minus 1 (i.e., the number of double-word memory locations).

The descriptor SIZE field is converted to number of bytes by appending three 1 bits as the least significant bits, producing a 20-bit byte size (SIZE-bytes). Accordingly, a memory operand SIZE field of zero means frame one descriptor. Using the 20-bit SIZE-bytes, the instruction proceeds as follows (shaded area is ignored):

If memory operand bit 27 = 0, ASR flag bit 27 and ASR bound field are set to zero and the instruction terminates.

If memory operand bit 27 = 1, the SIZE-bytes is compared to the bound field of the ASR:

If SIZE-bytes < Bound, then SIZE-bytes replaces contents of ASR Bound field.

If SIZE-bytes \geq Bound, then ASR remains unchanged.

NOTE: C(HWMR) is unchanged for all cases.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

PAS

NOTE:

An IPR fault occurs with modifications DU, DL, CI, SC, SCR, and execution of illegal repeats RPT, RPD, RPL.

EXAMPLE:

1	8	16	32

	INHIB	ON	
SVPTR1	STAS	SAVE1	store argument stack
	SDR	P1,0	save descriptor register 1
	STP	P1,SAV11	store pointer to descriptor
*			register 1
	TRA	0,5	
*			
RTPTR1	NULL		
	LDP	P1,SAV11	locates and restores descriptor
*			register 1
	PAS	SAVE1	restores argument stack
	TRA	0,5	

PULS1**12.3.2 PULS1**

PULS1	Pulse One	012(0)
-------	-----------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

No operation takes place

EXPLANATION:

The PULS1 instruction is identical to the NOP instruction except it increments a PULS1 counter in the CPU state variables.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words. The Tally Runout indicator may be set ON.

PULS1

NOTES:

1. An Illegal Procedure fault occurs when illegal repeats are used.
2. This instruction is for use only in external hardware monitoring equipment and not in normal coding.

PULS2**12.3.3 PULS2**

PULS2	Pulse Two	013(0)
-------	-----------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

No operation takes place

EXPLANATION:

The PULS2 instruction is identical to the NOP instruction except it increments a PULS2 counter in the CPU state variables.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The use of Indirect then Tally modifiers ID, DI, IDC, DIC, SCR, or SC changes the address and tally fields of the referenced indirect words. The Tally Runout indicator may be set ON.

PULS2

NOTES:

1. An Illegal Procedure fault occurs when illegal repeats are used.
2. This instruction is for use only in external hardware monitoring equipment and not in normal coding.

12.4 Machine Instruction Descriptions (Q)

Following is a detailed description of the processor instructions and operation codes beginning with the letter Q.

12.4.1 QFAD

QFAD	Quadruple Precision Floating Add	476(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ, LOR) + C(Y\{4 \text{ words}\})]$ normalized $\rightarrow C(EAQ, LOR)$

EXPLANATION:

The exponent underflow indicator is not set when the low-order exponent (E(L)) is in underflow ($E(U)-15 < -128$). At this time, the correct value + 256 is loaded into E(L) and the correct value into the low-order mantissa (M(L) - LOR(8-71)).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E(U) and E(L).

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E(L).

In any other case, E(U) -15 is loaded into E(L).

QFAD

In quadruple precision arithmetic operations, an additional digit (4 bits) called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If $[C(AQ)(0-63), C(LOR)(8-71)] = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -128 , then ON

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFLD**12.4.2 QFLD**

QFLD	Quadruple Precision Floating Load	432(0)
------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Y\{4 \text{ words}\}) \rightarrow C(EAQ, LOR)$
 $C(Y\{4 \text{ words}\})(0-7) \rightarrow C(E)$
 $C(Y\{4 \text{ words}\})(8-71) \rightarrow C(AQ)(0-63)$
 $00000000 \rightarrow C(AQ)(64-71)$
 $C(Y\{4 \text{ words}\})(72-143) \rightarrow C(LOR)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero If $[C(AQ)(0-71), C(LOR)(12-71)] = 0$, then ON;
otherwise OFF

Negative If $C(A)(0) = 1$, then ON; otherwise OFF

QFLD

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFMP**12.4.3 QFMP**

QFMP	Quadruple Precision Floating Multiply	462(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ, LOR) * C(Y\{4 \text{ words}\})]$ normalized $\rightarrow C(EAQ, LOR)$

EXPLANATION:

The exponent underflow indicator is not set when the low-order exponent (E(L)) is in underflow ($E(U)-15 < -128$). At this time, the correct value + 256 is loaded into E(L) and the correct value into the low-order mantissa (M(L) - LOR(8-71)).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E(U) and E(L).

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E(L).

In any other case, E(U) -15 is loaded into E(L).

In quadruple precision arithmetic operations, an additional digit (4 bits) called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

QFMP

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If [C(AQ)(0-63), C(LOR)(8-71)] = 0, then ON; otherwise, OFF
Negative	If C(A)(0) = 1, then ON; otherwise OFF
Exponent Overflow	If exponent > +127, then ON
Exponent Underflow	If exponent < - 128, then ON

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFSB**12.4.4 QFSB**

QFSB	Quadruple Precision Floating Subtract	576(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ, LOR) - C(Y\{4 \text{ words}\})]$ normalized $\rightarrow C(EAQ, LOR)$

EXPLANATION:

The exponent underflow indicator is not set when the low-order exponent (E(L)) is in underflow ($E(U) - 15 < -128$). At this time, the correct value + 256 is loaded into E(L) and the correct value into the low-order mantissa (M(L) - LOR(8-71)).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E(U) and E(L).

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E(L).

In any other case, $E(U) - 15$ is loaded into E(L).

In quadruple precision arithmetic operations, an additional digit (4 bits) called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

During the operation, a two's complement of the subtrahend is justified and added.

QFSB

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If [C(AQ)(0-63), C(LOR)(8-71)] = 0 ,then ON; otherwise, OFF
Negative	If C(A)(0) = 1, then ON; otherwise, OFF
Exponent Overflow	If exponent > +127, then ON
Exponent Underflow	If exponent < - 128, then ON

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFST**12.4.5 QFST**

QFST	Quadruple Precision Floating Store	453(0)
------	---------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

PROCEDURE MODE:

Any

SUMMARY:

[C(EAQ, LOR) -> C(Y{4 words})] normalized
 C(E) -> C(Y{4 words})(0-7)
 C(AQ)(0-63) -> C(Y{4 words})(8-71)
 C(AQ)(64-71) are ignored
 C(LOR) -> C(Y{4 words})(72-143))

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QFSTR

12.4.6 QFSTR

QFSTR	Quadruple Precision Floating Store Rounded	466(0)
-------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

PROCEDURE MODE:

Any

SUMMARY:

[C(AQ)(0-63), C(LOR)(12-71) Rounded,
normalized -> C(Y-pair)]

EXPLANATION:

Arithmetic operation procedure

[C(AQ)(0-63) + Carry] normalized -> C(Y-pair)
 If C(AQ, LOR) is positive, then Carry = 1
 If C(AQ, LOR) is negative;
 If C(LOR)(13-71) = 0, then Carry = 0
 If C(LOR)(13-71) <> 0, then Carry = C(LOR)(12)

Using the above processing, positive and negative data with an equal absolute value are rounded to give values with equal absolute value.

If the mantissa of the result = 0 by rounding, -128 is stored in C(Y-pair)(0-7).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

QFSTR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero If $C(Y\text{-pair})(8-71) = 0$, then ON; otherwise, OFF

Negative If $C(Y\text{-pair})(8) = 1$, then ON; otherwise, OFF

Exponent Overflow If exponent $> +127$, then ON

Exponent Underflow If exponent < -128 , then ON

NOTE:

An Illegal Procedure fault occurs when illegal address modifications or illegal repeats are used.

QLR

12.4.7 QLR

QLR	Q-Register Left Rotate	776(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Rotate C(Q) left by the number of positions indicated by bits 11-17 (NS mode) or 27-33 (ES/EI mode) of Y (Y modulo 128); enter each bit leaving bit position 0 of Q into bit position 35 of Q.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(Q) = 0, then ON; otherwise, OFF

Negative If C(Q)(0) = 1, then ON; otherwise, OFF

QLR

NOTES:

1. The rotate count in the instruction must be a decimal number. To "right-rotate" n bits, use QLR 36-n.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QLS

12.4.8 QLS

QLS	Q-Register Left Shift	736(0)
-----	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Shift C(Q) left by the number of positions indicated by bits 11-17 (NS mode) or 27-33 (ES/EI mode) of Y (Y modulo 128); fill vacated positions with zeros. The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

- Zero If C(Q) = 0, then ON; otherwise, OFF
- Negative If C(Q)(0) = 1, then ON; otherwise, OFF
- Carry If C(Q)(0) changes during the shift, then ON; otherwise, OFF. When the carry indicator is ON, the algebraic range of Q has been exceeded.

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QRS**12.4.10 QRS**

QRS	Q-Register Right Shift	732(0)
-----	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

Shift C(Q) right by the number of positions indicated by bits 11-17 or 27-33 (ES/EI mode) of Y (Y modulo 128); fill vacated positions with bit 0 of C(Q). The shift count in the instruction must be a decimal number.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero If C(Q) = 0, then ON; otherwise, OFF

Negative If C(Q)(0) = 1, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

QSMP

12.4.11 QSMP

QSMP	Quadruple Precision Floating Multiply with Double Precision Operands	460(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) * C(Y\{2 \text{ words}\})]$ normalized $\rightarrow C(EAQ, LOR)$

EXPLANATION:

The exponent underflow indicator is not set when the low-order exponent (E(L)) is in underflow ($E(U)-15 < -128$). At this time, the correct value + 256 is loaded into E(L) and the correct value into the low-order mantissa (M(L) - LOR(8-71)).

When the mantissa (both the high-order and the low-order portions) of the operation result is 0, then -128 is loaded into E(U) and E(L).

When the low-order mantissa, but not the high-order mantissa, of the operation result = 0, then -128 is loaded into E(L).

In any other case, E(U) -15 is loaded into E(L).

In quadruple precision arithmetic operations, an additional digit (4 bits) called a guard digit, is assumed next to the low-order position. An operation is performed in which the intermediate result that includes the guard digit is normalized. The high-order 124 bits are loaded into the EAQ and LOR registers.

The 72 bits of C(AQ)(0-71) are used for the mantissa of the multiplicand.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If $[C(AQ)(0-63), C(LOR)(8-71)] = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent $> +127$, then ON
Exponent Underflow	If exponent < -128 , then ON

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

RBFF

12.5 Machine Instruction Descriptions (R)

Following is a detailed description of the processor instructions and operation codes beginning with the letter R.

12.5.1 RBFF

RBFF	Reset Backup Fault Flag	153(1)
------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

Resets the CPU Backup Fault Flag (BUFF).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. This instruction is intended to be used following a Backup fault. The Backup fault occurs during execution of a Fault CLIMB.

The Backup fault CLIMB references the backup fault entry descriptor that will indicate to the software that this type fault has occurred. A safe store frame is not created, but a backup fault flag is set. If another fault occurs with the backup fault flag set, the CPU will be halted. An indication of the halt is reported to the Service Processor.

Following the indication of a backup fault, software may decide to continue and should reset the backup fault flag.

2. An IPR fault occurs if illegal address modifiers or repeats are specified.

RCCL

12.5.2 RCCL

RCCL	Read Calendar Clock	413(0)
------	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

00...0 ->(AQ)(00-19)
 C(Calendar Clock) ->(AQ)(20-71)

ILLEGAL ADDRESS MODIFICATIONS:

Address modification is not used

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. The calendar clock is a 52-bit binary counter that operates at intervals of 1 microsecond after its initialization.

RCRES**12.5.3 RCRES**

RCRES	Read and Clear Reserve Memory	251(0)
-------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y + \text{Reserve Memory Base Register}) \rightarrow C(A);$
 $00\dots 0 \rightarrow C(Y + \text{Reserve Memory Base Register})$

EXPLANATION:

The effective address Y is added to the Reserve Memory Base Register. The resulting address is used to read the contents of a Reserve Memory location without paging. The word is read into the A-Register and the RMS location used in the read is then cleared to zero. The read and clear is done as a RAR operation.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

RCRES

NOTES:

1. This instruction is intended only for use in Privileged Master mode. Use of this instruction in Master mode or Slave mode causes a Command fault.
2. Bit 29 should be zero to ensure compatibility with future systems. The value of bit 29 is ignored; no address register or descriptor register is used in the address formation.
3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

RDTCS**12.5.4 RDTCS**

RDTCS	Read TCS Clock Register	776(1)
-------	-------------------------	--------

NOTE: Timer Clock Synchronizer (TCS) is not implemented in the V9000, it is always off.

RET

12.5.5 RET

RET	Return	630(0)
-----	--------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS/ES Mode

C(Y)(0-17) -> (IC)
 C(Y)(18-32) -> (IR)
 C(Y)(33-35) is ignored
 C(Y) unchanged

EI Mode

C(Y)(2-35) -> (IC)(0-33)
 C(Y + 1)(18-33) -> (IR)(18-33)

where Y = even word location of a double word

RET**EXPLANATION:**If bit 29 = 0

This instruction loads the content of the location specified by Y into the instruction counter and indicator register. The RET instruction does not load the instruction segment register (ISR) and the SEGID(IS). The return is then within the current instruction segment. The RET instruction may be thought of as an LDI instruction followed by a transfer to the location specified by C(Y)(0-17).

If bit 29 = 1

This instruction loads the ISR and SEGID(IS) from the specified DRn and SEGIDn.

The relation between the bit positions of C(Y) and the indicators is shown below.

<u>C(Y) Bit Position</u>	<u>Indicator (or Mask)</u>
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	UNUSED
27	IGNORED
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Exponent underflow mask
32	Hexadecimal exponent mode
33	Fixed-Point Overflow mask
34-35	UNUSED

RET

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation in the following way.

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer in this case is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n and SEGID_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Master Mode	If C(Y)(28) is 1, then no change; otherwise, OFF
All other indicators	If corresponding bit in C(Y) is 1, then ON; otherwise, OFF

RET

NOTES:

1. An Overflow Fault does not occur when the overflow indicator, exponent overflow indicator, or exponent underflow indicator is set ON via the RET instruction, even if the Overflow Mask Indicator is OFF.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
3. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
4. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
5. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
6. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
7. Hexadecimal mode is exclusively controlled by bit 32 in the IR.

RICHR

12.5.6 RICHR

RICHR	Restart IC History Register	156(1)
-------	-----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

EXPLANATION:

After execution of the RICHR instruction, each Transfer Go or CLIMB causes the source IC value and SEGID to be pushed onto the top of the stack. The ICHR is locked and recording stops on all faults, including page faults and interrupts. RICHR must be executed again to restart recording.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

NOTES:

1. An IPR fault occurs if illegal repeats are used.
2. A command fault occurs if execution is attempted in the Slave or Master mode.

RIMR**12.5.7 RIMR**

RIMR	Read Interrupt Mask Register	233(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(IPMR_p)(0-3) \rightarrow C(A)(1,3,5,7)$
 $0,0,0,0 \rightarrow C(A)(0,2,4,6)$
 $ALLF_p \rightarrow C(A)(8)$
 $0 \rightarrow C(A)(9)$
 $00\dots0 \rightarrow C(A)(10-35)$

EXPLANATION:

Read the interrupt masks for system #p into the A register.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, SC, SCR, CI

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

RIMR

NOTES:

1. A Command fault occurs when an attempt is made to execute this instruction in Slave or Master mode.
2. To support compatibility with future systems, bits 0 to 17, 29, and 30 to 35 of the instruction should be 0. Presence of non-zero values in these fields is ignored and has no effect on execution.
3. An Illegal Procedure fault occurs if illegal repeats are used.

12.5.8 RIW

RIW	Read Interrupt Word Pair	412(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

Read the interrupt word pair for system #p into the AQ register.

EXPLANATION:

TAG = 00: If any unmasked interrupt is present for the current OS system in execution, read the OUT pointer defined interrupt word pair from the highest priority unmasked interrupt queue for this system into AQ and advance that interrupt queue's OUT pointer. The AQ register is set to zero if there are no unmasked interrupts present for this OS. Bit 0 of A is set = 1 if the interrupt queue could not be read because it was gated shut for 32 msec.

TAG = 01: If any unmasked interrupt is present for SID #15, read the highest priority unmasked interrupt queue for SID #15. Otherwise, operation is the same as for TAG = 00.

1. Check for an unmasked Interrupt Present for system "p", where p = the value in SIDR if TAG = 00 or p = 15 if TAG = 01. If there are no unmasked entries for "p", Set C(AQ)(0-71) = zero and terminate instruction.
2. Fetch the IN/OUT pointer for the highest priority unmasked interrupt type (T) for system "p" from the IPTBL.
3. Read the Interrupt Word pair from the Interrupt Queue defined by the OUT pointer and load this value into the AQ-register.

RIW

4. Increment the OUT pointer ENTRY# and store incremented OUT pointer back into the Pointer Table.
5. If the incremented OUT pointer = IN pointer, Reset IPFRp(T) = 0 and send a RPF(SID#,TYP) command on the SBUS to all processors.
6. Write back the IN pointer and terminate the instruction.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero If C(AQ)(0-71) = 0, then ON; otherwise, OFF.

Negative If C(A)(0) = 1, then ON; otherwise, OFF.

NOTES:

1. A Command fault occurs when an attempt is made to execute this instruction in Slave or Master mode.
2. An Illegal Procedure fault occurs if TAG is not equal to 00 or 01.
3. An Illegal Procedure fault occurs if illegal repeats are used.

RLPNR**12.5.9 RLPNR**

RLPNR	Read Logical Processor Number Register	366(1)
-------	--	--------

FORMAT:

Single word instruction

00	17 18	OP CODE	27 28 29 30	35
y	366(1)	I	A R	TAG

OPERATING MODES:

Any

SUMMARY:NS Mode

00...0 -> C(X7)(0-14)
 C(LP NR) -> C(X7)(15-17)

ES/EI Mode

00...0 -> C(GX7)(0-32)
 C(LP NR) -> C(GX7)(33-35)

EXPLANATION:

This instruction accesses the logical processor number register in XRAM and loads it into Index Register 7 (X7/GX7)

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

RLPNR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Not affected

RPD**12.5.10 RPD**

RPD	Repeat Double	560(0)
-----	---------------	--------

FORMAT:

00	07	08	09	10	11	17	18	OP CODE	26	27	28	29	30	35
TALLY	A	B	C	TERM COND	560(0)	0	1	0	DELTA					

OPERATING MODES:

Any

CODING FORMAT:

RPD N, I, k1, k2, ..., k7. (A=B=C=1.)

The command generated by the assembler from this format will cause the two instructions immediately following the RPD instruction to be iterated N times and the effective addresses of those two instructions to be incremented by the value I for each of N iterations. The meaning of the termination conditions of k1, k2, ..., k7 are the same as for the RPT instruction. Since the repeat double must fall in an odd location, the assembler will force this condition and a NOP instruction is used for a filler when needed.

RPDX , I. (A=B=C=0.)

This instruction operates just as the RPD instruction with the exception that A, B, N and the conditions for termination are loaded by the user into index register zero.

RPDA N, I, k1, k2, ..., k7. (A=C=1. B=0.)

This instruction operates just as the RPD instruction with the exception that only the effective address of the first instruction following the RPDA instruction will be incremented by the value of I for each of N iterations.

RPD

RPDB N, I, k1, k2, ..., k7. (A=0. B=C=1.)

This instruction operates in the same way as the RPD instruction except only the effective address of the second instruction following the RPDB instruction will be incremented by the value I for each of N iterations.

EXPLANATION:

The instructions from the next Y-pair are fetched and saved in the processor. They are executed repeatedly until a specified termination condition is met.

1. The RPD instruction must be stored in an odd memory location except when accessed via the XEC or XED instructions. In this case, the RPD instruction can be either even or odd, but the XEC or XED instruction must be in an odd memory location.

2. If C = 0, the tally and terminate conditions are loaded from X0/GX0.

NS Mode

Tally, terminate condition = C(X0)(0-17)

ES/EI Mode

Tally, terminate condition = C(GX0)(18-35);

C(GX0)(0-17) unchanged

3. If C = 1, then bits 0-17 of the RPD instruction are loaded into X0/GX0.

NS Mode

Bits 0-17 of the RPD instruction → C(X0/GX0)

ES/EI Mode

Bits 0-17 of the RPD instruction → C(GX0)(18-35)

00...0 → C(GX0)(0-17)

4. The terminate condition(s) and tally from X0 control the repetition for the instructions following the RPD instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.

5. The repetition cycle consists of the following steps:
- Execute the pair of repeated instructions.
 - $C(X0)(0-7) - 1 \quad \rightarrow C(X0)(0-7)$ **OR**
 $C(GX0)(18-25) - 1 \quad \rightarrow C(GX0)(18-25)$
 - If a terminate condition is met, set the Tally Runout indicator OFF and exit.
 - If bits 0-7 of C(X0) or bits 18-25 of C(GX0) = 0, set the Tally Runout indicator ON and exit.
 - If conditions in c. or d. are not met, go to a.
6. Many instructions cannot be repeated. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine whether or not a particular instruction can be repeated.
7. Address modification for the pair of repeated instructions is explained below.

For each of the two repeated instructions, only the modifiers R and RI and only the designators specifying X1,...,X7/GX1,...,GX7 are permitted. Address register modification is also permitted. All other modifier designations result in an IPR fault.

When the effective address for R modification is Y, and when the indirect word address for RI modification is YI, the address are determined by the following procedure.

When AR modification is not indicated (bit 29 = 0)

- a) For the first execution of each of the two repeated instructions:

$$Y + C(R) \quad \rightarrow Y(1) \text{ or } YI(1)$$

$$Y(1) \text{ or } YI(1) \rightarrow C(R)$$

- b) For any subsequent execution of the two repeated instructions:

For the first instruction of the pair:

$$\text{If } A=1, \text{ then } \Delta + C(R) \rightarrow Y(n) \text{ or } YI(n);$$

$$Y(n) \text{ or } YI(n) \rightarrow C(R)$$

$$\text{If } A=0, \text{ then } C(R) \rightarrow Y(n) \text{ or } YI(n), \text{ where } n>1$$

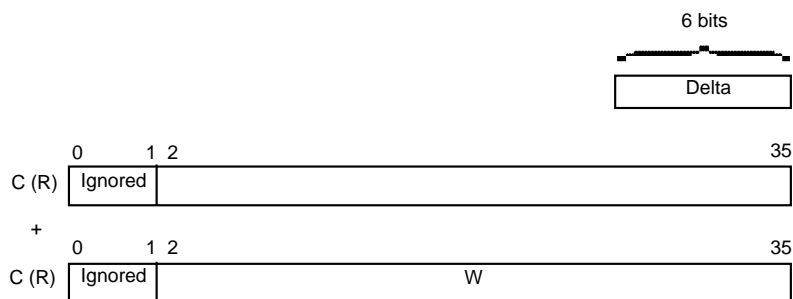
RPD

For the second instruction of the pair

If B=1, then DELTA + C(R) → Y(n) or YI(n);
 Y(n) or YI(n) → C(R)

If B=0, then C(R) → Y(n) or YI(n), where n>1

NOTE: In the ES/EI mode, the GXn used in generation of an operand address is processed as having 36 bits to which the delta is added.



When AR modification is indicated (bit 29 = 1)

- a) For the first execution of each of the two repeated instructions:

(se)Y + C(R) + C(ARm) → Y(1) or YI(1)
 (se)Y + C(R) → C(R)

(se) is the extended address with bit 3 of y.

ARm is the address register m selected by instruction bits 0, 1, 2.

- b) For any subsequent execution of the two repeated instructions:

For the first instruction of the pair:

If A=1, then DELTA + C(R) + C(ARm) → Y(n) or YI(n);
 DELTA + C(R) → C(R)

If A=0, then C(R) + C(AR) → Y(n) or YI(n)

For the second instruction of the pair:

If B=1, then DELTA + C(R) + C(ARm) → Y(n) or YI(n);
 DELTA + C(R) → C(R)

If B=0, then C(R) + C(ARm) → Y(n) or YI(n);

A and B are the contents of the X0 bits 8 and 9 or the GX0 bits 26 and 27.

When RI modification is specified in the repeated instruction, indirect reference is performed only once for each repeat. The tag field of the indirect word is ignored and processed as R modification (R = N).

8. The Exit Conditions

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 after the execution of the odd instruction of the repeated pair. An exit is also made when a fault occurs. The program-controlled exit conditions are:

- a) Tally = 0
- b) Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPD instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any one of the specified conditions is met. The carry, negative, and zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction pair.

- Bit 17 = 0: Ignore all overflows. The respective Overflow indicator is not set ON, and an overflow fault does not occur.
- Bit 17 = 1: Process overflows. If the Overflow Mask indicator is ON when an overflow occurs, then exit from the repetition cycle. If the Overflow Mask indicator is OFF when an overflow occurs, then an overflow fault occurs.
- Bit 16 = 1: Terminate if carry indicator is OFF
- Bit 15 = 1: Terminate if carry indicator is ON
- Bit 14 = 1: Terminate if negative indicator is OFF
- Bit 13 = 1: Terminate if negative indicator is ON
- Bit 12 = 1: Terminate if zero indicator is OFF
- Bit 11 = 1: Terminate if zero indicator is ON

RPD

c) Overflow Fault:

If bit 17 = 1 and an overflow occurs with the Overflow Mask indicator OFF, an overflow fault occurs and an exit is made from the repetition cycle after the execution of the current instruction when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than an overflow occurs. If any fault (overflow, divide check, parity error on indirect word or operand fetch, etc.) occurs on the even instruction, the odd instruction will not be executed.

9. Status at termination of repeat

Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue; that is, the number of repeats remaining until a tally runout would have occurred. The terminate conditions in bits 11-17 remain unchanged.

If the exit was caused by tally = 0 or a terminate condition, the X_n/GX_n specified by the designator of each of the two repeated instructions will contain either:

- a) The contents of the designated X_n/GX_n after the last execution of the repeated pair plus the DELTA associated with each instruction, as A or B, the DELTA designators (bits 8 and 9 of X0) = 1, or
- b) The contents of the designated X_n/GX_n after the last execution of the repeated pair if A or B, respectively, is zero.

If the exit was caused by a fault, the X_n/GX_n specified by the designator of each of the two repeated instructions may contain either:

- a) The contents of the designated X_n/GX_n when the fault occurred plus the DELTA associated with each instruction A and B = 1, or
- b) The contents of the designated X_n/GX_n when the fault occurred.

10. When X0(0-7)/GX0(18-25) contain zeros and the terminate condition is not satisfied, the tally runout indicator is set to ON. Otherwise, it is set to OFF.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed. Bit 29 is ignored.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The RPD instruction itself does not affect any of the indicators. However, the execution of the repeated instructions may affect indicators. The repeat mode entered as a result of the instruction affects the Tally Runout indicator.

NOTES:

1. A Repeat Double (RPD) of instructions that have long execution times may cause a Lockup fault (LUF) if the time involved is greater than the lockup time interval, which may be 2, 4, 8, or 16 milliseconds.
2. The repeated instruction must be modified by an index register.
3. The following conditions cause an IPR fault to occur:
 - If illegal address modifications or illegal repeats are used;
 - If the repeated instruction uses X0/GX0;
 - If R or RI modification is attempted with the repeated instruction with other than X1-X7/GX1-GX7;
 - If the RPD instruction (or the XEC instruction accessing the RPD instruction) is not at an odd location.

EXAMPLE:

1	8	16

	EAX6	FROM
	EAX7	TO
	RPD	100,2
	LDAQ	0,6
	STAQ	0,7
	.	
	.	
	.	
	EVEN	
FROM	BSS	200
TO	BSS	200

RPL

12.5.11 RPL

RPL	Repeat Link	500(0)
-----	-------------	--------

FORMAT:

00	07	08	09	10	11	17	18	OP CODE	26	27	28	29	30	35
TALLY	0	0	C	TERM COND	500(0)	0	1	0	0	0	0	0	0	0

OPERATING MODES:

Any

CODING FORMAT:

RPL N,k1,k2,...,k7. (C = 1.)

This format causes the instruction immediately following the RPL instruction to be repeated N times or until one of the conditions specified in k1,...,k7 is satisfied, or until the link address of zero is detected. The range of N is 0-255. If N = 0, the instruction will be iterated 256 times. If N is greater than 255, the instruction will cause an error flag (A) to be printed on the assembly listing. The fields k1, k2, ..., k7 may or may not be present. They represent conditions for termination which, when needed, are declared by the conditional transfer instructions TMI, TNC, TNZ, TOV, TPL, TRC, and TZE. These instructions affect the termination condition bits in position 11-17 of the Repeat instruction.

An octal number can be used rather than the transfer instructions to denote termination conditions. Thus, if the field for k1, k2, ..., k7 is found to be numeric, it will be interpreted as octal, and the low-order 7 bits will be ORed into bit positions 11-17 of the Repeat instruction. The variable field scan is terminated with the octal field.

RPLX (C = 0)

This instruction operates just as the RPL instruction except that N and the conditions for termination are loaded by the user into index register zero.

EXPLANATION:

The next instruction is executed either a specified number of times, until a specified termination condition is met, or until the link address of zero is detected.

1. If $C = 0$, the tally and terminate conditions are those loaded from $X0/GX0$.

NS Mode

Tally, terminate condition = $C(X0)(0-17)$

ES/EI Mode

Tally, terminate condition = $C(GX0)(18-35)$;
 $C(GX0)(0-17)$ is unchanged

2. If $C = 1$, then bits 0-17 of the RPL instruction $\rightarrow C(X0)/(GX0)$

NS Mode

Bits 0-17 of the RPL instruction $\rightarrow C(X0/GX0)$

ES/EI Mode

Bits 0-17 of the RPL instruction $\rightarrow GX0)(18-35)$
 $00...0 \rightarrow C(GX0)(0-17)$

3. The terminate condition(s) and tally from $X0$ control the repetition for the instruction following the RPL instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.
4. The repetition cycle consists of the following steps:
 - a. Execute the repeated instruction.
 - b. $C(X0)(0-7) - 1 \rightarrow$ bits 0-7 of $C(X0)$, **OR**
 $C(GX0)(18-25) - 1 \rightarrow C(GX0)(18-35)$
 - c. If a terminate condition is met, set the Tally Runout indicator OFF and exit.
 - d. If bits 0-7 of $C(X0)$ or bits 18-25 of $C(GX0) = 0$, or the link address bits 0-17 of $C(Y) = 0$ and no terminate condition is met, set the Tally Runout indicator ON and exit.
 - e. If conditions in step c. or d. are not met, the effective address $C(Y)$ is used as a link address to determine the $C(Y)$ to be used in the next iteration. Go to step a.

RPL

5. Many instructions cannot be repeat linked. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. Refer to the individual instruction descriptions to determine whether or not a particular instruction can be repeated.
6. Address modification for the repeated instruction is given below.
Only address register (AR) modification and R modification specifying X1-X7/GX1-GX7 are permitted for repeated instructions.

R modification is valid only for the first execution of the repeated instruction, AR modification is valid for all executions.

The effective address is generated in the following way.

When AR modification is not indicated (bit 29 = 0):

- a) For the first execution of the repeated instruction:

$$Y = Y(1) + C(R);$$

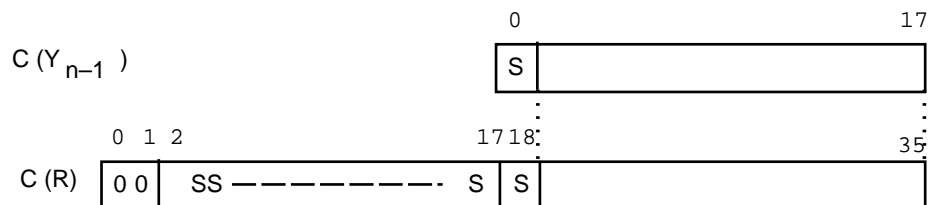
$$Y(1) \rightarrow C(R)$$

- b) For each successive execution of the repeated instruction:

$$Y(n) = C(Y_{n-1})(0-17);$$

$$Y_n \rightarrow C(R) \quad (\text{when } Y_n(0-17) \text{ does not contain zeros})$$

NOTE: When loading a link address $C(Y_{n-1})$ into $C(R)$ in the ES/EI mode, the value obtained by extending $C(Y_{n-1})(0)$ 16 bits to the left is loaded. Zeros are loaded in the two most significant bits.



The effective address Y_n is generated in the same way as in ES/EI mode for other instructions as a 34-bit effective word address.

When AR modification is indicated (bit 29 = 1):

- a) For the first execution of the repeated instruction:

$$\begin{aligned} Y(1) &= (se)Y + C(R) + C(ARm); \\ (se)Y + C(R) &\rightarrow C(R) \end{aligned}$$

(se)y is the extended address with bit 3 of y.

ARm is the address register m selected by instruction bits 0, 1, 2.

- b) For each successive execution of the repeated instruction:

$$\begin{aligned} Y_n &= C(Y_{n-1})(0-17) + C(AR); \\ C(Y_{n-1})(0-17) &\rightarrow C(R) \end{aligned}$$

when $Y_n(0-17)$ does not contain zeros

The effective address Y is the address of the next list word. The lower portion of the list word contains the operand to be used for this execution of the repeated instruction.

The operand is handled in one of the following formats:

Bits 0-17:	00...0
Bits 18-35:	C(Y)(18-35) for single-precision (1 word)

or as

Bits 0-17:	00...0
Bits 18-71:	C(Y) ₁₈₋₇₁ for double precision (2 words)

The upper 18 bits of the list word contain the link address; that is, the address of the next successive list word, and thus the effective address for the next successive execution of the repeated instruction.

RPL

7. Repeat Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 or link address = 0 after the execution of the repeated instruction. An exit is also made when a fault occurs. The program-controlled exit conditions are:

- a) Tally = 0
- b) Link Address = 0
- c) Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPL instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any one of the specified conditions is met.

The Carry, Negative, and Zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction.

- Bit 17 = 0: Ignore all overflows. The respective Overflow indicator is not set ON, and an overflow fault does not occur.
- Bit 17 = 1: Process overflows. If the Overflow Mask indicator is ON when an overflow occurs, then exit from the repetition cycle. If the Overflow Mask indicator is OFF when an overflow occurs, then an overflow fault occurs.
- Bit 16 = 1: Terminate if Carry indicator is OFF.
- Bit 15 = 1: Terminate if Carry indicator is ON.
- Bit 14 = 1: Terminate if Negative indicator is OFF.
- Bit 13 = 1: Terminate if Negative indicator is ON.
- Bit 12 = 1: Terminate if Zero indicator is OFF.
- Bit 11 = 1: Terminate if Zero indicator is ON.

RPL

d) Overflow Fault:

If bit 17 = 1 and an overflow occurs with the Overflow Mask indicator OFF, an overflow fault occurs and an exit is made from the repetition cycle when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than an overflow occurs (divide check, parity error on indirect word or operand fetch, etc.).

8. Status at termination of repeat

Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue; that is, the number of repeats remaining until a tally runout would have occurred. The terminate conditions in bits X0(11-17)/GX0(29-35) remain unchanged.

The X_n / GX_n specified by the designator of the repeated instruction contains the address of the list word that contains:

- a) in its lower half, the operand used in the last execution of the repeated instruction;
- b) in its upper-half, the address of the next list word.

9. When X0(0-7)/GX0(18-25) contain zeros, or when the link address (Y)(0-17) contains zeros, and the terminate condition is not satisfied, the Tally runout indicator is set to ON. Otherwise, it is set to OFF.

10. An exit will not occur if the effective address is 0 for the first execution of the linked instruction. This address specifies the location of the first word in the link table and is not interpreted as a link address.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed. Bits 29-35 are ignored.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The RPL instruction itself does not affect any of the indicators. However, the execution of the repeated instruction may affect the indicators. The repeat mode entered as a result of the instruction affects the Tally Runout indicator.

RPL

NOTES:

1. The repeated instruction must be modified by an index register.
2. The following conditions cause an Illegal Procedure fault.
 - Illegal address modifications or illegal repeats are used.
 - The repeated instruction uses X0/GX0.
 - Modification other than AR or R is attempted with the repeated instruction.
 - R modification other than X1-X7/GX1-GX7 is attempted with the repeated instruction.

EXAMPLE:

1	8	16

	EAX7	A
	LDQ	=07777777, DU
	LDA	=3HIDD, DL
	RPL	5, TZE
	CMK	0, 7
	TNZ	ERROR
	.	.
A	VFD	18/B, H18/IDA
	.	.
B	VFD	18/C, H18/IDB
	.	.
C	VFD	18/D, H18/IDC
	.	.
D	VFD	18/E, H18/IDD
	.	.
E	VFD	18/0, H18/IDE

12.5.12 RPN

RPN	Read Processor Number	367(1)
-----	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

SUMMARY:NS Mode

00...0 -> C(X7)(00-14)
 (CPU number) -> C(X7)(15-17)

ES/EI Mode

00...0 -> C(GX7)(00-32)
 (CPU number) -> C(GX7)(33-35)

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if an illegal repeat is used.

RPT

12.5.13 RPT

RPT	Repeat	520(0)
-----	--------	--------

FORMAT:

00	07 08 09 10 11	17 18	OP CODE	27 28 29 30	35
TALLY	0 0 C	TERM COND	520(0)	I	DELTA

OPERATING MODES:

Any

CODING FORMAT:

RPT N, I, k1, k2, ..., k7. (Bit C=1.)

The command generated by the assembler from this format will cause the instruction immediately following the RPT instruction to be iterated N times and that instruction's effective address to be incremented by the value I for each of N iterations. The range for N is 0-255. If N = 0, the instruction will be iterated 256 times. If N is greater than 256, the instruction will cause an error flag (A) to be printed on the assembly listing. The fields k1, k2, ..., k7 may or may not be present. They represent conditions for termination which, when needed, are declared by the conditional transfer instructions TMI, TNC, TNZ, TOV, TPL, TRC, and TZE. These instructions affect the termination condition bits in positions 11-17 of the Repeat instruction. See the discussion of terminate conditions below.

An octal number can also be used rather than the transfer instructions to denote termination conditions. Thus, if the field for k1, k2, ..., k7 is found to be numeric, it will be interpreted as octal and the low-order 7 bits will be ORed into bit positions 11-17 of the Repeat instruction. The variable-field scan will be terminated with the octal field.

RPTX , I (Bit C = 0)

This instruction operates in the same way as the RPT instruction except that N and the conditions for termination are loaded by the user into bit positions 0-7 and 11-17, respectively, of index register zero (instead of being embedded in the instruction).

RPT**EXPLANATION:**

The next instruction is executed either a specified number of times or until a specified termination condition is met.

1. If $C = 0$, the tally and terminate conditions are those loaded from $X0/GX0$.
 - NS Mode
Tally, terminate condition = $C(X0)(0-17)$
 - ES/EI Mode
Tally, terminate condition = $C(GX0)(18-35)$;
 $C(GX0)(0-17)$ unchanged
2. If $C = 1$, then bits 0-17 of the RPT instruction are loaded into $C(X0)/(GX0)$.
 - NS Mode
Bits 0-17 of the RPT instruction $\rightarrow C(X0/GX0)$
 - ES/EI Mode
Bits 0-17 of the RPT instruction $\rightarrow GX0)(18-35)$
 $00...0 \rightarrow C(GX0)(0-17)$
3. The terminate condition(s) and tally from $X0$ control the repetition for the instruction following the RPT instruction. An initial tally of zero is interpreted as 256. A fault also causes an exit from the cycle.
4. The repetition cycle consists of the following steps:
 - a) Execute the repeat instruction.
 - b) $C(X0)(0-7) - 1 \rightarrow$ bits 0-7 of $C(X0)$, **OR**
 $C(GX0)(18-25) - 1 \rightarrow C(GX0)(18-25)$
 - c) If a terminate condition is met, set the Tally Runout indicator OFF and exit.
 - d) If bits 0-7 of $C(X0)$ or bits 18-25 of $C(GX0) = 0$, set the Tally Runout indicator ON and exit.
 - e) If conditions in c. or d. are not met, go to a.
5. Many instructions cannot be repeated. If an instruction cannot be repeated, an illegal repeat causes an IPR fault to occur. (Refer to the individual instruction descriptions to determine whether a particular instruction can be repeated.)

RPT

6. Address modification for the repeated instruction works in the following way.

For the repeated instruction, only the modifiers R and RI and only the designators specifying X1,...,X7/GX1,...,GX7 are permitted. Address register modification is also permitted.

All other modifier designations result in an IPR fault.

When the effective address for R modification is Y, and when the indirect word address for RI modification is YI, the address are determined in the following way.

When AR modification is not indicated (bit 29 = 0)

- a) For the first execution of the repeated instruction:

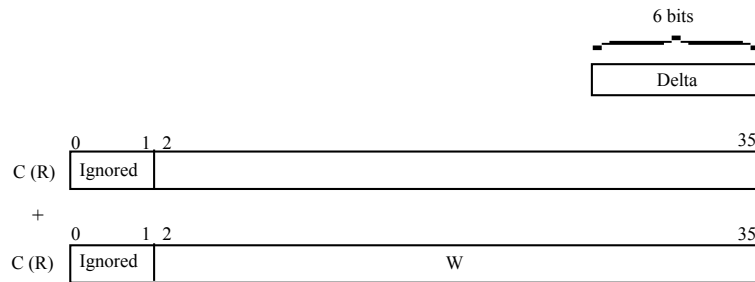
$$\begin{aligned} Y + C(R) &\rightarrow Y(1) \text{ or } YI(1) \\ Y(1) \text{ or } YI(1) &\rightarrow C(R) \end{aligned}$$

- b) For each successive execution of the repeated instruction:

$$\begin{aligned} \text{DELTA} + C(R) &\rightarrow Y(n) \text{ or } YI(n) \\ Y(n) \text{ or } YI(n) &\rightarrow C(R) \end{aligned}$$

DELTA - bits 30 to 35 of the RPT instruction

NOTE: In the ES/EI mode, the GXn used in generation of an operand address is processed as having 36 bits to which the delta is added.



When AR modification is indicated (bit 29 = 1):

- a) For the first execution of the repeated instruction:

$$\begin{aligned} (se)Y + C(R) + C(ARm) &\rightarrow Y(1) \text{ or } YI(1) \\ (se)Y + C(R) &\rightarrow C(R) \end{aligned}$$

(se) is the extended address with bit 3 of y

ARm is the address register m selected by instruction bits 0, 1, 2.

- b) For any subsequent execution of the repeated instruction:

$$\begin{aligned} \text{DELTA} + C(R) + C(ARm) &\rightarrow Y(n) \text{ or } YI(n) \\ \text{DELTA} + C(R) &\rightarrow C(R) \end{aligned}$$

RPT

When RI modification is specified in the repeated instruction, indirect reference is performed only once for each repeat. The tag field of the indirect word is ignored and processed as R modification (R = N).

7. Repeat Exit Conditions:

An exit is made from the repeat cycle if one of the terminate conditions exists or if tally = 0 after the execution of the odd instruction of the repeated pair. An exit is also made when a fault occurs.

The program-controlled exit conditions are:

a) Tally = 0

b) Terminate Conditions:

The bit configuration in bit positions 11-17 of the RPT instruction defines the terminate conditions. If more than one condition is specified, the repeat terminates if any one of the specified conditions is met.

The Carry, Negative, and Zero indicators each use two bits, one for the OFF condition and one for ON. A zero in both positions for one indicator causes this indicator to be ignored as a terminate condition. A 1 in both positions causes an exit after the first execution of the repeated instruction pair.

- | | |
|-------------|---|
| Bit 17 = 0: | Ignore all overflows. The respective Overflow indicator is not set ON, and an overflow fault does not occur. |
| Bit 17 = 1: | Process any overflows. If the Overflow Mask indicator is ON when an overflow occurs, then exit from the repetition cycle. If the Overflow Mask indicator is OFF when an overflow occurs, then an overflow fault occurs. |
| Bit 16 = 1: | Terminate if Carry indicator is OFF |
| Bit 15 = 1: | Terminate if Carry indicator is ON |
| Bit 14 = 1: | Terminate if Negative indicator is OFF |
| Bit 13 = 1: | Terminate if Negative indicator is ON |
| Bit 12 = 1: | Terminate if Zero indicator is OFF |
| Bit 11 = 1: | Terminate if Zero indicator is ON |

RPT

c) Overflow Fault:

If bit 17 = 1 and an overflow occurs with the Overflow Mask indicator OFF, an overflow fault occurs and an exit is made from the repetition cycle when the fault processor returns control.

A non-program-controlled exit from the repetition cycle occurs if any fault other than an overflow occurs.

8. Status at termination of repeat

Bits 0-7 of C(X0) or bits 18-25 of C(GX0) contain the tally residue; that is, the number of repeats remaining until a tally runout would have occurred. The terminate conditions in bits 11-17 remain unchanged.

If the exit was caused by tally = 0 or a terminate condition, the X_n/GX_n specified by the designator of the repeated instruction will contain the contents of the designated X_n/GX_n after the last execution of the repeated instruction plus the DELTA associated with each instruction;

If the exit was caused by a fault, the X_n/GX_n specified by the designator of the repeated instruction may contain one of the following:

- a) the contents of the designated X_n/GX_n when the fault occurred plus the DELTA;
- b) the contents of the designated X_n/GX_n when the fault occurred.

9. When X0(0-7)/GX0(18-25) contain zeros and the terminate condition is not satisfied, the tally runout indicator is set to ON. Otherwise, it is set to OFF.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed. Bit 29 is ignored.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

The RPT instruction itself does not affect any of the indicators. However, the execution of the repeated instruction may affect indicators. The repeat mode entered as a result of the instruction affects the Tally Runout indicator.

RPT**NOTES:**

1. The repeated instruction must be modified by an index register.
2. The following conditions cause an IPR fault to occur:
 - Illegal address modifications or illegal repeats are used;
 - The repeated instruction uses X0/GX0;
 - R or RI modification is attempted with the repeated instruction with other than X1-X7/GX1-GX7;
 - If other than R or RI modification or AR modification are attempted with the repeated instruction.

EXAMPLE:

1	8	16

	LDA	KEY
	EAX4	TABLE
	RPT	64, 1, TZE
	CMPA	0, 4
	TZE	FOUND
	.	
	.	
	.	
TABLE	BSS	64
KEY	BSS	1

RRES

12.5.14 RRES

RRES	Read Reserve Memory	231(0)
------	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(Y + \text{Reserve Memory Base Register}) \rightarrow C(A)$

EXPLANATION:

The effective address Y is added to the Reserve Memory Base Register. The resulting address is used to read the contents of a Reserve Memory location without paging.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT (I is allowed)

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

RRES

NOTES:

1. This instruction is intended only for use in Privileged Master mode. Use of this instruction in Master mode or Slave mode causes a Command fault.
2. Bit 29 should be zero to ensure compatibility with future systems. The value of bit 29 is ignored; no address register or descriptor register is used in the address formation.
3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

Notes

13. Machine Instruction Descriptions (S)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

Section 13.1, Machine Instruction Descriptions (S)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

13.1 Machine Instruction Descriptions (S)

Following is a detailed description of the processor instructions and operation codes beginning with the letter S.

13.1.1 S4BD/S4BDX

S4BD/ S4BDX	Subtract 4-bit Displacement from Address Register	522(1)
----------------	--	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{S4BD}		
	{S4BDX}	word displacement	,R,AR

S4BD/S4BDX

OPERATING MODES:

Any

EXPLANATION:

Description is the same as for A4BD except that y and C(DR) are added and the sum is subtracted from the content of AR_n.

When the mnemonic is coded with an X (S4BDX), bit 29 is forced to zero. If bit 29 is 0, the content of AR_n is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS:

If DU, DL, or IC are specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

Apply to NS mode only

1	8	16	32

	EAX3	10	
	S4BDX	2,3,4	AR4 octal contents - 77777460
	S4BD	0,3,4	AR4 octal contents - 77777340
	EAX6	7	
	S4BDX	3,6,2	AR2 octal contents - 77777405
	S4BD	0,6,2	AR2 octal contents - 77777320

S6BD/S6BDX**13.1.2 S6BD/S6BDX**

S6BD/ S6BDX	Subtract 6-bit Displacement from Address Register	521(1)
----------------	--	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	32

{S6BD}			
{S6BDX} word displacement, R, AR			

OPERATING MODES:

Any

EXPLANATION:

Description is the same as for A6BD except that y and C(DR) are added and the sum is subtracted from the content of AR_n.

When the mnemonic is coded with an X (S6BDX), bit 29 is forced to zero. If bit 29 is 0, the content of AR_n is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS:

If DU, DL, or IC are specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

S6BD/S6BDX

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

Applies to NS mode only

1	8	16	32

	EAX5	14	
	S6BDX	0,5,2	AR2 octal contents - 77777546
	S6BD	2,5,2	AR2 octal contents - 77777123
*			
	EAX6	5	
	S6BDX	1,6,7	AR7 octal contents - 77777605
	S6BD	0,6,7	AR7 octal contents - 77777523

S9BD/S9BDX**13.1.3 S9BD/S9BDX**

S9BD/ S9BDX	Subtract 9-bit Displacement from Address Register	520(1)
----------------	--	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{S9BD}		
	{S9BDX}	word displacement	R,AR

OPERATING MODES:

Any

SUMMARY:

Description is the same as for A9BD except that y and C(DR) are added and the sum is subtracted from the content of AR_n.

When the mnemonic is coded with an X (S9BDX), bit 29 is forced to zero. If bit 29 is 0, the content of AR_n is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS:

If DU, DL, or IC are specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

S9BD/S9BDX

INDICATORS:

None affected

EXAMPLES:

Applies to NS mode only

1	8	16	32

	EAX7	9	
	S9BDX	1,7,5	AR5 octal contents - 77777460
	S9BD	1,,5	AR5 octal contents - 77777360
	EAX2	7	
	S9BDX	2,2,6	AR6 octal contents - 77777420
	S9BD	0,2,6	AR6 octal contents - 77777240

SACK**13.1.4 SACK**

SACK	Transmit Sync Interrupt and Wait for Acknowledgement	734(1)
------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

Loads the A-register with zero to simulate a time-out on the SACK.

ILLEGAL ADDRESS MODIFICATIONS:

IT, RI, IR

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTE:

This instruction is used on Olympus to test shared cache synchronization. Since a commercial processor manages the processor cache on the V9000, the instruction is implemented as if the instruction timed out, i.e., synchronization failed.

SAR_n

13.1.5 SAR_n

SAR _n	Store Address Register <u>n</u>	74 _n (1)
------------------	---------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	SAR _n	LOCSYM, R, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n=0,1,...,7 as determined by op code:
 C(ARn) -> C(Y)(0-23); C(Y)(24-35), C(ARn) unchanged

ES/EI Mode

For n=0,1,...,7 as determined by op code:
 C(ARn) -> C(Y); C(ARn) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

SAR_n**INDICATORS:**

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

Applies to NS mode only

1	8	16	32	

	SAR5	ADDRWS	00175027	AR5 contents
	.			
	.			
	.			
ADDRWS	BSS	1	00175027xxxx	memory after

SAREG

13.1.6 SAREG

SAREG	Store Address Registers	443(1)
-------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	SAREG	LOCSYM, R, AR

OPERATING MODES:

Any

SUMMARY:

NS Mode

$C(AR0, AR1, \dots, AR7) \rightarrow C(Y, Y+1, \dots, Y+7) (0-23)$
 $00\dots0 \rightarrow C(Y, Y+1, \dots, Y+7) (24-35)$

ES/EI Mode

$C(AR0, AR1, \dots, AR7) \rightarrow C(Y, Y+1, \dots, Y+7)$

EXPLANATION:

The lower 3 bits of Y are assumed as 000 and the 8 words beginning from the 8-word boundary are accessed for storage. No check is performed to determine whether the lower 3 bits of Y are actually 000.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

SAREG

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16

	SAREG	REGWS
	.	
	.	
	.	
	EIGHT	
REGWS	BSS	8

SB2D

13.1.7 SB2D

SB2D	Subtract Using Two Decimal Operands	203(1)
------	-------------------------------------	--------

FORMAT:

00 01	08 09 10 11	17 18	27 28 29	35
P	00000000	T R D	MF2	203(1)
				I
				MF1

00 01 02	17 18	20 21 22 23 24	29 30	35
Y1	CN1	T N 1	S1	SF1
AR#	Y1			N1

00 01 02	17 18	20 21 22 23 24	29 30	35
Y2	CN2	T N 2	S2	SF2
AR#	Y2			N2

CODING FORMAT:

The SB2D instruction code is shown below.

1	8	16

SB2D	(MF1), (MF2), RD, P, T	
NDSC _n	LOCSYM, CN, N, S, SF, AM	
NDSC _n	LOCSYM, CN, N, S, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

`C(string 2) - C(string 1) -> C(string 2)`

EXPLANATION:

Same as SB3D except that the difference is stored using YC2, TN2, S2, and, if S2 indicates a scaled format, SF2

The zero indicator is set when the decimal number is zero; it does not indicate that all bits are zeros.

Refer to AD3D for a description of justifying the scaling factors.

Independent of the data type being used (either packed decimal or 9-bit numeric; floating-point or scaled) significant digits in the result may be lost if

1. The difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal-point alignment of source operands, followed by subtraction.
2. The result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

SB2D

INDICATORS:

Zero	If result equals zero, then ON; otherwise, OFF
Negative	If result is negative, then ON; otherwise, OFF
Truncation	If, in the preparation of the final result, one or more least significant digits (zero or nonzero) are lost and rounding is not specified, then ON; otherwise (i.e., no least significant digits lost or rounding is specified), OFF
Exponent Overflow	If exponent of floating-point result is > 127 , then ON; otherwise, unchanged
Exponent Underflow	If exponent of floating-point result is < -128 , then ON; otherwise, unchanged
Overflow	If fixed-point integer, or internal register overflow, then ON; otherwise, unchanged

NOTES:

1. Truncation fault same as for AD3D
2. Illegal Procedure fault same as for MVN
3. If an illegal digit or sign is detected, part or the entire receiving field may be changed before the IPR fault occurs.

EXAMPLES:

Applies to NS mode only

1	8	16	32

	SB2D	,,1	with rounding option
	NDSC4	FLD1,0,4,2,-3	subtrahend operand descriptor
	NDSC9	FLD2,0,8	minuend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	4P125+	1 2 5 +
FLD2	EDEC	8A+6543.21	+ 6 5 4 3 2 1 -2
	USE		+ 6 5 4 3 0 9 -2 (Result)
*			
	SB2D	,,,1	with truncation enable option
	NDSC4	FLD1,0,8,3,-4	subtrahend operand descriptor
	NDSC9	FLD2,0,8,3,-2	minuend operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8P12345678	12345678
FLD2	EDEC	8A87654321	87654321
	USE		87530864 (Result)
	*INSTRUCTION FAULT?	YES	WHAT KIND? truncation fault

SB2DX

13.1.8 SB2DX

SB2DX	Subtract Using Two Decimal Operands Extended	243(1)
-------	--	--------

FORMAT:

00	01	02	08	09	10	11	17	18	Op Code	27	28	29	35
C	N	0000000	T	R	MF2		243(1)			I	MF1		
S	S			D									

00	01	02	17	18	20	21	22	23	24	29	30	35
Y1				CN1	T	N	SX1	SF1	N1			
AR#	Y1				1							

00	01	02	17	18	20	21	22	23	24	29	30	35
Y2				CN2	T	N	SX2	SF2	N2			
AR#	Y2				2							

CODING FORMAT:

1	8	16	

	SB2DX	(MF1), (MF2), RD, CS, T, NS	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SB2DX

SUMMARY:

`C(string2) - C(string1) -> C(string2)`

EXPLANATION:

Same as for SB3DX except that the difference is stored using YC2, TN2, SX2, and if SX2 indicates a scaled format, SF2

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for AD3DX

NOTES:

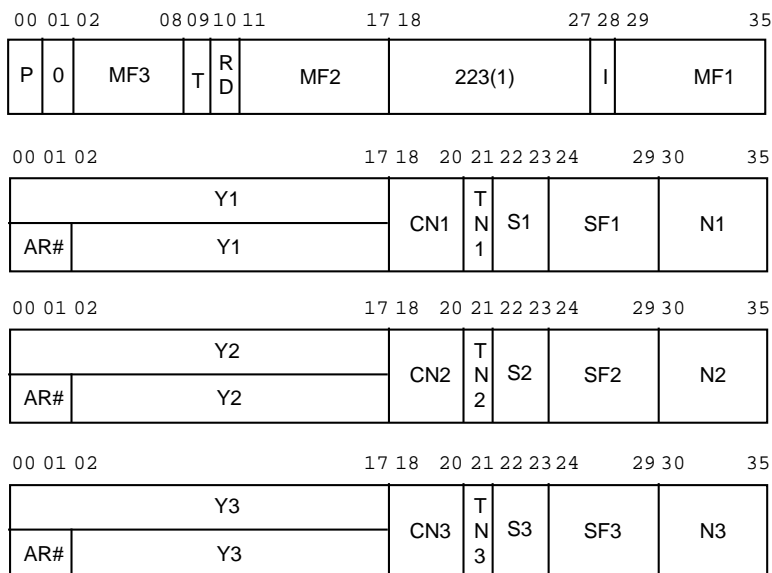
1. All notes for AD3DX apply to SB2DX.
2. See MVNX for information about coding of overpunched signs.

SB3D

13.1.9 SB3D

SB3D	Subtract Using Three Decimal Operands	223(1)
------	---------------------------------------	--------

FORMAT:



CODING FORMAT:

The SB3D instruction code is shown below.

1	8	16	
	SB3D		(MF1) , (MF2) , (MF3) , RD , P , T
	NDSC _n		LOCSYM , CN , N , S , SF , AM
	NDSC _n		LOCSYM , CN , N , S , SF , AM
	NDSC _n		LOCSYM , CN , N , S , SF , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

$C(\text{string } 2) - C(\text{string } 1) \rightarrow C(\text{string } 3)$

The decimal number of data type TN1, sign and decimal type S1, and starting location YC1, is subtracted from the decimal number of data type TN2, sign and decimal type S2, and starting location YC2. The difference is stored starting in location YC3 as a decimal number of data type TN3 and sign and decimal type S3.

If S3 indicates a fixed-point format, the results are stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If S3 indicates a floating-point format, the result is right-justified to preserve the most significant nonzero digits even if this causes least-significant truncation.

If P=1, positive signed 4-bit results are stored using octal 13 as the plus sign. If P=0, positive signed 4-bit results are stored with octal 14 as the plus sign. If RD is a 1, rounding takes place prior to storage.

Provided that strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, and MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for SB2D

SB3D

NOTES:

1. Truncation fault same as for AD3D
2. Illegal Procedure fault same as for MVN
3. The zero indicator is set when the decimal number is zero; it does not that all bits are zero.
4. Independent of the data type being used (either packed decimal or 9-bit numeric; floating-point or scaled) significant digits in the result may be lost if
 - a) the difference between the scaling factors (exponents) of the source operands is large enough to cause the expected length of the intermediate result to exceed 63 digits after decimal-point alignment of source operands, followed by subtraction;
 - b) the result field as defined by the result descriptor is not large enough to contain the calculated result after it has been aligned.
5. If an illegal digit or sign is detected, part or the entire receiving field may be changed before the IPR fault occurs.

EXAMPLES:

Applies to NS mode only

1	8	16	32

	SB3D	,, ,1	with rounding option
	NDSC4	FLD1,0,4,2	subtrahend operand descriptor
	NDSC4	FLD2,0,4,1	minuend operand descriptor
	NDSC9	FLD3,3,5	operand desc for result field
	USE	CONST.	memory contents
FLD1	EDEC	4P123-	123-
FLD2	EDEC	4P-123	-123
FLD3	BSS	2	X X X + 0 0 0 +127 (Result)
	USE		zero indicator ON

1	8	16	32

	SB3D		with truncation enable option
	NDSC9	FLD1,0,8	subtrahend operand descriptor
	NDSC9	FLD2,0,8	minuend operand descriptor
	NDSC4	FLD3,0,8,1,-2	result operand descriptor
	USE	CONST.	memory contents
FLD1	EDEC	8A-123456E-3	- 1 2 3 4 5 6 -3
FLD2	EDEC	8A-987654E-3	- 9 8 7 6 5 4 -3
FLD3	BSS	1	-0086419 (Result)
	USE		indicators on? - negative and truncation

SB3DX

13.1.10 SB3DX

SB3DX	Subtract Using Three Decimal Operands Extended	263(1)
-------	--	--------

FORMAT:

00	01	02	08	09	10	11	17	18	Op Code	27	28	29	35
C	N	MF3	T	R	D	MF2	263(1)			I	MF1		
S	S												

00	01	02	17	18	20	21	22	23	24	29	30	35
Y1						CN1	T	N	SX1	SF1	N1	
AR#	Y1						1					

00	01	02	17	18	20	21	22	23	24	29	30	35
Y2						CN2	T	N	SX2	SF2	N2	
AR#	Y2						2					

00	01	02	17	18	20	21	22	23	24	29	30	35
Y3						CN3	T	N	SX3	SF3	N3	
AR#	Y3						3					

CODING FORMAT:

1	8	16	

	SB3DX	(MF1), (MF2), (MF3), RD, CS, T, NS	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	
	NDSCn	LOCSYM, CN, N, SX, SF, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SUMMARY:

`C(string 2) - C(string 1) -> C(string 3)`

EXPLANATION:

The decimal number of data type TN1, sign and decimal type SX1, and starting location YC1, is subtracted from the decimal number of data type TN2, sign and decimal type SX2, and starting location YC2. The difference is stored starting in location YC3 as a decimal number of data type TN3 and a sign and decimal type SX3.

If SX3 indicates a fixed-point format, the difference is stored using scale factor SF3, which may cause leading or trailing zeros (4 bits - 0000, 9 bits - 000110000) to be supplied and/or most-significant-digit overflow or least-significant-digit truncation to occur.

If SX3 indicates a floating-point format, the result is right-justified to preserve the most-significant-nonzero digits even if this causes least-significant truncation. The character set is defined by CS. Placement of overpunched sign in the output is controlled by NS. (Refer to definition of NS in introductory pages of this section.)

If RD = 1, rounding takes place prior to storage.

Provided strings 1, 2, and 3 are not overlapped, the contents of the decimal numbers that start in locations YC1 and YC2 remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, MF2, or MF3

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for AD3D

SB3DX

NOTES:

1. All notes for AD3D apply to SB3DX.
2. See MVNX for information about coding of overpunched signs.

SBA**13.1.11 SBA**

SBA	Subtract from A-Register	175(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) - C(Y) \rightarrow C(A)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of bit 0 of $C(A)$ is generated, then ON; otherwise, OFF

SBAQ

13.1.12 SBAQ

SBAQ	Subtract from AQ-Register	177(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) - C(Y\text{-pair}) \rightarrow C(AQ)$; $C(Y\text{-pair})$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of AQ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

SBAR**13.1.13 SBAR**

SBAR	Store Base Address Register	550(0)
------	-----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

NS

SUMMARY:

$.WISR_{0-8} \rightarrow C(Y)_{9-17}$

$00\dots0 \rightarrow C(Y)_{0-8}$

$C(Y)_{18-35}$ unchanged

EXPLANATION:

The bounds of $.WISR$ are logically shifted nine places to the right and the results plus one are stored in the user's effective address.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

SBAR

NOTES:

1. An IPR fault occurs with modifications DU, DL, CI, SC, SCR, and execution of illegal repeats RPT, RPD, RPL.
2. An IPR fault occurs if the instruction is executed in the ES mode.
3. This simulated instruction involves many costly operations and should be avoided whenever possible.

SBD/SBDX**13.1.14 SBD/SBDX**

SBD/ SBDX	Subtract Bit Displacement from Address Register	523(1)
--------------	--	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16	

	{SBD }		
	{SBDX}	word displacement	,R,AR

OPERATING MODES:

Any

EXPLANATION:

Description is the same as for ABD except that y and C(DR) are added and the sum is subtracted from the AR.

When the mnemonic is coded with an X (SBDX), bit 29 is forced to zero. If bit 29 is 0, the content of ARn is assumed as 0.

ILLEGAL ADDRESS MODIFICATIONS:

If DU, DL, and IC are specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

SBD/SBDX

INDICATORS:

None affected

NOTES:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

Applies to NS mode only

1	8	16	32

	EAX1	48	
	SBDX	2,1,6	AR6 octal contents - 77777446
	SBD	0,1,6	AR6 octal contents - 77777323
	EAX2	75	
	SBDX	1,2,3	AR2 octal contents - 77777466
	SBD	0,2,3	AR2 octal contents - 77777263

SBLA**13.1.15 SBLA**

SBLA	Subtract Logical from A-Register	135(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) - C(Y) \rightarrow C(A); C(Y)$ unchanged

EXPLANATION:

This instruction is identical to SBA with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

SBLA

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(A)$ is generated, then ON; otherwise, OFF. When the Carry indicator is OFF, the range of A has been exceeded.

SBLAQ**13.1.16 SBLAQ**

SBLAQ	Subtract Logical from AQ-Register	137(0)
-------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) - C(Y\text{-pair}) \rightarrow C(AQ); C(Y\text{-pair})$ unchanged

EXPLANATION:

This instruction is identical to SBAQ with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

SBLAQ

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF. When the Carry indicator is OFF, the range of AQ has been exceeded.

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

SBLQ**13.1.17 SBLQ**

SBLQ	Subtract Logical from Q-Register	136(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) - C(Y) \rightarrow C(Q)$; $C(Y)$ unchanged

EXPLANATION:

This instruction is identical to SBQ except that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

SBLQ

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(Q)$ is generated, then ON; otherwise, OFF. When the Carry indicator is OFF, the range of Q has been exceeded.

SBLR**13.1.18 SBLR**

SBLR	Subtract Logical Register from Register	437(1)
------	---	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
R1	not used	OP CODE	I	mbz	R2

CODING FORMAT:

1	8	16	-----
SBLR	R1, R2		

OPERATING MODES:

Executes in ES/EI Mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q

$C(R1) - C(R2) \rightarrow C(R1);$

C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

SBLR

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(R1)$ is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to "Register to Register Instructions" in Section 7 for a description of the fields in the instruction word.

SBLX_n**13.1.19 SBLX_n**

SBLX _n	Subtract Logical from Index Register <u>n</u>	12 <u>n</u> (0)
-------------------	--	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, \text{ or } 7$ as determined by op code:
 $C(X_n) - C(Y)(0-17) \rightarrow C(X_n); C(Y)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, \text{ or } 7$ as determined by op code:
 $C(GX_n) - C(Y) \rightarrow C(GX_n); C(Y)$ unchanged

EXPLANATION:

This instruction is identical to SBX_n with the exception that the overflow indicator is not affected and an Overflow fault does not occur. Operands and results are treated as unsigned, positive binary integers.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of SBLX0

SBLX_n

INDICATORS:

Zero	If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF
Carry	If a carry out of bit 0 of $C(X_n/GX_n)$ is generated, then ON; otherwise, OFF

NOTES:

1. If DL modification is specified in the NS mode, all data is processed as 0.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SBQ**13.1.20 SBQ**

SBQ	Subtract from Q-Register	176(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) - C(Y) \rightarrow C(Q)$; $C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Q)$ is generated, then ON; otherwise, OFF

SBRR

13.1.21 SBRR

SBRR	Subtract Register from Register	436(1)
------	---------------------------------	--------

FORMAT:

00	03 04	17 18	27 28 29	31 32	35
----	-------	-------	----------	-------	----

R1	not used	OP CODE	I	mbz	R2
----	----------	---------	---	-----	----

CODING FORMAT:

1	8	16	-----	-----	-----
	SBRR	R1, R2			

OPERATING MODES:

Executes in ES/EI Mode only

SUMMARY:

R1, R2 = 0, 1, 2, 3, 4, 5, 6, 7, A, Q
 C(R1) - C(R2) -> C(R1);
 C(R2) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

None. The address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

Execution in NS mode

INDICATORS:

Zero	If $C(R1) = 0$, then ON; otherwise, OFF
Negative	If $C(R1)(0) = 1$, then ON; otherwise, OFF
Overflow	If the range of R1 is exceeded, ON
Carry	If a carry out of bit 0 of C(R1) is generated, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if illegal repeats are executed or if the instruction is executed in NS mode.
2. Refer to Register to Register Instructions in Section 7 for a description of the fields in the instruction word.

SBX_n

13.1.22 SBX_n

SBX _{<u>n</u>}	Subtract from Index Register <u>n</u>	16 _{<u>n</u>} (0)
-------------------------	---------------------------------------	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0, 1, ..., 7 as determined by op code:
 $C(X_n) - C(Y)(0-17) \rightarrow C(X_n); C(Y)$ unchanged

ES/EI Mode

For n = 0, 1, ..., 7 as determined by op code:
 $C(GX_n) - C(Y) \rightarrow C(GX_n); C(Y)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL of SBX0

INDICATORS:

Zero	If $C(X_n/GX_n) = 0$, then ON; otherwise, OFF
Negative	If $C(X_n/GX_n)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of X_n/GX_n is exceeded, then ON
Carry	If a carry out of bit 0 of $C(X_n/GX_n)$ is generated, then ON; otherwise, OFF

NOTES:

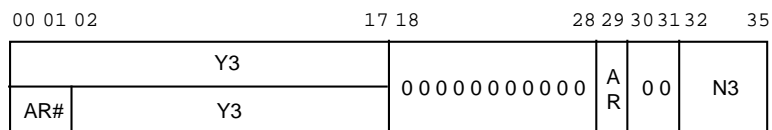
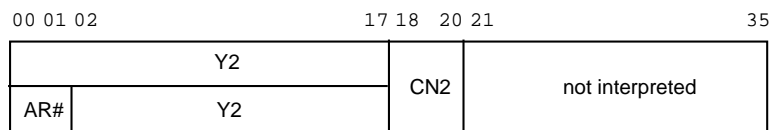
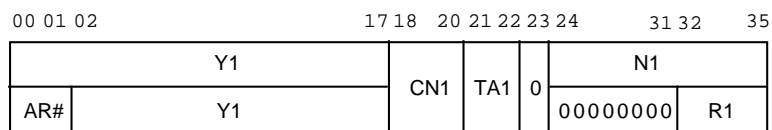
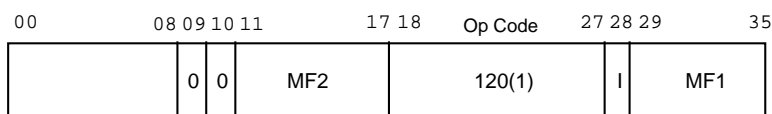
1. If DL modification is specified in the NS mode, all data is processed as 0.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SCD

13.1.23 SCD

SCD	Scan Characters Double	120(1)
-----	------------------------	--------

FORMAT:



CODING FORMAT:

The SCD instruction code is shown below.

1	8	16

SCD		(MF1) , (MF2)
ADSCn		LOCSYM , CN , N , AM
ADSCn		LOCSYM , CN , , AM
ARG		LOCSYM , RM , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Unless $N1 = 0$ or 1 , starting at location $YC1$, $L1-1$ concatenated pairs of type TA1 characters are compared with the two assumed type TA1 characters that are either stored in location $YC2$ and $YC2 + 1$ or contained in bits 0-7, bits 0-11, or; when the REG field of MF2 specifies DU modification, bits 0-17 of the address field of operand descriptor 2.

The compare continues until an identical match is found or until the $L1-1$ tally is exhausted. A count of compares is kept and for each unsuccessful match; the count is incremented by 1. When a match is found or the tally is exhausted, the compare count is stored in bits 12-35 of Y3 and bits 0-11 of Y3 are zeroed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or the Y3 REG field; DL for MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Tally	If the tally ($L1-1$) is exhausted without a successful match, then ON; otherwise, OFF
-------	--

NOTES:

1. The RL bit in the MF2 field is not used.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SCD

EXAMPLES:

1	8	16	32

	SCD		with no options
	ADSC6	FLD1,,6	scanned string operand descriptor
	ADSC6	FLD2,3	character pair operand descriptor
	ZERO	FLD3	FLD3 operand descriptor pointer
	TF	HAVE1	match found - tally runout OFF
	USE	CONST.	characters compared
FLD1	BCI	1,123456	123456
FLD2	BCI	1,654321	32
FLD3	BSS	1	unmatched count - 5
	USE		Result - no match found

1	8	16	32

	SCD		with no options
	ADSC6	DATA,,24	24 characters fetched from lower
	ADSC6	COMPmm2	DATA in units of 2 chars. and
	ARG	COUNT	compared with HH when HH found in
*			DATA, count stored as binary
*			number before HH detection and
*			instruction terminated
	.	.	
	.	.	
DATA	BCI	2,AABBCCDDEEFF	
	BCI	2,GGHHIIJJKKLL	
COUNT	BSS	1	COUNT contains decimal 14
COMP	BCI	1,HH	

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX5	5	load 5 into X5
	EAX7	7	load 7 into X7
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	SCD	(1,1,,5),(,,DU)	- with address modification
	ADSC9	0,0,X7,4	FLD1 operand pointer (FLD1+1,1,7)
FLD2	VFD	A18/45	FLD2 operand
	ARG	FLD3	pointer to count FLD3
	TTN	*+2	no match found
	NULL		match found
	USE	CONST.	characters compared
FLD1	EDEC	12A1234567	000001234567
FLD3	DEC	0	unmatched count - 3
	USE		Result - match found on 4th pair

SCDR**13.1.24 SCDR**

SCDR	Scan Characters Double in Reverse	121(1)
------	--------------------------------------	--------

FORMAT:

Same as Scan Characters Double (SCD) format

CODING FORMAT:

The SCDR instruction code is shown below.

1	8	16

	SCDR	(MF1) , (MF2)
	ADSC _n	LOCSYM, CN, N, AM
	ADSC _n	LOCSYM, CN, , AM
	ARG	LOCSYM, RM, AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Same as for SCD except that start is at location YC1 + (L1-1) and pairs are scanned in reverse to location YC1

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or the Y3 REG field; DL for MF2

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

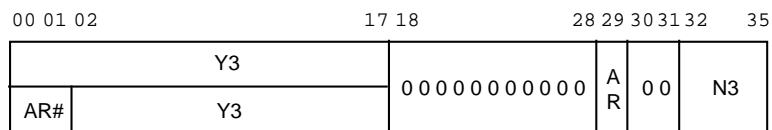
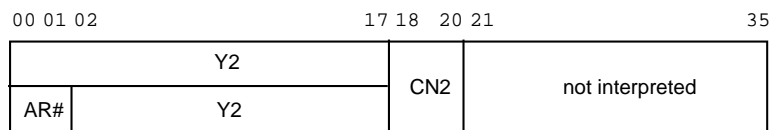
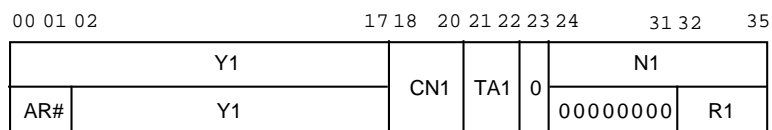
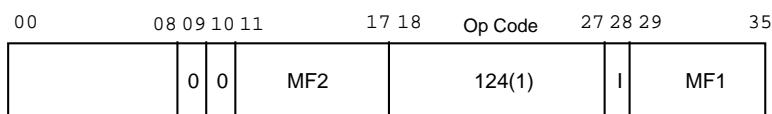
K0	EQU	0	
K7	EQU	7	
	EAX2	1	
	EAX3	FLD1	load FLD1 address into X3
	AWDX	0,3,4	put FLD1 address into AR4
	SCDR	(1,,2),(,,DU)	- with address modification
	ADSC4	0,K0,K7,4	FLD1 operand descriptor -
*			(FLD 1,1,7)
	EDEC	2PL23	FLD2 operand descriptor pointer
	ARG	FLD3	pointer to count word
	TTN	OOPS	no match - tally runout ON
	NULL		match found
	USE	CONST.	characters compared
FLD1	EDEC	8P123456	0123456 VS 23
FLD3	BSS	1	unmatched count - 3
	USE		Result - match found on 4th pair

SCM

13.1.25 SCM

SCM	Scan with Mask	124(1)
-----	----------------	--------

FORMAT:



CODING FORMAT:

The SCM instruction code is shown below.

1	8	16

SCM	(MF1)	(MF2), MASK
ADSC _n	LOCSYM	CN, N, AM
ADSC _n	LOCSYM	CN, , AM
ARG	LOCSYM	RM, AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Starting at location YC1, the L1 type TA1 characters are masked and compared with the assumed type TA1 character contained either in location YC2 or in bits 0-8 or 0-5 of the address field of operand descriptor 2 (when the REG field of MF2 specifies DU modification). The mask is right-justified in bit positions 0-8 of the instruction word. Each bit position of the mask that is a 1 prevents that bit position in the two characters from entering into the compare.

The masked compare operation continues until either a match is found or the tally (L1) is exhausted. For each unsuccessful match, a count is incremented by 1. When a match is found or when the L1 tally runs out, this count is stored right-justified in bits 12-35 of location Y3 and bits 0-11 of Y3 are zeroed. The contents of location YC2 and the source string remain unchanged. The RL bit of the MF2 field is not used.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or Y3 REG field; DL for MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Tally	If the tally (L1) is exhausted without a successful match, then ON; otherwise, OFF
-------	--

NOTES:

1. If L1 = 0, zero is stored in Y3 (bits 12-35) and the tally indicator is affected.
2. If L1 \neq 0 and a match is found in the first character, zero is stored in Y3 (bits 12-35) and the tally indicator is set to OFF.
3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SCM

EXAMPLES:

1	8	16	32

	SCM	,,760	mask to eliminate zone bits
	ADSC9	FLD1,0,4	character string operand desc
	ADSC9	FLD2,3	compare character operand desc
	ARG	FLD3	pointer to unmatched count word
	TTF	GOT.IT	match found
	NULL		no match - tally runout ON
	USE	CONST.	octal repr. of scanned chars.
FLD1	ASCII	1,ABCD	141 142 143 144 (before mask)
			001 002 003 004 (after mask)
*			octal representation of compare character
FLD2	ASCII	1,0004	064 (before mask)
FLD3	BSS	1	004 (after mask)
	USE		unmatched compare count - 3
*			Result - match found on 4th character
	SCM	,(,,DU)	DU type REG modifier on FLD2
	ADSC4	FLD1,3,5	character string operand
*			descriptor
	EDEC	8PL-1	FLD2's compare character -
	ARG	FLD3	pointer to unmatched count word
	TTF	GOT.IT	match found
	NULL		no match - tally runout ON
	USE	CONST.	character scanned
FLD1	EDEC	8P-1234	0,1,2,3,4
FLD3	BSS	1	unmatched compare count - 5
	USE		Result - no match found

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX1	1	load FLD2 character mod into X1
	EAX2	2	load FLD1 character mod into X2
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	SCM	(1,1,1,2),(1,,1,1),010	with all options
	ARG	INDSC1	pointer to FLD1 indirect desc
	ARG	INDSC2	pointer to FLD2 indirect desc
	ARG	FLD3	pointer to unmatched count word
	TTN	0Y	no match - tally runout ON
	USE	CONST.	character compared
FLD1	EDEC	8PL4321	2 1
FLD2	EDEC	4P0987	1
FLD3	BSS	1	unmatched compare count - 1
INDSC1	ADSC4	0,,X2,4	FLD1 operand desc (FLD1,2,2)
INDSC2	ADSC9	FLD2,0	FLD2 operand desc (FLD2,1)
	USE		Result - match found on 2nd character

SCMR**13.1.26 SCMR**

SCMR	Scan with Mask in Reverse	125(1)
------	---------------------------	--------

FORMAT:

Same as Scan with Mask (SCM) format

CODING FORMAT:

The SCMR instruction code is shown below.

1	8	16

	SCMR	(MF1) , (MF2) , MASK
	ADSC _n	LOCSYM , CN , N , AM
	ADSC _n	LOCSYM , CN , , AM
	ARG	LOCSYM , RM , AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Same as SCM except start at location YC1 + (L1-1) and progress toward location YC1

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 or the Y3 REG; DL for MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

	EAX6	6	load FLD1 length into X6
	EAX2	2	load character modifier into X2
	EAX4	FLD1	load FLD1 address into X4
	AWDX	0,4,4	put FLD1 address into AR4
	SCMR	(1,1,1,2),,760	with all options
	ARG	FLD3+1	pointer to FLD1 indirect desc
	ADSC4	FLD2,0	pointer to compare character
	ARG	FLD3	pointer to unmatched count word
	TTN	OUCH	no match - tally runout ON
	TRA	WHEW	match found
	USE	CONST.	characters compared
FLD1	EDEC	8P0123456-	2,3,4,5,6,-
FLD3	DEC	0	unmatched compare count - 4
	ADSC4	0,,X6,4	FLD1 operand desc(FLD 1,2,6)
FLD2	EDEC	4PL3	FLD2 compare character 3
	USE		Result - match found on 5th compare

SCTR

13.1.27 SCTR

SCTR	Store Weighted Instruction Counter	414(0)
------	------------------------------------	--------

NOTE: The SCTR instruction functions as a NOP in the V9000.

SDR_n**13.1.28 SDR_n**

SDR _n	Save Descriptor Register <u>n</u>	11 <u>n</u> (1)
------------------	-----------------------------------	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(DR_n) → Argument Stack (AS)

SEGID_n is set to indicate the stored segment descriptor.

EXPLANATION:

These eight instructions store the operand descriptor in the next available location on the argument stack and adjust the argument stack bound.

The instructions are executed in the following way.

1. The following checks are performed.
 - a) The ASR (Argument Stack Register) flag bit 28 is checked. If it is zero, the argument segment is not present, a Missing Segment fault occurs and the instruction is terminated.
 - b) The ASR bound field is checked.
 - c) If bound + 1 ≥ 2048 words, a Bound fault occurs and the instruction is terminated.
 - d) The ASR flag bit 21 (write permit) is not checked. This instruction always permits write operation for the argument segment.

SDR_n

2. If the conditions described under (1) are satisfied, execution continues. It generates the effective byte address indicating the next available double-word location on the AS. The ASR flag bit 27 is then checked.
 - a) If the ASR flag bit 27 = zero, the argument segment is empty. The ASR base indicates the first double-word location.
 - b) If the ASR flag bit 27 = 1, ASR bound + base + 1 is executed to generate a virtual address.
3. After the DR_n content has been stored in AS, the following operations are executed and the instruction is completed.
 - a) The ASR flag bit 27 is checked.

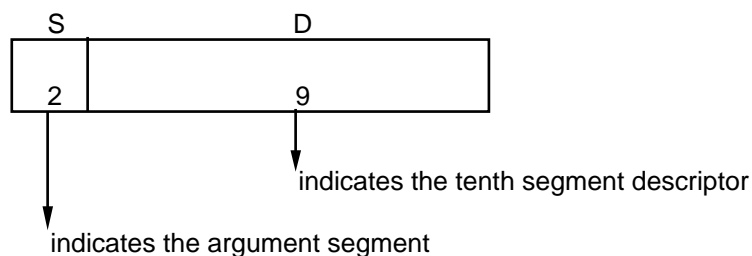
If ASR bit 27 = 1, 8 is added to the ASR bound field. It indicates that the new segment has been stored and the segment size has increased.

If ASR bit 27 = 0, the argument segment indicates that it was empty when the instruction was begun. The bound field is then set to seven bytes to indicate that a segment descriptor has been stored. The ASR flag bit 27 is set to 1 to indicate that this segment is no longer empty.

- b) SEGID_n is set to indicate the location in which the segment descriptor is stored.

For example, if the ASR bound field is 117 (octal) bytes (= 80 bytes = 20 words = 10 double-words) after 8 is added, SEGID_n is set as shown below.

SEGID_n



ILLEGAL ADDRESS MODIFICATIONS:

Ignored. Address modification is not executed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. A Missing Segment or Missing Page fault may occur.
2. A Security Fault, Class 1 may occur but not within housekeeping pages.
3. A Bound fault may occur.
4. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SFR

13.1.29 SFR

SFR	Store Fault Register	452(0)
-----	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

C(Fault Register) → C(Y-pair)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. A Command fault occurs when this instruction is executed in the Slave or Master Mode.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
3. Refer to Section 4 for the content of the Fault Register.

SICHR**13.1.30 SICHR**

SICHR	Store IC History Register	154(1)
-------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:NS mode:

$C(ICHR) \rightarrow C(Y)(0-71)$

EXPLANATION:

Stores the most recent entry of the ICHR into the C(Y-pair). The ICHR is then popped up by one entry with the entry just stored circulated to the bottom of the stack.

Instruction Counter History Register (ICHR)

The ICHR is a register consisting of 1024 words, each with 72 bits. It stacks the content of the instruction counter (IC) of a transfer of control instruction when update is performed by such an instruction (unconditional or conditional). The IC stack is cyclic modulo 16. This register is used for software debugging.

Mode	bit(s)	Description
NS/ES:	0-1	zeros (00)
	2-17	zeros (00...0)
	18-35	From IC value
	36-47	From SEGID(IS)
	48-71	zeros (00...0)
EI:	0-1	zeros (00)
	2-35	Transfer From IC value
	36-47	From SEGID(IS)
	48-71	zeros (00...0)

SICHR

ILLEGAL ADDRESS MODIFICATION:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

1. A Command fault occurs when this instruction is executed in the Slave or Master Mode.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

13.1.31 SIW

SIW	Set Interrupt Word Pair	451(0)
-----	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(SIDR) \rightarrow C(A)(32-35)$; otherwise, $C(A)(32-35)$ unchanged.
 $C(AQ) \rightarrow \text{interrupt queue}(\text{defined by } C(A)(29-30)+C(A)(32-35))$

EXPLANATION:

1. Fetch the IN/OUT pointer for the interrupt queue defined by the type number "T", A(29-30), and the system number "p", A(32-35). The IN/OUT pointer is formed by adding the quantity " $16*p+2*T$ " to the IPTBL address contained in the CPU XRAM.
2. Store the C(AQ) into the interrupt queue location defined by the IN pointer fetched in step 2.
3. Transmit a SIW command on the system bus with type field, A(29-30), and the SID field, A(32-35).
4. Increment the IN pointer, ENTRY#. If, after the increment, the IN pointer = the OUT pointer, the original IN pointer is written back to memory to unlock the gate, and a Queue overflow fault occurs. Otherwise, the new IN pointer value is stored into the Pointer Table.
5. Terminate the instruction.

ILLEGAL ADDRESS MODIFICATIONS:

None. Address modification is not executed.

SIW

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero If $C(AQ)(0-71) = 0$, then ON; otherwise, OFF.

Negative If $C(A)(0) = 1$, then ON; otherwise, OFF.

NOTES:

1. A Command fault occurs when an attempt is made to execute this instruction in Slave or Master mode.
2. An Illegal Procedure fault occurs if illegal repeats are used.

SLAR**13.1.32 SLAR**

SLAR	Store Limit Address Register	725(1)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

00 → C(Y) (00-01)
 C(ULAR) (00-15) → C(Y) (02-17)
 00 → C(Y) (18-19)
 C(LLAR) (00-15) → C(Y) (20-35)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, or RPL

INDICATORS:

None affected

NOTES:

1. A Command fault occurs if execution attempted in slave or master mode.
2. An Illegal Procedure fault occurs if illegal address modification is used.

SPDBR

13.1.33 SPDBR

SPDBR	Store Page Table Directory Base Register	151(1)
-------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

SV mode:

C(PDBR) → C(Y)(0-18)
00...0 → C(Y)(19-35)

SVMX mode:

C(PDBR) → C(Y)(0-35)

EXPLANATION:

The PDBR content is stored into Y. Zero is stored in unspecified bits. The PDBR content remains unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if execution of this instruction is attempted in Slave or Master Mode.

SPL

13.1.34 SPL

SPL	Store Pointers and Lengths	447(1)
-----	----------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	SPL	LOCSYM, R, AR

OPERATING MODES:

Any

SUMMARY:

C(Pointer and Length registers) → C(Y), C(Y+1), ..., C(Y+5)
 C(LOR) → C(Y+6), C(Y+7)

EXPLANATION:

The pointers and lengths registers are used by hardware to store control information when an interruptible multiword instruction is interrupted during execution. These registers enable hardware to resume processing an interrupted instruction after a return from servicing the interrupt.

Y must be a multiple of 8. However, a fault does not occur when the lower 3 bits of Y are not 000. For purposes of execution, the hardware assumes that these bits are 000.

Refer to Explanation under LPL instruction for relationship between the register and the operand.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTIONS:

XEC, XED

INDICATORS:

Multiword Instruction Interrupt indicator (bit 30), OFF

NOTES:

1. An Illegal Procedure fault occurs if illegal address modifications, illegal repeats, or illegal executions are used. An IPR fault also occurs if target of XEC, XED.
2. The content of the pointer and length registers are changed if RPT, RPD, RPL, XEC, or XED or indirect modification (IT) are executed.
3. The SPL instruction is normally only used by routines that process interrupts.
4. After an interrupt occurs, the SPL must be executed before any multiword instruction to avoid destruction of the pointer and length information.

SREG

13.1.35 SREG

SREG	Store Registers	753(0)
------	-----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

The registers are stored as follows:

$C(X0) \rightarrow C(Y) (00-17);$
 $C(X1) \rightarrow C(Y) (18-35)$
 $C(X2) \rightarrow C(Y+1) (00-17);$
 $C(X3) \rightarrow C(Y+1) (18-35)$
 $C(X4) \rightarrow C(Y+2) (00-17);$
 $C(X5) \rightarrow C(Y+2) (18-35)$
 $C(X6) \rightarrow C(Y+3) (00-17);$
 $C(X7) \rightarrow C(Y+3) (18-35)$
 $C(A) \rightarrow C(Y+4) (00-35)$
 $C(Q) \rightarrow C(Y+5) (00-35)$
 $C(E) \rightarrow C(Y+6) (00-07);$
 $00...0 \rightarrow C(Y+6) (08-35)$
 $C(TR) \rightarrow C(Y+7) (00-26);$
 $00...0 \rightarrow C(Y+7) (27-35)$

ES/EI Mode

The registers are stored as follows:

```

C(GX0) -> C(Y)
C(GX1) -> C(Y+1)
C(GX2) -> C(Y+2)
C(GX3) -> C(Y+3)
C(GX4) -> C(Y+4)
C(GX5) -> C(Y+5)
C(GX6) -> C(Y+6)
C(GX7) -> C(Y+7)
C(A)    -> C(Y+8)
C(Q)    -> C(Y+9)
C(E)    -> C(Y+10)(00-07);
00...0  -> C(Y+10)(08-35)
C(TR)   -> C(Y+11)(00-26);
00...0  -> C(Y+11)(27-35)

```

In all modes the register content remains unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Location Y must be forced to a multiple of 8 by entering an 8 in column 7 of the statement that defines Y, or by means of the EIGHT pseudo-operation.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SRMB

13.1.36 SRMB

SRMB	Store Reserve Memory Base	714(0)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(RMBR)(00-35) \rightarrow C(Y)(00-35)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Command fault occurs if execution is attempted in slave or master mode.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

13.1.37 SSA

SSA	Subtract Stored from A-Register	155(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) - C(Y) \rightarrow C(Y)$; $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of $C(Y)$ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

SSA

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

13.1.38 SSQ

SSQ	Subtract Stored from Q-Register	156(0)
-----	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) - C(Y) \rightarrow C(Y)$; $C(Q)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of $C(Y)$ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

SSQ

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SSX_n**13.1.39 SSX_n**

SSX _n	Subtract Stored from Index Register <u>n</u>	14 <u>n</u> (0)
------------------	---	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(X_n) - C(Y)(0-17) \rightarrow C(Y)(0-17); C(X_n)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(GX_n) - C(Y) \rightarrow C(Y); C(GX_n)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT or RPD of SSX0

RPL of any SSX_n

SSX_n

INDICATORS:

NS Mode

Zero	If $C(Y)(0-17) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of $C(Y)$ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

ES/EI Mode

Zero	If $C(Y) = 0$, then ON; otherwise, OFF
Negative	If $C(Y)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of $C(Y)$ is exceeded, then ON
Carry	If a carry out of bit 0 of $C(Y)$ is generated, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STA**13.1.40 STA**

STA	Store A-Register	755(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(A) \rightarrow C(Y); C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAB

13.1.41 STAB

STAB	Store A-Register by Byte	561(0)
------	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any, except NS mode which will cause IPR to occur.

SUMMARY:

$C(A)(28-35) \rightarrow C(Y)(\text{byte as specified});$
 $C(A)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, AU, QU, RI, IR, IT

ILLEGAL REPEATS:

RPL, RPD, RPT

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAC

NOTE:

An Illegal Procedure fault occurs if illegal address modifications are specified or if the target of a RPL instruction.

STACQ**13.1.43 STACQ**

STACQ	Store A Conditional on Q	654(0)
-------	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

If $C(Y) = C(Q)$,
set Zero Indicator ON and write-unlock. $C(A) \rightarrow C(Y)$

If $C(Y) \neq C(Q)$,
set Zero Indicator OFF and write-unlock. $C(Y)$ is unchanged.

EXPLANATION:

If the initial $C(Y)$ is $\neq C(Q)$, then $C(Y)$ is not changed by the STACQ instruction.

This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPD, RPL, RPT

STAH**13.1.44 STAH**

STAH	Store A-Register by Halfword	571(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any, except NS mode which will cause IPR to occur.

SUMMARY:

C(A)(low order halfword) → C(Y)(halfword as specified);
C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAQ

13.1.45 STAQ

STAQ	Store AQ-Register	757(0)
------	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(AQ) \rightarrow C(Y\text{-pair}); C(AQ)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STAS**13.1.46 STAS**

STAS	Store Argument Stack Register	750(1)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(ASR) \rightarrow C(Y\text{-pair}); C(ASR)$ unchanged

EXPLANATION:

The execution of this instruction causes the current contents of the argument stack register (ASR) to be stored in even and odd memory locations Y and Y+1. The contents of the ASR remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STAS

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16

	STAS	SVASR
	SDR	P0
	STP	P0 , SVP0
	SDR	P1
	STP	P1 , SVP1
	.	.
	.	.
	.	.
	LDP	P0 , SVP0
	LDP	P1 , SVP1
	PAS	SVASR

STBA**13.1.47 STBA**

STBA	Store 9-Bit Bytes of A-Register	551(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

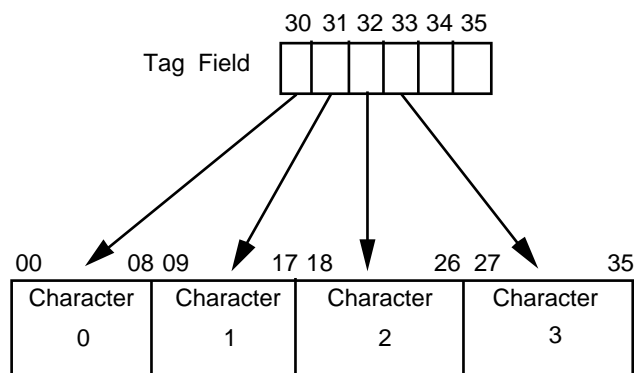
Any

SUMMARY:

9-bit bytes of C(A) —> corresponding characters of C(Y); the byte positions affected are specified in the tag field; C(A) is unchanged.

EXPLANATION:

Binary ones in the tag field specify the byte positions of A and Y affected as indicated in the diagram below. The tag field is entered as one 2-digit octal number. Bit positions 34 and 35 are ignored.



9-bit Byte Positions of A and Y

STBA

ILLEGAL ADDRESS MODIFICATIONS:

The tag field cannot be used for address modification. AR modification is permitted.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

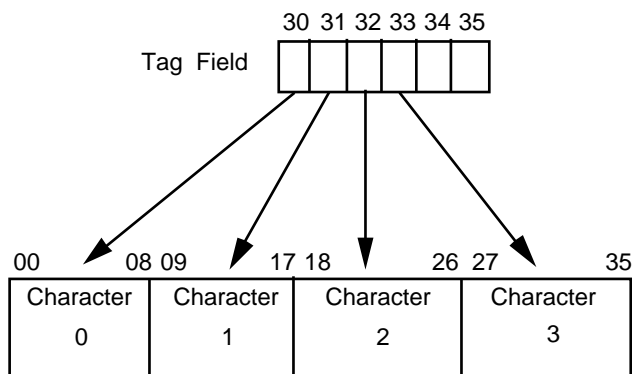
None affected

NOTE:

An Illegal Procedure fault occurs if an illegal repeat is used.

EXAMPLE:

The instruction STBA LOC,04 moves byte 3 of C(A) to the corresponding byte position of C(LOC) (04 octal = 000100 binary). All other byte positions of C(LOC) are unaffected.



9-bit Byte Positions of A and Y

STBQ**13.1.48 STBQ**

STBQ	Store 9-Bit Bytes of Q-Register	552(0)
------	------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

9-bit bytes of C(Q) —> corresponding bytes of C(Y); the byte positions affected are specified in the tag field; C(Q) is unchanged

EXPLANATION:

Binary ones in the tag field specify the byte positions of Q and Y affected as indicated in the diagram below. The tag field is entered as one 2-digit octal number. Bit positions 34 and 35 are ignored.

STBQ

ILLEGAL ADDRESS MODIFICATIONS:

The TAG field cannot be used for address modification. AR modification is permitted.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if an illegal repeat is used.

EXAMPLE:

The instruction STBQ LOC,04 moves byte 3 of C(Q) to the corresponding byte position of C(LOC) (04 octal = 000100 binary). All other byte positions of C(LOC) are unaffected.

STC1**13.1.49 STC1**

STC1	Store Instruction Counter Plus 1	554(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS/ES Mode

$C(IC)+1 \rightarrow C(Y)(0-17)$
 $C(IR) \rightarrow C(Y)(18-32)$
 $000 \rightarrow C(Y)(33-35)$
 $C(IC), C(IR)$ unchanged

EI Mode

$00 \rightarrow C(Y)(0-1)$
 $C(IC)(0-33)+1 \rightarrow C(Y)(2-35)$
 $00...0 \rightarrow C(Y+1)(0-17)$
 $C(IR) \rightarrow C(Y+1)(18-33)$
 $00 \rightarrow C(Y+1)(34-35)$

where Y = the even word of a double-word location

STC1

EXPLANATION:

The relation between the bit positions of C(Y) and the indicators is given below.

<u>Bit Position</u>	<u>Indicator</u>
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	UNUSED
27	IGNORED
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Exponent underflow mask
32	Hexadecimal exponent mode
33	Fixed-point overflow mask
34–35	00

The ON state corresponds to a 1 bit; the OFF state corresponds to a 0 bit. Bit 25 of C(Y) will contain the state of the Tally Runout indicator before address modification of the STC1 instruction (for tally operations).

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STC2**13.1.50 STC2**

STC2	Store Instruction Counter Plus 2	750(0)
------	-------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NS/ES Mode

$C(IC)+2 \rightarrow C(Y)(0-17)$

$C(Y)(18-35), C(IC)$ unchanged

EI Mode

00 $\rightarrow C(Y)(0-1)$

$C(IC)(0-33)+2 \rightarrow C(Y)(2-35)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STC2

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STCA**13.1.51 STCA**

STCA	Store 6-Bit Characters of A-Register	751(0)
------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

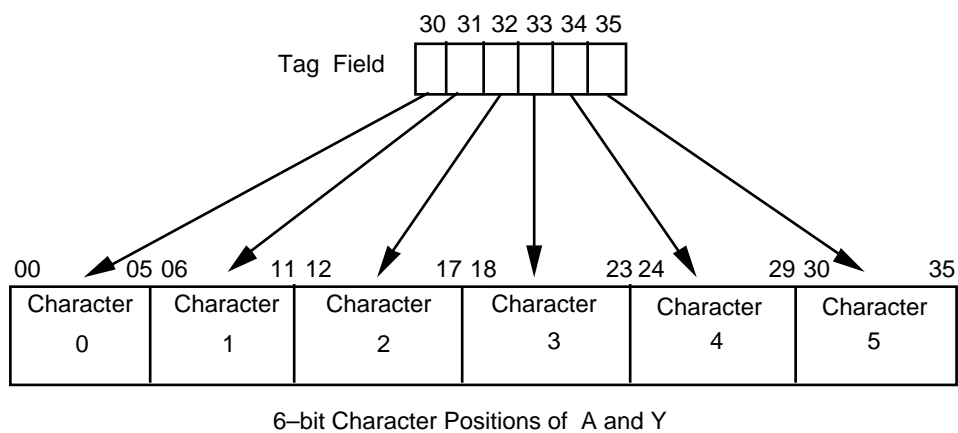
Any

SUMMARY:

6-bit characters of C(A) → corresponding characters of C(Y); the character positions affected are specified in the tag field; C(A) is unchanged

EXPLANATION:

Binary (1) bits in the tag field specify the affected A and Y character locations as follows. The TAG field is entered as one 2-digit octal number. (See Example below.)



STCA

The CPU reads one word from memory, embeds a character specified in the CPU into the word, and then writes this word back in memory. Therefore, while the CPU reads a word and writes it, it is possible that the word's content can be lost if another CPU writes the same word. To prevent multiprocessor contention, gating is necessary.

ILLEGAL ADDRESS MODIFICATIONS:

No modification except AR allowed

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. The tag field cannot be used for address modification. AR modification is permitted.
2. An Illegal Procedure fault occurs if illegal repeats are used.

EXAMPLE:

The instruction STCA LOC,07 moves characters 3, 4, and 5 of C(A) to corresponding character positions of C(LOC) (07 octal = 000111 binary). Character positions 0, 1, and 2 of C(LOC) are unaffected.

STCQ**13.1.52 STCQ**

STCQ	Store 6-Bit Characters of Q-Register	752(0)
------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

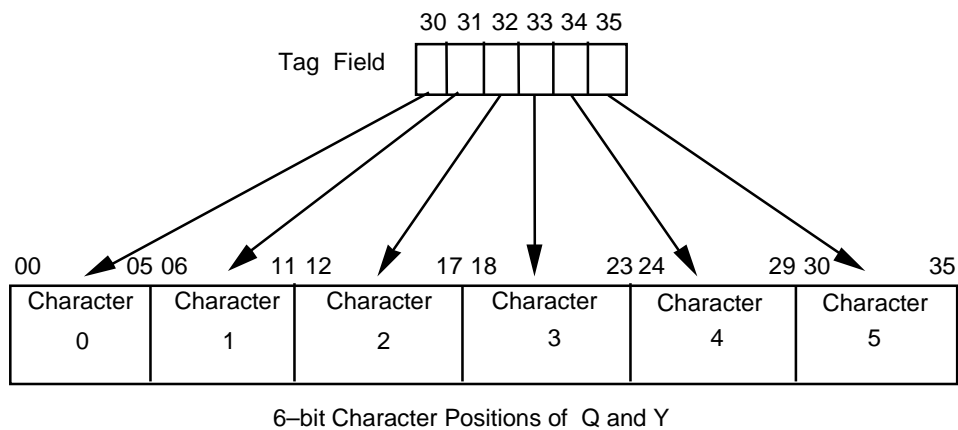
Any

SUMMARY:

6-bit characters of C(Q) → corresponding characters of C(Y); the character positions affected are specified in the tag field.

EXPLANATION:

Binary (1) bits in the tag field specify the affected Q and Y character locations as follows. The tag field is entered as one 2-digit octal number. (See Example below.)



STCQ

The CPU reads one word from memory, embeds a character specified in the CPU into the word, and then writes this word back in memory. Therefore, while the CPU reads a word and writes it, it is possible that the word's content can be lost if another CPU writes the same word. To prevent multiprocessor contention, gating is necessary.

ILLEGAL ADDRESS MODIFICATIONS:

No modification except AR allowed.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. The tag field cannot be used for address modification. AR modification is permitted.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

The instruction STCQ LOC,07 moves characters 3, 4, and 5 of C(Q) to corresponding character positions of C(LOC) (07 octal = 000111 binary). Character positions 0, 1, and 2 of C(LOC) are unaffected.

STD_n**13.1.53 STD_n**

STD _n	Store Descriptor Register <u>n</u>	05 _n (1)
------------------	------------------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(DR_n) \rightarrow C(Y), C(Y+1); C(DR_n)$ unchanged

EXPLANATION:

This instruction stores the DR_n content in an even/odd location of the segment descriptor segment or the operand segment.

If instruction bit 29 = 0 then $C(DR_n) \rightarrow C(Y\text{-pair})$ in the instruction segment.

If instruction bit 29 = 1 and DR_m descriptor type T = 1,3 (m is selected by instruction bits 0,1,2) then $C(DR_n) \rightarrow C(Y\text{-pair})$ of descriptor segment. Note: DR_n store permission is required.

If instruction bit 29 = 1 and DR_m descriptor type T = 0, 2, 4, 6, 12, 14 then $C(DR_n) \rightarrow C(Y\text{-pair})$ in the operand segment.
Note: DR_n store permission is not required.

To summarize the differences in processing performed due to the differing types of segment descriptors--

If the DR_n segment descriptor is stored in a segment descriptor segment (T = 1 or 3), the page must be a housekeeping page (PTW bit 32 must = 1). When all other conditions (e.g., write permission) are satisfied, the segment descriptor is stored, irrespective of the CPU mode.

STD_n

If an attempt is made to store in the operand segment, the write operation for the housekeeping page is dependent upon the CPU mode as the store flag is not examined by hardware.

ILLEGAL ADDRESS MODIFICATIONS:

If the DR_m type T = 1 or 3, only R type modification is permitted. An IPR fault occurs if DU, DL, RI, IR, or IT is specified.

If the DR_m type T = 0, 2, 4, 6, 12, or 14, an IPR fault occurs when DU, DL, SC, SCR, or CI is specified.

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs when illegal address modification or an illegal repeat is used.
2. If DR_n does not have store permission (bit 18 for T = 8, 9, 11; bit 22 for all other types), an SCL2 fault occurs.
3. If DR_m page is not housekeeping, an SCL1 fault occurs.
4. If DR_m segment or page does not have write permission, an SCL2 fault occurs.
5. If processor is in Master or Slave mode and DR_m page is housekeeping, an SCL1 fault occurs.
6. If DR_m segment or page does not have write permission, an SCL2 fault occurs.
7. If instruction bit 29 = 1 and DR_m descriptor type T = 5 or 7-11, 13, 15, an IPR fault occurs.

STDSA**13.1.54 STDSA**

STDSA	Store Data Stack Address Register	150(1)
-------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

C(DSAR) -> C(Y)(0-16)
00...0 -> C(Y)(17-35)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if this instruction is executed in Slave or Master mode.

STDSA

EXAMPLE:

1	8	16

	STDSD	SVREG
	STDSA	SVREG+2
	LDX0	SVREG+2
	ADLX0	NWPS , DU
	CMPX0	SVREG
	TPNZ	NOGOOD
	LDD	P . DS , DSVEC
	.	
	.	
	.	
SVREG	8BSS	8
DSVEC	FVEC	NWDS , (ALL)

STDS**13.1.55 STDS**

STDS	Store Data Stack Descriptor Register	551(1)
------	--------------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(DSDR) \rightarrow C(Y\text{-pair}); C(DSDR)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if illegal address modifications or illegal repeats are used.
2. A Command fault occurs if this instruction executed in Slave or Master mode.

STE

13.1.56 STE

STE	Store Exponent Register	456(0)
-----	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(E) → C(Y)(0-7);
 00...0 → C(Y)(8-17);
 C(Y)(18-35), C(E) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used.

STI**13.1.57 STI**

STI	Store Indicator Register	754(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:SV mode:

C(IR) -> C(Y)(18-33);
 00 -> C(Y)(43-35);
 C(Y)(0-17), C(IR) unchanged

STI

EXPLANATION:

The content of the indicator register is stored in C(Y)(18-33) after address modification. The value stored in C(Y)(25) is the Tally Runout status before address modification. The relation between bit positions of C(Y) and indicators is shown below.

<u>Bit Location</u>	<u>Indicator</u>
18	Zero
19	Negative
20	Carry
21	Overflow
22	Exponent overflow
23	Exponent underflow
24	Overflow mask
25	Tally runout
26	UNUSED
27	IGNORED
28	Master mode
29	Truncation
30	Multiword instruction interrupt
31	Exponent underflow mask
32	Hexadecimal exponent mode
33	Fixed-point overflow mask
34-35	00

The ON state corresponds to a 1 bit; the OFF state to a 0 bit.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STI

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STMB

13.1.58 STMB

STMB	Store Performance Monitor Mode	775(1)
------	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

where $n = y(14-17)$:
 $C(PMCh) \rightarrow C(Y)$

EXPLANATION:

The content of the performance counter selected by bits 14-17 of the instruction y field is stored in memory.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STMB

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. The control must be set to IDLE, READ mode (counting stopped) by the LDMB instruction; else the STMB instruction operation is not defined.

STO

13.1.59 STO

STO	Store Option Register	152(1)
-----	-----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

C(DSCF) → bit 18 of C(Y)
 C(SSBF) → bit 19 of C(Y)
 00...0 → remaining 34 bits of C(Y)

EXPLANATION:

This instruction stores the two flag bits of the option register in memory.

DSCF	<u>Data stack clear flag</u> 0 = do not clear 1 = clear
SSBF	<u>Safe store bypass flag</u> 0 = bypass safe store during ICLIMB 1 = perform safe store during ICLIMB

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

STO

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

STP_n

13.1.60 STP_n

STP _n	Store Pointer _n	45 _n (1)
------------------	----------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

C(AR_n) → C(Y)(0-23)
 C(SEGID_n) → C(Y)(24-35)

ES/EI Mode

C(AR_n) → C(Y)
 C(SEGID_n) → C(Y+1)(0-11)
 00...0 → C(Y+1)(12-35)

EXPLANATION:

These instructions store the address register (AR_n) and the associated segment identity register, (SEGID_n), in memory. The contents of the registers remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

EXAMPLE:

1	8	16	32
NEPR	EPPR	P0,FANY	error handler
	STP	P0,.SVFLT,,P.SSA	store pointer 0
	LDP	P0,.PS,DL	old argument segment
	LDP	P1,.SSR,DL	safe store
	LDD	P0,0,,P0	get argument 0
	LDD	P1,.WLSR,,P1	get original linkage segment
	LDA	0,,P0	get EPPA pointer
	CNAA	=020160,DL	test null descriptor
	TZE	FANY	

STPS

13.1.61 STPS

STPS	Store Parameter Segment Register	751(1)
------	----------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(PSR) \rightarrow C(Y, Y+1)$

EXPLANATION:

The current contents of the parameter segment register (PSR) are to be stored in even and odd memory locations Y and Y+1. The contents of the PSR remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STPS**NOTE:**

An IPR fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

(PMME processing)

1	8	16	32

	STPS	.STEMP,,P.SSA	STASH PSR
	LDA	STEMP,,P.SSA	
	CANA	.FBT27,DL	ANY PARAMETERS?
	TZE	NOPARM	NO,XFER
	LDP	P1,.PS	0,DL+YES, GET FIRST

STQ

13.1.62 STQ

STQ	Store Q-Register	756(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(Q) \rightarrow C(Y); C(Q)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An IPR fault occurs if illegal address modifications or illegal repeats are used.

STQB**13.1.63 STQB**

STQB	Store Q-Register by Byte	562(0)
------	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any, except NS mode which will cause IPR to occur.

SUMMARY:

C(Q)(low order byte) -> C(Y)(byte as specified);
C(Q) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STQH

13.1.64 STQH

STQH	Store Q-Register by Halfword	572(0)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any, except NS mode which will cause IPR to occur.

SUMMARY:

C(Q)(low order halfword) -> C(Y)(halfword as specified);
C(A) unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STSS**13.1.65 STSS**

STSS	Store Safe Store Register	753(1)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(SSR)(0-35) \rightarrow C(Y)(0-35)$
 $C(SSR)(36-69) \rightarrow C(Y+1)(0-33)$

The SCR content is stored in memory in the following way.

$C(SCR) \rightarrow C(Y+1)(34, 35)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

STSS

NOTES:

1. An Illegal Procedure fault occurs when illegal address modification or an illegal repeat is used.
2. A Command fault occurs if the processor is in Slave or Master mode and this instruction is executed.

EXAMPLE:

1	8	16	32

SOVTE	NULL		
	LDP	P0,SD.PSH,DL	copy push segment descr to P0
	LPD	P0,.CTYP,DL	change push descriptor type
	STSS	.SSSR,,P.SSA	store SSR
	LDA	.SSSR+1,,P.SSA	SSR base
	ADA	1K*4,DL	+1K words
	ORA	=O7777,DL	adjust page bounds
	STA	.SVFLT+1,,P.SSA	save it
	SBA	192*4,DL	
	EAX2	1,3	
	LDQ	PH.SS,,P0	original SSR bound + base
	QRL	16	
	ADQ	PH.SS,,P0	max virtual addr, safe store
	CMPQ	.SVFLT+1,,P.SSA	
	EAX2	0	
	SBA	.SSSR+1,,P.SSA	get new bound
	ALS	16	
	STA	.SVFLT+1,,P.SSA	store new bound
	LDP	P1,SD.DGS,DL	load DGS segment descriptor
	LDP	P0,SD.DGS,DL	
	LDP	P0,.CTYP,DL	change type GDS descriptor
	LXL0	POINT,7	
	LDAQ	0,0,P0	
	STAQ	.SSSR,,P.SSA	store current contents
	STSS	0,0,P0	store SSR to generate page load segment
*			
	LDA	0,0,P0	
	ANA	=O177777,DL	
	ORA	.SVFLT+1,,P.SSA	set new bound
	STA	0,0,P0	
	LDD	P2,0,0,P1	load new safe store descr.

STT**13.1.66 STT**

STT	Store Timer Register	454(0)
-----	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:NVM mode:

C(TR) -> C(Y)(0-26)
00...0 -> C(Y)(27-35)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. Bit 26 has a significance of 1/512 millisecond.
2. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STWS

13.1.67 STWS

STWS	Store Working Space Registers	752(1)
------	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

The content of four WSRn registers are stored into a single (if SV mode) memory location specified by Y. Selection of the four (n = 0,1,2,3 or n = 4,5,6,7) is determined by the lower two bits of the EA as defined below.

For the table below: if NS mode, Effective Double Word (EDW) = EA(16-17); else, EDW = EA(32-33).

SV Mode		
C(Y) bits:	EDW = x0	EDW = x1
00-08	WSR0	WSR4
09-17	WSR1	WSR5
18-26	WSR2	WSR6
27-35	WSR3	WSR7

EXPLANATION:

The contents of WSR0 to WSR3, or WSR4 to WSR7 are stored in memory location Y, in accordance with the setting of the EA value.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

STWS**ILLEGAL REPEATS:**

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.
2. A Command fault occurs if the processor is in Slave or Master mode and this instruction is executed.

EXAMPLE:

1	8	16	32

TODES	NULL		
	STWS	WSR	store WSR 0-3
	STWS	WSR+1	store WSR 4-7, store contents
	.		
	.		
	EVEN		
WSR	BSS	2	

STX_n

13.1.68 STX_n

STX _{<u>n</u>}	Store Index Register <u>n</u> in Upper	74 _{<u>n</u>} (0)
-------------------------	---	----------------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(X_n) \rightarrow C(Y)(0-17)$
 $C(Y)(18-35)$ unchanged

ES/EI Mode

For $n = 0, 1, \dots, 7$ as determined by op code:
 $C(GX_n) \rightarrow C(Y)$

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT or RPD of SSSX0

RPL of any SSSX_n

INDICATORS:

None affected

STX_n

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

STZ

13.1.69 STZ

STZ	Store Zero	450(0)
-----	------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

00...0 → C(Y)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL

ILLEGAL REPEATS:

RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SVMMR**13.1.70 SVMMR**

SVMMR	Store VM Mode Register	721(1)
-------	------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

C(VMMR) → C(Y)(00-01)
 00...0 → C(Y)(02-35)

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SVMOS

13.1.71 SVMOS

SVMOS	Start VM Operating System Mode	711(1)
-------	--------------------------------	--------

NOTE: Virtual Machine Facility (VMF) is not implemented on the V9000. Any attempt to enter VMM or VMOS mode causes an IPR fault.

SVMTR**13.1.72 SVMTR**

SVMTR	Store Virtual Machine Timer Register	723(1)
-------	--------------------------------------	--------

NOTE: Virtual Machine Facility (VMF) is not implemented on the V9000. Any attempt to enter VMM or VMOS mode causes an IPR fault. An Illegal Procedure fault occurs if execution is attempted in any mode other than VMM mode.

SWCA

13.1.73 SWCA

SWCA	Subtract With Carry from A-Register	171(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

If carry indicator is ON:

$C(A) - C(Y) \rightarrow C(A)$
 $C(Y)$ unchanged

If carry indicator is OFF:

$C(A) - C(Y) - 00\dots1 \rightarrow C(A)$
 $C(Y)$ unchanged

EXPLANATION:

This instruction is identical to SBA except that, when the carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least-significant position.

This instruction is intended for use with multiword precision arithmetic.

If carry indicator is ON, then $C(A) + 1$'s complement of:

$C(Y) + 00\dots1 \rightarrow C(A)$

If carry indicator is OFF, then $C(A) + 1$'s complement of:

$C(Y) \rightarrow C(A)$

The positive 1 is added when ON represents the carry from the next less-significant part of the multiword subtraction.

ILLEGAL ADDRESS MODIFICATIONS:

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(A) = 0$, then ON; otherwise, OFF
Negative	If $C(A)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of A is exceeded, then ON
Carry	If a carry out of bit 0 of C(A) is generated, then ON; otherwise, OFF

SWCQ

13.1.74 SWCQ

SWCQ	Subtract With Carry from Q-Register	172(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

If carry indicator is ON:

$C(Q) - C(Y) \rightarrow C(Q)$
 $C(Y)$ unchanged

If carry indicator is OFF:

$C(Q) - C(Y) - 0\dots1 \rightarrow C(Q)$
 $C(Y)$ unchanged

EXPLANATION:

This instruction is identical to SBQ except that, when the carry indicator is OFF at the beginning of the instruction, a positive 1 is subtracted from the least-significant position.

This instruction is intended for multiword-precision arithmetic.

If carry indicator is ON, then $C(Q) + 1$'s complement of:

$C(Y) + 00\dots1 \rightarrow C(Q)$

If carry indicator is OFF, then $C(Q) + 1$'s complement of

$(Y) \rightarrow C(Q)$

The positive 1 is added when ON represents the carry from the next less-significant part of the multiword subtraction.

SWCQ**ILLEGAL ADDRESS MODIFICATIONS:**

None

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(Q) = 0$, then ON; otherwise, OFF
Negative	If $C(Q)(0) = 1$, then ON; otherwise, OFF
Overflow	If range of Q is exceeded, then ON
Carry	If a carry out of bit 0 of C(Q) is generated, then ON; otherwise, OFF

EXAMPLE:

(Triple-precision binary fixed-point subtraction)

1	8	16	32

	STI	C	set overflow mask ON
	LDA	=1B24,DL	
	ORSA	C	
	LDI	C	
	LDQ	A+2	subtract low-order bits
	SBLQ	B+2	
	STQ	C+2	
	LDQ	A+1	subtract intermediate bits
	SWCQ	B+1	
	STQ	C+1	
	STI	C	set overflow/overflow mask OFF
	LDA	=0733777,DL	
	ANSA	C	
	LDI	C	
	LDQ	A	subtract high-order bit
	SWCQ	B	
	STQ	C	
A	DEC	9,8,7	
B	DEC	6,5,4	
C	BSS	3	

SWD/SWDX

13.1.75 SWD/SWDX

SWD/ SWDX	Subtract Word Displacement from Address Register	527(1)
--------------	---	--------

FORMAT:

Special arithmetic instruction format (see Figure 8-3)

CODING FORMAT:

1	8	16

	{SWD} {SWDX}	word displacement, R, AR

When the mnemonic is coded with X (AWDX), bit 29 is forced to zero.

OPERATING MODES:

Any

SUMMARY:

If bit 29 = 1:

$$C(\underline{AR}_n)(0-17) - (y + C(DR)) \rightarrow \underline{AR}_n(0-17)$$

If bit 29 = 0:

$$-(y + C(DR)) \rightarrow \underline{AR}_n(0-17)$$

In either case:

$$00\dots0 \rightarrow \underline{AR}_n(18-23)$$

SWD/SWDX**EXPLANATION:**

The y field (with bit 3 extended) is added to the contents of the register specified by the code in the DR field. Then, if bit 29 = 0, this value replaces bits 0-17 of the AR specified by bits 0-2 of the y field. If bit 29 = 1, this value is subtracted from bits 0-17 of the specified AR and the result is stored in bits 0-17 of the specified AR. In either case, bits 18-23 of the specified AR are zeroed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, or IC specified in DR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

Applies to NS mode only

1	8	16	32

	EAX5	2	
	SWDX	2, 5, 4	AR4 octal contents - 77777400
	SWD	0, 5, 4	AR4 octal contents - 77777200
	EAX4	1	
	SWDX	4, 4, 7	AR7 octal contents - 77777300
	SWD	1, 4, 7	AR7 octal contents - 77777100

SXL_n

13.1.76 SXL_n

SXL _{<u>n</u>}	Store Index Register <u>n</u> in Lower	44 <u>n</u> (0)
-------------------------	---	-----------------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

NS Mode

For N=0,1,...,7 as determined by op code:
 $C(X_n) \rightarrow C(Y)(18-35);$
 $C(Y)(0-17)$ unchanged

ES/EI Mode

For N=0,1,...,7 as determined by op code:
 $C(GX_n)(18-35) \rightarrow C(Y)(18-35);$
 $C(Y)(0-17)$ unchanged

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT or RPD of SXL0

RPL of any SXL_n

INDICATORS:

None affected

NOTE:

An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

SZNC**13.1.78 SZNC**

SZNC	Set Zero and Negative Indicators from Storage and Clear	214(0)
------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

EXPLANATION:

C(Y) is tested and the indicators are set in accordance with the result. C(Y) is then zeroed.

This instruction is used for a gating operation in multiple CPU systems. Execution of the next instruction is delayed until the cache-flush request applied to all CPUs has completed.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

None

SZNC

INDICATORS:

Zero If $C(Z) = 0$, then ON; otherwise, OFF

Negative If $C(Z)(0) = 1$, then ON; otherwise, OFF

<u>Zero</u>	<u>Negative</u>	<u>Relationship</u>
0	0	Number $C(Y) > 0$
1	0	Number $C(Y) = 0$
0	1	Number $C(Y) < 0$

NOTE:

An Illegal Procedure fault occurs if illegal address modification is used.

SZTL**13.1.79 SZTL**

SZTL	Set Zero and Truncation Indicators with Bit Strings Left	064(1)
------	--	--------

FORMAT:

00 01	04 05	08 09	10 11	17 18	27 28	29	35	
F	0000	BOLR	T	0	MF2	064(1)	I	MF1

00	02 03	17 18	19 20	23 24	32	35
Y1		C1	B1	N1		
AR#	Y1			00000000	R1	

00	02 03	17 18	19 20	23 24	32	35
Y2		C2	B2	N2		
AR#	Y2			00000000	R2	

CODING FORMAT:

1	8	16	

	SZTL	(MF1), (MF2), BOLR, F, T	
	BDSC	LOCSYM, N, C, B, AM	
	BDSC	LOCSYM, N, C, B, AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SZTL

SUMMARY:

`C(string 1) : (BOLR) : C(string 2)`

EXPLANATION:

The string of bits starting at location YCB1 is evaluated, bit by bit, with the string starting at location YCB2 until either the resultant bit from the BOLR field is a 1 or until L2 is exhausted. If L1 is greater than L2, the Truncation indicator is set.

If L1 is less than L2, the fill bit (F) is used as the L2-L1 least-significant bits of string 1. The contents of both strings remain unchanged.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero	If all the resultant bits generated are zero, then ON; otherwise, OFF
Truncation	If L1 is > L2, then ON; otherwise, OFF

NOTES:

1. An Illegal Procedure fault occurs if DU or DL modification is used for MF1 or MF2 or if illegal repeats are used.
2. An IPR fault does not occur even when $L_1 = 0$ or $L_2 = 0$. In this case, the zero and truncation indicators are affected.

SZTL**EXAMPLES:**

1	8	16	32
	SZTL	,,6	exclusive OR operation
	BDSC	FLD1,36,0,0	FLD1 operand descriptor
	BDSC	FLD2,35,0,1	FLD2 operand descriptor
	TZE	ALLOFF	zero indicator ON
	TRTN	TRUNC	truncation indicator ON
	USE	CONST.	memory contents in octal
FLD1	DEC	-1	777777777777
FLD2	DEC	-1	777777777777
	USE		indicators set? - zero/trunc
	LDI	0,DL	
	LDX7	-1,DU	load negative value into X7
	STI	FLD1	store processor indicators
	SZTL	,,1	AND operation
	BDSC	FLD1,1,2,1	FLD1 operand descriptor
	BDSC	FLD2,1,2,1	FLD2 operand descriptor
	TNZ	19ON	not zero - neg indicator ON
	USE	CONST.	memory contents in octal
FLD1	BSS	1	x x x x x x 2 0 0 0 0 0
FLD2	DEC	1B19	0 0 0 0 0 0 2 0 0 0 0 0
	USE		indicators set? - none

SZTR

13.1.80 SZTR

SZTR	Set Zero and Truncation Indicators with Bit Strings Right	065(1)
------	---	--------

FORMAT:

00 01	04 05	08 09	10 11	17 18	27 28	29	35
F	0000	BOLR	T	0	MF2	065(1)	MF1

00	02 03	17 18	19 20	23 24	32	35
Y1		C1	B1	N1		
AR#	Y1			00000000	R1	

00	02 03	17 18	19 20	23 24	32	35
Y2		C2	B2	N2		
AR#	Y2			00000000	R2	

CODING FORMAT:

1	8	16

SZTR	(MF1) , (MF2) , BOLR , F , T	
BDSC	LOCSYM , N , C , B , AM	
BDSC	LOCSYM , N , C , B , AM	

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

SZTR

SUMMARY:

`C(string 1) :(BOLR): C(string 2)`

EXPLANATION:

Same as for SZTL except that starting locations are YCB1 + (L1-1) and YCB2 + (L2-1) and the evaluation is from right to left (least-significant bit to most significant bit). Any fill (used in comparison) is of most-significant bits.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1 and MF2

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Same as for SZTL

NOTE:

Notes for SZTR are the same as for SZTL.

SZTR

EXAMPLES:

1	8	16	32

	SZTR	,,3,1	evaluate FLD1 as is (move)
	BDSC	FLD1,1,2,1	FLD1 operand descr. (bit 19)
	BDSC	0,1	FLD2 operand descriptor
	TNZ	19ON	
	USE	CONST.	memory contents in octal
FLD1	DEC	1B19	0 0 0 0 0 0 2 0 0 0 0 0
	USE		indicators set? - zero
	LDI	0,DL	clear processor indicators
	LDX7	0,DU	load zeros into X7
	STI	FLD1	store processor indicators
	SZTR	,,14	invert
	BDSC	FLD1,1,2,0	FLD1 operand descr. (bit 18)
	BDSC	0,1	FLD2 operand descriptor
	TZE	18ON	zero indicator ON
	USE	CONST.	memory contents in octal
FLD1	BSS	1	x x x x x x 4 0 0 0 0 0

14. Machine Instruction Descriptions (T-U)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 14.1, Machine Instruction Descriptions (T)
- Section 14.2, Machine Instruction Descriptions (U)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

TCT

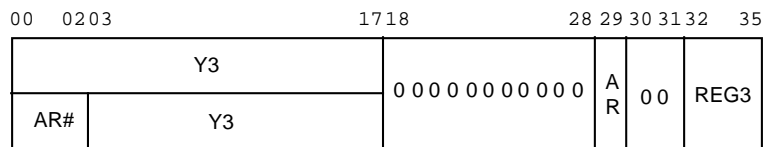
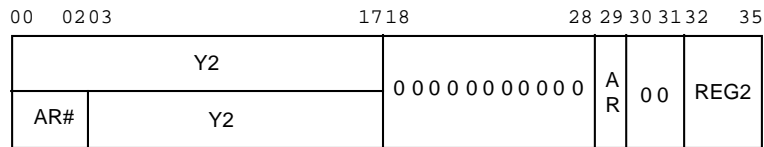
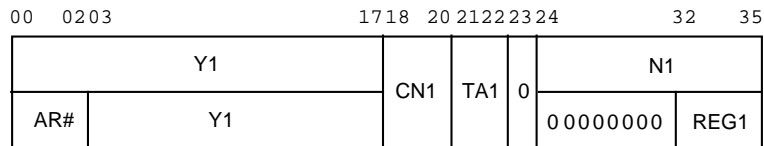
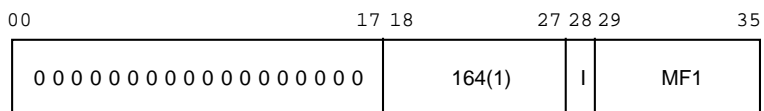
14.1 Machine Instruction Descriptions (T)

Following is a detailed description of the processor instructions and operation codes beginning with the letter T.

14.1.1 TCT

TCT	Test Character and Translate	164(1)
-----	------------------------------	--------

FORMAT:



TCT

NOTES:

1. If N1=0, zero is stored in Y3 (bits 12-35) and the tally indicator is affected.
2. If N1>0 and a match is found in the first character, zero is stored in Y3 (bits 12-35) and the tally indicator is not affected.
3. An Illegal procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16	32

	TCT		no modification
	ADSC6	FLD1,0,12	indexing string operand descr.
	ARG	TABLE	pointer to table
	ARG	FLD3	pointer to char. & count word
	TTF	FOUND	nonzero character found
	USE	CONST.	memory contents
FLD1	BCI	2, 1234567890#	200102030405060710110013 (oct)
FLD3	BSS	1	char. & count - 020000000013
*			Octal
*		0 1 2 3 4 5 6 7	Index
TABLE	OCT	000000000000,000000000000	0X
	OCT	000000020020,020020020020	1X
	OCT	000000000000	2X
	USE		Result - nonzero char. found

NOTE: The highest possible value in FLD1 is an octal 20, a "blank".

EXAMPLE WITH ADDRESS MODIFICATION:

1	8	16	32

X6	BOOL	16	
	EAX2	2	put 2 into X2
	EAX3	FLD1	put FLD1 address into X3
	EAX6	6	put 6 into X6
	AWDX	0,3,7	put FLD1 address into AR7
	TCT	(1,1,1,2)	with all modification options
	ARG	INDSCR	ptr. indirect operand descr.
	ARG	FLD3	pointer to FLD3
	TTF	*+2	nonzero found
	NULL		tally runout ON
	USE	CONST.	memory contents
*			
FLD1	ASCII	2, 1234;5	040040061062063064073065 (oct)
FLD3	BSS	1	char. and count 040000000004
INDSCR	ADSC9	0,0,X7,7	indexing FLD1 operand descr.
TABLE	BSS	12	generate 60 (oct) table chars.
	OCT	0000000000000,0000000000000	(060-067)
	OCT	0000000000040	(070-073)
	USE		Result - nonzero found

NOTE: The highest possible value in FLD1 is an octal 073, a ";".

TCTR

14.1.2 TCTR

TCTR	Test Character and Translate in Reverse	165(1)
------	--	--------

FORMAT:

Same as Test Character and Translate (TCT) format

CODING FORMAT:

1	8	16

	TCTR	(MF1)
	ADSC _n	LOCSYM, CN, N, AM
	ARG	LOCSYM, RM, AM
	ARG	LOCSYM, RM, AM

(Refer to Section 7 under Multiword Instructions for description of Multiword Modification Field.)

OPERATING MODES:

Any

EXPLANATION:

Same as TCT except start at location YC1 + (L1-1) and progress toward YC1

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL for MF1, REG2, REG3

ILLEGAL REPEATS:

RPT, RPD, RPL

TEO

14.1.3 TEO

TEO	Transfer on Exponent Overflow	614(0)
-----	-------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TEO	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

If exponent overflow indicator ON, then:
 Y -> C(IC)

If exponent overflow indicator ON,
 and instruction bit 29=1 then:

n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If exponent overflow indicator ON, then:
 Y(16-33) -> C(IC)

If exponent overflow indicator ON,
 and instruction bit 29=1 then:

n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Exponent Overflow - Set OFF

TEO

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TEU**14.1.4 TEU**

TEU	Transfer on Exponent Underflow	615(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TEU	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If exponent underflow indicator ON, then:
 Y -> C(IC);

If exponent underflow indicator ON,
 and instruction bit 29=1, then:

n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If exponent underflow indicator ON, then:
 Y(16-33) -> C(IC);

If exponent underflow indicator ON,
 and instruction bit 29=1, then:

n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TEU**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation as follows:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Exponent Underflow - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TMI

14.1.5 TMI

TMI	Transfer on Minus	604(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TMI	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

If negative indicator ON, then:
 Y -> C(IC);

If negative indicator ON and instruction bit 29=1, then:
 n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If negative indicator ON, then:
 Y(16-33) -> C(IC);

If negative indicator ON and instruction bit 29=1, then:
 n = Y(0-2);
 C(DR_n) -> C(ISR);
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- When bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TMI

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TMOZ**14.1.6 TMOZ**

TMOZ	Transfer on Minus or Zero	604(1)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TMOZ	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If negative indicator ON or Zero indicator ON, then:
Y → C(IC);

If negative indicator ON or Zero indicator ON; and
instruction bit 29=1, then:

n = Y(0-2);
C(DR_n) → C(ISR);
C(SEGID_n) → C(SEGID(IS))

ES Mode

If negative indicator ON or Zero indicator ON, then:
Y(16-33) → C(IC);

If negative indicator ON or Zero indicator ON, and
instruction bit 29=1, then:

n = Y(0-2);
C(DR_n) → C(ISR);
C(SEGID_n) → C(SEGID(IS))

EI Mode

Y(0-33) → C(IC)(0-33)

TMOZ

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TMOZ**NOTES:**

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

1	8	16	32
	LCQ	2, DL	
	TMOZ	NOPLUS	transfer on minus or zero
	NULL		plus routine
*	DID TRANSFER OCCUR?	YES	TO WHAT LOCATION? NOPLUS

TNC

14.1.7 TNC

TNC	Transfer on No Carry	602(0)
-----	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	

	TNC	LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

NS Mode

If carry indicator OFF, then:
 Y -> C(IC)

If carry indicator OFF and instruction bit 29=1, then:
 n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If carry indicator OFF, then:
 Y(16-33) -> C(IC)

If carry indicator OFF and instruction bit 29=1, then:
 n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TNC

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TNZ**14.1.8 TNZ**

TNZ	Transfer on Nonzero	601(0)
-----	---------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TNZ	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If zero indicator OFF, then:
Y -> C(IC)

If zero indicator OFF and instruction bit 29=1, then:

n = Y(0-2)
C(DR_n) -> C(ISR)
C(SEGID_n) -> C(SEGID(IS))

ES Mode

If zero indicator OFF, then:
Y(16-33) -> C(IC)

If zero indicator OFF and instruction bit 29=1, then:

n = Y(0-2)
C(DR_n) -> C(ISR)
C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TNZ**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TOV

14.1.9 TOV

TOV	Transfer on Overflow	617(0)
-----	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TOV	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

If overflow indicator ON, then:
 Y -> C(IC)

If overflow indicator ON and instruction bit 29=1, then:
 n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If overflow indicator ON, then:
 Y(16-33) -> C(IC)

If overflow indicator ON and instruction bit 29=1, then:
 n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(I) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Overflow - Set OFF

TOV

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TPL**14.1.10 TPL**

TPL	Transfer on Plus	605(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TPL	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If negative indicator OFF, then:

Y -> C(IC)

If negative indicator OFF and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

ES Mode

If negative indicator OFF, then:

Y(16-33) -> C(IC)

If negative indicator OFF and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TPL**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TPNZ

14.1.11 TPNZ

TPNZ	Transfer on Plus and Nonzero	605(1)
------	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TPNZ	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

If negative indicator OFF and Zero indicator OFF, then:
 Y -> C(IC)

If negative indicator OFF and Zero indicator OFF
 and instruction bit 29=1, then:

n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If negative indicator OFF and Zero indicator OFF, then:
 Y(16-33) -> C(IC)

If negative indicator OFF and Zero indicator OFF
 and instruction bit 29=1, then:

n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TPNZ

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

1	8	16	32

	EAX5	6	load address modifier into X5
	EAX6	PLUSRT	load transfer address into X6
	AWDX	0,6,6	put transfer address into AR6
	LDA	5,DL	load +5 into A-register
	TPNZ	0,5,6	transfer on plus and nonzero
	NULL		zero and negative routine
*			
*	DID TRANSFER OCCUR?	YES	TO WHAT LOCATION? PLUSRT+6
*			
	EAX2	3	load address modifier into X2
	LDX7	4,DU	load +4 into X7
	TPNZ	TRANS,2	transfer on plus and nonzero
	NULL		zero and negative routine
*			
*	DID TRANSFER OCCUR?	YES	TO WHAT LOCATION? TRANS+3

TRA**14.1.12 TRA**

TRA	Transfer Unconditionally	710(0)
-----	--------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TRA	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

Y -> C(IC);

If instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

ES Mode

Y(16-33) -> C(IC);

If instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TRA

EXPLANATION:

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer in this case is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TRC

14.1.13 TRC

TRC	Transfer on Carry	603(0)
-----	-------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16	

	TRC	LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

NS Mode

If carry indicator ON, then:
 Y -> C(IC)

If carry indicator ON and instruction bit 29=1, then:

n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

ES Mode

If carry indicator ON, then:
 Y(16-33) -> C(IC)

If carry indicator ON and instruction bit 29=1, then:

n = Y(0-2)
 C(DR_n) -> C(ISR)
 C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TRC

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TRCT_n**14.1.14 TRCT_n**

TRCT _n	Transfer on Count <u>n</u>	54 _n (0)
-------------------	----------------------------	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TRCT _n	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

For $n = 0, 1, \dots, 7$ as determined by op code:

If zero indicator OFF and negative indicator ON, then:

$C(X_n) - 1 \rightarrow C(X_n)$

If $C(X_n) \neq 0$,

Y $\rightarrow C(IC)$

If zero indicator OFF and negative indicator ON and instruction bit 29=1, then:

$m = Y(0-2)$

$C(DR_m) \rightarrow C(ISR)$

$C(SEGID_m) \rightarrow C(SEGID(IS))$

TRCT_nES Mode

For $n = 0, 1, \dots, 7$ as determined by op code:

If zero indicator OFF and negative indicator ON, then:

$C(GX_n) - 1 \rightarrow C(X_n)$

If $C(GX_n) \neq 0$,

$Y(16-33) \rightarrow C(IC)$

If zero indicator OFF and negative indicator ON and instruction bit 29=1, then:

$m = Y(0-2)$

$C(DR_m) \rightarrow C(ISR)$

$C(SEGID_m) \rightarrow C(SEGID(IS))$

EI Mode

$Y(0-33) \rightarrow C(IC)(0-33)$

EXPLANATION:

A value of 1 is subtracted from the content of X_n/GX_n and the result is loaded into X_n/GX_n . Unless the content of the result in X_n/GX_n is zero, control is transferred to the location specified by the y field. If the result is 0, the next instruction is executed.

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_m selected with bits 0, 1, 2, and the corresponding $SEGID_m$, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_m (not the ISR) are used in developing the addresses of indirect words.

TRCT_n

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

ILLEGAL EXECUTES:

XEC, XED

INDICATORS:

Zero	If C(X _n /GX _n) = 0, then ON; otherwise, OFF
Negative	If C(X _n /GX _n) = 1, then ON; otherwise, OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications, illegal repeats, or illegal executes are used.

TRCT_n

EXAMPLE:

1	8	16	32

	LDX0	10 ,DU	
	.		
	.		
A	LDA		
	.		
	.		
	TRTC0	A	

TRTF**14.1.15 TRTF**

TRTF	Transfer on Truncation Indicator OFF	601(1)
------	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TRTF	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If truncation indicator OFF, then:
Y → C(IC)

If truncation indicator OFF and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

ES Mode

If truncation indicator OFF, then:
Y(16-33) → C(IC)

If truncation indicator OFF and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

EI Mode

Y(0-33) → C(IC)(0-33)

TRTF**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16	32

	MLR		move alphanumeric left to right
	ADSC9	FLD1,0,4	sending operand descriptor
	ADSC4	FLD2,0,4	receiving operand descriptor
	TRTF	NTRUNC	truncation indicator OFF
	NULL		
*			
*	Did transfer to NTRUNC occur?		YES
*			
*	State of truncation indicator after?		OFF

TRTN

14.1.16 TRTN

TRTN	Transfer on Truncation Indicator ON	600(1)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TRTN	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

If truncation indicator ON, then:
Y → C(IC)

If truncation indicator ON and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

ES Mode

If truncation indicator ON, then:
Y(16-33) → C(IC)

If truncation indicator ON and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

EI Mode

Y(0-33) → C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Truncation - If ON, it is turned OFF

TRTN

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLE:

1	8	16	32

	MLR		move alphanumeric left to right
	ADSC4	FLD1,0,8	sending operand descriptor
	ADSC6	FLD2,0,6	receiving operand descriptor
	TRTN	TRUNC	truncation indicator ON
	TRA	TRUNC+6	truncation indicator OFF
*			
*	To where was transfer?		TRUNC
*			
*	State of truncation indicator after?		OFF
*			
	MLR		move alphanumeric left to right
	ADSC9	FLD1,0,8	sending operand descriptor
	ADSC4	FLD2,0,4	receiving operand descriptor
	TRTN	TRUNC	truncation indicator ON
	NULL		no truncation routine
*	Did transfer of control occur?	yes where to?	TRUNC
*	State of truncation indicator after?		OFF

TSS**14.1.17 TSS**

TSS	Transfer after Setting Slave	715(0)
-----	------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TSS	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

Y -> C(IC)

If instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

ES Mode

Y(16-33) -> C(IC)

If instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TSS

EXPLANATION:

All outstanding memory requests are checked for completion before the Master Mode indicator is reset on the TSS instruction.

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur even when bit 29 of the TSS instruction word is 0.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer in this case is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Master Mode - Set OFF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.
6. For a fault that occurs as a result of execution of a TSS instruction in Master mode, the state of bit 28 (Master Mode indicator) in the copy of the indicator register stored in the safe store frame is determined by one of the following conditions:
 - a. If IPR or Fault Tag fault, caused by the tag field in the instruction or indirect word, then IR28 = 1.
 - b. If Bound fault, caused by attempt to access an indirect word, then IR28 = 1.
 - c. If Bound fault, caused by attempt to access the target location, then IR28 = 1.
7. Using the TSS instruction does not change the contents of the DR or AR registers that may have been set by a previous Master Mode Entry (MME/PMME) and/or user code.

TSX_n

14.1.18 TSX_n

TSX _n	Transfer and Set Index Register <u>n</u>	70 _n (0)
------------------	---	---------------------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

TSX _n		LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:

NS Mode

For n = 0,1,...,7 as determined by op code:

C(IC) + 0...01 → C(X_n);
Y → C(IC)

If instruction bit 29=1, then:

n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

ES Mode (no transfer of a carry from bit 18 of GX_n to high-order bit.)

For n = 0,1,...,7 as determined by op code:

00...0 → C(GX_n)(0-17)
C(IC) + 0...01 → C(GX_n)(18-35);
Y(16-33) → C(IC)

If instruction bit 29=1, then:

n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

EI Mode

00 → C(GX_n) (0-1)
 C(IC) (0-33)+1 → C(GX_n) (2-35)
 Y(0-33) → C(IC) (0-33)

EXPLANATION:

With unconditional transfer of control instructions, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not affected. An IPR fault does not occur.
- When bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(IS). The transfer in this case is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TSX_n

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TSYNC**14.1.19 TSYNC**

TSYNC	Transmit Sync Interrupt	731(1)
-------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

EXPLANATION:

The Transmit Sync Interrupt (TSYNC) instruction loads the A-register with zero to simulate a time-out on the TSYNC.

ILLEGAL ADDRESS MODIFICATIONS:

IT, RI, IR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTE:

This instruction is used on Olympus to test shared cache synchronization. Since a commercial processor manages the processor cache on the V9000, the instruction is implemented as if the instruction timed out, i.e., synchronization failed.

TTF

14.1.20 TTF

TTF	Transfer on Tally Runout Indicator OFF	607(0)
-----	---	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

TTF	LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:

NS Mode

If tally runout indicator OFF, then:
Y → C(IC)

If tally runout indicator OFF and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

ES Mode

If tally runout indicator OFF, then:
Y(16-33) → C(IC)

If tally runout indicator OFF and instruction bit 29=1, then:
n = Y(0-2)
C(DR_n) → C(ISR)
C(SEGID_n) → C(SEGID(IS))

EI Mode

Y(0-33) → C(IC)(0-33)

EXPLANATION:

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

TTF

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

TTN**14.1.21 TTN**

TTN	Transfer on Tally Runout Indicator ON	606(1)
-----	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

The TTN instruction code is shown below.

1	8	16

TTN	LOCSYM, RM, AM	

OPERATING MODES:

Any

SUMMARY:NS Mode

If tally runout indicator ON, then:

Y → C(IC)

If tally runout indicator ON and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) → C(ISR)

C(SEGID_n) → C(SEGID(IS))

ES Mode

If tally runout indicator ON, then:

Y(16-33) → C(IC)

If tally runout indicator ON and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) → C(ISR)

C(SEGID_n) → C(SEGID(IS))

EI Mode

Y(0-33) → C(IC)(0-33)

TTN**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

1	8	16	32

	TCT		test character and translate
	ADSC6	FLD1,0,12	indexing string operand descr
	ARG	TABLE	pointer to table
	ARG	FLD3	operand pointer to count word
	TTN	NMATCH	tally runout ON-nonzero entry
	NULL		tally runout OFF
	USE	CONST.	
TABLE	OCT	,,20020,020020020020,0	
FLD1	BCI	2, 1234567890#	
FLD3	BSS	1	
	USE		
	* Did transfer occur?	no	
	TCT		test character and translate
	ADSC4	FLD1,0,8	indexing string operand descr
	ARG	TABLE	pointer to table
	ARG	FLD3	pointer to char and count word
	TTN	CHAROK	tally runout ON
	TRA	ERROR	tally runout OFF
	USE	CONST.	
TABLE	OCT	,,14014,14014	
FLD1	OCT	022064126317	
	USE		
	* To what location was transfer made?	ERROR	

TZE**14.1.23 TZE**

TZE	Transfer on Zero	600(0)
-----	------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

CODING FORMAT:

1	8	16

	TZE	LOCSYM, RM, AM

OPERATING MODES:

Any

SUMMARY:NS Mode

If zero indicator ON, then:
Y -> C(IC)

If zero indicator ON and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

ES Mode

If zero indicator ON, then:
Y(16-33) -> C(IC)

If zero indicator ON and instruction bit 29=1, then:

n = Y(0-2)

C(DR_n) -> C(ISR)

C(SEGID_n) -> C(SEGID(IS))

EI Mode

Y(0-33) -> C(IC)(0-33)

TZE**EXPLANATION:**

With conditional transfer instructions, if the transfer condition is not satisfied (transfer does not occur), the ISR and the SEGID(IS) are not changed. When transfer occurs, bit 29 of the instruction word affects the operation in the following way:

- when bit 29 of the instruction word = 0, the ISR and SEGID(IS) are not changed;
- when bit 29 of the instruction word = 1, the DR_n selected with bits 0, 1, 2, and the corresponding SEGID_n, are loaded into the ISR and SEGID(S). The transfer, in this case, is the transfer to another segment.

If instruction bit 29=1, and if any form of indirect addressing is specified in the tag field, then the base, bound, and working space from DR_n (not the ISR) are used in developing the addresses of indirect words.

When the transfer instruction attempts to load the ISR, the ISR bit 24 (NS/ES mode specification bit) and the ISR type field cannot be altered. If bit 24 of the ISR before execution of the transfer is not equal to bit 24 of the segment descriptor from the DR_n, an IPR fault occurs. Similarly, if the type field of the ISR before execution of the transfer is not equal to the type field of DR_n, an IPR fault occurs. The ISR bit and type field can be altered only with the CLIMB instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTES:

1. An IPR fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor that is not type T=0; or has a base that is not 0 modulo 32 bytes, or has a bound that is not 31 modulo 32 bytes. If the CPU is in EI mode, and the instruction attempts to load the ISR from a descriptor that is not type 12, an IPR fault occurs.
2. A Security Fault, Class 2 occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 25=0.
3. A Store or Bound fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 27=0.
4. A Missing Segment fault occurs if instruction bit 29=1 and the instruction attempts to load the ISR from a descriptor for which flag bit 28=0.
5. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

UFA

14.2 Machine Instruction Descriptions (U)

Following is a detailed description of the processor instructions and operation codes beginning with the letter U.

14.2.1 UFA

UFA	Unnormalized Floating Add	435(0)
-----	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) + C(Y)]$ not normalized $\rightarrow C(EAQ)$

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR cause an IPR to occur.

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If C(AQ) = 0, then ON; otherwise OFF
Negative	If C(AQ)(0) = 1, then ON; otherwise OFF
Exponent Overflow	If exponent is > +127, then ON
Exponent Underflow	If exponent is < -128, then ON
Carry	If a carry out of bit 0 of C(AQ) is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

EXAMPLE:

(Convert from floating to fixed)

1	8	16	32

FIXIT	MACRO		
	INE	#1, '.EAQ.', 1	
	FLD	#1	
	FCMP	-0110400, DU	2**35
	TMI	2, IC	
	NOF	, F	
	FCMP	=0107000, DU	-2**35
	TMI	02, IC	
	UFA	=71B25, DU	
	INE	#2, '.QR.', 1	
	STQ	#2	
	ENDM	FIXIT	
	FIXIT	X, I	I=X

UFM

14.2.2 UFM

UFM	Unnormalized Floating Multiply	421(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) * C(Y)]$ not normalized $\rightarrow C(EAQ)$

EXPLANATION:

This multiplication is executed like the FMP instruction except that the final normalization is performed only if both factor mantissas are $= -1.00...0$. The definition of normalization is located under the description of the FNO instruction.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON

NOTES:

1. When indicator bit 32=1, the floating-point alignment and normalization is hexadecimal. Otherwise, the floating-point alignment and normalization are binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

UFS

14.2.3 UFS

UFS	Unnormalized Floating Subtract	535(0)
-----	--------------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$[C(EAQ) - C(Y)]$ not normalized $\rightarrow C(EAQ)$

EXPLANATION:

The two's complement of the subtrahend is first taken and the smaller value is then right shifted to equalize it. The shifted out portion is truncated and addition is executed.

ILLEGAL ADDRESS MODIFICATIONS:

CI, SC, SCR

ILLEGAL REPEATS:

None

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF
Exponent Overflow	If exponent is $> +127$, then ON
Exponent Underflow	If exponent is < -128 , then ON
Carry	If a carry out of bit 0 of $C(AQ)$ is generated, then ON; otherwise, OFF

NOTES:

1. When indicator bit 32=1, the floating-point alignment is hexadecimal. Otherwise, the floating-point alignment is binary.
2. An Illegal Procedure fault occurs if illegal address modification is used.

UFTR

14.2.4 UFTR

UFTR	Unnormalized Floating Truncate Fraction	434(0)
------	--	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

$C(EAQ) \text{ fraction-truncated} \rightarrow C(EAQ)$

EXPLANATION:

This instruction truncates the fraction part of the floating-point data of C(EAQ) to obtain an integer. The result is unnormalized and stored into C(EAQ). A proper truncation to an integer is such that truncating the fraction parts of two numbers with the same absolute and different sign and adding the results produces 0.

ILLEGAL ADDRESS MODIFICATIONS:

None The address modification does not affect instruction operations, but the modification is executed.

ILLEGAL REPEATS:

RPL

INDICATORS:

Zero	If $C(AQ) = 0$, then ON; otherwise, OFF
Negative	If $C(AQ)(0) = 1$, then ON; otherwise, OFF

NOTE:

An Illegal Procedure fault occurs if an illegal repeat is used.

Notes

15. Machine Instruction Descriptions (V-X)

This section of the Programmer's Guide continues the alphabetical presentation of the V9000 instruction repertoire begun in Section 8, organized as follows:

- Section 15.1, Machine Instruction Descriptions (V)
- Section 15.2, Machine Instruction Descriptions (W and X)

NOTE: All of the V9000 machine instructions are listed alphabetically by their mnemonics at the end of Section 7 and in Appendix A. This list includes the instruction name and operating code.

15.1 Machine Instruction Descriptions (V)

Following is a detailed description of the processor instructions and operation codes beginning with the letter V.

VMME

15.1.1 VMME

VMME	Virtual Machine Monitor Mode Entry	710(1)
------	---------------------------------------	--------

NOTE: Virtual Machine Facility (VMF) is not implemented on the V9000. Any attempt to enter VMM or VMOS mode causes an IPR fault.

15.2 Machine Instruction Descriptions (W and X)

Following is a detailed description of the processor instructions and operation codes beginning with the letters W and X.

15.2.1 WRES

WRES	Write Reserve Memory	232(0)
------	----------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode

SUMMARY:

$C(A) \rightarrow C[Y + C(\text{Reserve Memory Base Register})]$

EXPLANATION:

The effective address Y is added to the contents of the reserve memory base register. The result produces a real memory address without paging. The contents of the A register is then stored in this address.

Bit 29 of this instruction is ignored. Neither the address register nor the segment descriptor register is used.

The WRES instruction forms a real memory address equal to $RMBR + EA$, where RMBR contains the base of the real RMS.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, RI, IR, IT

WRES

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

None affected

NOTES:

1. A Command fault occurs if execution is attempted in Slave or Master mode.
2. An Illegal Procedure fault occurs with illegal address modification or illegal repeats.

WSYNC**15.2.2 WSYNC**

WSYNC	Wait for Sync Interrupt	732(1)
-------	-------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Privileged Master Mode only

SUMMARY:

Loads the A-register with zero to simulate a time-out on the WSYNC.

ILLEGAL ADDRESS MODIFICATIONS:

IT, RI, IR

ILLEGAL REPEATS:

RPT, RPD, RPL

NOTE:

This instruction is used on Olympus to test shared cache synchronization. Since a commercial processor manages the processor cache on the V9000, the instruction is implemented as if the instruction timed out, i.e., synchronization failed.

XEC

15.2.3 XEC

XEC	Execute	716(0)
-----	---------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Obtain and execute the instruction stored at memory location Y.

EXPLANATION:

The next instruction to be executed is obtained from $C(IC)+1$. This instruction is located in memory immediately following the location containing the XEC instruction. This case does not apply if the execution of the instruction obtained from location Y changes the content of the IC.

To execute a repeat instruction with the XEC instruction, the XEC must reside at an odd location. The instructions to be repeated using the RPT, RPD, or RPL instructions must immediately follow the XEC instruction.

With the exceptions noted in Note 1, an XEC instruction may point to a multiword instruction. However, the descriptors for the multiword instruction must be stored immediately following the XEC instruction. The next instruction to be executed is obtained from $C(IC)+n+1$, where n is the number of descriptors for the multiword instruction.

If IC modification is used with the instruction being executed, the value of IC will be the same as the location of the XEC instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

XEC**ILLEGAL REPEATS:**

RPT, RPD, RPL

INDICATORS:

The XEC instruction itself does not affect any indicator. However, the execution of the instruction from Y may affect indicators.

NOTES:

1. An Illegal Procedure fault occurs if illegal address modification or illegal repeats are used when the XEC instruction is executing an SPL, LPL, CLIMB, or TRCTn instruction.
2. An Illegal Procedure fault occurs if a CLIMB instruction is executed via an XEC instruction.

EXAMPLE:

1	8	16	32
	REM		X7 has value 0 or 1
	REM		X6 has value 1, 2, 3, 4 or 5
	XEC	DOIT,7	add or subtract
	USE	SMARTS	
DOIT	ADQ	FF	
	SBQ	FF	
	USE		
	XEC	BRANCH-1,6	5-way branch
	USE	YERHED	
BRANCH	NOF		
	AOS	FLAG2	
	TRA	.S3	
	TRA	.S4	
	TRA	WRAPUP	
	USE		

XED

15.2.4 XED

XED	Execute Double	717(0)
-----	----------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

Obtain and execute the two instructions stored at the memory Y-pair locations (must be even and next odd location).

EXPLANATION:

The first instruction obtained from Y-pair must not alter the memory location from which the second instruction is obtained, and must not be another XED instruction. If the first instruction obtained from Y-pair alters the contents of the instruction counter, this transfer of control is effective immediately, and the second instruction of the pair is not executed.

After execution of the two instructions obtained from the Y-pair, the next instruction to be executed is obtained from C(IC)+1. This location immediately follows the XED instruction. This does not apply if the execution of the two instructions obtained from the y-pair alters the content of the IC.

To Execute Double (XED) the RPD instruction, the RPD must be the second instruction at an odd-numbered address. When RPD is at the odd-numbered address of the pair, the XED instruction must be at an odd location. In this case, the repeated instructions are those that immediately follow the XED instruction. If RPD is specified within a sequence of XEDs, the original and all subsequent XEDs in the sequence must be in odd locations.

XED

When repeat instructions RPT or RPL are executed with an XED instruction and the first instruction specified by the XED resides an even-numbered location, the repeated instruction is that immediately following the RPT or RPL. When the RPT or RPL instruction resides at an odd-numbered address, the repeated instruction is that immediately following the XED instruction.

With the exceptions noted in Note 1, multiword instructions are executed with the XED instruction. The multiword instruction (second instruction) must be located at an odd-numbered address. If it is not, an IPR fault occurs. The data descriptors for this multiword instruction immediately follow the XED instruction.

If IC modification is used with either of the instructions being executed, the value of IC will be the same as the location of the XED instruction.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

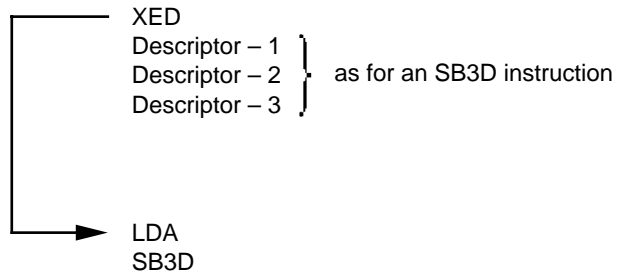
INDICATORS:

The XED instruction itself does not affect any indicator. However, the execution of the two instructions from Y-pair may affect indicators.

XED

NOTES:

1. An IPR fault occurs if XED instruction is used with SPL, LPL, CLIMB, or TRCTn instructions.
2. When multiword instructions other than those indicated in Note 1 are executed with an XED instruction, the multiword instruction must be located at an odd-numbered address (second instruction). If it is not, an IPR fault occurs. The data descriptors for this multiword instruction are those immediately following the XED instruction as indicated below:



3. An Illegal Procedure fault occurs if illegal address modifications or illegal repeats are used.

EXAMPLES:

1	8	16	32
	REM		x7 0 = 0,2,4, or 6
	XED	ENTRY, 7	
	.		
	.		
	EVEN		
ENTRY	NULL		
	STC1	SAVE1	
	TRA	FIRST	
	STC1	SAVE2	
	TRA	SECOND	
	STC1	SAVE3	
	TRA	THIRD	
	STC1	SAVE4	
	TRA	FOURTH	

XRAN**15.2.5 XRAN**

XRAN	Fixed Point Random Number	363(1)
------	---------------------------	--------

FORMAT:

Single-word instruction format (see Figure 8-1)

OPERATING MODES:

Any

SUMMARY:

```

00000000000000000000    -> C(Y)(0-16),
C(Y)(71), C(Y)(72), ...,
  C(Y)(143)                -> C(Y)(17-88),
C(Y)(17-71)               -> C(Y)(89-143),
0                          -> C(Q)(0),
C(Y)(18-52)               -> C(Q)(1-35)

```

[Note: C(Y) is the post-shift value.]

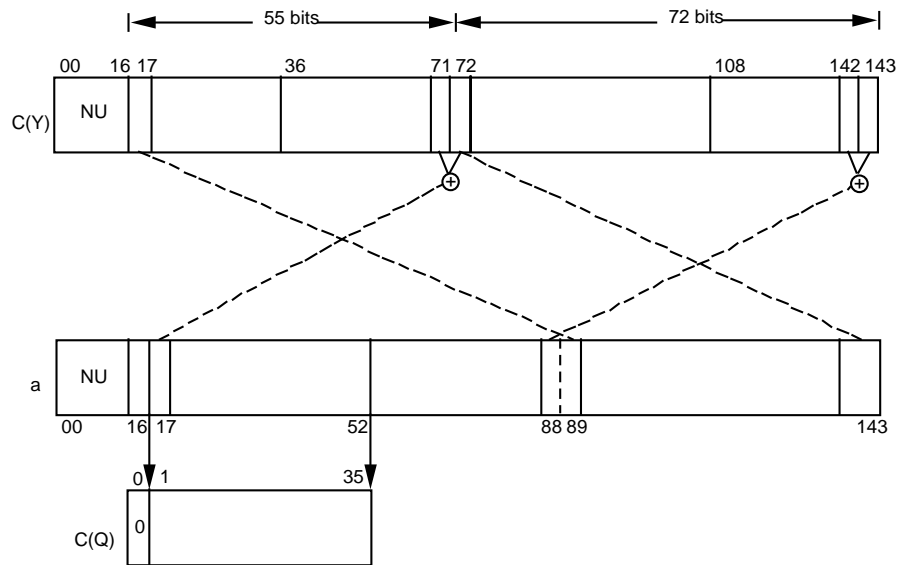
XRAN

EXPLANATION:

The operation shown in the figure below is performed on the low-order 127 bits of C(Y-4 words). The 127 bits of the result (a) are stored into bits 17-143 of the four words beginning at memory location Y. Zero is stored into the high-order bits 0-16 of memory location Y.

Zero is loaded into bit 0 of Q. Bits 18-52 of the result (a) are loaded into bits 1-35 of Q.

The memory location Y must be on a double-word boundary.



This instruction is not an RAR (Read-Alter-Rewrite) type of instruction. It does not cause a read-lock/write-lock command sequence.

A random number is generated by this instruction using a so-called maximum-length shift register sequence (M-sequence). The primitive polynomial is given by $f(X) = x^{127} + x^{126} + 1$, and the period by $2^{127} - 1 \sim 10^{34}$. Execution of this instruction produces uniform random numbers between 0 and $2^{35} - 1$.

ILLEGAL ADDRESS MODIFICATIONS:

DU, DL, CI, SC, SCR

ILLEGAL REPEATS:

RPT, RPD, RPL

INDICATORS:

Zero If $C(Q) = 0$, then ON; otherwise, OFF

Negative OFF (a positive number is always obtained)

NOTE:

An Illegal Procedure fault occurs if illegal address modification or an illegal repeat is used.

Notes

A. NovaScale 9000 Machine Instructions

The following is an alphabetical presentation of all of the machine instructions described in Sections 8-15.

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
A4BD(X)	Add 4-bit Displacement to Address Register	502 (1)	8
A6BD(X)	Add 6-bit Displacement to Address Register	501 (1)	8
A9BD(X)	Add 9-bit Displacement to Address Register	500 (1)	8
AAR \underline{n}	Alphanumeric Descriptor to Address Register \underline{n}	56 \underline{n} (1)	8
ABD(X)	Add Bit Displacement to Address Register	503 (1)	8
ACKS	Acknowledge Sync Interrupt	735 (1)	8
AD2D	Add Using Two Decimal Operands	202 (1)	8
AD2DX	Add Using Two Decimal Operands Extended	242 (1)	8
AD3D	Add Using Three Decimal Operands	222 (1)	8
AD3DX	Add Using Three Decimal Operands Extended	262 (1)	8
ADA	Add to A-Register	075 (0)	8
ADAQ	Add to AQ-Register	077 (0)	8
ADE	Add to Exponent Register	415 (0)	8
ADL	Add Low to AQ Register	033 (0)	8
ADLA	Add Logical to A-Register	035 (0)	8
ADLAQ	Add Logical to AQ-Register	037 (0)	8
ADLQ	Add Logical to Q-Register	036 (0)	8
ADLR	Add Logical Register to Register	435 (1)	8
ADLX \underline{n}	Add Logical to Index Register \underline{n}	02 \underline{n} (0)	8
ADQ	Add to Q-Register	076 (0)	8
ADRR	Add Register to Register	434 (1)	8
ADTCS	Add TCS Clock Register	756 (1)	8
ADX \underline{n}	Add to Index Register \underline{n}	06 \underline{n} (0)	8
ALR	A-Register Left Rotate	775 (0)	8
ALS	A-Register Left Shift	735 (0)	8
ANA	AND to A-Register	375 (0)	8
ANAQ	AND to AQ-Register	377 (0)	8
ANQ	AND to Q-Register	376 (0)	8
ANRR	AND Register to Register	535 (1)	8
ANSA	AND to Storage from A-Register	355 (0)	8
ANSQ	AND to Storage from Q-Register	356 (0)	8
ANSX \underline{n}	AND to Storage from Index Register \underline{n}	34 \underline{n} (0)	8
ANX \underline{n}	AND to Index Register \underline{n}	36 \underline{n} (0)	8
AOS	Add One to Storage	054 (0)	8
ARA \underline{n}	Address Register \underline{n} to Alphanumeric Descriptor	54 \underline{n} (1)	8
ARL	A-Register Right Logical Shift	771 (0)	8
ARN \underline{n}	Address Register \underline{n} to Numeric Descriptor	64 \underline{n} (1)	8
ARS	A-Register Right Shift	731 (0)	8

NovaScale 9000 Assembly Instructions Programmer's Guide

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
ASA	Add to Storage From A-Register	055 (0)	8
ASQ	Add to Storage From Q-Register	056 (0)	8
ASX _n	Add to Storage from Index Register <u>n</u>	04 _n (0)	8
AW0D	Add Word and 0 byte Displacement to AR _n	000 (1)	8
AW1D	Add Word and 1 byte Displacement to AR _n	001 (1)	8
AW2D	Add Word and 2 byte Displacement to AR _n	002 (1)	8
AW3D	Add Word and 3 byte Displacement to AR _n	003 (1)	8
AWCA	Add With Carry to A-Register	071 (0)	8
AWCQ	Add With Carry to Q-Register	072 (0)	8
AWD(X)	Add Word Displacement to Address Register	507 (1)	8
BCD	Binary-to-BCD Convert	505 (0)	8
BNCT	Binary Normalize and Count	410 (1)	8
BTD	Binary-to-Decimal Convert	301 (1)	8
CAMP	Clear Associative Memory Pages	532 (1)	9
CANA	Comparative AND with A-Register	315 (0)	9
CANAQ	Comparative AND with AQ-Register	317 (0)	9
CANQ	Comparative AND with Q-Register	316 (0)	9
CANX _n	Comparative AND with Index Register <u>n</u>	30 _n (0)	9
CBMX _n	Compare Byte Masked with X _n	07 _n (1)	9
CCAC	Clear Cache	011 (1)	9
CIOC	Connect Input/Output Channel	015 (0)	9
CLIMB	Domain Transfer	713 (1)	9
CMG	Compare Magnitude	405 (0)	9
CMK	Compare Masked	211 (0)	9
COMPA	Compare with A-Register	115 (0)	9
COMPAQ	Compare with AQ-register	117 (0)	9
COMPB	Compare Bit Strings	066 (1)	9
COMPBX	Compare Bit Strings Extended	067 (1)	9
CMPC	Compare Alphanumeric Character Strings	106 (1)	9
CMPCT	Compare Characters and Translate	166 (1)	9
CMPN	Compare Numeric	303 (1)	9
CMPNX	Compare Numeric Extended	343 (1)	9
CMPQ	Compare with Q-Register	116 (0)	9
CMPX _n	Compare with Index Register <u>n</u>	10 _n (0)	9
CMRR	Compare Register to Register	534 (1)	9
CNA A	Comparative NOT AND with A-Register	215 (0)	9
CNA AQ	Comparative NOT AND with AQ-Register	217 (0)	9
CNA Q	Compare NOT AND with Q-Register	216 (0)	9
CNA X _n	Comparative NOT AND with Index Register <u>n</u>	20 _n (0)	9
CSL	Combine Bit Strings Left	060 (1)	9
CSR	Combine Bit Strings Right	061 (1)	9
CWL	Compare with Limits	111 (0)	9
DFAD	Double-Precision Floating Add	477 (0)	9
DFCMG	Double-Precision Floating Compare Magnitude	427 (0)	9
DFCMP	Double-Precision Floating Compare	517 (0)	9
DFDI	Double-Precision Floating Divide Inverted	527 (0)	9
DFDV	Double-Precision Floating Divide	567 (0)	9
DFLD	Double-Precision Floating Load	433 (0)	9
DFLP	Double-Precision Floating Load Positive	532 (0)	9
DFMP	Double-Precision Floating Multiply	463 (0)	9
DFRD	Double-Precision Floating Round	473 (0)	9

NovaScale 9000 Machine Instructions

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
DFSB	Double-Precision Floating Subtract	577 (0)	9
DFSBI	Double-Precision Floating Subtract Inverted	467 (0)	9
DFST	Double-Precision Floating Store	457 (0)	9
DFSTR	Double-Precision Floating Store Rounded	472 (0)	9
DIAG	Diagnose	612 (0)	9
DIS	Delay Until Interrupt Signal	616 (0)	9
DIV	Divide Integer	506 (0)	9
DIVN	Divide Integer with No Remainder	504 (0)	9
DLY	Delay	730 (1)	9
DRL	Derail Fault	002 (0)	9
DTB	Decimal-to-Binary Convert	305 (1)	9
DTRACE	Dump Trace Table	733 (1)	9
DUFA	Double-Precision Unnormalized Floating Add	437 (0)	9
DUFM	Double-Precision Unnormalized Floating Multiply	423 (0)	9
DUFS	Double-Precision Unnormalized Floating Subtract	537 (0)	9
DV2D	Divide Using Two Decimal Operands	207 (1)	9
DV2DX	Divide Using Two Decimal Operands Extended	247 (1)	9
DV3D	Divide Using Three Decimal Operands	227 (1)	9
DV3DX	Divide Using Three Decimal Operands Extended	267 (1)	9
DVF	Divide Fraction	507 (0)	9
DVRR	Divide Register by Register	533 (1)	9
DVRRN	Divide Register with No Remainder	414 (1)	9
EAA	Effective Address to A-Register	635 (0)	10
EAQ	Effective Address to Q-Register	636 (0)	10
EAX _n	Effective Address to Index Register _n	62 _n (0)	10
EPAT	Effective Pointer and Address to Test	412 (1)	10
EPPR _n	Effective Pointer to Pointer Register _n	63 _n (1)	10
ERA	EXCLUSIVE OR to A-Register	675 (0)	10
ERAQ	EXCLUSIVE OR to AQ-Register	677 (0)	10
ERQ	EXCLUSIVE OR to Q-Register	676 (0)	10
ERRR	EXCLUSIVE OR Register to Register	537 (1)	10
ERSA	EXCLUSIVE OR to Storage with A-Register	655 (0)	10
ERSQ	EXCLUSIVE OR to Storage with Q-Register	656 (0)	10
ERSX _n	EXCLUSIVE OR to Storage with Index Register _n	64 _n (0)	10
ERX _n	EXCLUSIVE OR to Index Register _n	66 _n (0)	10
FAD	Floating Add	475 (0)	10
FCMG	Floating Compare Magnitude	425 (0)	10
FCMP	Floating Compare	515 (0)	10
FDI	Floating Divide Inverted	525 (0)	10
FDV	Floating Divide	565 (0)	10
FLD	Floating Load	431 (0)	10
FLP	Floating Load Positive	530 (0)	10
FMP	Floating Multiply	461 (0)	10
FNEG	Floating Negate	513 (0)	10
FNO	Floating Normalize	573 (0)	10
FRAN	Floating-Point Random Number	362 (1)	10
FRD	Floating Round	471 (0)	10
FSB	Floating Subtract	575 (0)	10
FSBI	Floating Subtract Inverted	465 (0)	10
FST	Floating Store	455 (0)	10
FSTR	Floating Store Rounded	470 (0)	10

NovaScale 9000 Assembly Instructions Programmer's Guide

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
FSZN	Floating Set Zero and Negative Indicators from Storage	430 (0)	10
FTR	Floating Truncate Fraction	474 (0)	10
GLDD0	Load Double to GX0	320 (1)	10
GLDD2	Load Double to GX2	322 (1)	10
GLDD4	Load Double to GX4	324 (1)	10
GLDD6	Load Double to GX6	326 (1)	10
GLLR	GX _n Long Left Rotate	446 (1)	10
GLLS	GX _n Long Left Shift	466 (1)	10
GLR	GX _n Left Rotate	442 (1)	10
GLRL	GX _n Long Right Logic	465 (1)	10
GLRS	GX _n Long Right Shift	464 (1)	10
GLS	GX _n Left Shift	462 (1)	10
GRL	GX _n Right Logic	461 (1)	10
GRS	GX _n Right Shift	460 (1)	10
GSTD0	Store Double from GX0	140 (1)	10
GSTD2	Store Double from GX2	142 (1)	10
GSTD4	Store Double from GX4	144 (1)	10
GSTD6	Store Double from GX6	146 (1)	10
GTB	Gray-to-Binary Convert	774 (0)	10
LAREG	Load Address Registers	463 (1)	11
LAR _n	Load Address Register _n	76 _n (1)	11
LCA	Load Complement into A-Register	335 (0)	11
LCAQ	Load Complement into AQ Register	337 (0)	11
LCCL	Load Calendar Clock	057 (0)	11
LCQ	Load Complement to Q-Register	336 (0)	11
LCTR	Load Weighted Instruction Counter	417 (0)	11
LCX _n	Load Complement into Index Register _n	32 _n (0)	11
LDA	Load A- Register	235 (0)	11
LDAB	Load A-Register with Byte	501 (0)	11
LDAC	Load A- Register and Clear	034 (0)	11
LDAH	Load A-Register with Halfword	511 (0)	11
LDAQ	Load AQ Register	237 (0)	11
LDAS	Load Argument Stack Register	770 (1)	11
LDCR	Load Complement Register from Register	431 (1)	11
LDD _n	Load Descriptor Register _n	67 _n (1)	11
LDDR	Load Double Register to Register Pair	433 (1)	11
LDDSA	Load Data Stack Address Register	170 (1)	11
LDDSD	Load Data Stack Descriptor Register	571 (1)	11
LDE	Load Exponent Register	411 (0)	11
LDEA _n	Load Extended Address _n	61 _n (1)	11
LDI	Load Indicator Register	634 (0)	11
LDMB	Load Performance Monitor Mode	755 (1)	11
LDO	Load Option Register	172 (1)	11
LDP _n	Load Pointer Register _n	47 _n (1)	11
LDPR	Load Positive Register to Register	432 (1)	11
LDPS	Load Parameter Segment Register	771 (1)	11
LDQ	Load Q-Register	236 (0)	11
LDQB	Load Q-Register with Byte	502 (0)	11
LDQC	Load Q-Register and Clear	032 (0)	11
LDQH	Load Q-Register with Halfword	512 (0)	11

NovaScale 9000 Machine Instructions

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
LDRR	Load Register from Register	430 (1)	11
LDSS	Load Safe Store Register	773 (1)	11
LDT	Load Timer Register	637 (0)	11
LDWS	Load Working Space Registers	772 (1)	11
LDX _n	Load Index Register <u>n</u> from Upper	22 _n (0)	11
LIMR	Load Interrupt Mask Register	553 (0)	11
LLAR	Load Limit Address Register	724 (1)	11
LLPNR	Load Logical Processor Number Register	364 (1)	11
LLR	Long Left Rotate	777 (0)	11
LLS	Long Left Shift	737 (0)	11
LLUF	Load Lockup Fault Register	674 (0)	11
LPDBR	Load Page Table Directory Base Register	171 (1)	11
LPL	Load Pointers and Lengths	467 (1)	11
LREG	Load Registers	073 (0)	11
LRL	Long Right Logical Shift	773 (0)	11
LRMB	Load Reserve Memory Base	712 (0)	11
LRS	Long Right Shift	733 (0)	11
LVMMR	Load Virtual Machine Mode Register	720 (1)	11
LVMTR	Load Virtual Machine Timer Register	722 (1)	11
LXL _n	Load Index Register <u>n</u> from Lower	72 _n (0)	11
MLR	Move Alphanumeric Left to Right	100 (1)	11
MME	Master Mode Entry Fault	001 (0)	11
MP2D	Multiply Using Two Decimal Operands	206 (1)	11
MP2DX	Multiply Using Two Decimal Operands Extended	246 (1)	11
MP3D	Multiply Using Three Decimal Operands	226 (1)	11
MP3DX	Multiply Using Three Decimal Operands Extended	266 (1)	11
MPF	Multiply Fraction	401 (0)	11
MPRR	Multiply Register Pair by Register	530 (1)	11
MPRS	Multiply Single Register by Register	531 (1)	11
MPX _n	Multiply GX _n	04 _n (1)	11
MPY	Multiply Integer	402 (0)	11
MRL	Move Alphanumeric Right to Left	101 (1)	11
MTM	Move to Memory	365 (1)	11
MTR	Move to Register	361 (1)	11
MVE	Move Alphanumeric Edited	020 (1)	11
MVN	Move Numeric	300 (1)	11
MVNE	Move Numeric Edited	024 (1)	11
MVNEX	Move Numeric Edited Extended	004 (1)	11
MVNX	Move Numeric Extended	340 (1)	11
MVT	Move Alphanumeric with Translation	160 (1)	11
NAR _n	Numeric Descriptor to Address Register <u>n</u>	66 _n (1)	12
NEG	Negate (A-Register)	531 (0)	12
NEGL	Negate Long (AQ-Register)	533 (0)	12
NOP	No Operation	011 (0)	12
ORA	OR to A-Register	275 (0)	12
ORAQ	OR to AQ-Register	277 (0)	12
ORQ	OR to Q-Register	276 (0)	12
ORRR	OR Register to Register	536 (1)	12
ORSA	OR to Storage from A-Register	255 (0)	12
ORSQ	OR to Storage from Q-Register	256 (0)	12
ORSX _n	OR to Storage from Index Register <u>n</u>	24 _n (0)	12

NovaScale 9000 Assembly Instructions Programmer's Guide

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
ORX \underline{n}	OR to Index Register \underline{n}	26 \underline{n} (0)	12
OWA	OR-Write Memory from A-Register	555 (1)	12
OWQ	OR-Write Memory from Q-Register	556 (1)	12
OWRES	OR Write Reserve Memory	252 (0)	12
PAS	Pop Argument Stack	176 (1)	12
PULS1	Pulse One	012 (0)	12
PULS2	Pulse Two	013 (0)	12
QFAD	Quadruple-Precision Floating Add	476 (0)	12
QFLD	Quadruple-Precision Floating Load	432 (0)	12
QFMP	Quadruple-Precision Floating Multiply	462 (0)	12
QFSB	Quadruple-Precision Floating Subtract	576 (0)	12
QFST	Quadruple-Precision Floating Store	453 (0)	12
QFSTR	Quadruple-Precision Floating Store Rounded	466 (0)	12
QLR	Q-Register Left Rotate	776 (0)	12
QLS	Q-Register Left Shift	736 (0)	12
QRL	Q-Register Right Logical Shift	772 (0)	12
QRS	Q-Register Right Shift	732 (0)	12
QSMP	Quadruple-Precision Floating Multiply with Double-Precision Operands	460 (0)	12
RBF	Reset Backup Fault Flag	153 (1)	12
RCCL	Read Calendar Clock	413 (0)	12
RCRES	Read and Clear Reserve Memory	251 (0)	12
RDTCS	Read TCS Clock Register	776 (1)	12
RET	Return	630 (0)	12
RICHR	Restart IC History Register	156 (1)	12
RIMR	Read Interrupt Mask Register	233 (0)	12
RIW	Read Interrupt Word Pair	412 (0)	12
RLPNR	Read Logical Processor Number Register	366 (1)	12
RPD	Repeat Double	560 (0)	12
RPL	Repeat Link	500 (0)	12
RPN	Read Processor Number	367 (1)	12
RPT	Repeat	520 (0)	12
RRES	Read Reserve Memory	231 (0)	12
S4BD(X)	Subtract 4-Bit Displacement from Address Register	522 (1)	13
S6BD(X)	Subtract 6-Bit Displacement from Address Register	521 (1)	13
S9BD(X)	Subtract 9-Bit Displacement from Address Register	520 (1)	13
SACK	Transmit Sync Interrupt and Wait for Acknowledgement	734 (1)	13
SAR \underline{n}	Store Address Register \underline{n}	74 \underline{n} (1)	13
SAREG	Store Address Registers	443 (1)	13
SB2D	Subtract Using Two Decimal Operands	203 (1)	13
SB2DX	Subtract Using Two Decimal Operands Extended	243 (1)	13
SB3D	Subtract Using Three Decimal Operands	223 (1)	13
SB3DX	Subtract Using Three Decimal Operands Extended	263 (1)	13
SBA	Subtract from A-Register	175 (0)	13
SBAQ	Subtract from AQ-Register	177 (0)	13
SBAR	Store Base Address Register	550 (0)	13
SBD(X)	Subtract Bit Displacement from Address Register	523 (1)	13
SBLA	Subtract Logical from A-Register	135 (0)	13
SBLAQ	Subtract Logical from AQ-Register	137 (0)	13
SBLQ	Subtract Logical from Q-Register	136 (0)	13
SBLR	Subtract Logical Register from Register	437 (1)	13

NovaScale 9000 Machine Instructions

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
SBLX _n	Subtract Logical from Index Register _n	12 _n (0)	13
SBQ	Subtract from Q-Register	176 (0)	13
SBRR	Subtract Register from Register	436 (1)	13
SBX _n	Subtract from Index Register _n	16 _n (0)	13
SCD	Scan Characters Double	120 (1)	13
SCDR	Scan Characters Double in Reverse	121 (1)	13
SCM	Scan With Mask	124 (1)	13
SCMR	Scan With Mask in Reverse	125 (1)	13
SCTR	Store Weighted Instruction Counter	414 (0)	13
SDR _n	Save Descriptor Register _n	11 _n (1)	13
SFR	Store Fault Register	452 (0)	13
SICHR	Store IC History Register	154 (1)	13
SIW	Set Interrupt Word Pair	451 (0)	13
SLAR	Store Limit Address Register	725 (1)	13
SPDBR	Store Page Table Directory Base Register	151 (1)	13
SPL	Store Pointers and Lengths	447 (1)	13
SREG	Store Registers	753 (0)	13
SRMB	Store Reserve Memory Base	714 (0)	13
SSA	Subtract Stored from A-Register	155 (0)	13
SSQ	Subtract Stored from Q-Register	156 (0)	13
SSX _n	Subtract Stored from Index Register _n	14 _n (0)	13
STA	Store A-Register	755 (0)	13
STAB	Store A-Register by Byte	561 (0)	13
STAC	Store A Conditional	354 (0)	13
STACQ	Store A Conditional on Q	654 (0)	13
STAH	Store A-Register by Halfword	571(0)	13
STAQ	Store AQ-Register	757 (0)	13
STAS	Store Argument Stack Register	750 (1)	13
STBA	Store 9-Bit Bytes of A-Register	551 (0)	13
STBQ	Store 9-Bit Bytes of Q-Register	552 (0)	13
STC1	Store Instruction Counter Plus 1	554 (0)	13
STC2	Store Instruction Counter Plus 2	750 (0)	13
STCA	Store 6-Bit Characters of A-Register	751 (0)	13
STCQ	Store 6-Bit Characters of Q-Register	752 (0)	13
STD _n	Store Descriptor _n	05 _n (1)	13
STDSA	Store Data Stack Address Register	150 (1)	13
STDSD	Store Data Stack Descriptor Register	551 (1)	13
STE	Store Exponent Register	456 (0)	13
STI	Store Indicator Register	754 (0)	13
STMB	Store Performance Monitor Mode	775 (1)	13
STO	Store Option Register	152 (1)	13
STP _n	Store Pointer _n	45 _n (1)	13
STPS	Store Parameter Segment Register	751 (1)	13
STQ	Store Q-Register	756 (0)	13
STQB	Store Q-Register by Byte	562 (0)	13
STQH	Store Q-Register by Halfword	572 (0)	13
STSS	Store Safe Store Register	753 (1)	13
STT	Store Timer Register	454 (0)	13
STWS	Store Working Space Registers	752 (1)	13
STX _n	Store Index Register _n in Upper	74 _n (0)	13
STZ	Store Zero	450 (0)	13

NovaScale 9000 Assembly Instructions Programmer's Guide

<u>Mnemonic</u>	<u>Instruction Name</u>	<u>Opcode</u>	<u>Section</u>
SVMMR	Store Virtual Machine Mode Register	721 (1)	13
SVMOS	Start VM Operating System Mode	711 (1)	13
SVMTR	Store Virtual Machine Timer Register	723 (1)	13
SWCA	Subtract With Carry from A-Register	171 (0)	13
SWCQ	Subtract With Carry from Q-Register	172 (0)	13
SWD(X)	Subtract Word Displacement from Address Register	527 (1)	13
SXL _n	Store Index Register <u>n</u> in Lower	44 _n (0)	13
SZNC	Set Zero and Negative Indicators from Storage	234 (0)	13
SZNC	Set Zero and Negative Indicators from Storage and Clear	214 (0)	13
SZTL	Set Zero and Truncation Indicators With Bit Strings Left	064 (1)	13
SZTR	Set Zero & Truncation Indicators With Bit Strings Right	065 (1)	13
TCT	Test Character and Translate	164 (1)	14
TCTR	Test Character and Translate in Reverse	165 (1)	14
TEO	Transfer on Exponent Overflow	614 (0)	14
TEU	Transfer on Exponent Underflow	615 (0)	14
TMI	Transfer on Minus	604 (0)	14
TMOZ	Transfer on Minus or Zero	604 (1)	14
TNC	Transfer on No Carry	602 (0)	14
TNZ	Transfer on Nonzero	601 (0)	14
TOV	Transfer on Overflow	617 (0)	14
TPL	Transfer on Plus	605 (0)	14
TPNZ	Transfer on Plus and Nonzero	605 (1)	14
TRA	Transfer Unconditionally	710 (0)	14
TRC	Transfer on Carry	603 (0)	14
TRCT _n	Transfer on Count <u>n</u>	54 _n (0)	14
TRTF	Transfer on Truncation Indicator OFF	601 (1)	14
TRTN	Transfer on Truncation Indicator ON	600 (1)	14
TSS	Transfer After Setting Slave	715 (0)	14
TSX _n	Transfer and Set Index Register <u>n</u>	70 _n (0)	14
TSYNC	Transmit Sync Interrupt	731 (1)	14
TTF	Transfer on Tally Runout Indicator OFF	607 (0)	14
TTN	Transfer on Tally Runout Indicator ON	606 (1)	14
TXIP	Test for Interrupt Present	416 (0)	14
TZE	Transfer on Zero	600 (0)	14
UFA	Unnormalized Floating Add	435 (0)	14
UFM	Unnormalized Floating Multiply	421 (0)	14
UFS	Unnormalized Floating Subtract	535 (0)	14
UFTR	Unnormalized Floating Truncate Fraction	434 (0)	14
VMME	Virtual Machine Monitor Mode Entry	710 (1)	15
WRES	Write Reserve Memory	232 (0)	15
WSYNC	Wait for Sync Interrupt	732 (1)	15
XEC	Execute	716 (0)	15
XED	Execute Double	717 (0)	15
XRAN	Fixed-Point Random Number	363 (1)	15

B. Operation Code Maps

The operation code maps for the V9000 processor are shown in Tables B-1 and B-2. The operation codes are separated into sections: the first section lists operation codes with bit 27 = 0 and the second section with bit 27 = 1.

Table B-1. Operation Code Map (Bit 27 = 0) (1 of 2)

Upper 6 bits ▼	Lower 4 bits							
	0(0)	1(0)	2(0)	3(0)	4(0)	5(0)	6(0)	7(0)
00 ₈		MME	DRL					
01 ₈		NOP	PULS1	PULS2		CIOC		
02 ₈	ADLX0	ADLX1	ADLX2	ADLX3	ADLX4	ADLX5	ADLX6	ADLX7
03 ₈			LDQC	ADL	LDAC	ADLA	ADLQ	ADLAQ
04 ₈	ASX0	ASX1	ASX2	ASX3	ASX4	ASX5	ASX6	ASX7
05 ₈					AOS	ASA	ASQ	LCCL
06 ₈	ADX0	ADX1	ADX2	ADX3	ADX4	ADX5	ADX6	ADX7
07 ₈		AWCA	AWCQ	LREG		ADA	ADQ	ADAQ
10 ₈	CMPX0	CMPX1	CMPX2	CMPX3	CMPX4	CMPX5	CMPX6	CMPX7
11 ₈		CWL				CMPA	CMPQ	CMPAQ
12 ₈	SBLX0	SBLX1	SBLX2	SBLX3	SBLX4	SBLX5	SBLX6	SBLX7
13 ₈						SBLA	SBLQ	SBLAQ
14 ₈	SSX0	SSX1	SSX2	SSX3	SSX4	SSX5	SSX6	SSX7
15 ₈						SSA	SSQ	
16 ₈	SBX0	SBX1	SBX2	SBX3	SBX4	SBX5	SBX6	SBX7
17 ₈		SWCA	SWCQ			SBA	SBQ	SBAQ
20 ₈	CNAX0	CNAX1	CNAX2	CNAX3	CNAX4	CNAX5	CNAX6	CNAX7
21 ₈		CMK			SZNC	CNAA	CNAQ	CNAAQ
22 ₈	LDX0	LDX1	LDX2	LDX3	LDX4	LDX5	LDX6	LDX7
23 ₈		RRES	WRES	RIMR	SZN	LDA	LDQ	LDAQ
24 ₈	ORSX0	ORSX1	ORSX2	ORSX3	ORSX4	ORSX5	ORSX6	ORSX7
25 ₈		RCRES	OWRES			ORSA	ORSQ	
26 ₈	ORX0	ORX1	ORX2	ORX3	ORX4	ORX5	ORX6	ORX7
27 ₈						ORA	ORQ	ORAQ
30 ₈	CANX0	CANX1	CANX2	CANX3	CANX4	CANX5	CANX6	CANX7
31 ₈						CANA	CANQ	CANAQ
32 ₈	LCX0	LCX1	LCX2	LCX3	LCX4	LCX5	LCX6	LCX7
33 ₈						LCA	LCQ	LCAQ
34 ₈	ANSX0	ANSX1	ANSX2	ANSX3	ANSX4	ANSX5	ANSX6	ANSX7
35 ₈					STAC	ANSA	ANSQ	
36 ₈	ANX0	ANX1	ANX2	ANX3	ANX4	ANX5	ANX6	ANX7
37 ₈						ANA	ANQ	ANAQ

Operation Code Maps

Table B-1. Operation Code Map (Bit 27 = 0) (2 of 2)

Upper 6 bits ▼	Lower 4 bits							
	0(0)	1(0)	2(0)	3(0)	4(0)	5(0)	6(0)	7(0)
40 ₈		MPF	MPY			CMG		
41 ₈		LDE	RIW	RCCL	SCTR	ADE	TXIP	LCTR
42 ₈		UFM		DUFM		FCMG		DFCMG
43 ₈	FSZN	FLD	QFLD	DFLD	UFTR	UFA		DUFA
44 ₈	SXL0	SXL1	SXL2	SXL3	SXL4	SXL5	SXL6	SXL7
45 ₈	STZ	SIW	SFR	QFST	STT	FST	STE	DFST
46 ₈	QSMP	FMP	QFMP	DFMP		FSBI	QFSTR	DFSBI
47 ₈	FSTR	FRD	DFSTR	DFRD	FTR	FAD	QFAD	DFAD
50 ₈	RPL	LDAB	LDQB		DIVN	BCD	DIV	DVF
51 ₈		LDAB	LDQB	FNEG		FCMP		DFCMP
52 ₈	RPT					FDI		DFDI
53 ₈	FLP	NEG	DFLP	NEGL		UFS		DUFS
54 ₈	TRCT0	TRCT1	TRCT2	TRCT3	TRCT4	TRCT5	TRCT6	TRCT7
55 ₈	SBAR	STBA	STBQ	LIMR	STC1			
56 ₈	RPD	STAB	STQB			FDV		DFDV
57 ₈		STAH	STQH	FNO		FSB	QFSB	DFSB
60 ₈	TZE	TNZ	TNC	TRC	TMI	TPL		TTF
61 ₈			DIAG		TEO	TEU	DIS	TOV
62 ₈	EAX0	EAX1	EAX2	EAX3	EAX4	EAX5	EAX6	EAX7
63 ₈	RET				LDI	EAA	EAQ	LDT
64 ₈	ERSX0	ERSX1	ERSX2	ERSX3	ERSX4	ERSX5	ERSX6	ERSX7
65 ₈					STACQ	ERSA	ERSQ	
66 ₈	ERX0	ERX1	ERX2	ERX3	ERX4	ERX5	ERX6	ERX7
67 ₈					LLUF	ERA	ERQ	ERAQ
70 ₈	TSX0	TSX1	TSX2	TSX3	TSX4	TSX5	TSX6	TSX7
71 ₈	TRA		LRMB		SRMB	TSS	XEC	XED
72 ₈	LXL0	LXL1	LXL2	LXL3	LXL4	LXL5	LXL6	LXL7
73 ₈		ARS	QRS	LRs		ALS	QLS	LLS
74 ₈	STX0	STX1	STX2	STX3	STX4	STX5	STX6	STX7
75 ₈	STC2	STCA	STCQ	SREG	STI	STA	STQ	STAQ
76 ₈								
77 ₈		ARL	QRL	LRL	GTB	ALR	QLR	LLR

Table B-2. Operation Code Map (Bit 27 = 1) (1 of 2)

Upper 6 bits ▼	Lower 4 bits							
	0(1)	1(1)	2(1)	3(1)	4(1)	5(1)	6(1)	7(1)
00 ₈	AW0D	AW1D	AW2D	AW3D	MVNEX		MMEP	MEMP
01 ₈		CCAC						
02 ₈	MVE				MVNE			
03 ₈								
04 ₈	MPX0	MPX1	MPX2	MPX3	MPX4	MPX5	MPX6	MPX7
05 ₈	STD0	STD1	STD2	STD3	STD4	STD5	STD6	STD7
06 ₈	CSL	CSR			SZTL	SZTR	CMPB	CMPBX
07 ₈	CBMX0	CBMX1	CBMX2	CBMX3	CBMX4	CBMX5	CBMX6	CBMX7
10 ₈	MLR	MRL					CMPC	
11 ₈	SDR0	SDR1	SDR2	SDR3	SDR4	SDR5	SDR6	SDR7
12 ₈	SCD	SCDR			SCM	SCMR		
13 ₈								
14 ₈	GSTD0		GSTD2		GSTD4		GSTD6	
15 ₈	STDSA	SPDBR	STO	RBFf	SICHR		RICHR	
16 ₈	MVT				TCT	TCTR	CMPCT	
17 ₈	LDDSA	LPDBR	LDO				PAS	
20 ₈			AD2D	SB2D			MP2D	DV2D
21 ₈								
22 ₈			AD3D	SB3D			MP3D	DV3D
23 ₈								
24 ₈			AD2DX	SB2DX			MP2DX	DV2DX
25 ₈								
26 ₈			AD3DX	SB3DX			MP3DX	DV3DX
27 ₈								
30 ₈	MVN	BTD		CMPN		DTB		
31 ₈								
32 ₈	GLDD0	MTRX	GLDD2		GLDD4	MTMX	GLDD6	
33 ₈								
34 ₈	MVNX	FMTR		CMPNX		FMTM		
35 ₈								
36 ₈		MTR	FRAN	XRAN	LLPNR	MTM	RLPNR	RPN
37 ₈								

Operation Code Maps

Table B-2. Operation Code Map (Bit 27 = 1) (2 of 2)

Upper 6 bits ▼	Lower 4 bits							
	0(1)	1(1)	2(1)	3(1)	4(1)	5(1)	6(1)	7(1)
40 ₈								
41 ₈	BNCT		EPAT		DVRRN			
42 ₈								
43 ₈	LDRR	LDCR	LDPR	LDDR	ADRR	ADLR	SBRR	SBLR
44 ₈			GLR	SAREG			GLLR	SPL
45 ₈	STP0	STP1	STP2	STP3	STP4	STP5	STP6	STP7
46 ₈	GRS	GRL	GLS	LAREG	GLRS	GLRL	GLLS	LPL
47 ₈	LDP0	LDP1	LDP2	LDP3	LDP4	LDP5	LDP6	LDP7
50 ₈	A9BD	A6BD	A4BD	ABD				AWD
51 ₈								
52 ₈	S9BD	S6BD	S4BD	SBD				SWD
53 ₈	MPRR	MPRS	CAMP	DVRR	CMRR	ANRR	ORRR	ERRR
54 ₈	ARA0	ARA1	ARA2	ARA3	ARA4	ARA5	ARA6	ARA7
55 ₈	STD	STDS		STTA		OWA	OWQ	
56 ₈	AAR0	AAR1	AAR2	AAR3	AAR4	AAR5	AAR6	AAR7
57 ₈		LDDSD						
60 ₈	TRTN	TRTF			TMOZ	TPNZ	TTN	
61 ₈	LDEA0	LDEA1	LDEA2	LDEA3	LDEA4	LDEA5	LDEA6	LDEA7
62 ₈								
63 ₈	EPPR0	EPPR1	EPPR2	EPPR3	EPPR4	EPPR5	EPPR6	EPPR7
64 ₈	ARN0	ARN1	ARN2	ARN3	ARN4	ARN5	ARN6	ARN7
65 ₈								
66 ₈	NAR0	NAR1	NAR2	NAR3	NAR4	NAR5	NAR6	NAR7
67 ₈	LDD0	LDD1	LDD2	LDD3	LDD4	LDD5	LDD6	LDD7
70 ₈								
71 ₈	VMME	SVMOS		CLIMB				
72 ₈	LVMMR	SVMMR	LVMTR	SVMTR	LLAR	SLAR		
73 ₈	DLY	TSYNC	WSYNC	DTRACE	SACK	ACKS		
74 ₈	SAR0	SAR1	SAR2	SAR3	SAR4	SAR5	SAR6	SAR7
75 ₈	STAS	STPS	STWS	STSS		LDMB	ADTCS	
76 ₈	LAR0	LAR1	LAR2	LAR3	LAR4	LAR5	LAR6	LAR7
77 ₈	LDAS	LDPS	LDWS	LDSS	SICPM	STMB	RDTCS	RICPM

Notes

C. ASCII Sequence

This section contains the listing of the Unified Character Set of the ASCII Sequence.

NOTE	SYMBOL	NAME	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
3	NUL	Null	00 000	00 000	1F 37	1E 36	12-0-1-8-9		
3	SOH	Start of Heading	01 001	01 001	1F 37	1E 36	12-1-9		
3	STX	Start of Text	02 002	02 002	1F 37	1E 36	12-2-9		
3	ETX	End of Text	03 003	03 003	1F 37	1E 36	12-3-9		
3	EOT	End of Transmission	04 004	37 067	1F 37	1E 36	7-9		
3	ENQ	Enquiry	05 005	2D 055	1F 37	1E 36	0-5-8-9		
3	ACK	Acknowledge	06 006	2E 056	1F 37	1E 36	0-6-8-9		
3	BEL	Bell (Audible Signal)	07 007	2F 057	1F 37	1E 36	0-7-8-9		
3	BS	Backspace	08 010	1G 026	1F 37	1E 36	11-6-9		
3	HT	Horizontal Tab (Punch Card Skip)	09 011	0G 005	1F 37	1E 36	12-5-9		
3	LF	Line Feed	0A 012	2G 045	1F 37	1E 36	0-5-9		
3	VT	Vertical Tabulation	0B 013	0B 013	1F 37	1E 36	12-3-8-9		
3	FF	Form Feed	0C 014	0C 014	1F 37	1E 36	12-4-8-9		
3	CR	Carriage Return	0D 015	0D 015	1F 37	1E 36	12-5-8-9		
0	SO	Shift Out	0E 016	0E 016	1F 37	1E 36	12-6-8-9		
3	SI	Shift In	0F 017	0F 017	1F 37	1E 36	12-7-8-9		
3	DLE	Data Link Escape	10 020	10 020	1F 37	1E 36	12-11-1-8-9		
3	DC1	Device Control 1	11 021	11 021	1F 37	1E 36	11-1-9		
3	DC2	Device Control 2	12 022	12 022	1F 37	1E 36	11-2-9		
3,6	DC3	Device Control 3	13 023	13 023	1F 37	1E 36	11-3-9		
3	DC4	Device Control 4 (step)	14 024	3C 074	1F 37	1E 36	4-8-9		
3	NAK	Negative Acknowledge	15 025	3D 075	1F 37	1E 36	5-8-9		
3	SYN	Synchronous Idle	16 026	32 062	1F 37	1E 36	2-9		
3	ETB	End of Transmission Block	17 027	26 046	1F 37	1E 36	0-6-9		
3	CAN	Cancel	18 030	18 030	1F 37	1E 36	11-8-9		
3	EM	End of Medium	19 031	19 031	1F 37	1E 36	11-1-8-9		
3	SUB	Substitute	1A 032	3F 077	1F 37	1E 36	7-8-9		
3	ESC	Escape	1B 033	27 047	1F 37	1E 36	0-7-9		
3	IFS	File Separator	1C 034	1C 034	1F 37	1E 36	11-4-8-9		
3	IGS	Group Separator	1D 035	1D 035	1F 37	1E 36	11-5-8-9		
3	IRS	Record Separator	1E 036	1E 036	1F 37	1E 36	11-6-8-9		
3	IUS	Unit Separator	1F 037	1F 037	1F 37	1E 36	11-7-8-9		

NOTE	SYMBOL	NAME	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC	GBCD	HBCD
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
		Space, Blank	20 040	40 100	10 20	0D 16	Blank	Blank	Blank
	!	Exclamation Point (ASCII)	21 041	4F 117	CI 3F 77	R 3D 75	12-7-8	0-7-8	0-5-8
	"	Double Quote	22 042	7F 177	3E 76	2D 55	7-8	0-6-8	11-5-8
2,9	#	Number Sign	23 043	7B 173	0B 13	2A 52	3-8	0-8	11-2-8
2,9	\$	Dollar Sign	24 044	6B 133	2B 53	2B 53	11-3-8	11-3-8	11-3-8
	%	Percent Sign	25 045	6C 164	3C 74	1D 35	0-4-8	0-4-8	12-5-8
	&	Ampersand	26 046	60 120	1A 32	0F 17	12-2	12	7-8
	'	Apostrophe	27 047	7D 175	2F 57	0A 12	5-8	11-7-8	2-8
	(Left Parenthesis	28 050	4D 115	1D 36	3C 74	12-5-8	12-8-8	0-4-8
)	Right Parenthesis	29 051	5D 135	2D 66	1C 34	11-5-8	11-8-8	12-4-8
	*	Asterisk	2A 052	5C 134	2C 64	2C 54	11-4-8	11-4-8	11-4-6
7	+	Plus	2B 053	4E 116	30 60	10 20	12-6-8	12-0	(12-0)(12)
	,	Comma	2C 054	6B 163	3B 73	3B 73	0-3-8	0-3-8	0-3-8
	-	Hyphen, Minus	2D 055	60 140	2A 62	20 40	11	11	(11-0)(11)
	.	Period	2E 056	4B 113	1B 33	1B 33	12-3-8	12-3-8	12-3-8
	/	Slash	2F 057	61 141	31 61	30 61	0-1	0-1	0-1
	0		30 060	F0 360	00 00	00 00	0	0	0
	1		31 061	F1 361	01 01	01 01	1	1	1
	2		32 062	F2 362	02 02	02 02	2	2	2
	3		33 063	F3 363	03 03	03 03	3	3	3
	4		34 064	F4 364	04 04	04 04	4	4	4
	5		35 065	F5 365	05 05	05 05	5	5	5
	6		36 066	F6 366	06 06	06 06	6	6	6
	7		37 067	F7 367	07 07	07 07	7	7	7
	8		38 070	F8 370	08 10	08 10	8	8	8
	9		39 071	F9 371	09 11	09 11	9	9	9
	:	Colon	3A 072	7A 172	0D 15	0C 11	2-8	5-8	4-8
	;	Semi-Colon	3B 073	6E 136	2E 56	1A 32	11-8-9	11-6-8	12-2-8
	<	Less Than	3C 074	4C 114	1E 36	30 60	12-4-8	12-6-8	5-8
	=	Equal	3D 075	7E 176	3D 75	0B 13	6-8	0-5-8	3-8
	>	Greater Than	3E 076	6E 156	0E 16	0E 16	0-6-8	6-8	6-8
7	?	Question Mark	3F 077	6F 157	0F 17	1F 37	0-7-8	7-8	(12)(12-0)

NOTE	SYMBOL	NAME	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC	GBCD	HBCD
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
2,9	@	At Sign	40 100	7C 174	0C 14	3A 72	4-8	4-8	0-2-8
	A		41 101	C1 301	11 21	11 21	12-1	12-1	12-1
	B		42 102	C2 302	12 22	12 22	12-2	12-2	12-2
	C		43 103	C3 303	13 23	13 23	12-3	12-3	12-3
	D		44 104	C4 304	14 24	14 24	12-4	12-4	12-4
	E		45 105	C5 305	15 25	15 25	12-5	12-5	12-5
	F		46 106	C6 306	16 26	16 26	12-6	12-6	12-6
	G		47 107	C7 307	17 27	17 27	12-7	12-7	12-7
	H		48 110	C8 310	18 30	18 30	12-8	12-8	12-8
	I		49 111	C9 311	19 31	19 31	12-9	12-9	12-9
	J		4A 112	D1 321	21 41	21 41	11-1	11-1	11-1
	K		4B 113	D2 322	22 42	22 42	11-2	11-2	11-2
	L		4C 114	D3 323	23 43	23 43	11-3	11-3	11-3
	M		4D 115	D4 324	24 44	24 44	11-4	11-4	11-4
	N		4E 116	D5 325	25 45	25 45	11-5	11-5	11-5
	O		4F 117	D6 326	26 46	26 46	11-6	11-6	11-6
	P		50 120	D7 327	27 47	27 47	11-7	11-7	11-7
	Q		51 121	D8 330	28 50	28 50	11-8	11-8	11-8
	R		52 122	D9 331	29 51	29 51	11-9	11-9	11-9
	S		53 123	E2 342	32 62	32 62	0-2	0-2	0-2
	T		54 124	E3 343	33 63	33 63	0-3	0-3	0-3
	U		55 125	E4 344	34 64	34 64	0-4	0-4	0-4
	V		56 126	E5 345	35 65	35 65	0-5	0-5	0-5
	W		57 127	E6 346	36 66	36 66	0-6	0-6	0-6
	X		58 130	E7 347	37 67	37 67	0-7	0-7	0-7
	Y		59 131	E8 350	38 70	38 70	0-8	0-8	0-8
	Z		5A 132	E9 351	39 71	39 71	0-9	0-9	0-9
2	[Left Bracket	[5B 133	c 4A 112	[0A 12	c 3F 77	12-2-8	2-8	0-7-8
2	\	Reverse Slant	\ 5C 134	\ 1F 37	\ 1F 37	o 2E 56	0-2-8	12-7-8	11-9-8
2,4,7]	Right Bracket] 5D 135] 1C 34] 1C 34	2F 57	11-2-8	12-4-8	(11)(11-0)
2	^	Circumflex Accent	^ 5E 136	^ 20 40	^ 20 40	1E 36	11-7-8	11-0	12-6-8
	-	Underline	5F 137	+ 3A 72	+ 3A 72	3E 76	0-5-8	0-2-8	0-6-8

NOTE	SYMBOL	NAME	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC	GBCD	HBCD
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
3	`	Grave Accent	60 140	79 171	1F 37	1E 36	1-8	12-7-8	12-6-8
1,8	a		61 141	81 201	11 21	11 21	12-0-1	12-1	12-1
1,8	b		62 142	82 202	12 22	12 22	12-0-2	12-2	12-2
1,8	c		63 143	83 203	13 23	13 23	12-0-3	12-3	12-3
1,8	d		64 144	84 204	14 24	14 24	12-0-4	12-4	12-4
1,8	e		65 145	85 205	15 25	15 25	12-0-5	12-5	12-5
1,8	f		66 146	86 206	16 26	16 26	12-0-6	12-6	12-6
1,8	g		67 147	87 207	17 27	17 27	12-0-7	12-7	12-7
1,8	h		68 150	88 210	18 30	18 30	12-0-8	12-8	12-8
1,8	i		69 151	89 211	19 31	19 31	12-0-9	12-9	12-9
1,8	j		6A 152	91 221	21 41	21 41	12-11-1	11-1	11-1
1,8	k		6B 153	92 222	22 42	22 42	12-11-2	11-2	11-2
1,8	l		6C 154	93 223	23 43	23 43	12-11-3	11-3	11-3
1,8	m		6D 155	94 224	24 44	24 44	12-11-4	11-4	11-4
1,8	n		6E 156	95 225	25 45	25 45	12-11-5	11-5	11-5
1,8	o		6F 157	96 226	26 46	26 46	12-11-6	11-6	11-6
1,8	p		70 160	97 227	27 47	27 47	12-11-7	11-7	11-7
1,8	q		71 161	98 230	28 50	28 50	12-11-8	11-8	11-8
1,8	r		72 162	99 231	29 51	29 51	12-11-9	11-9	11-9
1,8	s		73 163	A2 242	32 62	32 62	11-0-2	0-2	0-2
1,8	t		74 164	A3 243	33 63	33 63	11-0-3	0-3	0-3
1,8	u		75 165	A4 244	34 64	34 64	11-0-4	0-4	0-4
1,8	v		76 166	A5 245	35 65	35 65	11-0-5	0-5	0-5
1,8	w		77 167	A6 246	36 66	36 66	11-0-6	0-6	0-6
1,8	x		78 170	A7 247	37 67	37 67	11-0-7	0-7	0-7
1,8	y		79 171	A8 250	38 70	38 70	11-0-8	0-8	0-8
1,8	z		7A 172	A9 251	39 71	39 71	11-0-9	0-9	0-9
1,2	{	Left Brace	{ 7B 173	{ C0 300	0 00 00	+ 10 20	12-0	0	(12-0)(12)
2,3		Broken Vertical Line	7C 174	6A 152	1F 37	1E 36	12-11	12-7-8	12-6-8
1,2,7		Right Brace	} 7D 175	} D0 320	! 20 40	- 20 40	11-0	11-0	(11-0)(11)
2	~	Tilde	7E 176	A1 241	1F 37	1E 36	11-0-1	11-0-1	11-0-1
3	DEL	Delete	7F 177	07 007	1F 37	1E 36	12-7-9	12-7-8	12-6-8

- NOTES:
1. From EBCDIC or ASCII to HBCD or GBCD, this is a one-way correspondence.
 2. ISO defines these ASCII codes as variable for national usage.
 3. Since there is no corresponding character, a default character is substituted here; 36(octal)(#) for HBCD and 37(oct)(\) for GBCD.
 4. In HBCD, the code 57(octal) may represent 1/2 or 1.
 5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
 6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabets.
 7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (J). For the HBCD1 set, (+) and (-) are represented with the punch codes 12-0 and 11-0 while (?) and (J) are represented by 12 and 11. For the HBCD2 set, (+) and (-) are represented with punch codes 12 and 11 while (?) and (J) are represented with punch codes 12-0 and 11-0.
 8. These are EBCDIC control characters and are not defined in the ASCII standard.
 9. IBM defines these EBCDIC codes as national alphabetic extenders.

D. EBCDIC Sequence

This section contains the listing of the Unified Character Set of the EBCDIC Sequence.

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
3	NUL	Null	00 000	00 000	1F 37	1E 36	12-0-1-8-9		
3	SOH	Start of Heading	01 001	01 001	1F 37	1E 36	12-1-9		
3	STX	Start of Text	02 002	02 002	1F 37	1E 36	12-2-9		
3	ETX	End of Text	03 003	03 003	1F 37	1E 36	12-3-9		
3,8	PF	Punch Off	04 004	9C 234	1F 37	1E 36	12-4-9		
3	HT	Horizontal Tab	05 005	09 011	1F 37	1E 36	12-5-9		
3,8	LC	Lower Case	06 006	86 206	1F 37	1E 36	12-6-9		
3	DEL	Delete	07 007	7F 177	1F 37	1E 36	12-7-9		
3	GE	Graphic Escape	08 010	97 227	1F 37	1E 36	12-8-9		
3,8	RLF	Reverse Line Feed	09 011	8D 215	1F 37	1E 36	12-1-8-9		
3,8	SMM	Start of Manual Message	0A 012	8E 216	1F 37	1E 36	12-2-8-9		
3	VT	Vertical Tabulation	0B 013	0B 013	1F 37	1E 36	12-3-8-9		
3	FF	Form Feed	0C 014	0C 014	1F 37	1E 36	12-4-8-9		
3	CR	Carriage Return	0D 015	0D 015	1F 37	1E 36	12-5-8-9		
0	SO	Shift Out	0E 016	0E 016	1F 37	1E 36	12-6-8-9		
3	SI	Shift In	0F 017	0F 017	1F 37	1E 36	12-7-8-9		
3	DLE	Data Link Escape	10 020	10 020	1F 37	1E 36	12-11-1-8-9		
3	DC1	Device Control 1	11 021	11 021	1F 37	1E 36	11-1-9		
3	DC2	Device Control 2	12 022	12 022	1F 37	1E 36	11-2-9		
3,8	TM	Tape Mark	13 023	13 023	1F 37	1E 36	11-3-9		
3,8	RES	Restore	14 024	9D 235	1F 37	1E 36	4-8-9		
3,8	NL	New Line	15 025	85 205	1F 37	1E 36	5-8-9		
3	BS	Backspace	16 026	08 010	1F 37	1E 36	2-9		
3,8	IL	Idle	17 027	87 207	1F 37	1E 36	0-6-9		
3	CAN	Cancel	18 030	18 030	1F 37	1E 36	11-8-9		
3	EM	End of Medium	19 031	19 031	1F 37	1E 36	11-1-8-9		
3,8	CC	Cursor Control	1A 032	92 222	1F 37	1E 36	11-2-8-9		
3,8	CU1	Customer Use 1	1B 033	8F 217	1F 37	1E 36	11-3-8-9		
3	IFS	Interchange File Separator	1C 034	1C 034	1F 37	1E 36	11-4-8-9		
3	IGS	Interchange Group Separator	1D 035	1D 035	1F 37	1E 36	11-5-8-9		
3	IRS	Interchange Record Separator	1E 036	1E 036	1F 37	1E 36	11-6-8-9		
3	IUS	Interchange Unit Separator	1F 037	1F 037	1F 37	1E 36	11-7-8-9		

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
3,8	DS	Digit Select	20 040	80 200	1F 37	1E 36	11-0-1-8-9		
3,8	SOS	Start of Significance	21 041	81 201	1F 37	1E 36	0-1-9		
3,8	FS	File Separator	22 042	82 202	1F 37	1E 36	0-2-9		
3		UNDEFINED CODES	23 043	83 203	1F 37	1E 36	0-3-9		
3,8	BYP	Bypass	24 044	84 204	1F 37	1E 36	0-4-9		
3	LF	Line Feed	25 045	0A 012	1F 37	1E 36	0-5-9		
3	ETB	End of Transmission Block	26 046	17 027	1F 37	1E 36	0-6-9		
3	ESC	Escape	27 047	1B 033	1F 37	1E 36	0-7-9		
3		UNDEFINED CODES	28 050	88 210	1F 37	1E 36	0-8-9		
3		UNDEFINED CODES	29 051	89 211	1F 37	1E 36	0-1-8-9		
3,8	SM	Set Mode	2A 052	8A 212	1F 37	1E 36	0-2-8-9		
3,8	CU2	Customer Use 2	2B 053	8B 213	1F 37	1E 36	0-3-8-9		
3		UNDEFINED CODES	2C 054	8C 214	1F 37	1E 36	0-4-8-9		
3	ENQ	Enquiry	2D 055	05 005	1F 37	1E 36	0-5-8-9		
3	ACK	Acknowledge	2E 056	06 006	1F 37	1E 36	0-6-8-9		
3	BEL	Bell	2F 057	07 007	1F 37	1E 36	0-7-8-9		
3		UNDEFINED CODES	30 060	90 220	1F 37	1E 36	12-11-0-1-8-9		
3		UNDEFINED CODES	31 061	91 221	1F 37	1E 36	1-9		
3	SYN	Synchronous Idle	32 062	16 026	1F 37	1E 36	2-9		
3		UNDEFINED CODES	33 063	93 223	1F 37	1E 36	3-9		
3,8	PN	Punch On	34 064	94 224	1F 37	1E 36	4-9		
3,8	RS	Reader Stop	35 065	95 225	1F 37	1E 36	5-9		
3,8	UC	Upper Case	36 066	96 226	1F 37	1E 36	6-9		
3	EOT	End of Transmission	37 067	04 004	1F 37	1E 36	7-9		
3		UNDEFINED CODES	38 070	98 230	1F 37	1E 36	8-9		
3		UNDEFINED CODES	39 071	99 231	1F 37	1E 36	1-8-9		
3		UNDEFINED CODES	3A 072	9A 232	1F 37	1E 36	2-8-9		
3,8	CU3	Customer Use 3	3B 073	9B 233	1F 37	1E 36	3-8-9		
3	DC4	Device Control 4	3C 074	14 024	1F 37	1E 36	4-8-9		
3	NAK	Negative Acknowledge	3D 075	15 025	1F 37	1E 36	5-8-9		
3		UNDEFINED CODES	3E 076	9E 236	1F 37	1E 36	6-8-9		
3	SUB	Substitute	3F 077	1A 032	1F 37	1E 36	7-8-9		

EBCDIC Sequence

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
		Space, Blank	40 100	20 040	10 20	0D 15	Blank	Blank	Blank
3		UNDEFINED CODES	41 101	A0 240	1F 37	1E 36	12-0-1-9		
3		UNDEFINED CODES	42 102	A1 241	1F 37	1E 36	12-0-2-9		
3		UNDEFINED CODES	43 103	A2 242	1F 37	1E 36	12-0-3-9		
3		UNDEFINED CODES	44 104	A3 243	1F 37	1E 36	12-0-4-9		
3		UNDEFINED CODES	45 105	A4 244	1F 37	1E 36	12-0-5-9		
3		UNDEFINED CODES	46 106	A5 245	1F 37	1E 36	12-0-6-9		
3		UNDEFINED CODES	47 107	A6 246	1F 37	1E 36	12-0-7-9		
3		UNDEFINED CODES	48 110	A7 247	1F 37	1E 36	12-0-8-9		
3		UNDEFINED CODES	49 111	A8 250	1F 37	1E 36	12-1-8		
9	¢	Cents Sign	¢ 4A 112	[5B 133	[0A 12	¢ 3F 77	12-2-8	2-8	0-7-8
	.	Period, Decimal Point	4B 113	2E 056	1B 33	1B 33	12-3-8	12-3-8	12-3-8
	<	Less Than	4C 114	3C 074	1E 36	30 60	12-4-8	12-6-8	5-8
	(Left Parenthesis	4D 115	28 050	1D 35	3C 74	12-5-8	12-5-8	0-4-8
7	+	Plus Sign	4E 116	2B 053	30 60	10 20	12-6-8	12-0	(12-0) (12)
		Logical OR	4F 117	21 041	c 3F 77	R 3D 75	12-7-8	0-7-8	0-5-8
	&	Ampersand	50 120	26 046	1A 32	0F 17	12	12	7-8
3		UNDEFINED CODES	51 121	A9 251	1F 37	1E 36	12-11-1-9		
3		UNDEFINED CODES	52 122	AA 252	1F 37	1E 36	12-11-2-9		
3		UNDEFINED CODES	53 123	AB 253	1F 37	1E 36	12-11-3-9		
3		UNDEFINED CODES	54 124	AC 254	1F 37	1E 36	12-11-4-9		
3		UNDEFINED CODES	55 125	AD 255	1F 37	1E 36	12-11-5-9		
3		UNDEFINED CODES	56 126	AE 256	1F 37	1E 36	12-11-6-9		
3		UNDEFINED CODES	57 127	AF 257	1F 37	1E 36	12-11-7-9		
3		UNDEFINED CODES	58 130	B0 260	1F 37	1E 36	12-11-8-9		
3		UNDEFINED CODES	59 131	B1 261	1F 37	1E 36	11-1-8		
4,7,9	!	Exclamation Point EBCDIC	5A 132] 5D 135] 1C 34	2F 57	11-2-8	12-4-8	(11) (11-0)
2,9	\$	Dollar Sign	5B 133	24 044	2B 53	2B 53	11-3-8	11-3-8	11-3-8
	*	Asterisk	5C 134	2A 052	2C 54	2C 54	11-4-8	11-4-8	11-4-8
)	Right Parenthesis	5D 135	2B 053	2D 55	1C 34	11-5-8	11-5-8	12-4-8
	;	Semi-Colon	5E 136	3B 073	2E 56	1A 32	11-6-8	11-6-8	12-2-8
		Logical NOT	5F 137	* 5E 136	! 20 40	1E 36	11-7-8	11-0	12-6-8

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
7	-	Minus Sign, Hyphen	60 140	2D 055	2A 52	20 40	11	11	(11-0)(11)
		Slash	61 141	2F 057	31 61	31 61	0-1	0-1	0-1
3		UNDEFINED CODES	62 142	B2 262	1F 37	1E 36	11-0-2-9		
3		UNDEFINED CODES	63 143	B3 263	1F 37	1E 36	11-0-3-9		
3		UNDEFINED CODES	64 144	B4 264	1F 37	1E 36	11-0-4-9		
3		UNDEFINED CODES	65 145	B5 265	1F 37	1E 36	11-0-5-9		
3		UNDEFINED CODES	66 146	B6 266	1F 37	1E 36	11-0-6-9		
3		UNDEFINED CODES	67 147	B7 267	1F 37	1E 36	11-0-7-9		
3		UNDEFINED CODES	68 150	B8 270	1F 37	1E 36	11-0-8-9		
3		UNDEFINED CODES	69 151	B9 271	1F 37	1E 36	0-1-8		
3		Vertical Line	6A 152	7C 174	1F 37	1E 36	12-11	12-7-8	12-6-8
	,	Comma	6B 153	2C 054	3B 73	3B 73	0-3-8	0-3-8	0-3-8
	%	Percent Sign	6C 154	23 045	3C 74	3C 74	0-4-8	0-4-8	12-5-8
	_	Underscore	6D 155	5F 137	+ 3A 72	+ 3A 72	0-5-8	0-2-8	0-6-8
	>	Greater Than Sign	6E 156	3E 076	0E 16	0E 16	0-6-8	6-8	6-8
7	?	Question Mark	6F 157	3F 077	0F 17	1F 37	0-7-8	7-8	(12)(12-0)
3		UNDEFINED CODES	70 160	BA 272	1F 37	1E 36	12-11-0		
3		UNDEFINED CODES	71 161	BB 273	1F 37	1E 36	12-11-0-1-9		
3		UNDEFINED CODES	72 162	BC 274	1F 37	1E 36	12-11-0-2-9		
3		UNDEFINED CODES	73 163	BD 275	1F 37	1E 36	12-11-0-3-9		
3		UNDEFINED CODES	74 164	BE 276	1F 37	1E 36	12-11-0-4-9		
3		UNDEFINED CODES	75 165	BF 277	1F 37	1E 36	12-11-0-5-9		
3		UNDEFINED CODES	76 166	C0 300	1F 37	1E 36	12-11-0-6-9		
3		UNDEFINED CODES	77 167	C1 301	1F 37	1E 36	12-11-0-7-9		
3		UNDEFINED CODES	78 170	C2 302	1F 37	1E 36	12-11-0-8-9		
3	'	Grave Accent	79 171	60 140	1F 37	1E 36	1-8	12-7-8	12-6-8
	:	Colon	7A 172	3A 072	0D 15	0C 14	2-8	5-8	4-6
2,9	#	Number Sign	7B 173	23 043	0B 13	2A 52	3-8	3-8	11-2-8
2,9	@	At Sign	7C 174	40 100	0C 14	3A 72	4-8	4-8	0-2-8
	'	Prime, Apostrophe	7D 175	27 047	2F 57	0A 12	5-8	11-7-8	2-8
	=	Equal Sign	7E 176	3D 075	3D 75	0B 13	6-8	0-5-8	3-8
9	"	Quotation Mark	7F 177	22 042	3E 76	2D 55	7-8	0-6-8	11-5-8

EBCDIC Sequence

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
3		UNDEFINED CODES	80 200	C3 303	1F 37	1E 36	12-0-1-8		
1,6	a		81 201	61 141	11 21	11 21	12-0-1	12-1	12-1
1,6	b		82 202	62 142	12 22	12 22	12-0-2	12-2	12-2
1,6	c		83 203	63 143	13 23	13 23	12-0-3	12-3	12-3
1,6	d		84 204	64 144	14 24	14 24	12-0-4	12-4	12-4
1,6	e		85 205	65 145	15 25	15 25	12-0-5	12-5	12-5
1,6	f		86 206	66 146	16 26	16 26	12-0-6	12-6	12-6
1,6	g		87 207	67 147	17 27	17 27	12-0-7	12-7	12-7
1,6	h		88 210	68 150	18 30	18 30	12-0-8	12-8	12-8
1,6	i		89 211	69 151	19 31	19 31	12-0-9	12-9	12-9
3		UNDEFINED CODES	8A 212	C4 304	1F 37	1E 36	12-0-2-8		
3		UNDEFINED CODES	8B 213	C5 305	1F 37	1E 36	12-0-3-8		
3		UNDEFINED CODES	8C 214	C6 306	1F 37	1E 36	12-0-4-8		
3		UNDEFINED CODES	8D 215	C7 307	1F 37	1E 36	12-0-5-8		
3		UNDEFINED CODES	8E 216	C8 310	1F 37	1E 36	12-0-6-8		
3		UNDEFINED CODES	8F 217	C9 311	1F 37	1E 36	12-0-7-8		
3		UNDEFINED CODES	90 220	CA 312	1F 37	1E 36	12-11-1-8		
1,6	j		91 221	6A 152	21 41	21 41	12-11-1	11-1	11-1
1,6	k		92 222	6B 153	22 42	22 42	12-11-2	11-2	11-2
1,6	l		93 223	6C 154	23 43	23 43	12-11-3	11-3	11-3
1,6	m		94 224	6D 155	24 44	24 44	12-11-4	11-4	11-4
1,6	n		95 225	6E 156	25 45	25 45	12-11-5	11-5	11-5
1,6	o		96 226	6F 157	26 46	26 46	12-11-6	11-6	11-6
1,6	p		97 227	70 160	27 47	27 47	12-11-7	11-7	11-7
1,6	q		98 230	71 161	28 50	28 50	12-11-8	11-8	11-8
1,6	r		99 231	72 162	29 51	29 51	12-11-9	11-9	11-9
3		UNDEFINED CODES	9A 232	CB 313	1F 37	1E 36	12-11-2-8		
3		UNDEFINED CODES	9B 233	CC 314	1F 37	1E 36	12-11-3-8		
3		UNDEFINED CODES	9C 234	CD 315	1F 37	1E 36	12-11-4-8		
3		UNDEFINED CODES	9D 235	CE 316	1F 37	1E 36	12-11-5-8		
3		UNDEFINED CODES	9E 236	CF 317	1F 37	1E 36	12-11-6-8		
3		UNDEFINED CODES	9F 237	D0 320	1F 37	1E 36	12-11-7-8		

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD			
			CODE	CODE	CODE	CODE	ASCII/EBCDIC	GBCD	HBCD
			16 8	16 8	16 8	16 8	CARD CODE	CARD CODE	CARD CODE
3		UNDEFINED CODES	A0 240	D1 321	1F 37	1E 36	11-0-1-8		
3	~	Tilde	A1 241	7E 176	1F 37	1E 36	11-0-1	12-7-8	12-6-8
1,6	s		A2 242	73 163	32 63	32 62	11-0-2	0-2	0-2
1,6	t		A3 243	74 164	33 63	33 63	11-0-3	0-3	0-3
1,6	u		A4 244	75 165	34 64	34 64	11-0-4	0-4	0-4
1,6	v		A5 245	76 166	35 65	35 65	11-0-5	0-5	0-5
1,6	w		A6 246	77 167	36 66	36 66	11-0-6	0-6	0-6
1,6	x		A7 247	78 170	37 67	37 67	11-0-7	0-7	0-7
1,6	y		A8 250	79 171	38 70	38 70	11-0-8	0-8	0-8
1,6	z		A9 251	7A 172	39 71	39 71	11-0-9	0-9	0-9
3		UNDEFINED CODES	AA 252	D2 322	1F 37	1E 36	11-0-2-8		
3		UNDEFINED CODES	AB 253	D3 323	1F 37	1E 36	11-0-3-8		
3		UNDEFINED CODES	AC 254	D4 324	1F 37	1E 36	11-0-4-8		
3		UNDEFINED CODES	AD 255	D5 325	1F 37	1E 36	11-0-5-8		
3		UNDEFINED CODES	AE 256	D6 326	1F 37	1E 36	11-0-6-8		
3		UNDEFINED CODES	AF 257	D7 327	1F 37	1E 36	11-0-7-8		
3		UNDEFINED CODES	B0 260	D8 330	1F 37	1E 36	12-11-0-1-8		
3		UNDEFINED CODES	B1 261	D9 331	1F 37	1E 36	12-11-0-1		
3		UNDEFINED CODES	B2 262	DA 332	1F 37	1E 36	12-11-0-2		
3		UNDEFINED CODES	B3 263	DB 333	1F 37	1E 36	12-11-0-3		
3		UNDEFINED CODES	B4 264	DC 334	1F 37	1E 36	12-11-0-4		
3		UNDEFINED CODES	B5 265	DD 335	1F 37	1E 36	12-11-0-5		
3		UNDEFINED CODES	B6 266	DE 336	1F 37	1E 36	12-11-0-6		
3		UNDEFINED CODES	B7 267	DF 337	1F 37	1E 36	12-11-0-7		
3		UNDEFINED CODES	B8 270	E0 340	1F 37	1E 36	12-11-0-8		
3		UNDEFINED CODES	B9 271	E1 341	1F 37	1E 36	12-11-0-9		
3		UNDEFINED CODES	BA 272	E2 342	1F 37	1E 36	12-11-0-2-8		
3		UNDEFINED CODES	BB 273	E3 343	1F 37	1E 36	12-11-0-3-8		
3		UNDEFINED CODES	BC 274	E4 344	1F 37	1E 36	12-11-0-4-8		
3		UNDEFINED CODES	BD 275	E5 345	1F 37	1E 36	12-11-0-5-8		
3		UNDEFINED CODES	BE 276	E6 346	1F 37	1E 36	12-11-0-6-8		
3		UNDEFINED CODES	BF 277	E7 347	1F 37	1E 36	12-11-0-7-8		

EBCDIC Sequence

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			
1	{	Opening Brace	{ C0 300	{ 7B 173	0 00 00	+ 10 20	12-0	0	(12-0)(12)
	A		C1 301	41 101	11 21	11 21	12-1	12-1	12-1
	B		C2 302	42 102	12 22	12 22	12-2	12-2	12-2
	C		C3 303	43 103	13 23	13 23	12-3	12-3	12-3
	D		C4 304	44 104	14 24	14 24	12-4	12-4	12-4
	E		C5 305	45 105	15 25	15 25	12-5	12-5	12-5
	F		C6 306	46 106	16 26	16 26	12-6	12-6	12-6
	G		C7 307	47 107	17 27	17 27	12-7	12-7	12-7
	H		C8 310	48 110	18 30	18 30	12-8	12-8	12-8
	I		C9 311	49 111	19 31	19 31	12-9	12-9	12-9
3		UNDEFINED CODES	CA 312	E8 350	1F 37	1E 36	12-0-2-8-9		
3		UNDEFINED CODES	CB 313	E9 351	1F 37	1E 36	12-0-3-8-9		
3		UNDEFINED CODES	CC 314	EA 352	1F 37	1E 36	12-0-4-8-9		
3		UNDEFINED CODES	CD 315	EB 353	1F 37	1E 36	12-0-5-8-9		
3		UNDEFINED CODES	CE 316	EC 354	1F 37	1E 36	12-0-6-8-9		
3		UNDEFINED CODES	CF 317	ED 355	1F 37	1E 36	12-0-7-8-9		
1,7	}	Closing Brace	} D0 320	} 7D 175	! 20 40	- 20 40	11-0	11-0	(11-0)(11)
	J		D1 321	4A 112	21 41	21 41	11-1	11-1	11-1
	K		D2 322	4B 113	22 42	22 42	11-2	11-2	11-2
	L		D3 323	4C 114	23 43	23 43	11-3	11-3	11-3
	M		D4 324	4D 115	24 44	24 44	11-4	11-4	11-4
	N		D5 325	4E 116	25 45	25 45	11-5	11-5	11-5
	O		D6 326	4F 117	26 46	26 46	11-6	11-6	11-6
	P		D7 327	50 120	27 47	27 47	11-7	11-7	11-7
	Q		D8 330	51 121	28 50	28 50	11-8	11-8	11-8
	R		D9 331	52 122	29 51	29 51	11-9	11-9	11-9
3		UNDEFINED CODES	DA 332	EE 356	1F 37	1E 36	12-11-2-8-9		
3		UNDEFINED CODES	DB 333	EF 357	1F 37	1E 36	12-11-3-8-9		
3		UNDEFINED CODES	DC 334	F0 360	1F 37	1E 36	12-11-4-8-9		
3		UNDEFINED CODES	DD 335	F1 361	1F 37	1E 36	12-11-5-8-9		
3		UNDEFINED CODES	DE 336	F2 362	1F 37	1E 36	12-11-6-8-9		
3		UNDEFINED CODES	DF 337	F3 363	1F 37	1E 36	12-11-7-8-9		

NOTE	SYMBOL	NAME	EBCDIC	ASCII	GBCD	HBCD	ASCII/EBCDIC CARD CODE	GBCD CARD CODE	HBCD CARD CODE
			CODE 16 8	CODE 16 8	CODE 16 8	CODE 16 8			

		Reverse Slant	\ E0 340	\ 5C 134	1F 37	1E 36	0-2-8		
3		UNDEFINED CODES	E1 341	9F 237	1F 37	1E 36	11-0-1-9		
	S		E2 342	53 123	32 63	32 62	0-2	0-2	0-2
	T		E3 343	54 124	33 63	33 63	0-3	0-3	0-3
	U		E4 344	55 125	34 64	34 64	0-4	0-4	0-4
	V		E5 345	56 126	35 65	35 65	0-5	0-5	0-5
	W		E6 346	57 127	36 66	36 66	0-6	0-6	0-6
	X		E7 347	58 130	37 67	37 67	0-7	0-7	0-7
	Y		E8 350	59 131	38 70	38 70	0-8	0-8	0-8
	Z		E9 351	5A 132	39 71	39 71	0-9	0-9	0-9
3		UNDEFINED CODES	EA 352	F4 364	1F 37	1E 36	11-0-2-8-9		
3		UNDEFINED CODES	EB 353	F5 365	1F 37	1E 36	11-0-3-8-9		
3		UNDEFINED CODES	EC 354	F6 366	1F 37	1E 36	11-0-4-8-9		
3		UNDEFINED CODES	ED 355	F7 367	1F 37	1E 36	11-0-5-8-9		
3		UNDEFINED CODES	EE 356	F8 370	1F 37	1E 36	11-0-6-8-9		
3		UNDEFINED CODES	EF 357	F9 371	1F 37	1E 36	11-0-7-8-9		
	0		F0 360	30 060	00 00	00 00	0	0	0
	1		F1 361	31 061	01 01	01 01	1	1	1
	2		F2 362	32 062	02 02	02 02	2	2	2
	3		F3 363	33 063	03 03	03 03	3	3	3
	4		F4 364	34 064	04 04	04 04	4	4	4
	5		F5 365	35 065	05 05	05 05	5	5	5
	6		F6 366	36 066	06 06	06 06	6	6	6
	7		F7 367	37 067	07 07	07 07	7	7	7
	8		F8 370	38 070	08 10	08 10	8	8	8
	9		F9 371	39 071	09 11	09 11	9	9	9
3		Long Vertical Mark	FA 372	FA 372	1F 37	1E 36	12-11-0-2-8-9		
3		UNDEFINED CODES	FB 373	FB 373	1F 37	1E 36	12-11-0-3-8-9		
3		UNDEFINED CODES	FC 374	FC 374	1F 37	1E 36	12-11-0-4-8-9		
3		UNDEFINED CODES	FD 375	FD 375	1F 37	1E 36	12-11-0-5-8-9		
3		UNDEFINED CODES	FE 376	FE 376	1F 37	1E 36	12-11-0-6-8-9		
3	EO	Eight Ones	FF 377	FF 377	1F 37	1E 36	12-11-0-7-8-9		

EBCDIC Sequence

- NOTES:
1. From EBCDIC or ASCII to HBCD or GBCD, this is a one-way correspondence.
 2. ISO defines these ASCII codes as variable for national usage.
 3. Since there is no corresponding character, a default character is substituted here; 36(octal)(#) for HBCD and 37(oct)(\) for GBCD.
 4. In HBCD, the code 57(octal) may represent 1/2 or 1.
 5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
 6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabets.
 7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (J). For the HBCD1 set, (+) and (-) are represented with the punch codes 12-0 and 11-0 while (?) and (J) are represented by 12 and 11. For the HBCD2 set, (+) and (-) are represented with punch codes 12 and 11 while (?) and (J) are represented with punch codes 12-0 and 11-0.
 8. These are EBCDIC control characters and are not defined in the ASCII standard.
 9. IBM defines these EBCDIC codes as national alphabetic extenders.

E. GBCD Sequence

This section contains the listing of the Unified Character Set of the GBCD Sequence.

NOTE	SYMBOL	NAME	GBCD	HBCD	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
	0		00 00	00 00	30 060	F0 360	0	0	0
	1		01 01	01 01	31 061	F1 361	1	1	1
	2		02 02	02 02	32 062	F2 362	2	2	2
	3		03 03	03 03	33 063	F3 363	3	3	3
	4		04 04	04 04	34 064	F4 364	4	4	4
	5		05 05	05 05	35 065	F5 365	5	5	5
	6		06 06	06 06	36 066	F6 366	6	6	6
	7		07 07	07 07	37 067	F7 367	7	7	7
	8		08 10	08 10	38 070	F8 370	8	8	8
	9		09 11	09 11	39 071	F9 371	9	9	9
	[Left Bracket	[0A 12	c 3F 77	[5B 133	c 4A 112	2-8	0-7-8	12-2-8
	#	Number Sign	0B 13	2A 52	23 043	7B 173	3-8	11-2-8	3-8
	@	At Sign	0C 14	3A 72	40 100	7C 174	4-8	0-2-8	4-8
	:	Colon	0D 15	0C 14	3A 072	7A 172	5-8	4-8	2-8
	>	Greater Than	0E 16	0E 16	3E 076	6E 156	6-8	6-8	0-6-8
7	?	Question Mark	0F 17	1F 37	3F 077	6F 157	7-8	(12)(12-0)	0-7-8
		Space. Blank	10 20	0D 15	20 040	40 100	Blank	Blank	Blank
	A		11 21	11 21	41 101	C1 301	12-1	12-1	12-1
	B		12 22	12 22	42 102	C2 302	12-2	12-2	12-2
	C		13 23	13 23	43 103	C3 303	12-3	12-3	12-3
	D		14 24	14 24	44 104	C4 304	12-4	12-4	12-4
	E		15 25	15 25	45 105	C5 305	12-5	12-5	12-5
	F		16 26	16 26	46 106	C6 306	12-6	12-6	12-6
	G		17 27	17 27	47 107	C7 307	12-7	12-7	12-7
	H		18 30	18 30	48 110	C8 310	12-8	12-8	12-8
	I		19 31	19 31	49 111	C9 311	12-9	12-9	12-9
	&	Ampersand	1A 32	0F 17	26 046	50 120	12	7-8	12
	.	Period	1B 33	1B 33	2E 056	4B 113	12-3-8	12-3-8	12-3-8
4,7]	Right Bracket] 1C 34	2F 57] 5D 135	5A 132	12-4-8	(11)(11-0)	11-2-8
	(Left Parenthesis	1D 35	3C 74	28 050	4D 115	12-5-8	0-4-8	12-5-8
	<	Less Than	1E 36	30 60	3C 074	4C 114	12-6-8	5-8	12-4-8
	\	Reverse Slant	\ 1F 37	c 2E 56	\ 5C 134	\ E0 340	12-7-8	11-6-8	0-2-8

NOTE	SYMBOL	NAME	GBCD	HBCD	ASCII	EBCDIC	GBCD	HBCD	ASCII/EBCDIC
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
	I	Upward Arrow	! 20 40	1E 36	' 5E 136	5F 137	11-0	12-6-8	11-7-8
	J		21 41	21 41	4A 112	D1 321	11-1	11-1	11-1
	K		22 42	22 42	4B 113	D2 322	11-2	11-2	11-2
	L		23 43	23 43	4C 114	D3 323	11-3	11-3	11-3
	M		24 44	24 44	4D 115	D4 324	11-4	11-4	11-4
	N		25 45	25 45	4E 116	D5 325	11-5	11-5	11-5
	O		26 46	26 46	4F 117	D6 326	11-6	11-6	11-6
	P		27 47	27 47	50 120	D7 327	11-7	11-7	11-7
	Q		28 50	28 50	51 121	D8 330	11-8	11-8	11-8
	R		29 51	29 51	52 122	D9 331	11-9	11-9	11-9
7	-	Hyphen Minus	2A 52	20 40	2D 055	60 140	11	(11-0)(11)	11
	\$	Dollar Sign	2B 53	2B 53	24 044	5B 133	11-3-8	11-3-8	11-3-8
	*	Asterisk	2C 54	2C 54	2A 052	5C 134	11-4-8	11-4-8	11-4-8
)	Right Parenthesis	2D 55	1C 34	29 051	5D 135	11-5-8	12-4-8	11-5-8
	;	Semi-Colon	2E 56	1A 32	3B 073	5E 136	11-6-8	12-2-8	11-6-8
	'	Apostrophe	2F 57	0A 12	27 047	7D 175	11-7-8	2-8	5-8
7	+	Plus	30 60	10 20	2B 053	4E 116	12-0	(12-0)(12)	12-6-8
	/	Slash	31 61	31 61	2F 057	61 141	0-1	0-1	0-1
	S		32 62	32 62	53 123	E2 342	0-2	0-2	0-2
	T		33 63	33 63	54 124	E3 343	0-3	0-3	0-3
	U		34 64	34 64	55 125	E4 344	0-4	0-4	0-4
	V		35 65	35 65	56 126	E5 345	0-5	0-5	0-5
	W		36 66	36 66	57 127	E6 346	0-6	0-6	0-6
	X		37 67	37 67	58 130	E7 347	0-7	0-7	0-7
	Y		38 70	38 70	59 131	E8 350	0-8	0-8	0-8
	Z		39 71	39 71	5A 132	E9 351	0-9	0-9	0-9
	<-	Left Arrow	<- 3A 72	3E 76	- 5F 137	- 6D 155	0-2-8	0-6-8	0-5-8
	,	Comma	3B 73	3B 73	2C 054	6B 153	0-3-8	0-3-8	0-3-8
	%	Percent Sign	3C 74	1D 35	25 045	6C 154	0-4-8	12-5-8	0-4-8
	=	Equal	3D 75	0B 13	3D 075	7E 176	0-5-8	3-8	6-8
	"	Double Quote	3E 76	2D 55	22 042	7F 177	0-6-8	11-5-8	7-8
	!	Exclamation Point (ASCII)	! 3F 77	R 3D 75	C 21 041	4F 117	0-7-8	0-5-8	12-7-8

GBCD Sequence

- NOTES:
1. From EBCDIC or ASCII to HBCD or GBCD, this is a one-way correspondence.
 2. ISO defines these ASCII codes as variable for national usage.
 3. Since there is no corresponding character, a default character is substituted here; 36(octal)(#) for HBCD and 37(oct)(\) for GBCD.
 4. In HBCD, the code 57(octal) may represent 1/2 or 1.
 5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
 6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabets.
 7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (J). For the HBCD1 set, (+) and (-) are represented with the punch codes 12-0 and 11-0 while (?) and (J) are represented by 12 and 11. For the HBCD2 set, (+) and (-) are represented with punch codes 12 and 11 while (?) and (J) are represented with punch codes 12-0 and 11-0.
 8. These are EBCDIC control characters and are not defined in the ASCII standard.
 9. IBM defines these EBCDIC codes as national alphabetic extenders.

F. HBCD Sequence

This section contains the listing of the Unified Character Set of the HBCD Sequence.

NOTE	SYMBOL	NAME	HBCD	GBCD	ASCII	EBCDIC	HBCD	GBCD	ASCII/EBCDIC
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
	0		00 00	00 00	30 060	F0 360	0	0	0
	1		01 01	01 01	31 061	F1 361	1	1	1
	2		02 02	02 02	32 062	F2 362	2	2	2
	3		03 03	03 03	33 063	F3 363	3	3	3
	4		04 04	04 04	34 064	F4 364	4	4	4
	5		05 05	05 05	35 065	F5 365	5	5	5
	6		06 06	06 06	36 066	F6 366	6	6	6
	7		07 07	07 07	37 067	F7 367	7	7	7
	8		08 10	08 10	38 070	F8 370	8	8	8
	9		09 11	09 11	39 071	F9 371	9	9	9
	'	Apostrophe	0A 12	2F 57	27 047	7D 175	2-8	11-7-8	5-8
	=	Equal	0B 13	3D 75	3D 075	7E 176	3-8	0-5-6	6-8
	:	Colon	0C 14	0D 15	3A 072	7A 172	4-8	5-8	2-8
		Space, Blank	0D 15	10 20	20 040	40 100	Blank	Blank	Blank
	>	Greater Than	0E 16	0E 16	3E 076	6E 156	6-8	6-8	0-6-8
	&	Ampersand	0F 17	1A 32	26 046	50 120	7-8	12	12
	+	Plus	10 20	30 60	2B 053	4E 116	(12-0)(12)	12-0	12-6-8
	A		11 21	11 21	41 101	C1 301	12-1	12-1	12-1
	B		12 22	12 22	42 102	C2 302	12-2	12-2	12-2
	C		13 23	13 23	43 103	C3 303	12-3	12-3	12-3
	D		14 24	14 24	44 104	C4 304	12-4	12-4	12-4
	E		15 25	15 25	45 105	C5 305	12-5	12-5	12-5
	F		16 26	16 26	46 106	C6 306	12-6	12-6	12-6
	G		17 27	17 27	47 107	C7 307	12-7	12-7	12-7
	H		18 30	18 30	48 110	C8 310	12-8	12-8	12-8
	I		19 31	19 31	49 111	C9 311	12-9	12-9	12-9
	;	Ampersand	1A 32	2E 56	3B 073	5E 136	12-2-8	11-6-8	11-6-8
	.	Period	1B 33	1B 33	2E 056	4B 113	12-3-8	12-3-8	12-3-8
)	Right Parenthesis	1C 34	2D 55	29 051	5D 135	12-4-8	11-5-8	11-5-8
	%	Percent Sign	1D 35	3C 74	25 045	6C 154	12-5-8	0-4-8	0-4-8
	#	Closed Box	1E 36	20 40	5E 136	5F 137	12-6-8	11-0	11-7-8
	?	Question Mark	1F 37	0F 17	3F 077	6F 157	(12)(12-0)	7-8	0-7-8

NOTE	SYMBOL	NAME	HBCD	GBCD	ASCII	EBCDIC	HBCD	GBCD	ASCII/EBCDIC
			CODE	CODE	CODE	CODE	CARD CODE	CARD CODE	CARD CODE
			16 8	16 8	16 8	16 8			
7	-	Hyphen, Minus	20 40	2A 52	' 2D 055	60 140	(11-0)(11)	11	11
	J		21 41	21 41	4A 112	D1 321	11-1	11-1	11-1
	K		22 42	22 42	4B 113	D2 322	11-2	11-2	11-2
	L		23 43	23 43	4C 114	D3 323	11-3	11-3	11-3
	M		24 44	24 44	4D 115	D4 324	11-4	11-4	11-4
	N		25 45	25 45	4E 116	D5 325	11-5	11-5	11-5
	O		26 46	26 46	4F 117	D6 326	11-6	11-6	11-6
	P		27 47	27 47	50 120	D7 327	11-7	11-7	11-7
	Q		28 50	28 50	51 121	D8 330	11-8	11-8	11-8
	R		29 51	29 51	52 122	D9 331	11-9	11-9	11-9
	#	Number Sign	2A 52	0B 13	23 043	7B 173	11-2-8	3-8	3-8
	\$	Dollar Sign	2B 53	2B 53	24 044	5B 133	11-3-8	11-3-8	11-3-8
	*	Asterisk	2C 54	2C 54	2A 052	5C 134	11-4-8	11-4-8	11-4-8
	"	Double Quote	2D 55	3E 76	22 042	7F 137	11-5-8	0-6-8	7-8
	≠	Not Equal	= 2E 56	\ 1F 37	\ 5C 134	\ E0 340	11-6-8	12-7-8	0-2-8
4,7	!	Exclamation Point (EBCDIC)	2F 57] 1C 84] 5D 135	} 5A 132	(11)(11-0)	12-4-8	11-2-0
	<	Less Than	30 60	1E 36	3C 074	4C 114	5-8	12-6-8	12-4-8
	/	Slash	31 61	31 61	2F 057	61 141	0-1	0-1	0-1
	S		32 62	32 62	53 123	E2 342	0-2	0-2	0-2
	T		33 63	33 63	54 124	E3 343	0-3	0-3	0-3
	U		34 64	34 64	55 125	E4 344	0-4	0-4	0-4
	V		35 65	35 65	56 126	E5 345	0-5	0-5	0-5
	W		36 66	36 66	57 127	E6 346	0-6	0-6	0-6
	X		37 67	37 67	58 130	E7 347	0-7	0-7	0-7
	Y		38 70	38 70	59 131	E8 350	0-8	0-8	0-8
	Z		39 71	39 71	5A 132	E9 351	0-9	0-9	0-9
	@	At Sign	3A 72	0C 14	40 100	7C 174	0-2-8	4-8	4-8
	,	Comma	3B 73	3B 73	2C 054	6B 153	0-3-8	0-3-8	0-3-8
	(Left Parenthesis	3C 74	1D 35	28 050	4D 115	0-4-8	12-5-8	12-5-8
	CR	Credit Sign	R 3D 75	3F 77	C 21 041	4F 117	0-5-8	0-7-8	12-7-8
	"	Open Box	3E 76	+ 3A 72	- 5F 137	- 6D 115	0-6-8	0-2-8	0-5-8
	¢	Cents Sign	¢ 3F 77	[0A 12	[5B 133	c 4A 112	0-7-8	2-8	12-2-8

HBCD Sequence

- NOTES:
1. From EBCDIC or ASCII to HBCD or GBCD, this is a one-way correspondence.
 2. ISO defines these ASCII codes as variable for national usage.
 3. Since there is no corresponding character, a default character is substituted here; 36(octal)(#) for HBCD and 37(oct)(\) for GBCD.
 4. In HBCD, the code 57(octal) may represent 1/2 or 1.
 5. TM occupies the same position as DC3. TM is an EBCDIC control character while DC3 is an ASCII control character.
 6. The internal and punched card codes shown for HBCD and GBCD are for capital alphabets.
 7. There are two HBCD card code sets (HBCD1 and HBCD2), the difference being the card punch representation for (+) and (-), and for (?) and (J). For the HBCD1 set, (+) and (-) are represented with the punch codes 12-0 and 11-0 while (?) and (J) are represented by 12 and 11. For the HBCD2 set, (+) and (-) are represented with punch codes 12 and 11 while (?) and (J) are represented with punch codes 12-0 and 11-0.
 8. These are EBCDIC control characters and are not defined in the ASCII standard.
 9. IBM defines these EBCDIC codes as national alphabetic extenders.

Index

9

9-BIT

9-bit output 7-30

A

ABBREVIATIONS

abbreviations and symbols 8-5

ACCESS

accessing virtual memory 3-3

ADDRESS

address register alter
contents 7-11
address register
instructions 7-10
address register special arithmetic
instructions 8-13
address truncation 5-81
address wraparound 7-73
alphanumeric/numeric address
preparation 5-43
effective address
generation 5-51
effective address to register
instructions 7-4
ES address modification with
AR 5-51
generating a memory address in
virtual memory 3-3
instruction address
procedure 5-60
operand address procedure 5-60
operand descriptor address
preparation 5-40

single-word address
modification 5-28
virtual address 5-68
virtual address generation
(NS) 5-61
virtual address is a combination of
working space number and
offset 3-5
word address 5-34

ADDRESS MODIFICATION

Indirect Then Register (IR)
address modification (coding
examples) 5-11
register (R) address modification
variation types (list of coding
examples) 5-6

ADDRESS TRAP

address trap fault 6-36

ADDRESSING

virtual memory addressing 5-59

ADT

address trap fault 6-36

ALL MODES

all mode faults 6-40

ALPHANUMERIC

alphanumeric instructions 7-7,
7-25
alphanumeric operand descriptor
format 7-26
alphanumeric/numeric address
preparation 5-43

ALTER

address register alter
contents 7-11

ARITHMETIC

- address register special arithmetic instructions 8-13
- fixed-point arithmetic instructions 7-4
- floating-point arithmetic instructions 7-5

ASCII

- character codes for ascii and ebcdic overpunched sign 11-178

ATTRIBUTES

- common attributes of instructions 8-9

B

BASE

- base value 5-59
- Page Directory Base Registers (PDBR) (format, description, function) 4-32
- paging (vs/xa) 5-69
- relationship of base and bound to a segment (description, definitions) 3-5

BASIC

- basic features 7-1

BCD

- Binary-To-BCD Conversion 7-68

BINARY

- Binary-To-BCD Conversion 7-68
- conversions between binary and decimal numbers 7-35

BIT

- bit operations 5-46
- bit string instructions 7-7, 7-33
- bit string operand descriptor format 7-34
- bit strings and index table of translate instruction 5-83

- housekeeping bit 6-39, 7-61
- page access bit 6-41
- page modify bit 6-41

BLANK-WHEN-ZERO

- blank-when-zero flag 7-41

BND

- bound fault 6-19

BOLR

- bolr control field 9-81
- boolean result 7-33

BOOL

- boolean operators 7-14

BOOLEAN

- boolean expressions 7-14
- boolean operation instructions 7-14
- boolean operations 7-3, 7-33
- evaluation of boolean expressions 7-14

BOUND

- bound check equations 5-83
- bound field 11-25
- bound value 5-59
- bounds checking 5-81
- description of segment size 3-5
- modifying the bound field 12-26

BOUND FAULTS

- bound faults 11-56

BOUND VALID,FLAG

- bound valid flag 6-38

BOUNDARY

- byte boundary 5-84

BYTE

- BYTE boundary 5-84
- byte checks 5-83
- byte operations 5-82
- byte positions 13-91, 13-93

C**CHARACTER**

- character codes for ascii and ebcidic overpunched sign 11-178
- character move to/from register instructions 8-15
- character operations 5-48
- character positions 13-102

CIRCUITRY

- processor logic circuitry 12-31

CLIMB

- change of domain instruction (general procedures and exceptions) 3-15
- Domain Transfer (CLIMB) 7-61
- processor interprets wired-in climb 6-44
- transfer of domain from one process to another 3-3

CMD

- command fault 6-17

CODE

- OPERATION code maps for the DPS 9000 (tables separated by either bit 27 = 0 or bit 27 = 1) B-1

CODES

- character codes for ascii and ebcidic overpunched sign 11-178
- Micro Operation Code Assignment Map 7-60
- mnemonic code 8-3
- octal value of the operation code 8-3

COMMAND

- command faults 11-56

COMPARE

- comparison operations 7-3
- data comparison 7-9

CON

- connect fault 6-32

CONSTANTS

- conversion constants 8-105

CONVERSION

- Binary-To-BCD Conversion 7-68
- conversion constants 8-105
- conversion instructions 7-8
- conversions between binary and decimal numbers 7-35
- data conversion instructions 7-35
- Gray-To-Binary Conversion 7-68

COPY

- copy option 11-73

D**DATA**

- data comparison 7-9
- data conversion instructions 7-35
- data movement capability 7-9
- data movement instructions 7-3
- data shifting instructions 7-3
- procedure for transfer of double-precision data 2-2

DATA STACK SEGMENT

- explanation of domains in the virtual environment 3-15

DECIMAL

- conversions between binary and decimal numbers 7-35

DESCRIPTOR

- alphanumeric operand descriptor format 7-26
- bit string operand descriptor format 7-34
- Descriptor Register Instructions 7-61

DESCRIPTOR (cont.)

- descriptors for defining a domain
(contained in the linkage
segment) 3-15
- distinction between vector
arguments and
descriptors 3-15
- entry descriptor 6-1
- general description of segment
descriptors 3-5
- numeric operand descriptor
format 7-30
- operand descriptor address
preparation 5-40
- operand descriptor indirect pointer
format 7-25
- operand descriptor modification
(ES) 5-57
- operand descriptors 5-35
- operand descriptors and indirect
pointers 7-25
- passing segment descriptors to a
process 3-3
- segment descriptor 5-59
- standard descriptor 5-61, 9-20,
11-91

DIRECT OPERAND

- direct operand address
modification (explanation) 5-5

DIVISION

- division arithmetic
instructions 7-4

DLF

- dynamic linking fault 6-25

DOMAIN

- distinction between calling and
called domains 3-15
- distinction of domain as a logical
element of memory in the
hardware 3-3
- Domain Transfer (CLIMB) 7-61
- explanation of domains in the
virtual environment
(figure) 3-15

DOUBLE-WORD

- word and double-word
operations 5-81

DRL

- derail fault 6-21

DVCF

- divide check fault 6-15

E

EBCDIC

- character codes for ascii and
ebcdic overpunched
sign 11-178

EDIT

- edit flags 7-41
- edit insertion table 7-38
- edited move micro
operations 7-8
- micro operations for edit
instructions mve and
mvne 7-37

EFFECTIVE

- effective address
generation 5-51
- effective address to register
instructions 7-4

END

- end suppression flag 7-41

ENTRY

- entry descriptor 6-1

ENTRY DESCRIPTOR

- function of entry descriptor in
domains 3-15

EQUATIONS

- bound check equations 5-83

ES

- effective address
generation 5-51

ES (cont.)

ES address modification with
AR 5-51
operand descriptor modification
(ES) 5-57
tag field modification ES 5-53

EXECUTE

Execute Instructions 7-68
execute permission flag 6-38

EXF

execute fault 6-10

EXPRESSIONS

boolean expressions 7-14
evaluation of boolean
expressions 7-14

F**FACTOR**

scaling factor 7-31

FAULT

address trap fault 6-36
backup 6-3
bound fault 6-19
class 1 security fault 6-24
class 2 fault security fault 6-29
command fault 6-17
connect fault 6-32
derail fault 6-21
description of faults and
interrupts 6-1
divide check fault 6-15
dynamic linking fault 6-25
execute fault 6-10
fault procedure 6-2
fault tag fault 6-23
highest priority fault 6-4
illegal procedure fault 6-22
lockup fault 6-12
master mode entry fault 6-20
meep fault 6-35
memsys fault 6-13
missing page fault 6-28
missing section fault 6-27
missing segment fault 6-18

missing working space
fault 6-26
operation not completed
fault 6-11
overflow fault 6-16
safe store stack fault 6-30
shutdown fault 6-34
startup fault 6-9
timer runout fault 6-33

FAULTS

all modes faults 6-40
command faults 11-56
illegal procedure (ipr)
faults 11-55
master mode faults 6-40
privileged master mode
faults 6-39
slave mode faults 6-40

FDIV

divide check fault 6-15

FIELD

bolr control field 9-81
bound field 11-25
modifying the bound field 12-26
multiword modification
field 7-24
tag field modification ES 5-53

FIXED-POINT

fixed-point arithmetic
instructions 7-4
fixed-point numbers
(description,illustration) 2-5

FLAG

blank-when-zero flag 7-41
end suppression flag 7-41
execute permission flag 6-38
privileged flag 6-38
save permission flag 6-38
segment present flag 6-38
sign flag 7-41
zero flag 7-41

FLAGS

edit flags 7-41

FLOATING-POINT

floating-point arithmetic
instructions 7-5

FORMAT

alphanumeric operand descriptor
format 7-26
bit string operand descriptor
format 7-34
format of instruction
description 8-2
instruction word formats 8-9
numeric operand descriptor
format 7-30
operand descriptor indirect pointer
format 7-25
page table word (ptw) format
(vs/xa) 5-74

FRAMED

framed stack space 9-22

FTAG

fault tag fault 6-23

G

GENERATION

effective address
generation 5-51

GRAY-TO-BINARY

Gray-To-Binary
Conversion 7-68

GXN

GXn register in R
modification 5-50

H

HOUSEKEEPING

housekeeping bit 6-39, 7-61

I

ICLIMB

iclimb version of climb 6-1

ILLEGAL

illegal modification 8-9
illegal procedure (ipr)
faults 11-55

INDEX

bit strings and index table of
translate instruction 5-83
index register symbols 5-34

INDICATOR

Index Registers (IR) (format,
description, function) 4-24
parity indicator 8-9
Transfer on Tally Runout
Indicator OFF 14-58
transfer on tally runout indicator
on 14-61

INDIRECT

indirect word 5-39
operand descriptor indirect pointer
format 7-25
operand descriptors and indirect
pointers 7-25

INDIRECT ADDRESSING

Indirect Then Register (IR)
address modification (coding
examples) 5-11

INDIRECT THEN REGISTER (IR)

procedure for IR address
modification (coding
examples) 5-11

INSERTION

edit insertion table 7-38

INSTRUCTION

single-word instruction
interrupt 6-42

INSTRUCTIONS

- address register
 - instructions 7-10
- address register special arithmetic
 - instructions 8-13
- alphanumeric instructions 7-7, 7-25
- bit string instructions 7-7, 7-33
- bit strings and index table of
 - translate instruction 5-83
- boolean operation
 - instructions 7-14
- character move to/from register
 - instructions 8-15
- common attributes of
 - instructions 8-9
- conversion instructions 7-8
- data conversion
 - instructions 7-35
- data movement instructions 7-3
- data shifting instructions 7-3
- Descriptor Register
 - Instructions 7-61
- effective address to register
 - instructions 7-4
- Execute Instructions 7-68
- fixed-point arithmetic
 - instructions 7-4
- floating-point arithmetic
 - instructions 7-5
- format of instruction
 - description 8-2
- instruction address
 - procedure 5-60
- instruction word formats 8-9
- micro operations for edit
 - instructions mve and mvne 7-37
- multiword instruction
 - capabilities 7-9
- multiword instructions 8-11
- numeric instructions 7-7, 7-30
- Privileged Instructions 7-61
- register to register
 - instructions 8-16
- Register to register
 - Instructions 7-64
- Repeat Instructions 7-69
- single-word instructions 7-2, 8-9
- special address register
 - instructions 7-13

- Transfer Instructions 7-67
- Virtual Memory
 - Instructions 7-61

INTERPRET

- processor interprets wired-in
 - climb 6-44

INTERRUPT

- description of faults and
 - interrupts 6-1
- interrupt procedures 6-41
- interval timer contained in the
 - processor for program
 - interrupt 1-10
- single-word instruction
 - interrupt 6-42

INTERVAL TIMER

- interval timer contained in the
 - processor for program
 - interrupt 1-10

IPR

- illegal procedure (ipr)
 - faults 11-55
- illegal procedure fault 6-22

L**LENGTH**

- translation table length 11-181

LENGTHS

- load pointers and lengths 11-110

LINKAGE SEGMENT

- segment and descriptors for
 - defining a domain (explanation, figure) 3-15

LLUF

- lluf 11-14, 11-107, 11-119, 11-120, 13-58

LOAD

- load pointers and lengths 11-110

LOAD LOCKUP FAULT

load lockup fault register 11-14,
11-107, 11-119, 11-120, 13-58

LOCKUP

lockup fault 6-12

LOGIC

logical operations 7-3, 7-14
processor logic circuitry 12-31

LOGICAL SHIFT

long right logical shift 11-114

LONG

long right logical shift 11-114

LOWER-BOUND

lower-bound check 5-82

LPL

lpl 11-110

LREG

lreg 5-81

LRL

lrl 11-114

LUF

lockup fault 6-12

M

MAGNITUDE

sign and magnitude
operands 7-30

MANAGEMENT

multiprocessor memory
management 5-84

MAP

Micro Operation Code
Assignment Map 7-60

MASTER

master mode faults 6-40

privileged master mode
faults 6-39

MASTER MODE

master mode faults 6-40
master mode functions allowed
(list, explanations) 1-6

MEEP

meep fault 6-35

MEMORY

multiprocessor memory
management 5-84
virtual memory addressing 5-59
Virtual Memory
Instructions 7-61

MEMORY SYSTEM

fault 6-13

MEMSYS

fault 6-13
memory system fault 6-13

MICRO

edited move micro
operations 7-8
Micro Operation Code
Assignment Map 7-60
micro operation sequence 7-37
micro operations for edit
instructions mve and
mvne 7-37
Terminating Micro
Operations 7-60

MME

master mode entry fault 6-20

MNEMONIC

mnemonic code 8-3

MODE

privileged master mode
faults 6-39

MODIFICATION

ES address modification with
AR 5-51
illegal modification 8-9

MODIFICATION (cont.)

- multiword modification
 - field 7-24
- operand descriptor modification (ES) 5-57
- single-word address modification 5-28
- tag field modification ES 5-53

MOP

- micro operation sequence 7-37

MOVE

- character move to/from register instructions 8-15
- data movement instructions 7-3
- edited move micro operations 7-8

MOVEMENT

- data movement capability 7-9

MPF

- missing page fault 6-28

MSCT

- missing section fault 6-27

MSG

- missing segment fault 6-18

MULTIPLICATION

- multiplication arithmetic instructions 7-4

MULTIPROCESSOR

- multiprocessor memory management 5-84

MULTIWORD

- multiword instruction capabilities 7-9
- multiword instructions 8-11
- multiword modification field 7-24

MVE

- micro operations for edit instructions mve and mvne 7-37

- mvne, mvnex and mve differences 7-39

MVNE

- micro operations for edit instructions mve and mvne 7-37
- mvne, mvnex and mve differences 7-39

MVNEX

- mvne, mvnex and mve differences 7-39

MWSF

- missing working space fault 6-26

N**NOT COMPLETED**

- operation not completed fault 6-11

NS

- virtual address generation (NS) 5-61

NUMERIC

- alphanumeric/numeric address preparation 5-43
- numeric instructions 7-7, 7-30
- numeric operand descriptor format 7-30

O**OCTAL**

- octal value of the operation code 8-3

OFFSET

- virtual address is a combination of working space number and offset 3-5

OFL

- overflow fault 6-16

ONC

operation not completed
fault 6-11

OPERAND

operand address procedure 5-60
operand descriptor address
preparation 5-40
operand descriptor indirect pointer
format 7-25
operand descriptor modification
(ES) 5-57
operand descriptors 5-35
operand descriptors and indirect
pointers 7-25

OPERATION

operation not completed
fault 6-11

OPERATION CODE

operation code maps for the DPS
9000 (tables separated by either
bit 27 = 0 or bit 27 = 1) B-1

OPERATIONS

boolean operations 7-3
comparison operations 7-3
logical operations 7-3, 7-14

OPTION

copy option 11-73

OUTPUT

9-bit output 7-30

OVERPUNCHED

character codes for ascii and
ebcdic overpunched
sign 11-178

P

PAGE

distinction of page as a physical
element of memory in the
hardware 3-3

explanation of domains in the
virtual environment
(figure) 3-15

Page Directory Base Registers
(PDBR) (format, description,
function) 4-32

page table word (ptw) format
(vs/xa) 5-74

PAGE ACCESS

page access bit 6-41

PAGE DIRECTORY BASE REGISTER

Page Directory Base Registers
(PDBR) (format, description,
function) 4-32

PAGE MODIFY

page modify bit 6-41

PAGE TABLE

paging (vs/xa) 5-69

PARITY

parity indicator 8-9

PATTERN

replicate a pattern across a
string 11-125

PBW

paging (vs/xa) 5-69

PDBR

pdbr 11-108

PERMISSION

execute permission flag 6-38
save permission flag 6-38

POINTER

operand descriptor indirect pointer
format 7-25
operand descriptors and indirect
pointers 7-25

POINTERS

load pointers and lengths 11-110

PRESENT

segment present flag 6-38

PRIORITY

highest priority fault 6-4

PRIVILEGED

privileged flag 6-38
 Privileged Instructions 7-61
 privileged master mode
 faults 6-39

PRIVILEGED MASTER MODE

privileged master mode functions
 allowed (list,
 explanations) 1-6

PROCEDURES

interrupt procedures 6-41

PROCESS

definition of process relative to
 generating memory addresses in
 virtual memory 3-3

PROCESSOR

processor interprets wired-in
 climb 6-44
 processor logic circuitry 12-31

R**REAL ADDRESS**

transformed from virtual
 address 3-4

REG

REG 7-24

REGISTER

address register alter
 contents 7-11
 address register
 instructions 7-10
 address register special arithmetic
 instructions 8-13
 character move to/from register
 instructions 8-15
 Descriptor Register
 Instructions 7-61
 effective address to register
 instructions 7-4

GXn register in R

modification 5-50

index register symbols 5-34

Indirect Then Register (IR)

address modification (coding
 examples) 5-11

Page Directory Base Registers
 (PDBR) (format, description,
 function) 4-32

procedure of safe storing register
 contents 3-15

register (R) address modification
 variation types (list of coding
 examples) 5-6

register selection 7-24

special address register
 instructions 7-13

stack control register (scr) 11-91

REGISTER-TO-REGISTER

register to register
 instructions 8-16
 Register to register
 Instructions 7-64

REPEAT

Repeat Instructions 7-69

REPLICATE

replicate a pattern across a
 string 11-125

RMS

backup fault vector 6-3

ROUND

true round 9-110, 10-46, 10-53

RUNOUT

Transfer on Tally Runout
 Indicator OFF 14-58
 transfer on tally runout indicator
 on 14-61

S**SAFE STORE STACK**

explanation of domains in the
 virtual environment 3-15

SAVE

save permission flag 6-38

SCALING

scaling factor 7-31

SCL1

class 1 security fault 6-24

SCL2

class 2 fault security fault 6-29

SCR

stack control register
(SCR) 11-91

SDF

shutdown fault 6-34

SECURITY

security through virtual memory
management 3-3

SEGMENT

description of segment size 3-5
general description of structure of
segments 3-5
passing segment descriptors to a
process 3-3
segment descriptor 5-59
segment present flag 6-38

SHIFT

data shifting instructions 7-3
long right logical shift 11-114

SIGN

sign and magnitude
operands 7-30
sign flag 7-41

SINGLE-WORD

single-word address
modification 5-28
single-word instruction
interrupt 6-42
single-word instructions 7-2, 8-9

SLAVE

slave mode faults 6-40

SLAVE MODE

slave mode faults 6-40

SPACE

framed stack space 9-22

SPECIAL-ADDRESS

special address register
instructions 7-13

SREG

sreg 5-81

SSSF

safe store stack fault 6-30

STACK

framed stack space 9-22
stack control register (scr) 11-91

STANDARD

standard descriptor 5-61, 9-20,
11-91

STRING

bit string instructions 7-7, 7-33
bit string operand descriptor
format 7-34
bit strings and index table of
translate instruction 5-83
replicate a pattern across a
string 11-125

STUP

startup fault 6-9

SUPPRESSION

end suppression flag 7-41

SV MODE

standard virtual standard real
memory mode 1-8

SVMX MODE

standard virtual real memory
extended mode 1-8

SYMBOLS

abbreviations and symbols 8-5
index register symbols 5-34

T**TABLE**

bit strings and index table of
translate instruction 5-83
edit insertion table 7-38
page table word (ptw) format
(vs/xa) 5-74
paging (vs/xa) 5-69
translation table length 11-181

TAG

tag field modification ES 5-53

TAG DESIGNATOR

one of two parts of a tag field
(description, illustration) 5-3

TALLY

Transfer on Tally Runout
Indicator OFF 14-58
transfer on tally runout indicator
on 14-61

TIMER

interval timer contained in the
processor for program
interrupt 1-10

TRANSFER

Domain Transfer (CLIMB) 7-61
Transfer Instructions 7-67
Transfer on Tally Runout
Indicator OFF 14-58
transfer on tally runout indicator
on 14-61

TRANSLATE

bit strings and index table of
translate instruction 5-83

TRANSLATION

translation table length 11-181

TRO

timer runout fault 6-33

TRUE ROUND

true round 9-110, 10-46, 10-53

TRUNCATION

address truncation 5-81

TSYNC

Transmit Sync Interrupt 14-57

TTF

TTF 14-58

TTN

ttn 14-61

U**UPPER-BOUND**

upper-bound check 5-82

V**VALUE**

base value 5-59
bound value 5-59
octal value of the operation
code 8-3

VECTOR

backup fault 6-3
distinction between vector
arguments and
descriptors 3-15

VIRTUAL

virtual address 5-68
virtual address generation
(NS) 5-61
virtual memory addressing 5-59
Virtual Memory
Instructions 7-61

VS/XA

page table word (ptw) format
(vs/xa) 5-74
paging (vs/xa) 5-69

W

WORD

- indirect word 5-39
- instruction word formats 8-9
- page table word (ptw) format
(vs/xa) 5-74
- word address 5-34
- word and double-word
operations 5-81

WORKING SPACE

- explanation of domains in the
virtual environment
(figure) 3-15
- virtual address is a combination of
working space number and
offset 3-5

WRAPAROUND

- address wraparound 7-73

Z

ZERO

- zero flag 7-41

