HONEYWELL

DPS 6 & LEVEL 6 GCOS 6 MOD 400 SYSTEM PROGRAMMER'S GUIDE — VOLUME II

SOFTWARE

DPS 6 & LEVEL 6
GCOS 6 MOD 400
SYSTEM PROGRAMMER'S GUIDE
VOLUME II

SUBJECT

Description of System Service Macro Calls

SPECIAL INSTRUCTIONS

This manual supersedes the System Service Macro Calls manual, order number CB08-02, dated December 1978.

SOFTWARE SUPPORTED

Refer to the MOD 400 Guide to Software Documentation for information regarding Executive releases supported by this manual.

ORDER NUMBER

CZ06-00 December 1982

Honeywell

Preface

The purpose of this manual is to enable Assembly language programmers to use system service macro calls in their applications. The manual presupposes knowledge of GCOS 6 Assembly language, which is described in the *GCOS 6 Assembly Language Reference*.

The manual consists of macro call descriptions, each of which includes the following information:

- Arguments (if any) to be supplied with the call
- Registers into which the system places supplied arguments
- Register contents, including error codes, returned by the call
- The function of the call
- Special procedures (if any) for using the call

Many calls include examples of their usage.

The macro call descriptions are arranged alphabetically by name. A more general discussion of system service macro calls, which groups the calls according to their function, is found in Volume I of the *System Programmer's Guide*.

In addition to describing individual macro calls, this volume provides detailed information about the following subjects:

- Macro call format and conventions
- Data structures referred to by macro call arguments.

File No.: 1R23, 1S23 CZ06-00

The following conventions are used to indicate the relative levels of topic headings used in this manual:

Level	Format
1 (highest)	ALL CAPITAL LETTERS, UNDERLINED
2	Initial Capital Letters, Underlined
3	ALL CAPITAL LETTERS, NOT UNDERLINED
4	Initial Capital Letters, Not Underlined

MANUAL DIRECTORY

The following publications constitute the GCOS 6 MOD 400 manual set. Refer to the "Software/Manual Directory" of the Guide to Software Documentation for the current revision number and addenda (if any) of relevant release-specific publications.

Manuals are obtained by submitting a Honeywell Publications Order Form to the following address:

Manuals can be ordered from :

Honeywell Information Systems Ltd.
10 Cullen Way
London NW10 6JZ.

Honeywell software reference manuals are periodically updated to support enhancements and improvements to the software. Before ordering any manuals, you should refer to the Guide to Software Documentation to obtain information concerning the specific edition of the manual that supports the software currently in use at your installation. If you use the four-character base publication number to order a document, you will receive the latest edition of the manual. The Publications Distribution Center can provide specific editions of a publication only when supplied with the seven- or eight-character order number listed in the Guide to Software Documentation.

Honeywell applications software packages, such as INFO 6, TOTAL 6, and TPS 6, provide specialized services. Contact your Honeywell representative for information concerning the availability of applications software and supporting documentation.

Base Publication Number	Manual Title
CZ01	GCOS 6 MOD 400 Guide to Software Documentation
CZ02	GCOS 6 MOD 400 System Building and Administration
CZ03	GCOS 6 MOD 400 System Concepts
CZ04	GCOS 6 MOD 400 System User's Guide
CZ05	GCOS 6 MOD 400 System Programmer's Guide - Volume I
CZ06	GCOS 6 MOD 400 System Programmer's Guide - Volume II
CZ07	GCOS 6 MOD 400 Programmer's Pocket Guide
CZ09	GCOS 6 MOD 400 System Maintenance Facility Administrator's Guide
CZ10	GCOS 6 MOD 400 Menu Management/Maintenance Guide
CZ15	GCOS 6 MOD 400 Application Developer's Guide
CZ16	GCOS 6 MOD 400 System Messages
CZ17	GCOS 6 MOD 400 Commands
CZ18	GCOS 6 Sort/Merge
CZ19	GCOS 6 Data File Organizations and Formats
CZ20	GCOS 6 MOD 400 Transaction Control Language Facility
CZ21	GCOS 6 MOD 400 Display Formatting and Control
CZ34	GCOS 6 Advanced COBOL Reference
CZ35	GCOS 6 Advanced COBOL Quick Reference Guide
CZ36	GCOS 6 BASIC Reference
CZ37	GCOS 6 BASIC Quick Reference Guide
CZ38	GCOS 6 Assembly Language (MAP) Reference
CZ39	GCOS 6 Advanced FORTRAN Reference
CZ40	GCOS 6 Pascal User's Guide
CZ41	GCOS 6 RPG-II Reference
CZ47	Data Entry Facility-II User's Guide
CZ48	Data Entry Facility-II Operator's Quick Reference Guide
CZ52	DM6 I-D-S/II Programmer's Guide
CZ53	DM6 I-D-S/II Data Base Administrator's Guide
CZ54	DM6 I-D-S/II Reference Card
CZ59	Level 6 to Level 6 File Transmission Facility User's Guide
CZ60	Level 6 to Level 66 File Transmission Facility User's Guide
CZ61	Level 6 to Level 62 File Transmission Facility User's Guide
CZ62	BSC Transport Facility User's Guide
CZ63	2780/3780 Workstation Facility User's Guide
CZ64	HASP Workstation Facility User's Guide

Base Publication Number	Manual Title
CZ65	Programmable Facility/3271 User's Guide
CZ66	Remote Batch Facility/66 User's Guide
CZ71	DM6 TP Development Reference
CZ72	DM6 TP Application User's Guide
CZ73	DM6 TP Forms Processing

In addition, the following publications provide supplementary information:

AS22	Level 6 Models 6/34, 6/36, and 6/43 Minicomputer Handbook
AT97	Level 6 Communications Handbook
CC71	Level 6 Minicomputer Systems Handbook
CD18	Level 6 MOD 400/600 Online Test and Verification Operator's Guide
FQ41	Writable Control Store User's Guide

Users should be aware that a Software Release Bulletin accompanies each software product ordered from Honeywell. You should consult the Software Release Bulletin before using the software. Contact your Honeywell representative if a copy of the Software Release Bulletin is not available.

CONTENTS

CONTENTS

CONTENTS

Page

CONTENTS

CONTENTS

Page

CONTENTS

ILLUSTRATIONS

TABLES

TABLES

Section 1
MACRO CALL SYNTAX AND CONVENTIONS

This section provides a brief definition of macro calls. It then describes at length macro call syntax and register conventions, which must be followed when using a call.

SYSTEM SERVICES

The macro calls described in this volume are system service macro calls. A system service is a routine executed by the Executive in behalf of a running user application. System services are functions frequently required by user applications, such as the reading or writing of records, requests for memory, loading of overlays, etc. System services save the programmer the labor of coding routines that perform the same function; system services also coordinate the execution of multiple applications on a single system.

An application can call a system service routine by means of a sequence of instructions (a monitor call) or a single instruction (a macro call).

MONITOR CALLS

A monitor call identifies the service being requested by means of a function code. A monitor call usually must supply information expected by the requested system service in certain registers. For example, to request the system service that releases an entire block of memory previously allocated to the user, the user codes the following sequence of instructions:

1. An instruction loading register B4 with the address of the memory block to be returned to the pool of available memory.

2. The instruction MCL, signifying "monitor call".

3. An instruction identifying the requested service by its function code, X'0404'.

Assuming that the address of the memory block to be returned is stored in a location labeled RET_MEM, the three instructions may take the following form:

```
LDB $B4,>RET_MEM
MCL
DC  X'0404'
```

MACRO CALLS

A system service macro call is an abbreviated form of monitor call. When requesting a system service by means of a macro call, the programmer codes a single instruction instead of several. This instruction consists of a macro call name and any arguments expected by the system service. For example, the following macro call requests the return of a memory block whose address is stored in location RET_MEM:

```
$RMEM  RET_MEM
```

Before the source text is compiled, this macro call is expanded by the macro preprocessor into the three instructions listed above. The programmer can request the Return Memory function by coding either the three-instruction monitor call or the macro call. The advantage of the macro call, besides its brevity, is its independence from registers. It is possible (though unlikely) that in some future version of the Executive, the Return Memory routine will expect the location of the memory to be loaded into register B5 rather than B4. In this case, the above monitor call will produce an error; the macro call will not.

MACRO CALL SYNTAX

Macro call syntax follows the conventions for Assembly language (described in detail by the Assembly Language Reference). The first field of the macro call can have an optional label. If no label is used, at least one blank must precede the macro call. User-selected items of data in a macro call are known as arguments; these arguments are passed to a system service macro routine by the macro processor.

Within the called system service macro routine (which is generalized to handle any set of data passed to it), the macro call arguments are associated with the service routine arguments -- the order of positional arguments in the macro call indicates the variables to which the data is applied. Thus, the order of your arguments must be the same as the positional arguments within the system service macro routine. Unless stated otherwise, omitted arguments that precede an included argument must be indicated by the presence of a replacing comma for each omission. One or more spaces must separate the macro call name from its arguments, with a comma between each argument. The horizontal tab character is equivalent to a space. A semicolon at the end of a line indicates that the next line is a continuation line.

REGISTER CONVENTIONS AND CONTENTS

Macro call arguments are often loaded into registers for access by the system services. An argument of a macro call can specify that the corresponding system service argument is either contained in memory or in a register. If an argument is omitted from the macro call, the system assumes that the register normally used to provide the value or address to the system service routine contains the required value or address. For this reason, it is important to know how the system service routines use the registers, as well as the conventions that exist for saving register contents.*

The system services use the following registers without preserving their contents:

R1	**R7**
R2	**B2**
R6	**B4**

Unless otherwise stated, the system services do not usually alter the contents of the following registers:

S	B1	T	S3
I	B3	RDBR	M1 through M7
R3	B5	CI	
R4	B6	SI	
R5	B7	S1	
		S2	

When coding a macro call that uses a register whose contents are not preserved, ensure that the contents of the register are appropriate for each occurrence of the macro call.

*The file system macro calls preserve the contents of all registers except R1. B4 is the only register loaded by the file system macro calls.

ADDRESSING CONVENTIONS

Any macro call argument definition that specifies an argument default of a specific register content will allow an argument specification in the form =$Rn or =$Bn (n designates the register to be specialized for the system service routine) to denote that the register has been previously set to the value to be used. When a macro call argument description specifies that the location of a value or an address may be provided, any assembly-level address syllable format that is valid for the type of register being specialized can be used; that is, the value (if less than or equal to two bytes) or address can be supplied as an immediate memory operand (IMO) address syllable form by prefacing the value or address with an equal sign (=). The !label macro notation is used only to distinguish between LDB and LAB instructions when specializing a base register.

For example, the $WAIT macro call has a single argument that specifies the location of the address of the request block to be waited on. This location must be placed in base register B4. The value specified for this argument in the $WAIT macro call can take any of the following forms (among others):

=label

The label refers to the request block to be waited on. An IMO address syllable format will be used by the LDB instruction generated to load $B4.

label

The label refers to a field that contains the address of the request block to be waited on. A P+DSP address syllable format will be used by the LDB instruction generated to load $B4.

<label

The label refers to a field that contains the address of the request block to be waited on. An IMA address syllable format will be used by the LDB instruction generated to load $B4.

=$B4

$B4 already contains the address of the request block to be waited on. No instruction will be generated to load $B4.

=$B3

$B3 contains the address of the request block to be waited on. A register addressing address syllable will be used by the LDB instruction generated to load $B4.

$B3

$B3 contains the address of a field that contains the address of the request block to be waited on. A direct base addressing address syllable will be used by the LDB instruction generated to load $B4.

*$B3

$B3 contains the address of a field that contains the address of a field that contains the address of the request block to be waited on. An indirect base addressing address syllable will be used by the LDB instruction generated to load $B4.

$B3.$R2

The address referred to by $B3 plus $R2 contains the address of the request block to be waited on. An indexed base addressing address syllable will be used by the LDB instruction generated to load $B4.

If the address syllable is preceded by an exclamation point (!), the instruction generated is a LAB rather than an LDB. For example:

!label

The label refers to the address of the request block to be waited on. An effective address syllable format will be used by the LAB instruction generated to load $B4.

!*label

The label refers to a field containing the address of the request block to be waited on. A "LAB $B4, *label" instruction will be generated to load $B4.

Thus, macro call "location address" arguments (which are to be loaded into base registers) can refer to the <u>location of the address</u> of the data or data structure or can refer to the <u>address</u> of the data or data structure. In the first case (location of address), the macro call loads the Bn register through an LDB instruction, thus requiring that the "location address" values in the macro call arguments be the label of a location where the address of the actual argument structure is located. In the second case (address), the macro call loads the effective address of the argument structure into $Bn directly (through a LAB instruction) when the first argument is a label and is preceded by an exclamation point (!) character. For example:

```
FIBPTR  DC      FIB
          .
          .
          .
FIB     RESV    16
          .
          .
          .
        $macro  FIBPTR  =  LDB $B4,FIBPTR
          .
          .
          .
        $macro  !FIB    =  LAB $B4,FIB
```

REGISTER CONTENTS AT TASK ACTIVATION

When a task is activated, the contents of $B4, $B5, $B6, and $B7 are the following:

$B4

Address of the task request block.

$B5

Address of the system-supplied termination routine (see the Return ($RETRN) macro call).

$B6

Address of the top of the root/data segment of the bound unit associated with the task. If this segment is larger than 32K words, $B6 contains the address of the 32Kth word of the segment.

$B7

Address of the parameter block containing the request block argument list.

REGISTER CONTENTS AT INITIAL TASK ACTIVATION

The M registers are set up as follows. When each task starts, the system establishes the following default values for registers M1, M3, M4, and M5:

M1 = 00

Trace trap and all R-register overflow traps disabled.

M3 = 00

CIP overflow trap and truncation trap disabled; CIP is under direct CPU firmware control (i.e., not in software test mode).

M4 = 03

Truncation mode in effect. Scientific accumulators $S1 and $S2 and associated memory operands are two words long; $S3 and associated memory operands are four words long.

M5 = 20

Significance error trap enabled; exponent overflow and precision error traps disabled.

Contents of these registers can be modified with the Assembly language instruction MTM.

RETURN STATUS CODES IN $R1

The descriptions of the macro calls in Section 2 include lists of status codes returned in $R1, together with an explanation of each code. These lists are not intended to include every possible return code; moreover, the explanations of these codes are briefer than error messages provided by the system. See the System Messages manual for a list of all $R1 return status codes, system messages, and additional definitions.

SYSTEM SERVICE MACRO CALLS AND FUNCTION CODES

Table 1-1 contains an alphabetic list, by macro call name, of the macro calls described in the next section.

The list includes the function codes associated with each macro call (data structure generation macro calls do not have function codes). The first two digits of the function code designate the major function, and are used by the macro call trap-handling routine to locate the entry point of the appropriate system service routine. The last two digits are a subfunction code used by the system service routine to provide the requested subfunction. When a macro call is executed, it generates the following:

```
        MCL
        DC      Z'mmss'
```

where mm is the 2-digit major function code and ss is the 2-digit subfunction code. The function codes are provided for information only; they will appear in program listings and dumps.

LOCATION OF MACRO ROUTINES

The macro routines are located either on cartridge disk or on mass storage unit in a library named >LDD>MACRO>EXEC_LIB. On diskette they are located in ^ZSYS02>LDD>MACRO>EXEC_LIB.

Table 1-1. System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$ABGRP	Abort group	0D/0A	Task group control
$ABGRQ	Abort group request	0D/07	Task group control
$ACTID	Account identification	14/02	Identification and information
$ACTVG	Activate group	0D/09	Task group control
$ASFIL	Associate file	10/10	File management
$BUAT	Bound unit, attach	0C/09	Task control
$BUDT	Bound unit, detach	0C/0B	Task control
$BUID	Bound unit identification	14/06	Identification and information
$BULD	Bound unit, load	0C/0A	Task control
$BUXFR	Bound unit transfer	0C/07	Task control
$CANRQ	Cancel request	0C/01	Task control
$CIN	Command in	08/02	Standard system file I/O
$CKPFL	Checkpoint file	0D/11	File management
$CKPT	Checkpoint	0D/0F	Task group control
$CLFIL	Close file	10/55-10/57	File management
$CLPNT	Clean point	0C/13	File management
$CLRSW	Clear external switches	0B/02	External switch

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$CMDLN	Command line process	0C/08	Task control
$CMSUP	Console message suppression	09/02,09/03	Operator interface
$CNCRQ	Cancel clock request	05/01	Clock
$CNSRQ	Cancel semaphore request	06/01	Semaphore handling
$CRB	Clock request block	-	Data structure generation
$CRBD	Clock request block offsets	-	Data structure generation
$CRDIR	Create directory	10/A0	File management
$CRFIL	Create file	10/30	File management
$CRGRP	Create group	0D/03	Task group control
$CRKDB	Create file key descriptor block offsets	-	Data structure generation
$CROAT	Create overlay area table	07/0A	Overlay handling
$CRPSB	Create file parameter structure block offsets	-	Data structure generation
$CRRDB	Create file record descriptor block offsets	-	Data structure generation
$CRSEG	Create segment	0C/0C	Task control
$CRTSK	Create task	0C/02,0C/03	Task control
$CWDIR	Change working directory	10/B0	File management
$DFCKP	Defer checkpoint	0C/19	Task control

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$DFRHD	Defer request on head	01/0D	Request and Return
$DFRTL	Defer request on tail	01/0C	Request and Return
$DFSM	Define semaphore	06/04	Semaphore handling
$DIPSB	Device information parameter structure block offsets	-	Data structure generation
$DLDIR	Delete directory	10/A5	File management
$DLFIL	Delete file	10/35	File management
$DLGRP	Delete group	0D/04	Task group control
$DLOAT	Delete overlay area table	07/0D	Overlay handling
$DLREC	Delete record	11/30,11/31	Data management
$DLSEG	Delete segment	0C/0D	Task control
$DLSM	Delete semaphore	06/07	Semaphore handling
$DLTSK	Delete task	0C/04	Task control
$DQPST	Dequeue and post	01/0B	Request and Return
$DSFIL	Dissociate file	10/15	File management
$DSTRP	Disable user trap	0A/02	Trap handling
$ELEND	Error logging end	02/09	Physical I/O
$ELEX	Error logging information, exchange	02/07	Physical I/O
$ELGT	Error logging information, get	02/08	Physical I/O
$ELOG	Error logging table	-	Data structure generation
$ELST	Error logging, start	02/05	Physical I/O

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$ENTID	Entry point identification	14/07	Identification and information
$ENTRP	Enable user trap	0A/01	Trap handling
$EROUT	Error out	08/03	Standard system file I/O
$EXTDT	External date/time, convert to	05/04	Date/time
$EXTIM	External time, convert to	05/05	Date/time
$FIB	File information block	-	Data structure generation
$FIBDM	File information block offsets (data management access)	-	Data structure generation
$FIBSM	File information block offsets (storage management access)	-	Data structure generation
$GAFIL	Get file access rights	10/7C	File management
$GAPSB	Get file access rights parameter structure block offsets	-	Data structure generation
$GDTM	Get date/time	05/06	Date/time
$GIDEV	Get device information	10/66	File management
$GIFAB	Get file information, file attribute block offsets	-	Data structure generation
$GIFIL	Get file information	10/60	File management

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$GIKDB	Get file information, key descriptor block offsets	-	Data structure generation
$GIPSB	Get file information, parameter structure block offsets	-	Data structure generation
$GIRDB	Get file record descriptor block offsets	-	Data structure generation
$GMEM	Get memory/get available memory	04/02,04/03	Memory allocation
$GNFIL	Get name	10/3C	File management
$GNPSB	Get names parameter structure block offsets	-	Data structure generation
$GRFIL	Grow file	10/38	File management
$GRPID	Group identification	14/08	Identification and information
$GRPSB	Grow file parameter structure block offsets	-	Data structure generation
$GTACT	Get file accounting information	10/42	File management
$GTFIL	Get file	10/20	File management
$GTPSB	Get file parameter structure block offsets	-	Data structure generation
$GWDIR	Get working directory	10/C0	File management
$HDIR	Home directory	14/0B	Identification and information
$INDTM	Internal date/time, convert to	05/07	Date/time

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$IORB	Input/output request block	-	Data structure generation
$IORBD	Input/output request block offsets	-	Data structure generation
$KILLT	Kill (abort) task	0C/11	Task control
$MACPT	Message group, accept	15/01	Intergroup message facility
$MCME	Message group, cancel enclosure	15/06	Intergroup message facility
$MCMG	Message group, count	15/07	Intergroup message facility
$MDFIL	Modify file	10/41	File management
$MDPSB	Modify file parameter structure block offsets	-	Data structure generation
$MGCRB	Message group, control request block	-	Data structure generation
$MGCRT	Message group control request block offsets	-	Data structure generation
$MGIRB	Message group, initialization request block	-	Data structure generation
$MGIRT	Message group initialization request block offsets	-	Data structure generation
$MGRRB	Message group, recovery request block	-	Data structure generation
$MGRRT	Message group recovery request block offsets	-	Data structure generation

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$MINIT	Message group, initiate	15/02	Intergroup message facility
$MODID	Mode identification	14/03	Identification and information
$MRECV	Message group, receive	15/03	Intergroup message facility
$MSEND	Message group, send	15/05	Intergroup message facility
$MTMG	Message group, terminate	15/04	Intergroup message facility
$NCIN	New command in	08/06	Standard system file I/O
$NMLF	New message library	08/08	Standard system file I/O
$NPROC	New process	0D/0B	Task group control
$NUIN	New user input	08/04	Standard system file I/O
$NUOUT	New user output	08/05	Standard system file I/O
$OPFIL	Open file	10/50,10/51	File management
$OPMSG	Operator information message	09/00	Operator interface
$OPRSP	Operator response message	09/01	Operator interface
$OVEXC	Overlay, execute	07/00	Overlay handling
$OVLD	Overlay, load	07/01	Overlay handling
$OVRCL	Overlay release, wait, and recall	07/07	Overlay handling
$OVRLS	Overlay area, release	07/06	Overlay handling

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$OVRSV	Overlay area reserve, and execute overlay	07/05	Overlay handling
$OVST	Overlay status	07/03	Overlay handling
$OVUN	Overlay, unload	07/0C	Overlay handling
$PERID	Person identification	14/01	Identification and information
$PPNTL	Postpone request on tail	01/0E	Request and Return
$PRBLK	Parameter block	-	Data structure generation
$PRFAU	Profile record, accounting update	24/42	User registration
$PRFCR	Profile record, create	24/20	User registration
$PRFDL	Profile record, delete	24/30	User registration
$PRFGT	Profile record, get	24/10	User registration
$PRFIF	Profile record, get user information	24/12	User registration
$PRFUP	Profile record, update	24/40	User registration
$RBADD	Return request block address	01/07	Request and return
$RBD	Request block displacements	-	Data structure generation
$RBOOT	Reboot	20/06	Software reboot
$RBPRM	Modify reboot parameters	20/05	Software reboot
$RCLHD	Recall from head	01/0F	Request and return
$RDBLK	Read block	12/00-12/04	Storage management

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$RDREC	Read record	11/10-11/16, 11/19	Data management
$RDSW	Read external switches	0B/00	External switch
$RETRN	Return	-	Request and return
$RLDMP	Unlock dumpfile	20/04	Software reboot
$RLSM	Release semaphore	06/03	Semaphore handling
$RLTML	Release terminal	17/04	Terminal function
$RMEM	Return memory/return partial block of memory	04/04,04/05	Memory allocation
$RMFIL	Remove file	10/25	File management
$RNFIL	Rename file/rename directory	10/40	File management
$ROLBK	Roll back (recover) files	0C/14	File management
$RPDFC	Report message, display formatting and control	0F/04	Message reporter
$RPMSG	Report message	0F/03	Message reporter
$RQBAT	Request batch	0E/00	Batch
$RQCL	Request clock	05/00	Clock
$RQGRP	Request group	0D/00	Task group control
$RQIO	Request I/O	02/00	Physical I/O
$RQSM	Request semaphore	06/00	Semaphore handling
$RQSPT	Request specific terminal	17/02	Terminal function
$RQTML	Request terminal	17/03	Terminal function

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$RQTSK	Request task	0C/00	Task control
$RS	Restart	0D/10	Task control
$RSVSM	Reserve semaphore	06/02	Semaphore handling
$RWREC	Rewrite record	11/40,11/41	Data management
$RVFPW	Reverify password	24/01	User registration
$SDL	Set dial	1B/00	Communications
$SETSW	Set external switches	0B/01	External switch
$SGRPA	Set group attributes	0D/13	Task group control
$SGTRP	Signal trap	0A/03	Trap handling
$SHFIL	Shrink file	10/37	File management
$SHPSB	Shrink file parameter structure block offsets	-	Data structure generation
$SPGRP	Spawn group	0D/05	Task group control
$SPTSK	Spawn task	0C/05,0C/06, 0C/15	Task control
$SRB	Semaphore request block	-	Data structure generation
$SRBD	Semaphore request block offsets	-	Data structure generation
$STMP	Status memory pool	04/06	Memory allocation
$STTY	Set terminal file characteristics	10/45	File management
$SUSPG	Suspend group	0D/08	Task group control

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$SUSPN	Suspend for interval; suspend until time	05/02,05/03	Clock
$SWFIL	Swap file	10/5A	File management
$SYSAT	System attribute information, get	14/11	Identification and information
$SYSID	System identification	14/04	Identification and information
$TEST	Test completion status	01/02	Request and return
$TFIB	File information block offsets (data and storage management access)	-	Data structure generation
$TGIN	Task group input	14/0C	Identification and information
$TIFIL	Test file for input	10/62	File management
$TOFIL	Test file for output	10/63	File management
$TRB	Task request block	-	Data structure generation
$TRBD	Task request block offsets	-	Data structure generation
$TRMRQ	Terminate request	01/03,01/04	Request and return
$TRPHD	Trap handler connect	0A/00	Trap handling
$USIN	User input	08/00	Standard system file I/O
$USOUT	User output	08/01	Standard system file I/O
$USRID	User identification	14/00	Identification and information
$VLCKP	Validate checkpoint	0D/12	Task group control

Table 1-1 (cont). System Service Macro Calls

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$WAIT	Wait	01/00	Request and return
$WAITA	Wait any	01/01	Request and return
$WAITL	Wait on request list	01/01	Request and return
$WAITM	Wait on multiple requests	01/01	Request and return
$WIFIL	Wait file (input)	10/64	File management
$WLIST	Wait list generate	-	Data structure generation
$WLSTM	Wait list, generate multiple	-	Data structure generation
$WOFIL	Wait file (output)	10/65	File management
$WRBLK	Write block	12/10,12/11	Storage management
$WRREC	Write record	11/20-11/26	Data management
$WTBLK	Wait block	12/20	Storage management
$XFERU	Transfer user	17/06	Terminal function
$XPATH	Expand pathname	10/D0	File management
$XRETU	Transfer and re- turn user	17/07	Terminal function

Section 2
MACRO CALL DESCRIPTIONS

This section describes in detail the system service macro calls listed in the previous section (Table 1-1). The descriptions are ordered alphabetically by function name (see Column 2, Table 1-1).

Each description explains the purpose of the system service routine invoked by the macro call. The description also defines any arguments that the user supplies with the macro call. Explanation of the routine's logic is limited to points that are pertinent to proper use of the call. A section of notes in the description provides the following information:

- Registers used by the macro call
- Possible error codes returned in register R1 and their significance.

The list of error codes included in macro call descriptions is only partial; moreover, the explanations of the codes are briefer than the error messages provided by the system. For a complete listing of error codes and accompanying system messages, see the System Messages manual.

The following notational conventions are used in macro call formats:

Convention	Meaning
UPPERCASE CHARACTERS	Required word; i.e., must be used in the form specified.
lowercase characters	Symbolic name; i.e., must be replaced by user-specified word(s)
[ ]	Brackets. The item enclosed in the brackets is optional.
{ }	Braces. An enclosed entry must be selected.

NOTE

Brackets and braces can be combined as shown in the following example:

$$\left[\begin{Bmatrix} \text{,PRESERVE} \\ \text{,RENEW} \end{Bmatrix}\right]$$

The argument is optional; if specified, it must take the form PRESERVE or RENEW.

...	Ellipses. The immediately preceding item may be repeated one or more times.
Δ	Required space. This character is used to indicate a required space at the end of a string; embedded spaces are usally represented visually (i.e., by the absence of any character).

ABORT GROUP ($ABGRP)

Function Code: 0D/0A

Equivalent Command: Abort Group (ABORT_GROUP)

Terminate the indicated task group and delete it.

FORMAT:

```
[label]  $ABGRP  [location of abort status],
                 [location of group id]
```

ARGUMENTS:

location of abort status

Any address form valid for a data register; provides a completion status code that is posted when the task group is terminated. The abort status code is used as the termination code of the lead task of the aborted group.

location of group id

Any address form valid for a data register; provides the group identification of the task group to be aborted. If this argument is omitted, the task group issuing the macro call is aborted. If a group id is specified, it must be the same as that used in the Create Group macro call that initialized that task group.

DESCRIPTION:

This function terminates an existing task group, whether the group is active or dormant. The Abort Group macro call removes all data structures that define and control execution of the task group, and returns all memory used by the group to the appropriate memory pool. Any files that were open during execution of the task group are closed. Any requests pending against the group are canceled. The group is deleted.

NOTES

1. The system places the abort status codes supplied by argument 1 in \$R6. If this argument is omitted, the system assumes that \$R6 contains the abort status code to be used.

2. The system places the group identification supplied by argument 2 in \$R2. If this argument is omitted, \$R2 is set to zero to designate that the issuing task group is to be aborted.

3. If a task group other than the issuing task group is aborted, \$R1 and \$R2 contain the following information upon return to the issuing task.

   \$R1 - Return status; one of the following:

   0000 - Abort task group status set
   0806 - Invalid group id

   \$R2 - Group id of aborted task group.

Example:

In this example, the Abort Group macro call causes the processing of the current group request to be aborted with a completion status of 40 (decimal). The task group is then deleted, and any requests that may be queued on the group are discarded.

```
    $ABGRP  = 40
```

ABORT GROUP REQUEST ($ABGRQ)

Function Code: 0D/07

Equivalent Command: Abort Group Request (AGR)

Terminate execution of the current request in the indicated task group.

FORMAT:

```
[label]  $ABGRQ  [location of abort status],
                 [location of group id]
```

ARGUMENTS:

location of abort status

Any address form valid for a data register; provides a completion status code that is posted when the request is marked as terminated. The abort status code is used as the termination code of the lead task of the aborted group.

location of group id

Any address form valid for a data register; provides the group identification of the task group whose current request is to be terminated. If this argument is omitted, the current request of the issuing task group is terminated. If a group identifier is specified, it must be the same as that used in the Create Group or Spawn Group macro call that initialized this task group.

DESCRIPTION:

This macro call terminates execution of the current request in the indicated task group. It removes all defining and controlling data structures except those associated with the lead task (as defined by the Create Group macro call that specified this group id), and returns the associated memory to the appropriate memory pool.

Open files for this task group are closed. The abort process is not achieved until all outstanding input/output orders are completed.

When the macro call has been executed, the abort status code is posted, the request is removed, and the next request for this group, if any, is processed by the lead task.

An Abort Group Request for a spawned group is equivalent to an Abort Group monitor call.

NOTES

1. The system places the abort status code specified by argument 1 in $R6. If this argument is omitted, the system assumes that $R6 contains the abort status code to be used.

2. The system places the group identification specified by argument 2 in $R2. If this argument is omitted, $R2 is set to zero to designate that the issuing task group request is to be aborted.

3. If the current request of a task group other than the issuing task group was aborted, $R1 and $R2 contain the following information upon return to the issuing task.

   $R1 - Return status; one of the following:

   0000 - Abort task group request status set
   0806 - Invalid id

   $R2 - Group id of task group whose current request was aborted.

Example:

In this example, the Abort Group Request macro call causes the processing of the current group request to be aborted with a completion status of 20 (hexadecimal). If additional requests are queued on the task group, the next (first) request in the queue is processed:

```
END2     $ABGRQ     =X'20'
```

ACCOUNT IDENTIFICATION ($ACTID)

Function Code: 14/02

Equivalent Command: None

Return the account component of the calling task group's user identification to a 12-character receiving field.

FORMAT:

[label] $ACTID [location of account id field address]

ARGUMENT:

location of account id field address

Any address form valid for an address register; provides the address of a 12-character, aligned, nonvarying field into which the system will place the account component of the user identification associated with the issuing task group.

DESCRIPTION:

This macro call returns the account component (i.e., the account under which the user is working) of the task group's user id to a field in the issuing task. See the System User's Guide for more details.

The entire user id is returned by the User Identification macro call.

NOTES

1. The system places in $B4 the address of the receiving account id field, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the receiving field.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0817 - Memory access violation.

Example:

In the following example, $B4 is loaded with the address (ACIDFL) of a 12-character field, and the $ACTID macro call is issued to place the account identification of the task group in that field.

```
ACIDFL    RESV    12,0
          LAB     $B4,ACIDFL
          $ACTID
```

ACTIVATE GROUP ($ACTVG)

Function Code: 0D/09

Equivalent Command: Activate Group (ACTG)

Reactivate a previously suspended task group.

FORMAT:

[label] $ACTVG [location of group id]

ARGUMENT:

location of group id

Any address form valid for a data register; provides the group id of the task group to be reactivated.

DESCRIPTION:

This macro call causes the system to reactivate the specified suspended task group. The task group must have been previously suspended through a Suspend Group macro call. The system requeues on the appropriate level queue all tasks that were active when the task group was suspended.

Before terminating, any online task group that has suspended another online task group (through a Suspend Group macro call) should reactivate that task group. If the suspending task group does not issue an Activate Group macro call, or if the suspended task group is aborted, the operator must issue an Activate Group command for the suspended task group to resume.

NOTES

1. The system places in $R2 the group id of the task group to be reactivated, supplied by the argument. If the argument is omitted, the system assumes that $R2 contains the correct group id.

2. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0806 - Invalid group id

   080D - Specified task group not currently suspended

   $R2 - Group id as supplied.

Example:

In this example, the Activate Group macro call is used to reactivate the previously suspended task group whose group id is G1. All tasks in task group G1 that were active when the group was suspended are requeued on the appropriate level queue.

```
ACTGAA    $ACTVG    =G1
```

ASSOCIATE FILE ($ASFIL)

Function Code: 10/10

Equivalent Command: Associate (ASSOC)

Associate a logical file number (LFN) with a specific pathname. This association is typically done outside program execution to allow the program to be run against a pathname that is not known until execution time. The Get File macro call or Get File command may be more useful.

FORMAT:

```
[label]   $ASFIL   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown.

logical file number

A 2-byte LFN used to refer to the file; must be a binary number in the range 0 through 255.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) to be associated with the LFN.

DESCRIPTION:

This macro call establishes a logical connection between an LFN and a pathname. It does not reserve a file or check to determine whether the pathname identifies an existing file or directory (i.e., the pathname entry may identify an incomplete pathname). Subsequent macro calls (e.g., Change Working Directory) have no effect on a previously associated pathname because the pathname identified in this macro call is fully expanded at the time of the call. Note that the association established is specific to a task group; that is, different task groups can associate different pathnames to the same LFN.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0210 - LFN already associated

   0222 - Pathname cannot be expanded, no working directory

   0226 - Not enough user memory for buffers or structures.

Example:

This example assumes that $B4 has been loaded with the address of the label FILE_A (i.e., LAB $B4,FILE_A). The macro call that associates the path identified in the Create File example (i.e., ^VOL03>SUBINDEX.A>FILE_A) with LFN 5 is coded as follows:

```
$ASFIL
```

FILE_A is defined in "Assumptions for File System Examples" in Appendix A; as a result of issuing the Associate File macro call, the first two entries in that structure are referred to by the system.

BOUND UNIT, ATTACH ($BUAT)

Function Code: OC/09

Equivalent Command: None

Load the root of the specified bound unit and start its execution.

FORMAT:

```
[label]  $BUAT  [location of root entry name address],
                [location of segmented code address],
                [location of code segment access rights],
                [location of data segment access rights]
```

ARGUMENT:

location of root entry name address

Any address form valid for an address register; provides the location of the address of the pathname of the bound unit to be executed. The bound unit pathname can have an optional suffix, in the form ?entry, where entry is the symolic start address within the root. If the suffix is not supplied, the execution of the bound unit begins at the default start address established at link time.

location of segmented code address

Any address form valid for an address register; provides the address of any word in the code segment of the bound unit to be attached. A null address specifies the segment number (if any) specified at link time. If no segment number was specified at link time, a null address directs the system to assign a segment number.

The system bypasses this argument if:

- The bound unit to be attached is globally sharable (i.e., linked with the GSHARE directive)
- The task that issues $BUAT is not running in a swap pool.

location of code segment access rights

Any address form valid for a data register; provides the access rights (read, write, and execute) for the code segment of the bound unit to be attached. Access rights are expressed as a string of six bits, which are used to specifiy the type of access as follows:

Bits	Access Type
0-1	Read
2-3	Write
4-5	Execute

Ring access is coded as follows:

Bit Values	Ring
00	3
01	2
10	1
11	0

The system bypasses this argument if:

- The bound unit to be attached is globally sharable (i.e., linked withthe GSHARE directive)
- The task that issues $BUAT is not running in a swap pool.

In either of the two cases stated above, omitting this argument does not cause the system to assign default access rights. Otherwise, omitting this argument causes the system to assign the following default access rights to the code segment:

Read = 3
Write = 0
Execute = 3

location of data segment access rights

Any address form valid for a data register; provides the access rights for the data segment of a bound unit to be attached. Access rights are specified in the same manner as for argument 3.

The system bypasses this argument if:

- The bound unit to be attached was not linked with the -R argument of the Linker command

- The task that issues $BUAT is not running in a swap pool.

In either of the two cases stated above, omitting this argument does not cause the system to assign default access rights; otherwise, omitting this argument causes the system to assign the following default access rights to the data segment:

Read = 3
Write = 3
Execute = 3

DESCRIPTION:

Arguments 2, 3, and 4 (which pertain to segments) are applicable only when the the call is issued from a swap pool. (Further limitations to the applicability of these arguments are noted in their descriptions).

When the call is issued from a swap pool, the address space of the issuing task is compared to the total address space of the bound unit to be attached. If the two address spaces overlap, a warning message is returned to the error out file.

The segment number specified by argument 2 overrides a segment number specified at link time. If the bound unit to be attached has been linked with the -R argument of the Linker command, the system assigns to that bound unit's data segment a segment number lower by 1 than the segment number specified for the code segment.

If the code or data segment of the bound unit to be attached has been assigned a ring number at link time, arguments 3 and 4 can only lower that number; they cannot assign a ring number higher than one previously assigned. That is, arguments 3 and 4 can only increase the protection already given to a bound unit; they cannot lower that protection.

Up to seven bound units can be attached to a task at a given time.

When a bound unit is attached, its bound unit index indentifier (a value from 1 to 7) is returned in $R6. The bound unit index id can be used later to execute a Bound Unit, Detach macro call; it must be supplied with all macro calls that handle the attached bound unit's overlays.

When a globally sharable or sharable bound unit (linked with the GSHARE or SHARE directives, respectively) is attached for the second time by a given task, the bound unit index id first returned is returned again; a new index id is not established. When, however, a nonsharable bound unit is attached for the second time, the bound unit is loaded a second time and a new bound unit index id is established.

An attached bound unit is loaded according to the Loader's general rules for allocation. When the bound unit to be attached is globally sharable, no additional copy of the bound unit is loaded; instead, the "number of BU users" is incremented. When the bound unit to be attached is sharable (i.e., linked with the SHARE directive) and has already been loaded into a different segment than that specified by argument 2, a private copy of the bound unit is loaded for the issuing task. When a nonsharable bound unit is attached, its external symbols are resolved with respect to the calling bound unit.

NOTES

1. The address of the root entry name supplied by the first argument is placed in $B2. If this argument is omitted, the system assumes that $B2 contains the address.

2. The address of any word in the code segment, supplied by argument 2, is placed in $B4. If this argument is applicable but omitted, the system assumes that $B4 contains the address.

3. The access rights value for the code segment, supplied by argument 3, is placed in $R7. When this argument is applicable but omitted, the access rights default to those given above in the description of argument 3.

4. The access rights value for the data segment, supplied by argument 4, is placed in $R6. When this argument is applicable but omitted, the access rights default to those given above in the description of argument 4.

5. On return, $R3, $R4, $R5, $B1, $B3, $B5, and $B7 are preserved. $R1, $B4, $B6, $R6 contain the following information:

$R1 - Return status; one of the following:

- 0000 - No error
- 0602- No memory available for attached bound units array
- 082C - Number of attachable bound units exceeded
- 1605 - Relocation error
- 1607 - Media error
- 1608 - Symbol resolution error
- 1609 - File not found
- 160A - No memory available for bound unit
- 160D - Bound unit entry point not defined
- 160F - Bound unit cannot run in System Group Task
- 1614 - Access violation
- 1615 - Invalid bound unit format
- 1619 - Concurrency violation

$B4 - Address of entry point

$B6 - Address of data section (if any)

$R6 - Index id of attached bound unit.

BOUND UNIT, DETACH ($BUDT)

Function Code: OC/OB

Equivalent Command: None

Unload a bound unit that has been attached or loaded by the issuing task.

FORMAT:

```
[label]  $BUDT  [location of pathname address],
                [location of bound unit index id]
```

ARGUMENTS:

location of pathname address

Any address form valid for an address register; provides the location of the pathname of the bound unit to be detached from the issuing task. A null address signifies that the bound unit to be detached is specified by its bound unit index id, supplied by argument 2. A null address must be used when detaching a nonsharable bound unit.

location of bound unit index id

Any address form valid for a data register; provides the bound unit index id (a value from 1 to 7) of the bound unit be to detached. The index id is returned in $R6 by the Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call that initially loaded the bound unit to be detached.

If argument 1 supplies a pathname, argument 2 may be omitted. If argument 1 supplies a null address, argument 2 must specifiy a bound unit index id.

DESCRIPTION:

The task issuing this call must have previously issued a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call that loaded the bound unit to be detached by this call.

To detach a nonsharable bound unit, the user must specify in argument 1 a null address and in argument 2 the bound unit's index id.

The bound unit specified by argument 1 or 2 is unloaded according to the loader's allocation rules: If the specified bound unit is sharable, it is physically unloaded when the count of its users is decremented to zero.

NOTES

1. The address of the pathname supplied by argument 1 is placed in $B2. If the argument is omitted, the system assumes that $B2 contains the address of the pathname. If $B2 is null, the bound unit index supplied by argument 2 is used to identify the bound unit.

2. If argument 1 supplies a non-null address of a pathname, argument 2 is bypassed. Otherwise, the bound unit index id supplied by argument 2 is placed $R6. If argument 2 is omitted, the system assumes that $R6 contains the index id to be used.

3. On return, $R1, $R2, and $R6 contain the following information:

   $R1 - Return status; one of the following:

   - 0000 - No error
   - 082A - No matching bound unit found to be detached
   - 0826 - May not detach the primary bound unit of a task
   - 0602 - No memory available to expand pathname
   - 0201 - Illegal pathname
   - 0222 - Pathname cannot be expanded; no working directory

   $R2 - 0

   $R6 - Bound unit index id.

   Other registers are preserved.

Example:

The issuing task requests that the sharable bound unit PROG1, previously attached by means of a Bound Unit, Attach macro call, be detached. The address occupied by PROG1 is made inactive relative to the issuing task; that is, it is removed from the address space defined for the issuing task.

```
DTBU1    $BUDT    !ROOT
           .
           .
           .
ROOT     TEXT     'PROG1Δ'
```

BOUND UNIT IDENTIFICATION ($BUID)

Function Code: 14/06

Equivalent Command: Name

Return the symbolic entry point name of the bound unit being executed by the issuing task to a 12-character receiving field.

FORMAT:

```
[label]  $BUID  [location of bound unit id field address]
```

ARGUMENT:

location of bound unit id field address

Any address form valid for an address register; provides the addess of a 12-character aligned, nonvarying receiving field into which the system will place the name of the current bound unit.

DESCRIPTION:

This macro call returns the symbolic entry point name of the currently executing bound unit to a specified field in the issuing task. The name returned is that specified in the first Linker EDEF directive whose address matches the entry point of the current task; if not found, the initial start address of the task.

NOTES

1. The system places in $B4 the address of the receiving bound unit id field supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the receiving field.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0817 - Memory access violation.

3. On return, $B4 contains the address of the receiving field. If not found, 12 blank characters are placed in the receiving field.

Example:

In this example, $B4 is loaded with the address (BUNAME) of a 6-word field. The Bound Unit, Identification macro call is issued to place the name of the currently executing bound unit in that field.

```
BUNAME  RESV 6,0
        LAB  $B4, BUNAME
          .
          .
          .
       $BUID
```

BOUND UNIT, LOAD ($BULD)

Function Code: 0C/0A

Equivalent Command: None

Load the root of the specified bound unit and return its start address to the caller.

FORMAT:

```
[label]  $BULD  [location of root entry name address],
                [location of segmented code address],
                [location of code segment access rights],
                [location of data segment access rights]
```

ARGUMENT:

location of root entry name address

Any address form valid for an address register; provides the location of the address of the pathname of the bound unit to be loaded. The bound unit pathname can have an optional suffix, in the form ?entry, where entry is the symolic start address within the root. If the suffix is not supplied, the start address returned is the default start address established at link time.

location of segmented code address

Any address form valid for an address register; provides the address of any word in the code segment of the bound unit to be loaded. A null address specifies the segment number (if any) specified at link time. If no segment number was specified at link time, a null address directs the system to assign a segment number.

The system bypasses this argument if:

- The bound unit to be loaded is globally sharable (i.e., linked with the GSHARE directive)
- The task that issues $BULD is not running in a swap pool.

location of code segment access rights

Any address form valid for a data register; provides the access rights (read, write, and execute) for the code segment of the bound unit to be loaded. Access rights are expressed as a string of six bits, which are used to specifiy the type of access as follows:

Bits	Access Type
0-1	Read
2-3	Write
4-5	Execute

Ring access is coded as follows:

Bit Values	Ring
00	3
01	2
10	1
11	0

The system bypasses this argument if:

- The bound unit to be loaded is globally sharable (i.e., linked withthe GSHARE directive)

- The task that issues $BULD is not running in a swap pool.

In either of the two cases stated above, omitting this argument does not cause the system to assign default access rights. Otherwise, omitting this argument causes the system to assign the following default access rights to the code segment:

Read = 3
Write = 0
Execute = 3

location of data segment access rights

Any address form valid for a data register; provides the access rights for the data segment of a bound unit to be loaded. Access rights are specified in the same manner as for argument 3.

The system disregards this argument if:

- The bound unit to be loaded was not linked with the -R argument of the Linker command

* The task that issues $BULD is not running in a swap pool.

In either of the two cases stated above, omitting this argument does not cause the system to assign default access rights; otherwise, omitting this argument causes the system to assign the following default access rights to the data segment:

Read = 3
Write = 3
Execute = 3

DESCRIPTION:

Bound Unit, Load performs the same functions as Bound Unit Attach ($BUAT) with this excecption: $BUAT both loads the specified bound unit and starts its execution; $BULD only loads the specified bound unit, returning its start address in $B4. After issuing $BULD, a task resumes execution at the next sequential instruction following the macro call. To start execution of the loaded bound unit, the user should employ the instruction JMP $B4.

Arguments 2, 3, and 4 (which pertain to segments) are applicable only when the the call is issued from a swap pool. (Further limitations to the applicability of these arguments are noted in their descriptions).

When the call is issued from a swap pool, the address space of the issuing task is compared to the total address space of the bound unit to be loaded. If the two address spaces overlap, a warning message is returned to the error out file.

The segment number specified by argument 2 overrides a segment number specified at link time. If the bound unit to be loaded has been linked with the -R argument of the Linker command, the system assigns to that bound unit's data segment a segment number lower by 1 than the segment number specified for the code segment.

If the code or data segment of the bound unit to be loaded has been assigned a ring number at link time, arguments 3 and 4 can only lower that number; they cannot assign a ring number higher than one previously assigned. That is, arguments 3 and 4 can only increase the protection already given to a bound unit; they cannot lower that protection.

Up to seven bound units can be loaded to a task at a given time.

When a bound unit is loaded, its bound unit index indentifier (a value from 1 to 7) is returned in $R6. The bound unit index id can be used later to execute a Bound Unit, Detach macro call; it must be supplied with macro calls that handle the loaded bound unit's overlays.

When a globally sharable or sharable bound unit (linked with the GSHARE or SHARE directives, respectively) is loaded for the second time by a given task, the bound unit index id first returned is returned again; a new index id is not established. When, however, a nonsharable bound unit is loaded for the second time, the bound unit is loaded a second time and a new bound unit index id is established.

A bound unit is loaded according to the Loader's general rules for allocation. When the bound unit to be loaded is globally sharable, no additional copy of the bound unit is loaded; instead, the "number of BU users" is incremented. When the bound unit to be loaded is sharable (i.e., linked with the SHARE directive) and has already been loaded into a different segment than that specified by argument 2, a private copy of the bound unit is loaded for the issuing task. When a nonsharable bound unit is loaded, its external symbols may be resolved with respect to the calling bound unit.

NOTES

1. The address of the root entry name supplied by the first argument is placed in $B2. If this argument is omitted, the system assumes that $B2 contains the address.

2. The address of any word in the code segment, supplied by argument 2, is placed in $B4. If this argument is applicable but omitted, the system assumes that $B4 contains the address.

3. The access rights value for the code segment, supplied by argument 3, is placed in $R7. When this argument is applicable but omitted, the access rights default to those given above in the description of argument 3.

4. The access rights value for the data segment, supplied by argument 4, is placed in $R6. When this argument is applicable but omitted, the access rights default to those given above in the description of argument 4.

5. On return, $R3, $R4, $R5, $B1, $B3, $B5, and $B7 are preserved. $R1, $B4, $B6, $R6 contain the following information:

$R1 - Return status; one of the following:

0000 - No error

0602- No memory available for attached bound units array

082C - Number of attachable bound units exceeded

1605 - Relocation error

1607 - Media error

1608 - Symbol resolution error

1609 - File not found

160A - No memory available for bound unit

160D - Bound unit entry point not defined

160F - Bound unit cannot run in System Group Task

1614 - Access violation

1615 - Invalid bound unit format

1619 - Concurrency violation

$B4 - Address of entry point

$B6 - Address of data section (if any)

$R6 - Index id of loaded bound unit.

Example:

Bound unit PROG2 is loaded for use by the issuing task. When execution of the bound unit is requested, the start address is the default address. Bound unit PROG2 is found by applying the system search rules currently defined for the issuing task group.

```
          LDBU2     $BULD     !ROOT
                    .
                    .
                    .
          ROOT      TEXT      'PROG2Δ'
```

BOUND UNIT TRANSFER ($BUXFR)

Function Code: 0C/07

Equivalent Command: None

Terminate the issuing task's execution of the current bound unit. Return memory allocated for that bound unit and all currently attached or loaded bound units. Initiate execution of the specified bound unit.

FORMAT:

```
[label]  $BUXFR  [location of command line address],
                 [location of command line size],
                 [location of memory area]
```

ARGUMENTS:

location of command line address

Any address form valid for an address register; provides the pathname of the bound unit to be executed as the first ASCII string in the command line.

location of command line size

Any address form valid for an address register; provides the 2-byte size of the command line, including a blank which terminates the pathname.

location of memory area

Any address form valid for an address register; provides the address of the memory to be returned.

DESCRIPTION:

This macro call terminates execution of the current bound unit and initiates execution of a specified bound unit. If the resident bound unit is a sharable bound unit, the system increments the count of tasks that are currently associated with it.

NOTES

1. The system places in $B4 the address of the pathname supplied by argument 1. When this argument is omitted, the system assumes $B4 contains the address.

2. The system places in $R6 the size of the command line supplied by argument 2. When this argument is omitted, the system assumes $R6 contains the size.

3. The system places in $B2 the address of the memory area supplied by argument 3. When this argument is omitted, $B2 is set to null (i.e., no memory is returned).

4. On entry to the transferred bound unit, data registers and address registers contain the following information:

   $R1 - Unspecified

   $R2 - Set to zero

   $B2 - Unspecified

   $B4 - Unspecified

   $B6 - Address of the data space of the bound unit

   $R6, $R7 - Unspecified; preserved when $BUXFR was issued.

   Remaining registers are preserved when the Bound Unit Transfer macro call is issued.

5. Any error encountered during processing results in termination or deletion of the issuing task, with appropriate status.

Example:

In this example, control is transferred to the bound unit "nxtbu" at entry point "xntry". The memory block whose address is contained in $B3 is returned to the caller's memory pool.

```
                $BUXFR     !cmdln,!cmdsz,=$b3

    cmdln       text       'nxtbu?xntry '

    cmdsz       equ        ($-cmdln)*2
```

CANCEL CLOCK REQUEST ($CNCRQ)

Function Code: 05/01

Equivalent Command: None

Cancel a previously issued clock request.

FORMAT:

```
[label] $CNCRQ [location of CRB address]
```

ARGUMENT:

location of CRB address

Any address form valid for an address register; provides the address of the clock request block (CRB) to be removed from the timer queue.

DESCRIPTION:

This macro call removes a queued CRB that is no longer needed from the timer queue. The CRB must have previously been placed on the queue by a Request Clock macro call.

The Cancel Clock Request macro call is the only way to remove a cyclic CRB from the timer queue. A noncyclic CRB will also be removed when its interval elapses.

NOTES

1. The system places in $B4 the address of the CRB to be disconnected from the queue, supplied by the argument. If the argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0401 - Invalid date, time, interval value

   0403 - Invalid interval unit

0404 - CRB not connected to basic timer queue

$B4 - Address of CRB.

Example:

See the example given for the Wait on Request List macro call.

CANCEL REQUEST ($CANRQ)

Function Code: 0C/01

Equivalent Command: None

Cancel a previously issued request made through a Request Terminal, Request Specific Terminal, or Request Task macro call.

FORMAT:

```
[label] $CANRQ [location of address of request block]
```

ARGUMENT:

location of address of request block

Any address form valid for an address register; provides the address of the request block whose request is to be canceled.

DESCRIPTION:

This macro call cancels a request previously issued by a Request Terminal, Request Specific Terminal, or Request Task macro call.

NOTES

1. The system places in $B4 the address of the request block containing the request to be canceled, supplied by the argument . If this argument is omitted, the system assumes that $B4 contains the address of the request block.

2. On return, $R1 contains one of the following status codes:

   0000 - Request canceled

   0803 - Invalid wait on request block attempted

   0817 - Memory access violation

   083C - Request block not active

   083D - Request in process; unable to cancel.

3. When $R1 contains an 083C return code, $R6 contains the posted return code. The request block was completed before this macro call was issued.

Example:

In this example, the Cancel Request macro call is used to cancel the request established by a Request Terminal ($RQTML) macro call. (See the example for the Request Terminal macro call.)

```
END_RQ    $CANRQ    !IORB
```

CANCEL SEMAPHORE REQUEST ($CNSRQ)

Function Code: 06/01

Equivalent Command: None

If a previously issued Request Semaphore macro call caused a semaphore request block (SRB) to be queued, cancel the effect of that macro call by removing the SRB from the semaphore request queue. Return to the issuing task.

FORMAT:

```
[label]   $CNSRQ   [location of SRB address]
```

ARGUMENT:

location of SRB address

Any address form valid for an address register; provides the address of the semaphore request block to be removed from the semaphore request queue.

DESCRIPTION:

This macro call removes a specified SRB from its semaphore request queue. The SRB must have been queued as the result of a previously issued Request Semaphore macro call. The SRB address specified in the argument of the Cancel Semaphore Request macro call must be the same SRB address used in the Request Semaphore macro call.

When executed, this function increments the counter established by the Define Semaphore macro call, and previously decremented by the Request Semaphore macro call.

When the SRB is removed from the semaphore request queue, the memory required for its structure is returned to the system memory area.

NOTES

1. The system places in $B4 the address of the SRB supplied by the argument. If this argument is omitted, the system assumes that $B4 contains the SRB address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0502 - Invalid SRB

   $B4 - Address of SRB (as supplied).

Example:

In this example, the Cancel Semaphore Request macro call is used to cancel the semaphore request used in the example for the Request Semaphore ($RQSM) macro call. It is assumed that the task did not need the resource.

```
    $CNSRQ    !SRB
```

CHANGE WORKING DIRECTORY ($CWDIR)

Function Code: 10/B0

Equivalent Command: Change Working Directory (CWD)

Change the working directory to the one specified in the macro call. This function is usually done outside program execution.

FORMAT:

[label] $CWDIR [argument structure address]

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entry.

new working directory

A 1- to 45-byte pathname, which includes and must end with an ASCII space character, identifying the new current working directory. At least one nonspace character must be specified.

DESCRIPTION:

The specified pathname, which may be absolute or relative, must point to an existing directory; that is, this macro call does not dynamically create a directory. If a return status code other than 0000 is returned (see Note 2, below), an attempt is made to reestablish the previous working directory; if a subsequent error results, future functions may return an 0222 error code.

The system issues a mount request when a disk volume containing the new working directory is not mounted. The task is suspended until the volume is mounted or the operator cancels the mount request.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4; if the argument is omitted the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following stauts codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0209 - Named file or directory not found

   020C - Volume not found

   0222 - Pathname cannot be expanded; no working directory

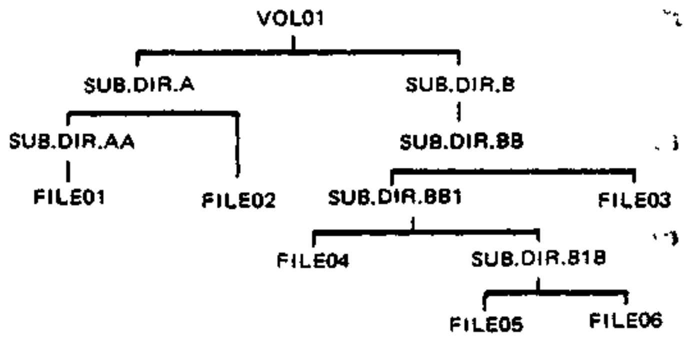
   0225 - Not enough system memory for buffers or structures

   0226 - Not enough user memory for buffers or structures

   0228 - Invalid file type (not a directory).

Example:

This example is based on the following file system hierarchy (see the System Concepts manual):



The current working directory is SUB.DIR.B1B, and FILE01 is to be accessed from subdirectory SUB.DIR.AA. It is not necessary to specify the absolute pathname to FILE01 if the Change Working Directory macro call ($CWDIR) to SUB.DIR.AA is specified, as shown below. The file can then be accessed with the simple pathname FILE01.

To change to this working directory, the Change Working Directory macro call can be specified:

```
    $CWDIR     !CHGPTH
```

to identify the path:

```
    CHGPTH     DC     '<<<<SUB.DIR.A>SUB.DIR.AA△'
```

or

```
    CHGPTH     DC     '^VOL01>SUB.DIR.A>SUB.DIR.AA△'
```

The first case uses the existing working directory as a base from which to expand the relative pathname; the second case produces the same result, but uses the absolute pathname. See the System Concepts manual for more information about relative and absolute pathnames.

CHECKPOINT

CHECKPOINT ($CKPT)

Function Code: 0D/0F

Equivalent Command: None

Cause a new checkpoint file image of the issuing task group to be recorded on the currently assigned checkpoint file.

FORMAT:

[label] $CKPT

ARGUMENTS:

None

DESCRIPTION:

This macro call causes a new checkpoint file image of the issuing task group to be recorded on the currently assigned checkpoint file; performs a file system cleanpoint; and constructs a checkpoint file image on which is recorded the curent status of task, files, and screen forms. Once recorded, the checkpioint file remains an available restart point until one of the following occurs:

- The next checkpoint for that task group is successfully completed
- The current checkpoint is disassigned
- The task group request is terminated normally.

The macro calls associated with the checkpoint/restart faciltiy are: Validate Checkpoint, Checkpoint, Restart, Defer Checkpoint, Checkpoint File.

NOTE

On return, $R1 and $R2 contain the following return codes:

If $R1=0, $R2 contains one of the following return codes:

0000 - No error

8000 - Return from restart

084A - Checkpoints disabled

0849 - No checkpoint file assigned

If $R2=0, $R1 contains one of the following return codes:

084B - Task group not in checkpointable state

0602 - Insufficient memory available to complete checkpoint

0107 - Physical I/O error writing to checkpoint file.

Example:

In this example, the current checkpoint image is recorded on the previously assigned checkpoint files.

```
                .
                .
                .
CPOINT          $CKPT      Record checkpoint image
                .
                .
                .
```

CHECKPOINT FILE

CHECKPOINT FILE ($CKPFL)

Function Code: 0D/11

Equivalent Command: Checkpoint File (CKPTFILE)

Establish or terminate the checkpoint file assignment for the task group request in which it is issued.

FORMAT:

```
[label]  $CKPFL   {ASSIGN   }
                  {DISASSIGN}
                    [location of pathname of checkpoint files]
```

ARGUMENTS:

ASSIGN
DISASSIGN

One of the following values is specified to indicate whether the checkpoint files are to be established or terminated:

ASSIGN

Establish the checkpoint files specified by the pathname supplied in argument 1.

DISASSIGN

Terminate the checkpoint files specified by the pathname supplied in argument 1.

location of pathname of checkpoint files

Any address form valid for an address register; provides the pathname of the checkpoint files to be assigned or disassigned.

DESCRIPTION:

This macro call establishes or terminates the checkpoint file assignment for the task group request in which it is issued. If checkpoint files are to be assigned, a pathname must be supplied in argument 2; if checkpoint files are to be disassigned, argument 2 is not required. The disassignment of checkpoint files invalidates any currently valid checkpoint.

The macro calls associated with the checkpoint/restart facility are: Validate Checkpoint, Checkpoint, Restart, Defer Checkpoint, Checkpoint File.

NOTES

1. If ASSIGN is specified in argument 1, $R2 is set to zero. If DISASSIGN is specified, $R2 is set to one.

2. The system places in $B4 the address of the pathname supplied by argument 2. When this argument is omitted, the system assumes that $B4 contains the address.

3. On return, $R1 contains one of the following return codes:

   0000 - No error

   0209 - File or directory not found (returned only when call is issued with DISASSIGN argument).

   0213 - Exclusive access not available

   0846 - Checkpoint file is not a sequential file

   0847 - Checkpoint file already assigned

   0848 - File contains a valid checkpoint; unable to assign

   0849 - No checkpoint/restart file assigned.

Example:

This example illustrates the use of the Checkpoint File macro in the ASSIGN and DISASSIGN operations. The assignment is made to a particular checkpoint file pair to establish the checkpoint session. If the assignment fails because the files already contain a valid checkpoint, this checkpoint can be ignored and files can be reused by disassigning those checkpoint files. This should be done if no restart is desired from the checkpoint. After disassigning, the ASSIGN can be reissued. At the end of the session, the current checkpoint files are disassigned, making them available for another checkpoint session.

```
        $CKPFL              ASSIGN, !Path        Assign check-
                                                 point files

        bez                 $rl,>alldne          Continue if
                                                 successful

        cmr                 $rl, =vlderr

        bne                 $rl,>errxit

        $CKPFL              DISASSIGN, !Path     Disassign
                                                 checkpoint
                                                 files

        bnez                $rl,>errxit

        $CKPFL              ASSIGN, !Path        Retry assign
          .
          .
          .
        checkpoint session
          .
          .
          .
        $CKPFL              DISASSIGN            Disassign cur-
          .                                      rent files
          .
          .
Path    text                '^myvol>ckptfile '

vlderr  dc                  Z'0848'              File contains
                                                 a valid check-
                                                 point
```

CLEAN POINT ($CLPNT)

Function Code: 0C/13

Equivalent Command: None

Define a clean and consistent point in program execution at which all file records updated by the program are valid. Make the updated records visible to other users sharing these files. Write out to disk the records updated by the issuing task group; unlock the records previously locked by the issuing task group, for all files assigned to the task group.

FORMAT:

```
[label]  $CLPNT
```

ARGUMENT:

None.

DESCRIPTION:

This macro call results in the following:

1. All disk buffers modified by the task group are written to disk.
2. If the end-of-data record for a disk file accessed by the task group is altered, the directory record for that file is updated.
3. All record locks set by this task group are unlocked, allowing other users to continue processing.
4. The call defines the last good state to which files, subsequently updated by the task group, can be rolled back (i.e., recovered).
5. The recovery file is reset; that is, the macro call deletes any "before" images previously recorded for all files in the task group. (See the Roll Back macro call for discussions about file recovery.)
6. Updates to files (i.e., after images) are written to a system journal if one has been defined and if files have the "restore" attribute.

The period of all I/O activity during which the user is altering and manipulating records is defined as a phase, or interval between clean point executions, when data is in an inconsistent or alterable condition. A phase change, when data is declared to be consistent, is accomplished by the Clean Point macro call. File recovery is done on a phase basis; that is, a phase rollback (recovery) to the last Clean Point execution, by means of the Roll Back macro call. The call also resets the recovery file.

Record locking, a file system mechanism, provides multi-user interference protection for shared file access. A record, when accessed by a user, is locked by a lock applied to the control interval(s) where the record is located. Locking is on a first-come, first-served basis. Another user (task group) sharing this file is denied access to that record and any other record in the same control interval, until the previous user unlocks the record.

The only limit to the number of locks at one time is the amount of memory dedicated to the lock pool at system building.

Record locks for a file may be requested when the file is reserved through a Get File macro call or by a GET command. Normally, record locking is an attribute set by the Create File or Modify File macro calls/commands. Once record locking for a file is requested, any access (read or write) causes a lock. Once locked, records are unlocked only when a Clean Point macro call is issued or when the file is closed. (Abnormal task group termination also causes records to be unlocked.)

Records should be unlocked when there is no further need to lock them. Otherwise, when records remain locked, lock pool overflow or deadlock record contention may result. The description of the Get File macro call has more details about record locking.

The Clean Point macro call allows a user to structure an application into steps. At the end of each step, successful execution of the macro call ensures that all the file updates have been written to disk, and that the resources used in record locking are released to the system.

NOTES

1. To perform the Clean Point function in a COBOL program, the user must call an Assembly language subroutine that contains the Clean Point macro call(s).

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   01xx - Physical I/O error
   023A - Recovery file I/O error
   0263 - Journal file I/O error.

CLEAR EXTERNAL SWITCHES

CLEAR EXTERNAL SWITCHES ($CLRSW)

Function Code: 0B/02

Equivalent Command: Modify External Switches (MSW)

Set the specified switches in the task group's external switch word to off; return the inclusive logical OR of the previous settings.

FORMAT:

```
[label]  $CLRSW  external switch name,
                 [external switch name],
                           .
                           .
                           .
                 [external switch name]
```

ARGUMENTS:

external switch name ... external switch name

A single hexadecimal digit specifying the external switch in the task group's external switch word to be set off. A maximum of 16 external switch names (0 through F) can be specified. If no arguments are supplied, the system assumes that $R2 contains a mask word specifying the switches to be set off. If ALL is specified for any argument, all external switches are set off.

DESCRIPTION:

This macro call provides a mask by which switches can be set off in the external switch word of the issuing task's task group. It also provides an indication of the previous settings of the switches.

The mask word is $R2. Each bit that is one in $R2 causes the corresponding bit in the external switch word to be set off; each bit that is zero causes the corresponding bit to remain unchanged.

When the Clear External Switches macro call is executed, $R2 contains the new settings of the external switch word. Bit 11 (bit-test indicator) or the I-register provides an indication of the previous setting of the switches, as follows:

- If bit 11 is zero, no switch set off had previously been set on.
- If bit 11 is one, at least one switch set off had previously been set on.

NOTES

1. The bits corresponding to the external switches in the arguments are set on in $R2; if no arguments are supplied, the system assumes that $R2 contains the mask to be used. If ALL is specified for any argument, all bits are set on in $R2.

2. On return, $R2 and the I-register contain the following information:

   $R2 - External switch word after modification

   I-register (Bit 11) - Inclusive OR of previous settings of switches set off:

   0 - No switch off was on
   1 - At least one switch set off was on.

Example:

In this example, the Clear External Switches macro call is used to turn off external switches 4, 8, and C of the task group in which the issuing task is executing.

```
CLR_AA    $CLRSW    4,8,C
```

CLOCK REQUEST BLOCK

CLOCK REQUEST BLOCK ($CRB)

Function Code: None

Equivalent Command: None

Generate a regular or cyclic clock request block (CRB) whose length is from six to nine words.

FORMAT:

```
[label]  $CRB  [CRB type],
               { ,                                  }
               { WAIT,                              }
               { NWAIT, [termination action]        }
               [interval value]
```

ARGUMENTS:

CRB type

A value specifying the type of CRB to be generated, as follows:

C - Generate a cyclic CRB
R - Generate a regular (noncyclic) CRB

[WAIT | NWAIT],

One of the following values is specified to indicate whether the requesting task is to be suspended until the clock request has been satisfied.

WAIT

Suspend the issuing task until the clock request has been satisfied (set W-bit to zero).

NWAIT

Do not suspend the issuing task (set W-bit to one).

If this argument is omitted, the value NWAIT is assumed.

If WAIT is specified, argument 3 (termination action) must be omitted.

termination action

One of the following values is specified to indicate the action to be taken when the clock request is satisfied.

SM=aa

Do not suspend the issuing task; release (V-op) the semaphore identified by aa (two ASCII characters) when timeout has occurred.

RB=label

Do not suspend the issuing task; issue a request for the request block identified by label, when timeout has occurred.

Note that the requesting task must be asynchronous, can not wait on the requested task later on, and can only point to a task request block (TRB). The requested task must have already been created (not spawned), be asynchronous, and have a valid LRN. When the requesting task terminates, the TRB pointed to by "label" must be inactive.

If this argument is omitted (or argument 2 is WAIT), the generated CRB contains no termination option.

interval value

Unit of time after which completion of the request is posted; has one of the following values:

MS=n
TS=m
SC=m
MN=m
CT=m
DT

MS indicates milliseconds; TS, tenths of seconds; SC, seconds; MN, minutes; CT, units of clock resolution; and DT, internal date/time.

n is an integer value from 1 through 65535; m is an integer value from 1 through 32767. If DT is selected, the application must store the 48-bit internal date/time value at offset C_TM of the created request block.

DESCRIPTION:

The CRB is used as the standard means of synchronizing events with the passage of time. A CRB contains the time at which, or the interval after which, completion of the request is to be posted (marked as complete).

There are two types of CRBs: regular and cyclic.

When the interval specified in a cyclic CRB has been satisfied, it is automatically recycled to begin a new clock request for the initially specified interval. This process continues until a Cancel Clock Request macro call is issued for this CRB.

A regular CRB is dequeued from the timer queue when the specified interval has been satisfied. A new Request Clock macro call must be issued to requeue the CRB.

Example:

In this example, the Clock Request Block macro call is used to generate a cyclic CRB with an interval of 500 milliseconds. The issuing task is not suspended. When the request has been satisfied, the issuing task releases semaphore XX.

```
    CLKAA    $CRB    C,NWAIT,SM=XX,MS=500
```

CLOCK REQUEST BLOCK OFFSETS ($CRBD)

Generated Label Prefixes:

	C_RRB/C_SEM
CRB label	offset 0
	C_CT1
	C_CT2
	C_TM

See Appendix C for the format of the clock request block.

DESCRIPTION:

See the Clock Request Block macro call.

CLOSE FILE ($CLFIL)

Function code: 10/55 (normal), 10/56 (leave), 10/57 (unload)

Equivalent Command: None

Terminates processing of the specified file. The file cannot be processed again until another Open File macro call is issued. The file to be closed is identified by supplying its logical file number.

FORMAT:

$$\text{[label]}\quad \text{\$CLFIL}\quad \text{[fib address]}\quad \left[\left\{\begin{array}{l}\text{,NORMAL}\\ \text{,LEAVE}\\ \text{,UNLOAD}\end{array}\right\}\right]$$

ARGUMENTS:

fib address

Any address form valid for an address register; provides the location of the file information block (FIB). The FIB must contain a valid logical file number (LFN).

$\left\{\begin{array}{l}\text{NORMAL}\\ \text{NOR}\end{array}\right\}$

Normal mode for closing files; the file can be reopened during execution of the task group.

If the file is tape-resident, the end-of-file (EOF) labels are written (if necessary) and the tape is rewound to its beginning-of-tape (BOT) position.

If the file is a terminal device, the line is disconnected according to the specifications made at system building time.

For card punch files, a file mark card is punched. This card is recognized as the end-of-file for read operations.

NORMAL is the default value for this macro call.

{LEAVE}
{LEV}

For tape files, the action is the same as for NORMAL mode, except that the tape is not rewound; that is, it remains at its current position.

For terminal device files, this argument indicates that the line is <u>not</u> to be hung up, regardless of the specification made at system building time.

For card punch files, this argument indicates that a file mark card is not to be punched.

{UNLOAD}
{UNL}

For tape-resident files, the action is the same as for NORMAL mode, except that after being rewound, the tape is unloaded (i.e., cycled down).

For terminal device files, the line is hung up (regardless of the specification made at system building time).

DESCRIPTION:

The FIB address specified by the first argument of this macro call can refer to the same structure specified in the Open File macro call with which this macro call is paired.

This macro call causes all unwritten buffers to be written, records to be unlocked, and the logical EOF label to be updated. However, the call does not remove the file (see the Remove File macro call) from the task group (i.e., the file remains reserved for the task group and can be reopened).

If the file being closed is a card punch, a file mark card is punched. A card reader/punch is considered to be a card punch if the FIB program view word at open time had bit 2 set to one (write permitted) and bit 1 set to zero (read not permitted).

The following information applies only to magnetic tape. The actions performed on closing a tape file are determined by these factors:

- Whether or not the write permit bit (bit 2) in the FIB program view word was set on when the file was opened.
- Whether or not write operations to the file were peformed.

Note that when a tape volume is opened for storage management access, and only a device name is specified, processing of labels is not performed. This is the user's responsibility.

1. Reserved and opened for writing:

   a. If the file has been opened in RENEW mode, the trailer label group is written, followed by an end-of-data (EOD) tape mark. This action is performed whether or not data records were actually written into the file.

   b. If the file has been opened in PRESERVE mode, the trailer label group and EOD tape mark are written only if write operations have been performed. In this case, data and/or files located beyond the current position of the tape are destroyed.

      If no write operations have been performed, the trailer label group is not written and existing data and/or files located beyond the current position of the tape are preserved.

   c. If the LEAVE option is specified, the tape is left at its current position.

2. Reserved and opened for reading:

   a. If the EOF tape mark has been detected, the trailer label group is processed and the action specified by NORMAL, LEAVE, or UNLOAD is taken.

      If the LEAVE option is specified, the tape is positioned at the end of the current trailer label group.

b. If the EOF tape mark has not been detected, the trailer label group is not processed. When the LEAVE option is specified, the tape is left at its current position. A subsequent OPEN will correctly reposition the tape before executing the Open function.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets macro call.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0207 - LFN not open

   0225 - Not enough system memory for buffers or structures

   0226 - Not enough user memory for buffers or structures

   0236 - Tape block count error.

Example:

In this example, it is assumed that the file opened in the example for the Open File macro call is to be closed. The macro call is coded as follows:

```
MYFIB     DC        5        LFN 5
CLFILA    $CLFIL    !MYFIB
```

Since the second argument is not specified, the system assumes NORMAL mode.

COMMAND IN ($CIN)

Function Code: 08/02

Equivalent Command: None

Read the next record from the standard command-in file for the issuing task.

FORMAT:

```
[label]  $CIN  [location of record area address],
               [location of record size],
               [byte offset of beginning of record area]
```

ARGUMENTS:

location of record area address

Any address form valid for an address register; provides the address of a record area in the issuing task into which the next record on the command-in file will be placed.

location of record size

Any address form valid for a data register; provides the size (in bytes) of the record whose address is given in argument 1.

byte offset of beginning of record area

Any address form valid for a data register; provides the byte offset of the beginning of the record area (from the address provided in argument 1).

DESCRIPTION:

This macro call allows a task to read the next record from the standard command-in file.

NOTES

1. The system places in $B4 the address of the command input record area supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the record area address.

2. The system places in $R6 the record area size supplied by argument 2; if this argument is omitted, the system assumes that $R6 contains the correct size.

3. If argument 3 is L, $R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, $R7 is set to one to designate that the record area begins in the right byte of the specified address. Any other value for argument 3 designates the location of the byte offset to be used, and is placed in $R7. If argument 3 is omitted, the record area begins in the left byte of the specified address, and $R7 is set to zero.

4. On return, $R1, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0817 - Memory access violation

   All data management read-next-record error codes may also be returned. See the <u>System Messages</u> manual.

   $R6 - Residual range (number of bytes left unfilled in record area).

   $R7 - File type: bits 10 through 15 of $R7 contain the hexadecimal value for the following file types:

Value	File Type
02	Fixed relative
10	Line/serial printer
11	Card reader
12	KSR (MDC-connected)
1A	Bidirectional MLCP
1B	BSC
1E	Output (only MLCP)
30	Variable sequential (spanned records)
32	Relative
33	Indexed (data)
34	Indexed (index)

$B4 - Input record area address.

Example:

In this example, the issuing task is to read the next record of the command-in file into a 128-byte record area whose address is in RECAD. The record area begins at an offset of 10 bytes from the indicated address.

```
    INDAD     $CIN     !RECAD,=128,=10
                       .
                       .
                       .
    RECAD     RESV     5+64,0
```

COMMAND LINE PROCESS ($CMDLN)

Function Code: 0C/08

Equivalent Command: None

Process the supplied command line by spawning a task to execute the command named in the first argument of the macro call, and wait for the task's termination.

FORMAT:

```
[label]   $CMDLN   [location of command line address],
                   [location of command line size]
```

ARGUMENTS:

location of command line address

Any address form valid for an address register; provides the address of the supplied command line.

location of command line size

Any address form valid for a data register; provides the size (in bytes) of the command line to be processed.

DESCRIPTION:

This macro call allows you to embed commands in your program (see the Commands manual). The same task that executes the particular command when given from the terminal is spawned to execute the command named in the macro call.

The task spawned on behalf of the macro call is provided with a request block that has been constructed by the system to contain the edited arguments in system-standard Task Request Block format. The task that issues this macro call waits for the completion of the spawned task before continuing its own processing. The spawned task passes the completion status ($R1) to the issuing task.

NOTES

1. The system places in $B4 the address of the command line, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the command line to be processed.

2. The system places in $R6 the size of the command line, supplied by argument 2. If this argument is omitted, the system assumes that $R6 contains the size.

3. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0000-FFFF - Completion status returned by spawned task

   0601 - Invalid size for memory pool

   0602 - Insufficient memory

   0805 - Unbalanced quotation marks, brackets, or parentheses

   080C - Unresolved symbolic entry point

   1609 - Invalid bound unit pathname for first argument

   160A - Insufficient memory

   FFFE - Honeywell component error previously reported; reported error code found in word following last pointer in task request block argument list.

   FFFF - Honeywell component error previously reported

   $B4 - Address of supplied command line.

Example:

In this example, the Command Line Process macro call causes a command line to be processed which will cause the Assembler to assemble the source program MYPROG, residing in the current working directory. The Assembler will use 5K words of memory, taken from the issuing task group's memory pool, for its symbol table. The assembly listing will be written on the device named LPT01, and the object code will be stored in the file MYPROG.O in the working directory. If MYPROG.O does not already exist, it will be created.

```
        $CMDLN   !LINE,=LENGTH
                 .
                 .
                 .
  LINE  TEXT     'MAP  MYPROG  -SZ  5  -COUT  >SPD>LPT01'
LENGTH  EQU      2*($-LINE)
```

CONSOLE MESSAGE SUPPRESSION ($CMSUP)

Function Code: 09/02 (suppression), 09/03 (no suppression)

Equivalent Command: None

Turn console message suppression on or off for the issuing task's task group.

FORMAT:

```
[label]   $CMSUP   [keyword]
```

ARGUMENT:

keyword

One of the following values:

ON

Turn on console message suppression (function code 09/02)

OFF

Turn off console message suppression (function code 09/03)

If this argument is omitted, OFF is assumed.

DESCRIPTION:

This macro call turns console message suppression on or off for the issuing task's task group.

When console message suppression is turned on, operating system components, such as Storage Management, do not issue error messages to the operator terminal, either directly (through the facility offered by the Operator Information Message macro call) or indirectly (through the facility offered by the Report Message macro call). Turning on console message suppression does not disable these facilities; rather, it prevents the system components from using the facilities to report anything other than catastrophic errors.

When console message suppression is turned on, the error code normally used in the operator message is returned in $R1 (assuming the message had an error code).

When console message suppression is turned off, messages are again issued in the normal manner.

NOTE

On return, $R1 contains one of the following sub-function codes:

0002 - Turn on suppression
0003 - Turn off suppression.

Example:

In this example, the issuing task turns on console message suppression for the task group under which it is running.

```
SUPON    $CMSUP    ON
```

CREATE DIRECTORY

CREATE DIRECTORY ($CRDIR)

Function Code: 10/A0

Equivalent Command: Create Directory (CD)

Create a new directory in the file system hierarchy. This function is usually done outside program execution.

FORMAT:

```
[label]   $CRDIR   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of an argument structure that must contain the following entries in the order shown.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII "space" character) that, when expanded, identifies the directory in the hierarchy in which the new directory is to be created and the name of the new directory itself.

reserved

A 4-byte entry containing zeros.

DESCRIPTION:

This macro call can be use only to create new directories, which are created with:

- An initial allocation of 8 physical sectors (allowing 32 entries) for diskette, 8 physical sectors (allowing 64 entries) for cartridge disk and storage module (except 19-surface, 200 tracks-per-inch), or 16 physical sectors (allowing 128 entries) for 19-surface, 200 tracks-per-inch storage module.

- An increment allocation of 4 physical sectors (allowing 16 entries each) for diskette, 8 physical sectors (allowing 64 entries) for cartridge disk and storage module (except 19-surface, 200 tracks-per-inch), or 16 physical sectors (allowing 128 entries for 19-surface, 200 tracks-per-inch storage module).

- A maximum allocation of 4000 physical sectors (allowing a maximum of 16,000 entries) for diskette, or 4000 physical sectors (allowing a maximum of 32,000 entries) for cartridge disk and storage module.

NOTES

1. If the argument is coded, the system reads the address of the parameter structure into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - Successful completion

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0209 - File or directory not found

   020C - Volume not found

   0212 - Attempted creation of existing file or directory

   0215 - Not enough contiguous logical sectors available

   0222 - Pathname cannot be expanded; no working directory

   0224 - Directory space limit reached or not expandable

   0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

022C - Access control list (ACL) violation.

Example:

In this example, the macro call is used to create the subdirectory, labeled SUBINDEX.A, identified in the Create File macro call description example. This subdirectory must exist before the path identified in that example (i.e., ^VOL03 SUBINDEX.A >FILE_A) can be used. Prior to issuing the Create Directory macro call, the following parameter structure and pathname must exist:

```
SUBDIR  DC    <DIRPTH
        RESV  2-$AF
        RESV  2,0
        .
        .
        .
DIRPTH  DC    '^VOL03>SUBINDEX.AΔ'
```

The macro call can be specified as follows:

```
$CRDIR  !SUBDIR
```

CREATE FILE ($CRFIL)

Function Code: 10/30

Equivalent Command: Create File (CR)

Create a new disk file by placing a description of the file in the file system hierarchy and, optionally, by allocating space for it. This function is normally done outside program execution. The user identifies this file by either a logical file number (LFN) or a pathname, or both. At completion of Create File execution, the file is reserved exclusively for the task group. If the user supplies both an LFN and a pathname, the file is created and reserved and, in addition, it is assigned to the LFN. Subsequent macro calls (Open File, Read Record, etc.) can then be directed to the file through this LFN. The Create File macro call can be used to create any of the disk files which are described in the Data File Organizations and Formats manual, including:

- Fixed-relative
- Relative
- Sequential
- Indexed
- Alternate index
- Random (CALC)
- Dynamic.

In addition, the Create File macro call can be used to create a temporary disk file that will exist only during this task group's execution.

FORMAT:

```
[label]    $CRFIL    [pathname structure address]
```

ARGUMENT:

pathname structure address

Any address form valid for an address register; provides the location of the parameter structure defined below in Table 2-1. The order of entries must follow the order shown in the table.

Table 2-1. Create File Parameter Structure

Field Name	Size (bytes)	Meaning
LFN	2	The logical file number (LFN) used to refer to the file. Must be a binary number from 0 through 255, ASCII blanks (which indicate that an LFN is not specified), or -1 (FFFF) (which indicates that the system should assign an LFN from the pool of those available).
Pathname Pointer	4	Pointer to the pathname of the file to be created. Zeros indicate that a pathname is not specified. If specified, the pathname must end with a space character. A pathname conisting of a single space character indicates that a temporary file is to be created.
File Organization	1	UFAS files: 'S' = Sequential file 'R' = Relative file 'I' = Indexed file 'V' = Dynamic file 'C' = CALC (random) 'X' = Alternate index Non-UFAS relative files: '2' = Fixed relative without deletable records '5' = Fixed relative with deletable records

Table 2-1 (cont). Create File Parameter Structure

Field Name	Size (bytes)	Meaning
Space Allocation Options	1	Bit 0: 0 = Space initially allocated need not be contiguous (i.e., may consist of more than one extent). 1 = Space initially allocated must be contiguous. Bits 1-2: MBZ Bits 3-7: (for multivolume sets): 00000 = Disk space is initially allocated on the volume having the most space available. nnnnn = Disk space is initially allocated on the "nnnnn"th volume in the set.
Logical Record Size	2	Length of the longest record in the file. For file organizations R, S, I, V, X, and C, this size does not include the logical record headers. For file types 2 and 5, this size includes the 2-byte record header. Zero in this entry takes the following defaults: R: No default; must be specified S: 16K bytes I: CI size - 32 bytes V: CI size - 18 bytes (dynamic files) X: Key size + 6 bytes C: CI size - 40 bytes (random files) 2: 256 bytes 5: 256 bytes

Table 2-1 (cont). Create File Parameter Structure

Field Name	Size (bytes)	Meaning
Control Interval size	2	For UFAS files, the size of data transfer to/from main memory (and thus buffer size); includes both control interval and logical record header information; must be a multiple of 256 bytes. Zeros indicate a CI size of 512 bytes. For fixed-relative files, defines only the unit of space allocation; includes record header information; must be a multiple of 128 bytes. Zeros indicate the device physical sector size (128 or 256 bytes).
Initial allocation size	2	Number of CIs to be allocated to the file when it is created. Zeros indicate that no space is to be allocated initially. For random files, an initial or maximum allocation size must be specified.
Allocation growth size	2	Number of additional CIs to be allocated whenever necessary. Zero indicates 40 physical sectors.
Maximum allocation size	2	The maximum number of CIs that can be allocated to the file. Zeros indicate no limit. For random files, either an initial or maximum allocation size must be specified.
Free space per CI or	2	For indexed files: the number of bytes to be left free in each data CI at file loading time. Records can be inserted into these bytes without causing overflow. For alternate indexes: the number of bytes to be left free in each index CI at index-loading time. New index entries can be inserted into these bytes without forcing a CI split.

Table 2-1 (cont). Create File Parameter Structure

Field Name	Size (bytes)	Meaning
Inventory threshold		For dynamic and random files: The percent of a data CI, which must be filled before inventory is updated. Specifying an inventory threshold causes the allocation of inventory CIs, which contain 1 byte per data CI. These inventory bytes, which describe the amount of free space in corresponding data CIs, facilitate the insertion of new records. For dynamic and random files, zero indicates a threshold of 75%. For other file types, this field is zero.
Local overflow allocation or Hash results	2	For indexed files: The frequency at which local overflow CIs are allocated at file-loading time. One local overflow CI is allocated after every n data CIs are allocated. For random files: The number of possible hash results; must be less than or equal to the maximum number of CIs. Zero indicates one hash result per CI.
Number of record descriptors	2	The number of record descriptors specified for the file. For indexed, dynamic, and random files and alternate indexes: The value must be Z'0001', since these files have only one record descriptor. For other file formats: MBZ.

Table 2-1 (cont). Create File Parameter Structure

Field Name	Size (bytes)	Meaning
Pointer to record descriptors	4	For indexed, dynamic, and random files; alternate indexes; and I-D-S/II areas: A pointer to the record descriptor structure shown below. For other file formats: A null value.
Reserved	8	Must be zeros.

Table 2-2 describes the record descriptor structure, which is pointed to by the Create File parameter structure.

Table 2-2. Record Descriptor Structure

Field Name	Size (bytes)	Meaning
Record descriptor size	2	The size (in words) of this structure. Zeros indicates a size of 9 words.
Record type	2	Bit 0: 1 = duplicate keys allowed 0 = duplicate keys not allowed Bits 1-3: MBZ Bits 4-15: A value that uniquely identifies the record type of the record described by this structure. A record's type is determined by the values of the remaining fields in this structure. Currently, the values of the remaining fields are the same for all records in a given random file, indexed file, and alternate index. Thus, for these files, bits 4-15 must be zero, indicating that all records in these files are of the same type.

Table 2-2 (cont). Record Descriptor Structure

Field Name	Size (bytes)	Meaning
Number of key components	1	The number of components in the record's key. Must be 1 for random and indexed files, in which there is only 1 component per key. Alternate indexes support more than one component per key.
Reserved	9	Must be zero.
Key component data type	1	The data type of the key component. 'C' = Character string 'D' = Decimal unpacked, trailing sign 'B' = Signed binary 'S' = Decimal packed, trailing sign 'U' = Decimal packed, unsigned For indexed files and alternate indexes, a data type in upper case (e.g., 'C') indicates ascending key sequence, and, in lower case (e.g., 'c'), indicates descending key sequence.
Key component size	1	The size of the key component. For character string, signed binary, or unpacked decimal fields: The size is expressed in bytes. For packed decimal fields: The size is expressed in half-bytes. For signed packed decimal fields: The size must include 1 half-byte for the sign.

Table 2-2 (cont). Record Descriptor Structure

Field Name	Size (bytes)	Meaning
Key component location	2	The offset of the key component from the beginning of the record. For character string, signed binary, or unpacked decimal fields: The offset is expressed in bytes. For packed decimal fields: The offset is expressed in half-bytes. The first byte or half-byte in the record is 1.

NOTE

The key component data type, size, and location fields constitute one key component descriptor that can be repeated within a record descriptor as many times as the number of components per key.

DESCRIPTION:

This macro call cannot be issued if the file already exists (i.e., if a Create File macro call with the same pathname has been previously issued and the file has not been released), or if the LFN is currently assigned to an open file in the same task group. When properly coded, the Create File macro call allocates space to the specified file in accordance with the entries in the argument structure (i.e., it "creates" an empty file, which can be loaded with data through data management or storage management macro calls).

The file can be specified (in the argument structure) by (1) an LFN only, (2) a pathname only, or (3) both an LFN and a pathname.

1. If only an LFN is specified, it must previously have been associated with a pathname (see the Associate File macro call).

2. If only a pathname is specified (i.e., the LFN field contains ASCII spaces (2020)), the file is reserved without a unique LFN. The only requests that can use the file are those that can refer to it by pathname. If a pathname is specified, and the LFN field contains a value of -1 (FFFF), the system assigns a unique LFN; it is the user's responsibility to return the LFN to the pool of available LFNs (through the Remove File macro call) when it is no longer needed. The unique LFN is assigned from the pool of available LFNs for the task group. The highest LFN not already assigned is set in the LFN entry of the argument structure, overlaying the previous contents (FFFF). You must move this value to other structures (i.e., argument structures or FIBs) as required.

3. If both an LFN and a pathname are specified, in addition to their creation, the file is assigned to the specified LFN.

Zeros are specified in the "initial allocation size" entry if space is allocated according to the value specified in the "allocation increment size" entry at file load time.

Allocation increment size, although stated in terms of CIs, cannot resolve to a value greater than 8191 logical sectors for mass storage units or 8191 physical sectors for diskettes and cartridge disks. Disk space initially allocated to the file may not be contiguous unless the contiguous allocation is specified. If the contiguous option is specified, initial space is also restricted to 8191 logical sectors. After the space is allocated, the system reserves it with "exclusive" concurrency control; as a result, it is not necessary to issue a Get File macro call before an Open File macro call. If the file being created is a temporary file (see the "pathname pointer" entry described in the argument structure description), it can be released (i.e., deleted) through the Remove File macro call.

Offset tags for the parameter structure can be defined by the Create File Parameter Structure Block Offsets and Create File Key Descriptors Block Offsets macro calls.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0208 - LFN or file already open

   0209 - Named file or directory not found

   020C - Volume not found

   0210 - LFN conflict

   0211 - Unable to establish unique LFN

   0212 - Attempted creation of existing file

   0215 - Not enough contiguous logical sectors available

   0222 - Pathname cannot be expanded; no working directory

   0224 - Directory space limit reached or not expandable

   0225 - Not enough system memory for buffers or control structures

   0226 - Not enough user memory for buffers or control structures

   022C - Access control list violation.

Example:

In this example, the argument structure labeled FILE_A, defined under "Assumptions for File System Examples" in Appendix A, describes the file to be created. In addition, the following key descriptor structure has been defined:

```
KEY    DC     Z'00000000'   RESERVED
       DC     Z'0100'       NO. OF COMPONENTS = 1
       RESV   4,0           RESERVED
       DC     Z'430A'       KEY COMP. DATA TYPE = C;
                            KEY LENGTH = 10
       DC     1             KEY LOC. IN RECD. = FIRST POSITION
```

Also, the pathname is defined as follows:

```
    IDX01    DC    '^VOL03>SUBINDEX.A>FILE_A
```

After the preceding definitions have been made, the following Create File macro call creates FILE_A:

```
    $CRFIL    !FILE_A
```

CREATE FILE KEY DESCRIPTOR BLOCK OFFSETS

CREATE FILE KEY DESCRIPTOR BLOCK OFFSETS ($CRKDB)

Associated Macro Calls:

Create File, Get File, Create File Parameter Structure Block Offsets, Get File Information Parameter Structure Block Offsets

FORMAT:

[label] $CRKDB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0 1	Reserved	
2	No. of Key Components	Reserved
3 4 5 6	Reserved	
7	Key Type	Key length
8	Key Offset	

NOTE

Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
Y_NKC	+2	Number of key components
Y_KTYP	+7	Key type (first byte)
Y_KLEN	+7	Key length, in bytes (second byte)
Y_KOFF	+8	Key offset, in bytes
Y_SZ	9	Size of structure (in words); not a field in the block

NOTE

This macro call has the same effect as the Get File Information, Key Descriptor Block Offsets macro call.

CREATE FILE PARAMETER STRUCTURE BLOCK OFFSETS

CREATE FILE PARAMETER STRUCTURE BLOCK OFFSETS ($CRPSB)

Associated Macro Calls: Create File, Create File Key Descriptor Block Offsets

FORMAT:

[label] $CRPSB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Logical File Number (LFN)	
1 2	Pathname Pointer	
3	File Organization	Allocation options
4	Logical Record Size	
5	Control Interval Size	
6	Initial Allocation Size	
7	Allocation Growth Size	
8	Maximum Allocation Size	
9	Free Space per Control Interval/ Inventory threshold	
10	Local Overflow Increment/ Hash results	

11	Number of Record Descriptors
12 13	Record Descriptor Pointer
14 15 16 17	Reserved

NOTES

1. Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

2. The last five fields of a record descriptor constitute a key descriptor describing the key by which the record is accessed.

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
R_LFN	0	Logical file number (LFN)
R_PTHP	+1	Pointer to path
R_ORG	+3	File Organization (first byte)
R_OPT	+3	Allocation options (second byte)
R_LRSZ	+4	Logical record size
R_CISZ	+5	Control interval size
R_INSZ	+6	Initial allocation size
R_GRSZ	+7	Allocation increment size
R_MXSZ	+8	Maximum allocation size
R_FPC	+9	Amount of free space per CI (indexed files)
R_INVT	+9	Inventory threshold (dynamic/random files
R_HASH	+10	Number of hash results (random files)
R_LOV	+10	Local overflow allocation increment (indexed files)

R_NRD	+11	Number of record descriptors
R_RDP	+12	Pointer to record descriptors
R_SZ	18	Size of structure (in words); not a field in the block

NOTE

To refer to a key descriptor within a record descriptor, use offset tags generated by the Create File Key Descriptor Block Offsets ($CRKDB) macro call.

CREATE FILE RECORD DESCRIPTOR BLOCK OFFSETS ($CRRDB)

Associated Macro Calls:

Create File, Get File Information, Create File Parameter Structure Block Offsets, Get File Information, Parameter Structure Block Offsets

Structure:

Word	Fields	
0	Size of Record Descriptor Block (including this field)	
0	Record Type	
1	Number of key components	Reserved
2 3	Reserved	
4 5	Reserved	

Generated Offset Tags:
(Basic record descriptor)

Tag	Corresponding Offsets (in Words)	Entry Name
R_RDSZ	0	Size of record descriptor block (including this field)
R_RT	0	Record Type
R_NKC	+1	Number of key components

(for each key component within basic record descriptor)

Tag	Corresponding Offsets (in Words)	Entry Name
R_KTYP	0	Key type (first byte)
R_KLEN	0	Key length, in bytes (second byte)
R_KOFF	+1	Key offset in bytes

NOTE

This macro call has the same effect as the Get File Record Descriptor Block Offsets macro call.

CREATE GROUP ($CRGRP)

Function Code: 0D/03

Equivalent Command: Create Group (CG)

Define a new task group. Allocate and initialize the data structures required to control the task group within the specified memory pool. Create the lead task as described under the Create Task macro call.

FORMAT:

```
[label]   $CRGRP  [location of group id],
                  [location of memory pool id],
                  [location of base level],
                  [location of high logical resource number],
                  [location of high logical file number],
                  [location of root entry name address]
```

ARGUMENTS:

location of group id

Any address form valid for a data register; provides the group id of the new task group. The group id must be a 2-character (ASCII) name that does not have the $ character as its first character.

location of memory pool id

Any address form valid for a data register; provides the id of the memory pool to be used to satisfy all memory requests emanating from the created task group. The memory pool id consists of two ASCII characters that name a pool defined at system building. If this argument is omitted, the new task group uses the memory pool associated with the issuing task group.

location of base level

Any address form valid for a data register; provides the base priority level, relative to the system level, at which the lead task executes.

The base level of 0, if specified, is the next higher level above the last system priority level. The sum of the highest system physical level plus 1, and the base level of a group, and the relative level of a task within that group, must not exceed 62_{10}.

location of high logical resource number

Any address form valid for a data register; provides the highest logical resource number (LRN) that is used by any task in the task group. The LRN can be a value from 0 through FC (hexadecimal). If this argument is omitted, or if the value specified is less than the highest LRN used by the system task group, the system task group's LRN is used.

location of high logical file number

Any address form valid for a data register; provides the highest LFN to be used by any task in the task group. The LFN can be a value from 0 through FF (hexadecimal). If this argument is omitted, the value F is assumed. (Refer to the Associate File macro call.)

location of root entry name address

Any address form valid for an address register; provides the address of the root entry name string that specifies the pathname of the bound unit to be executed as the lead task. The bound unit pathname can have an optional suffix in the form of ?entry, where entry is the symbolic start address within the root segment. If this suffix is not given, the default start address (established at assembly or link time) is used. EC?ZXECL specifies the command processor as the lead task.

DESCRIPTION:

This macro call causes the initialization and allocation of all data structures used by the system to define and control the execution of a task group. It also causes the loading of the root segment of the lead task of the task group. It does not cause the system to activate any task within the task group.

NOTES

1. The system places in $R2 the group id supplied by argument 1. If this argument is omitted, the system assumes that $R2 contains the group id to be used.

2. The system places in $R4 the memory pool id supplied by argument 2. If this argument is omitted, the system assumes that $R4 is set to zero to indicate that the memory pool of the issuing task group should be used by the newly created task group.

3. The system places in $R5 the base priority level supplied by argument 3. If this argument is omitted, the system assumes that $R5 contains the base priority level to be used.

4. The system places in $R6 the high LRN value supplied by argument 4. If this argument is omitted, the system sets $R6 to zero to indicate that the value of the highest LRN created for the system task is to be used.

5. The system places in $R7 the high LFN value specified by argument 5. If this argument is omitted, the system sets $R7 to 15.

6. The system places in $B2 the address of the root entry name supplied by argument 6. If this argument is omitted, the system assumes that $B2 contains the address of the bound unit to be executed by the lead task.

7. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   - 0000 - No error
   - 0601 - Invalid memory size or memory pool
   - 0602 - Insufficient memory
   - 0804 - Group id in use
   - 0806 - Invalid group id
   - 0807 - Invalid memory pool id
   - 0808 - Invalid base level
   - 0809 - Invalid high LRN
   - 080A - Invalid high LFN
   - 080C - Unresolved start address
   - 0E02 - No memory available for nonswappable task
   - 160A - Insufficient memory in pool for this group

160B - Invalid overlay nesting

$R2 - Group id of created group.

Example:

In this example, a new task group is created with a group id of G1; the group uses memory pool P1 and has level 40 (decimal) assigned as a relative base level. Both the high LRN and high LFN are defaulted (only the number of LRNs equivalent to that configured for the system task group is available, and the highest logical file number available is 15 decimal). The task group's lead task begins execution at the entry point ENTRY1 of the bound unit PROG1, as found by application of the system search rules.

```
GROUP1     $CRGRP     ='G1',='P1',=40,,,!ROOT
                          .
                          .
                          .
ROOT       TEXT       'PROG1?ENTRY1Δ'
```

CREATE OVERLAY AREA TABLE ($CROAT)

Function Code: 07/0A

Equivalent Command: None

Create an overlay table (OAT) to be used for sharing floatable overlays; create in memory the overlay area described by this OAT.

FORMAT:

```
[label]  $CROAT  [location of OAT address],
                 [location of size of overlay area entry],
                 [location of number of overlay area
                  entries],
                 [location of bound unit index id]
```

ARGUMENTS:

location of OAT address

Any address form valid for an address register; provides the location into which the system places the address of the OAT.

location of size of overlay area entry

Any address form valid for a data registers; provides the location of a value specifying the number of words to be contained in each entry in this overlay area. This value should be equal to or greater than the size of the overlays to be placed in the area for loading.

location of number of overlay area entries

Any address form valid for a data register; provides a value specifying the number of entries in this overlay area. (The size of each entry is defined by argument 2.) The correct value for this argument depends on the number of overlays of this size used by the bound unit and the frequency of their release.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit from which this $CROAT call is issued and with which the created OAT is associated; used only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the primary (i.e, attaching) bound unit is 0.

DESCRIPTION:

The overlay area and overlay area table created by $CROAT permit the sharing of floatable overlays by tasks in the same task group. (See the System Concepts manual for information about overlay area and overlay area tables.)

The memory space for the overlay area created by this call is obtained from the memory pool in which the current bound unit is loaded. If the current bound unit is not sharable, memory is obtained from the pool associated with the group of the issuing task. If the current bound unit is sharable, memory is obtained from the system pool. If the current bound unit is in a swap pool, memory is obtained from the bound unit's segment, which dynamically expands to accomodate the overlay area.

Once created, an OAT is associated with the current bound unit by means of a field in the bound unit's bound unit descriptor (BUD) block. That field points to a queue of OATs created by the bound unit; OATs in the queue are ordered by ascending area size.

Before an OAT is created, the bound unit's OAT queue is searched for an OAT whose entry size is equal to that specified by argument 2. If such an OAT is found, no OAT is created by this call. The address of the existing or created OAT is returned to the location specified by argument 1.

Argument 3 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a mulit-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id of the primary (i.e., attaching) bound unit is 0. If not applicable, this argument is bypassed.

The Overlay Area, Reserve and Execute Overlay ($OVRSV) and Overlay Area, Release, Wait, and Recall ($OVRCL) macro calls each require that an OAT and overlay area be present; thus, each call must be preceded by $CROAT.

NOTES

1. The system returns the address of the OAT in $B4 and stores it in the memory location specified by argument 1. If argument 1 is omitted, the system stores the address only in $B4.

2. The system places in $R2 the size of the entry supplied by argument 2. If this argument is omitted, the system assumes that $R2 contains the correct size.

3. The system places in $R6 the number of entries supplied by argument 3. If this argument is omitted, the system assumes that $R6 contains the correct number.

4. The system places in $R7 the bound unit index id supplied by argument 4. If this argument is omitted, the system assumes that $R7 contains the correct number.

5. On return, $R1, $R2, $R6, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   - 0000 - No error
   - 0602 - Insufficient memory; user system area or segment
   - 0E02 - No memory available for non-swappable task
   - 1602 - Invalid argument (size or number of overlay areas)
   - 160A - Insufficient memory

   $R2 - Actual size of overlay area entry (if $R1 is 0000); for overlay entry in a segment, rounded up to nearest 256 words.

   $R6 - Actual number of overlay areas allocated to this area (if $R1 is 0000).

CREATE SEGMENT

<u>CREATE SEGMENT ($CRSEG)</u>

Function Code: 0C/0C

Equivalent Command: None

Create a segment in the address space of the issuing task; assign the segment to the initial bound unit of that task.

FORMAT:

```
[label]  $CRSEG   [location of segment access rights],
                  [location of segment size],
                  [location of segmented address]
```

ARGUMENTS:

location of segment access rights

Any address form valid for a data register; provides the access rights (read, write, and execute) for the segment to be created. Access rights are expressed as a string of six bits, which are used to specifiy the type of access as follows:

Bits	Access Type
0-1	Read
2-3	Write
4-5	Execute

Ring access is coded as follows:

Bit Values	Ring
00	3
01	2
10	1
11	0

location of segment size

Any address form valid for a double word data register (i.e., an address, or hexadecimal string if a constant); provides the segment's size, in words. The actual size of the created segment is the specified size rounded up to the next 256-word increment.

location of segmented address

Any addres form valid for an address register; provides the address of any word in the segment. When null is specified, the system selects a segment number that is consistent with that specified by argument 2 and with the availability of segment numbers to users.

DESCRIPTION:

This call enables the issuing task to create dynamically a segment of the size specified by argument 2. The created segment is added to the issuing task's address space.

The segment's address, specified by argument 3, must be that of an available user segment. User segments may be any of the fifteen large segments; however, the assignment of user segments, made by the system administrator, can vary from one installation to another.

$CRSEG is appropriately issued by a task running in a swap pool. If issued from a nonswap pool, the call is converted into a Get Memory function with the DENY argument.

After execution of $CRSEG, $B2 contains a pointer to the start of the created segment; $R6,$R7 contain the size of the created segment.

The <u>System Concepts</u> manual describes in detail ring and segment access, segment size, and segment numbers.

NOTES:

1. The system places in $R2 the segment's access rights value supplied by argument 1. When the argument is omitted, the system assumes that $R2 contains this value.

2. The system places in $R6 and $R7 the size of the segment supplied by argument 2. When the argument is omitted, the system assumes that $R6 and $R7 contain the segment size.

3. The system places in $B2 the address of any word in the segment, supplied by argument 3. When the argument is omitted, the system assumes that $B2 contains the segmented address. When argument 3 specifies zero, the system selects the segment number.

4. On return, $R1, $R6, $R7, and $B2 contain the following. (Contents of these registers are undefined for a return with an error.)

   $R1 - Return status code; one of the following:

   0000 - No error

   0602 - Memory unavailable

   0817 - Memory access violation; attempt to destroy an address (with the created segment), without the right to do so, of the following:

   - Sharable bound unit root
   - System segment
   - Non-user-created segment

   082E - Argument error:

   - Size exceeds 64K

   - Size inconsistent with the specified segment number

   $R6, $R7 - Actual size of created segment

   $B2 - Pointer to start (offset = 0) of created segment.

Example:

In this example, the requesting task creates a 2K-word segment and assigns it to the inital bound unit. Ring 3 has read and write access rights, but execute access is restricted to ring 0. The segment number of the created segment is C. On successful return to the issuing task, $B2 contains the address of the first word of the created segment; $R6 and $R7 contain the segment's size. The address and size of the segment are saved in SEG_A and SEG_S, respectively. When the task has finished using the segment, the segment is deleted with a Delete Segment ($DLSEG) macro call.

```
            $CRSEG      =B'0000110000000000';
                        =2048,
                        +$A
              .
              .
              .
$A                      Z'000C0000'
*
*    CHECK FOR ERROR OR INSUFFICIENT MEMORY
*
            BNEZ        NO_GO
*
*    SAVE THE SEGMENT'S ADDRESS AND SIZE
*
            STB         $B2, SEG_A
            SDI         SEG_S
              .
              .
              .
*
*    NOW DELETE THE SEGMENT
*
            $DLSEG      SEG_A
              .
              .
              .
SEG_A       DC          <$
SEG_S       DC          0B(31,0)
```

CREATE TASK ($CRTSK)

Function Code: 0C/02 (same bound unit),
0C/03 (different bound unit)

Equivalent Command: Create Task (CT)

Add the supplied task definition to the set of currently defined tasks within the task group of the issuing task.

FORMAT:

```
[label]  $CRTSK  [location of logical resource number],
                 [location of relative priority level],
                 [location of start address],
                 [location of root entry name address]
```

ARGUMENTS:

location of logical resource number

Any address form valid for a data register; provides the location of the logical resource number (LRN) by which the issuing task group can refer to the created task. The LRN (a value from 0 through 252) cannot exceed the value used as the high LRN in the Create Group macro call that created the group of which this task is a member. If the LRN value is set to -1, the system selects an available LRN, starting with the maximum, and returns it to the user in $R2.

location of relative priority level

Any address form valid for a data register; provides the location of the priority level, relative to the task group's base priority level, at which the created task is to execute. If this argument is omitted or is -1, the priority level used is that of the issuing task.

location of start address

Any address form valid for an address register; provides the location of the task start address when the newly created task is to execute in the same bound unit as the task that issued the Create Task macro call. (Function code 0C/02.) Either the location of the start address or the location of the root entry name address, but not both, can be specified.

location of root entry name address

Any address form valid for an address register; provides the address of the pathname of the bound unit root segment to be loaded for execution by the newly created task. The bound unit pathname can have an optional suffix in the form of ?entry, where entry is the symbolic start address within the root segment. If this suffix is not given, the default start address (established at link time) is used. (Function code 0C/03.) Either the location of the start address or the location of the root entry name address, but not both, can be specified.

DESCRIPTION:

This macro call causes the allocation and initialization of the data structures that define and control task execution. The call does not activate the task; the Request Task macro call is required for task activation.

One or more Create Task macro calls can be issued to create one or more tasks within a task group.

When a Create Task macro call is executed, the system builds a resource control table (RCT) and a task control block (TCB) for the created task. The address of the RCT is placed in the logical resource table (LRT) in association with the appropriate LRN.

If the new task is to execute the same bound unit as the issuing task, the count of tasks associated with the unit is incremented (function code 0C/02) to prevent premature reuse of memory containing the bound unit.

If the specified bound unit is not a sharable bound unit that is currently resident in memory, the root segment of the bound unit is loaded into memory belonging to the task group. If the specified bound unit is both sharable and currently resident, the count of tasks associated with the unit is incremented. (Function code 0C/03.)

NOTES

1. The system places in $R2 the LRN supplied by argument 1. If this argument is omitted, the system asssumes that $R2 contains the LRN for the created task.

2. The system places in $R6 the relative priority level supplied by argument 2. If this argument is omitted, the system sets $R6 to the relative priority level of the task issuing this macro call.

3. Arguments 3 and 4 are mutually exclusive. If both are supplied, argument 3 is used and a diagnostic is issued. Information derived from either argument is placed in $B2. If these arguments are omitted, the system assumes that $B2 contains the start address to be used.

4. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   - 0000 - No error
   - 01xx - Media error
   - 0209 - File or directory not found
   - 0602 - Memory unavailable
   - 0809 - LRN too large
   - 0813 - Referenced LRN already in use or invalid
   - 0827 - Invalid bound unit file format
   - 0830 - LRN not available
   - 082D - Group's available memory quota exceeded
   - 0E02 - No memory available for nonswapable task
   - 1604 - Unresolved symbolic start address
   - 160A - Insufficient memory
   - 1611 - Zero length overlay referenced
   - 1613 - Invalid pathname format
   - 1615 - Invalid bound unit format

   $R2 - LRN of created task.

Examples:

In this example, the Create Task macro call makes a task known as logical resource number 10 (decimal) of the issuing group. The task will execute at priority level 2 relative to the group's relative base level. The task will execute the procedures contained in the bound unit PROG10, as found by application of search rules, and enters the bound unit at entry point PROG10.

```
        $CRTSK    =10,=2,,!ROOT
         .
         .
         .
   ROOT TEXT    'PROG10 '
```

In this example, the Create Task macro call makes a task known as logical resource number 12 (decimal) of the issuing group. The task executes at the same priority level as the issuing task. The task executes the same bound unit as the issuing task and starts at the address represented by the label SSA.

```
        $CRTSK    =12,,,!SSA
         .
         .
         .
```

DEFER CHECKPOINT

DEFER CHECKPOINT ($DFCKP)

Function Code: 0C/19

Equivalent Command: None

Enable or disable the ability of the issuing task group to take new checkpoints.

FORMAT:

```
[label]    $DFCKP   [ {E} ]
                    [ {D} ]
```

ARGUMENT:

E
D

One of the following values is specifed to indicate whether the issuing task's ability to issue checkpoints is to be enabled or disabled:

[E] - Enable the issuing task's ability to issue new checkpoints.

[D] - Disable the issuing task's ability to issue new checkpoints.

DESCRIPTION:

This macro call allows the issuing task to enable or disable the ability to create new checkpoints. A count of Defer Checkpoint calls is kept. New checkpoints can be created only when an equal number of enable checkpoint calls have been issued. If this argument is omitted, the current content of $R2 is issued.

The macro calls associated with the checkpoint/restart facility are: Validate Checkpoint, Checkpoint, Restart, Defer Checkpoint, Checkpoint File.

NOTES

1. If E is specified for argument 1, $R2 is set to zero. If D is specified for argument 1, $R2 is set to one.

2. On return, $R1 contains one of the following return codes:

   0000 - No error

   082E - Argument error ($R2 is not equal to zero or one).

Example:

This example uses the Defer Checkpoint macro call to protect an area or procedure from being checkpointed. The area is bounded by a disable/enable checkpoint sequence.

```
        .
        .
        .
    $DFCKP    D          Disable checkpoints
        .
        .
        .
    procedure protected from checkpoints
        .
        .
        .
    $DFCKP    E          Enable checkpoints
        .
        .
        .
```

DEFER REQUEST ON HEAD ($DFRHD)

Function Code: 01/0D

Equivalent Command: None

Dequeue the currently dispatched task request and requeue it at the head of requests previously deferred at the specified priority.

FORMAT:

```
[label]   $DFRHD   [location of defer priority],
                   [location of task start address]
```

ARGUMENTS:

location of defer priority

Any address form valid for a data register; specifies the frame priority number at which the currently dispatched, dequeued request is deferred. Must be a value between +1 and +32,767.

location of task start address

Any address form valid for an address register; provides the address of the instruction to which execution of the issuing task jumps after the Defer Request on Head macro calling sequence. If task execution continues with the instruction immediately following the call (i.e., does not jump), the value of this argument must be null.

DESCRIPTION:

This macro call dequeues the currently dispatched request and requeues it at the head of requests previously deferred at the specified priority. If there is another active request in the issuing task's request queue, that request is dispatched; otherwise, after completing execution, the issuing task is suspended until it receives another request.

Following this call, task execution can either continue with the next instruction or jump to a new task start specified by argument 2 of the call.

NOTES

1. The system places in $R5 the defer priority specified by argument 1. If argument 1 is omitted, the system assumes that $R5 contains the defer priority.

2. The system places in $B4 the task start address supplied by argument 2. If argument 2 is omitted, the system assumes that $B4 contains either the task start address or a null value.

3. On return (assuming that after the currently dispatched request is dequeued, there is another active request in the issuing task's request queue), $B4, $B5, and $B7 contain the following information:

   $B4 - Address of the request block for the new request

   $B5 - Address of the system-supplied termination sequence, which consists of the following code:

   ```
   LDR    $R2,=$R1
   MCL
   DC     Z'0103'
   ```

   $B7 - Address of the request block's argument list, equivalent to (RB_address+2*$AF=2).

DEFER REQUEST ON TAIL

DEFER REQUEST ON TAIL ($DFRTL)

Function Code: 01/0C

Equivalent Command: None

Dequeue the currently dispatched task request and requeue it at the end of the queue of requests previously deferred under the specified priority.

FORMAT:

```
[label]   $DFRTL   [location of defer priority],
                   [location of task start address]
```

ARGUMENTS:

location of defer priority

Any address form valid for a data register; specifies the frame priority number at which the currently dispatched, dequeued request is deferred. Must be a value between +1 and +32,767.

location of task start address

Any address form valid for an address register; provides the address of the instruction to which execution of the issuing task jumps after the Defer Request on Tail calling sequence. If task execution continues with the instruction that immediately follows the call (i.e., does not jump), the value of this argument must be null.

DESCRIPTION:

This macro call dequeues the currently dispatched request and requeues it at the tail of requests previously deferred at the specified priority. If there is another active request in the issuing task's request queue, that request is dispatched; otherwise, after completing execution, the issuing task is suspended until it receives another request.

Following this call, task execution can either continue with the next instruction or jump to a new task start specified by argument 2 of the call.

NOTES

1. The system places in $R5 the defer priority specified by argument 1. If argument 1 is omitted, the system assumes that $R5 contains the defer priority.

2. The system places in $B4 the task start address supplied by argument 2. If argument 2 is omitted, the system assumes that $B4 contains either the task start address or a null value.

3. On return (assuming that after the currently dispatched request is dequeued, there is another active request in the issuing task's request queue), $B4, $B5, and $B7 contain the following information:

   $B4 - Address of the request block for the new request

   $B5 - Address of the system-supplied termination sequence, which consists of the following code:

   ```
   LDR     $R2,=$R1
   MCL
   DC      Z'0103'
   ```

   $B7 - Address of the request block's argument list, equivalent to (RB_address+2*$AF=2).

DEFINE SEMAPHORE ($DFSM)

Function Code: 06/04

Equivalent Command: None

Define a semaphore for the issuing task group; assign the semaphore an identifier and an initial value.

FORMAT:

```
[label]   $DFSM  [location of semaphore id],
                 [location of initial value of semaphore]
```

ARGUMENTS:

location of semaphore id

Any address form valid for a data register; provides the two ASCII characters that identify this semaphore.

location of initial value of semaphore

Any address form valid for a data register; provides the initial value to which the semaphore is set. This value specifies the number of simultaneous requests for the resource identified by the semaphore. If this argument is omitted, the initial value of the semaphore is set to one (one user at a time).

DESCRIPTION:

This macro call allows different tasks within the same task group to coordinate the use of a resource (such as a task code, a device, or a file). The semaphore acts as a gating mechanism that allows a requesting task to obtain the use of a resource if the value of its associated semaphore is positive.

When a semaphore is defined by a task, it is available only to tasks within the task group of the defining task. See "Semaphore Functions" in Section 2, Vol. I for a discussion of semaphores.

The 2-character semaphore id indicated by argument 1 is a system symbol used by the operating system to coordinate requests for the resource being controlled. The initial value indicated by argument 2 specifies the type of control to be exercised. If this value is 1, the resource can be accessed by only one task at a time. A value of 2 allows two users, a value of 3 allows three users, and so on.

NOTES

1. The system places in $R6 the semaphore id supplied by argument 1. If argument 1 is omitted, the system assumes that $R6 contains the id to be used.

2. The system places in $R2 the initial semaphore value supplied by argument 2. If this parameter is omitted, $R2 is set to one.

3. On return, $R1 and $R6 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0503 - Semaphore id previously defined in issuing task group

   $R6 - Semaphore id (as supplied).

Example:

In this example, the Define Semaphore macro calls define two semaphores named TH and LK.

TH is a semaphore having an initial value of 10. It controls the allocation of 10 identical nonsharable resources, such as magnetic tape drives, that are called "resources" in this example. Any task requiring a resource does a P-op test (see reserve semaphore) on this semaphore. If no resources are available at the moment, the task is suspended until a resource becomes available. When a task finishes using a resource, it does a V-op (see release semaphore), thereby making the resource available for use by other tasks. If any other task is waiting for this semaphore when the V-op is done, the task that has waited the longest is awakened.

LK is a semaphore that has an initial value of one. It controls access to the free resource list by serving as a lock. After a task has reserved the right to use a resource by performing the P-op on the TH semaphore, as described above, the task unlinks (the description of) a particular resource from the free resource list. Upon entering a section where it examines or modifies the free resource list, the task does a P-op on the LK semaphore, thus ensuring the integrity of this data base. After it stops using this data base, the task does a V-op on the LK semaphore.

When the task finishes using the resource, it returns the resource by doing a P-op on LK, linking (the description of) the resource being returned into the free resource list, doing a V-op on LK, and then doing a V-op on TH.

```
*
*   DEFINE SEMAPHORES TO CONTROL RESOURCES
*
                    $DFSM    ='TH',=10
                    $DFSM    ='LK'
                             .
                             .
*                            .

*   ROUTINE TO GET A RESOURCE
*
*   FIRST GET RIGHTS TO TAKE A RESOURCE

                     $RSVSM   ='TH'
*
*   NOW LOCK THE FREE RESOURCE LIST
*
                     $RSVSM   ='LK'
*
*   TAKE A RESOURCE FROM THE FREE RESOURCE LIST

*                            .
                             .
                             .
*
*   THEN UNLOCK THE FREE RESOURCE LIST
*
                     $RLSM    ='LK'
*
*   END OF ROUTINE TO GET A RESOURCE
*
*   ROUTINE TO RETURN A RESOURCE
*
*   FIRST LOCK THE FREE RESOURCE LIST
*
                     $RSVSM   ='LK'
```

```
*  NOW LINK THE RESOURCE BACK INTO THE FREE RESOURCE LIST
*                               .
                                .
                                .
*
*  THEN UNLOCK THE FREE RESOURCE LIST
*
                   $RLSM    ='LK'
*
*  FINALLY RELEASE THE RESOURCE
*
                   $RLSM    ='TH'
*
*  END OF ROUTINE TO RETURN A RESOURCE
*
```

DELETE DIRECTORY ($DLDIR)

Function Code: 10/A5

Equivalent Command: Delete Directory (DD)

Delete a previously created directory from the system; remove all of the directory's attributes, including its name, from the immediately superior directory that describes it, and release all space allocated to the directory. This function is usually done outside program execution.

FORMAT:

```
[label]   $DLDIR   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of a parameter structure that must contain the following entry.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the directory to be released.

DESCRIPTION:

This macro call, in effect, reverses the Create Directory action, provided the directory has no subordinate directories or files (i.e., if the directory to be deleted contains a subordinate directory or file, it is not deleted and an error code is returned). In addition, if it is currently the working directory in any task group, the directory cannot be deleted.

NOTES

1. If the argument is coded, the system loads the address of the parameter structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - Successful completion

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - This function requires a pathname to be specified

   0205 - Invalid argument

   0209 - Named directory not found

   020C - Volume not found

   0213 - Cannot provide requested concurrency

   0220 - Directory not empty

   0222 - Pathname cannot be expanded; no working directory

   0225 - Not enough system memory for buffers or structures

   0226 - Not enough user memory for buffers or structures

   0228 - Invalid file type (not a directory)

   022C - Access control list (ACL) violation.

Example:

In this example, the Delete Directory macro call deletes the directory created in the Create Directory example (i.e., SUBINDEX.A). The system uses the first entry to identify the directory to be deleted. The Delete Directory macro call is coded as:

```
SUBDIR    DC       <DIRPTH
DIRPTH    DC       '^VOL03>SUBINDEX.A
          $DLDIR   !SUBDIR
```

DELETE FILE ($DLFIL)

Function Code: 10/35

Equivalent Command: Delete File (DL)

Delete a previously created file from the system. All the file's attributes, including its name, are removed from the directory that describes it, and all space allocated to the file is released. The file to be deleted is identified by supplying either a logical file number (LFN) or a pathname. This function is usually done outside program execution.

FORMAT:

[label] $DLFIL [argument structure address]

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of an argument structure that must contain the following entries in the order shown.

logical file number

A 2-byte LFN used to refer to the file; must be a binary number in the range 0 through 255, or blank (which indicates that an LFN is not specified).

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the directory in the file hierarchy in which the file to be released is found (as well as the name of the file itself).

Zeros in this entry indicate that a pathname is not specified.

DESCRIPTION:

This macro call, in effect, reverses the Create File action, provided the file is neither open in this task group nor reserved by another task group. In the former case, a return status code of 0208 is loaded in $R1; in the latter case, a return status of 0213 is loaded in $R1.

The file to be deleted can be specified by (1) an LFN only or (2) a pathname only. If only an LFN is specified, the file must have been created or reserved (through a Create File or Get File macro call, or equivalent command) with that LFN.

For files other than disk files, the Delete File function is equivalent to the Remove File function.

A restorable file (i.e., one created/modified with the -RESTORE attribute) cannot be deleted unless the system's after-image journal is open.

A disk file that contains accounting information cannot be deleted unless its retention period has expired.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - The LFN and pathname both were not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0208 - LFN or file currently open in same task group

   0209 - Named file or directory not found

   020C - Volume not found

   0210 - LFN conflict

   0213 - File in use by another task group

0222 - Pathname cannot be expanded; no working directory

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

0228 - Invalid file type (a directory)

022C - Access control list (ACL) violation

020E - File has not expired

0260 - Journal file is not open.

Example:

In this example, the macro call deletes the file created in the Create File ($CRFIL) macro call example. To do this, it references the same argument structure as the Create File macro call; the system, in turn, uses the first two entries to identify the file to be deleted. The Delete File macro call is coded as:

```
$DLFIL    !FILE_A
```

DELETE GROUP ($DLGRP)

Function Code: 0D/04

Equivalent Command: Delete Group (DG)

Mark the task group as eligible for deletion when it becomes dormant; then return all allocated memory to the associated memory pool.

FORMAT:

```
[label]   $DLGRP   [location of group id]
```

ARGUMENT:

location of group id

Any address form valid for a data register; provides the group id of the task group to be deleted. This task group must have previously been created by a Create Group macro call. If this argument is omitted, the issuing task group is deleted.

DESCRIPTION:

This macro call removes all data structures built by the Create Group macro call issued with this group id, when the group becomes dormant. No further Request Group macro calls can be issued for this task group once the Delete Group macro call has been issued.

When a task group is deleted, the memory occupied by the data structures defining the group, and any memory associated with the execution of the group, is returned to the appropriate memory pool.

The Delete Group macro call takes effect immediately if the task group is dormant when the command is issued. If the task group is active (i.e., its code is being executed and/or there are requests in its request queue), the Delete Group macro call takes effect when execution terminates and no requests remain in the queue.

NOTES

1. The system places in $R2 the group id supplied by argument 1. If this argument is omitted, $R2 is set to zero to designate that the issuing task group is to be deleted.

2. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - Delete task group status set
   0806 - Invalid group id

   $R2 - Group id of deleted task group.

Example:

In this example, the Delete Group macro call causes the task group in which the macro call is executed to be deleted when the group's tasks are all terminated and there are no queued group requests.

```
NOABA     $DLGRP
```

DELETE OVERLAY AREA TABLE

DELETE OVERLAY AREA TABLE ($DLOAT)

Function Code: 07/0D

Equivalent Command: None

Delete the named overlay area table (OAT).

FORMAT:

```
[label]   $DLOAT   [location of overlay area table address]
```

ARGUMENT:

location of overlay area table address

Any address form valid for an address register; provides the address of the OAT to be deleted.

DESCRIPTION:

This macro call deletes the specified OAT, and returns its memory space to the task group's memory pool. Deletion occurs only when all overlays in the overlay areas are inactive (i.e., have no users attached to them).

NOTES

1. The system places in $B4 the address of the OAT to be deleted, supplied in the argument. When the argument is omitted, the system assumes that $B4 contains that address.

2. On return, $R1 contains one of the following:

   0000 - No error

   0602 - Memory not available to release overlay areas

   1610 - Named OAT not found in the queue

   1618 - OAT cannot be deleted; overlay areas are active.

DELETE RECORD ($DLREC)

Function Code: 11/30 (current), 11/31 (key)

Equivalent Command: None

Remove the specified logical record from the file; valid for all file organizations except fixed-relative without deletable records, tape-resident sequential files, and device files.

FORMAT:

```
[label]   $DLREC   [fib address]   [{,CURRENT}]
                                   [{,KEY    }]
```

ARGUMENTS:

fib address

Any address form valid for an address register; provides the location of the file information block (FIB).

{CURRENT}
{CUR}

Indicates that the last record read by means of the Read Record macro call (with read next or read with key mode specified) is to be deleted. (This is the default value for this macro call.) The user must code the following FIB entry:

logical file number

KEY

Indicates that the record identified by the key value pointed to by the FIB is to be deleted. The user must code the following FIB entries:

logical file number
input key pointer
input key format

DESCRIPTION:

Before this macro call can be executed, the file must have been opened (see the Open File macro call) with a program view word that allows access through data management (bit 0 is zero) and allows delete operations (bit 4 is one). The file must have been reserved (see Get File macro call) with write access concurrency (type 3, 4, or 5). In addition, execution of this macro call has no effect on the next read or write pointer (i.e., it can be issued between a Read Next Record and Write Next Record macro call without disturbing the sequence of the records being read or written).

The Delete Record macro call does not apply to fixed-relative files with nondeletable records, tape files, and device files.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets macro call.

NOTES

1. If the argument is coded, the system loads the address of the FIB into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. None of the out values in the FIB are set by this macro call.

3. On return, $R1 contains one of the following status codes:

   0000 - No error
   01xx - Physical I/O error
   0203 - Invalid function
   0205 - Invalid argument
   0206 - Unknown or invalid LFN
   0207 - LFN not open
   020A - Address out of file
   020B - Invalid extent description information
   020E - Record not found
   0217 - Access violation
   0219 - No current record pointer
   021E - Key length or location error
   022A - Record lock area overflow
   022B - Record deadlock occurred
   022F - Unknown or invalid record type
   0237 - Invalid record or control interval format
   023A - Recovery file I/O error
   0263 - Journal file I/O error.

Example:

The macro call in this example identifies the FIB that is described under "Assumptions for File System Examples" in Appendix A. The Delete Record macro call indicates that the current record is to be deleted; it is assumed that the file is open and that a Read Record ($RDREC) macro call immediately precedes the Delete Record macro call. The macro call is:

```
    $DLREC    !MYFIB,CURRENT
```

The FIB identified by the address in the first argument is as defined in the example for the Open File macro call.

DELETE SEGMENT

DELETE SEGMENT ($DLSEG)

Function Code: 0C/0D

Equivalent Command: None

Delete the specified segment.

FORMAT:

```
[label]  $DLSEG  [location of segmented address]
```

ARGUMENT:

location of segmented address

Any address form valid for an address register; provides the location of any word within the segment to be deleted.

DESCRIPTION:

This macro call causes a previously created segment (identified by the argument) to be deleted. The segment may have been created at link time or created dynamically by the Create Segment ($CRSEG) macro call. When the segment is deleted, memory assigned to it is returned to the task group's memory pool.

The following segments cannot be deleted:

- Root segment of a sharable bound unit
- Protected segments (e.g., group system area segment, group work area segment, overlay area table, and/or system segments)
- Segments to which the user does not have write/execute access rights in the user ring.

NOTES

1. The system places in $B2 the address of a word within the segment to be deleted, supplied in the argument. When the argument is omitted, the system assumes that $B2 contains that address.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   0602 - Memory unavailable

   0817 - Memory access violation; cannot destroy addresses of:

   - System segment
   - Root of sharable bound unit
   - Segment to which user does not have access.

Example:

See the example for the Create Segment ($CRSEG) macro call.

DELETE SEMAPHORE

DELETE SEMAPHORE ($DLSM)

Function Code: 06/07

Equivalent Command: None

Delete a counting semaphore that is currently defined for the task group issuing this call.

FORMAT:

```
[label]   $DLSM   [location of semaphore id]
```

ARGUMENT:

location of semaphore id

Any address form valid for a data register; provides the semaphore id, as two ASCII characters, of the semaphore to be deleted.

DESCRIPTION:

This macro call deletes a counting semaphore that was previously defined for the issuing task group with a Define Semaphore macro call.

The semaphore will be deleted only when there are no tasks waiting for the resource controlled by the semaphore (see Reserve Semaphore macro call). If tasks are waiting, a return to the issuing task results and $R1 contains a 0504 status code. When there are no longer any tasks waiting on the semaphore, the Delete Semaphore macro call must be reissued.

When the semaphore is deleted, all system references to it are removed. Any attempt to use it results in a return to the issuing task, with status code 0502 in $R1.

NOTES

1. The system places in $R6 the semaphore id supplied by the argument. When the argument is omitted, the system assumes that $R6 contains the id to be used.

2. On return, registers $R1 and $R6 contain the following:

   $R1 - Return status; one of the following:

   0000 - No error

   0502 - Semaphore not defined

   0504 - Semaphore request canceled

   0506 - Semaphore is currently active and cannot be deleted.

   $R6 - Semaphore id (as supplied).

Example:

The issuing task group requests that semaphores TH and LK (as defined for the example given in the Define Semaphore ($DFSM) macro call) be deleted.

```
DLSAA     $DLSM     ='TH'
          CMR       $R1,=Z'0506'
          BE        TH_BSY
DLSBB     $DLSM     ='LK'
          CMR       $R1,=Z'0506'
          BE        LK_BSY
            .
```

DELETE TASK ($DLTSK)

Function Code: 0C/04

Equivalent Command: Delete Task (DT)

Delete the definition of a task from the task group of which the task issuing this macro call is a member.

FORMAT:

```
[label]   $DLTSK   [location of logical resource number]
```

ARGUMENT:

location of logical resource number

Any address form valid for a data register; provides the location of the logical resource number (LRN) of the task to be deleted. The LRN (a value from 0 through 252) must have been specified in a previously issued Create Task macro call. If this argument is omitted, the task issuing the macro call is deleted.

DESCRIPTION:

This macro call removes the data structures constructed by the Create Task macro call that was issued with the specified LRN.

If the task is executing, the macro call causes its definition to be deleted when the task next issues a Terminate Macro call that has no request blocks in its request queue. No further Request Task macro calls can be issued for this task after the Delete Task macro call has been issued.

If the task is not executing and there are no outstanding requests for it, its definition is deleted immediately. When the task is deleted, the memory occupied by its data structures is returned to the appropriate memory pool. The Delete Task function operates asynchronously. The issuing task does not wait until the referenced task is deleted.

NOTES

1. The system places in $R2 the LRN specified by argument 1. If this argument is omitted, $R2 is set to -1 to denote that the task issuing the macro call is to be deleted.

2. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0802 - Invalid LRN

   $R2 - LRN of deleted task.

Example:

In this example, the Delete Task macro call causes the task known as logical resource number 10 (decimal) within the issuing task's task group to be deleted. If the Delete Task macro call shown in this example has been executed in the same task group as the Create Task macro call used in the first example of the Create Task Macro call description, the task created by that example is deleted.

```
    DLE_AA     $DLTSK     =10
```

DEQUEUE AND POST

DEQUEUE AND POST ($DQPST)

Function Code: 01/0B

Equivalent Command: None

Dequeue the currently dispatched task request and post the specified completion status.

FORMAT:

[label] $DQPST [location of completion status]

ARGUMENT:

location of completion status

Any address form valid for a data register; provides the status of the dequeued request. The user may select any status code as the value of this argument.

DESCRIPTION:

This macro call dequeues the currently dispatched request and posts its completion status. The issuing task immediately continues execution at the instruction following the call, without dispatching another request.

NOTES

1. The system places in $R2 the completion status code specified by the argument. If the argument is omitted, the system assumes that $R2 contains the completion status code.

2. On return, $R1 contains the folowing information:

   0000 - Request successfully dequeued and posted
   0814 - No currently dispatched request exists.

DISABLE USER TRAP ($DSTRP)

Function Code: 0A/02

Equivalent Command: None

Disable the handling of the specified trap for the issuing task.

FORMAT:

```
[label]   $DSTRP   [location of trap number]
```

ARGUMENT:

location of trap number

Any address form valid for a data register; provides the trap number (0 through 63, decimal) of the trap to be disabled. A value of -1 designates that all traps are to be disabled. The trap number must have been specified in an Enable User Trap macro call.

DESCRIPTION:

This macro call disables the hardware trap vector specified by argument 1. All subsequent occurrences of the specified trap are handled by the system's default trap handling routine until an Enable User Trap macro call is later issued for the trap. (Appendix A, Vol. I describes trap handling.)

NOTES

1. The system places in $R2 the trap number of the trap to be disabled, supplied by argument 1. If this argument is omitted, the system assumes that $R2 contains the binary number of the trap to be disabled.

2. On return, $R1 and $R2 contain the following information:

$R1 - Return status:

0000 - No error

0342 - Invalid trap number

0343 - A previously signalled trap is still pending.

$R2 - Trap number supplied in macro call.

Example:

See the example given for the Trap Handler Connect macro call.

DISSOCIATE FILE ($DSFIL)

Function Code: 10/15

Equivalent Command: Dissociate (DISSOC)

Dissociate a previously associated logical file number (LFN) from a pathname. This dissociation is typically done outside program execution.

FORMAT:

```
[label]   $DSFIL   [parameter structure address]
```

ARGUMENT:

parameter structure address

Any address form valid for an address register; provides the location of an argument structure that must contain the following entry.

logical file number

A 2-byte LFN used to refer to the pathname; must be a binary number in the range 0 through 255.

DESCRIPTION:

This macro call breaks the logical connection between the specified LFN and its previously associated pathname (see the Associate File macro call). It does not remove the file from the task group (see the Remove File macro call).

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0205 - Invalid argument
   0206 - Unknown or invalid LFN.

Example:

In this example, the macro call identifies the same argument structure used in the Associate File macro call described earlier (i.e., FILE_A). The effect of the Dissociate File macro call is to remove the logical connection betwee the LFN and the pathname IDX01, as established by the Associate File macro call.

```
FILE_A     DC          5           LFN5
           $DSFIL      !FILE_A
```

ENABLE USER TRAP ($ENTRP)

Function Code: 0A/01

Equivalent Command: None

Enable a specified user trap for the issuing task.

FORMAT:

```
[label]   $ENTRP   [location of trap number]
```

ARGUMENT:

location of trap number

Any address form valid for a data register; provides the trap number of the trap to be enabled. The trap number is a decimal value from 0 through 63, or a value of -1. A -1 value designates that all user traps are to be enabled.

DESCRIPTION:

This macro call causes a specific hardware trap vector, whose number is derived from argument 1, to be enabled. All subsequent occurrences of the specified trap cause control to be transferred to a previously established trap handling routine for the task (see the Trap Handler Connect macro call).

When the task group's general trap handling routine is entered, $R3 contains the trap number assigned to the event that caused the entry to the routine. $B3 contains the location of the trap save area. The j-mode bit in the I-register has been set off. All other registers are unchanged. An return from trap (RTT) instruction is executed to return from the task's trap handler. (See Appendix A, Volume I for more information about trap handling.)

NOTES

1. The system places in $R2 the trap number of the trap to be enabled, supplied by argument 1. If this argument is omitted, the system assumes that $R2 contains the binary number of the trap to be enabled.

2. On return, $R1 and $R2 contain the folowing information:

   $R1 - Return status; one of the following:

   0000 - No error

   0341 - Trap handler entry not connected

   0342 - Invalid trap number (requested trap not a user class trap)

   $R2 - Trap number supplied in macro call.

3. This macro call is required in order to enable a software simulated trap in a task that the user interrupts with the break key function, and for which a PI or UW break response is entered.

Example:

See the example given for the Trap Handler Connect ($TRPHD) macro call.

ENTRY POINT IDENTIFICATION ($ENTID)

Function Code: 14/07

Equivalent Command: None

Return the address or value corresponding to a symbolic name that is defined in the bound unit currently executed by the issuing task or in a bound unit permanently resident in memory. The name must have been declared at link time by an EDEF statement.

FORMAT:

```
[label]  $ENTID  [location of symbolic name field address],
                 [location of id type],
                 [location of bound unit index id]
```

ARGUMENT:

location of symbolic name field address

Any address form valid for a data register; provides the address of an aligned character string that contains the symbolic name.

location of id type

Any address form valid for a data register; specifies whether the information to be retured is an entry point address or an overlay number. Possible values for the argument are 'A' (signifying "address") or 'V' (signifying "value").

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit currently executed by the issuing task; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

The call returns to the issuing task, in $B2, the entry point address or, in $R2, the overlay id corresponding to the symbolic name specified in the macro call.

Argument 3 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id number of the initial bound unit is 0. If not applicable, this argument is bypassed.

If argument 3 is applicable, $ENTID resolves the specified symbolic name by searching, in the order given, any attached, currently executing bound unit; the primary bound unit; and bound units made permanently resident in memory by a CLM LBDU statement. If argument 3 is not applicable, the currently executing bound unit and then memory resident bound units are searched.

NOTES

1. The system places in $B4 the address of the symbolic name field supplied by argument 1. When this argument is ommitted, the system assumes that $B4 contains the field's address.

2. If 'A' (address) is specified in argument 2, $R6 is loaded with 0. If 'V' (value) is specified in argument 2, $R6 is loaded with -1. If argument 2 is omitted, the system assumes that $R6 contains the id type.

3. The bound unit index id supplied by argument 3 is placed in $R7. If argument 3 is omitted, the system assumes that $R7 contains the bound unit index id.

4. On return, $R1, $R2, and $B2 contain the following:

   $R1 - Return status, one of:

   0000 - No error

   080C - Symbolic name not found; unresolved symbolic start address

   0817 - Memory access violation

   $R2 - Value definition (if $R6 =-1 on input)

   $B2 - Entry point address corresponding to the specified symbolic name (if $R6=0 on input).

Example:

The issuing task obtains the entry point address corresponding to the symbolic name ENTRY1. The address is returned to $B2, not stored in memory.

```
          $ENTID    !ENTNAM,='V'
             .
             .
             .
ENTNAM    TEXT      'ENTRY1'
```

ERROR LOGGING, END

ERROR LOGGING, END ($ELEND)

Function Code: 02/09

Equivalent Command: STOP_ELOG

Terminate the error logging function for the named device and place current logging information from the system's logging table into the user's logging table.

FORMAT:

```
[label]   $ELEND   [location of device-name],
                   [address of user's error logging table]
```

ARGUMENT:

location of device-name

Any address form valid for an address register; provides the address of the device-name for the peripheral (noncommuncations) device for which the logging function is to be terminated.

address of user's logging table

Any address form valid for an address register; provides the address of error logging table previously generated and initialized by the user. (See Table 2-3 in the discussion of the Error Logging, Start macro call.)

DESCRIPTION:

This call terminates the error logging function previously activated for this device. The system transfers logging information values from the system's logging table into the user's logging table. The information transferred is shown in Table 2-3, under the description of Error Logging Start ($ELST).

The device name specified by argument 1 must have been previously specified with the Error Logging, Start macro call that activated error logging for that device.

NOTES

1. When argument 1 is specified, the system places the location of the device-name in $B2. If this argument is omitted, the system assumes that $B2 contains a pointer to the device-name.

2. When argument 2 is specified, the system places the address of the user's logging table in $B4. When this argument is omitted, the system assumes that $B4 contains a pointer to that table.

3. On return, $R1 contains one of the following status codes:

   0000 - Error logging terminated successfully

   3B01 - Invalid argument (1 or 2)

   3B02 - Named device is nonexistent

   3B05 - Logging function for this device is not active

   3B08 - Invalid function code

   3B0A - Device-name refers to a communications device. Macro call cannot be executed.

ERROR LOGGING INFORMATION, EXCHANGE ($ELEX)

Function Code: 02/07

Equivalent Command: None

Verify, then save the values in the user's error logging table; transfer current logging values from system's error logging table to user's error logging table; move the saved user-supplied error logging values into the system's logging table.

FORMAT:

```
[label]   $ELEX   [location of device-name],
                  [address of user's error logging table]
```

ARGUMENTS:

location of device-name

Any address form valid for an address register; provides the address of the device-name (previously coded in the Error Logging Start macro call for this device) for the device whose error logging values are to be exchanged.

address of user's logging table

Any address form valid for an address register; provides the address of the previously generated user's error logging table. Table 2-3, found under the Error Logging, Start macro call, defines the error logging table, the first part of which the user must build and initialize before issuing any Error Logging macro call.

DESCRIPTION:

Like Error Logging Information Get, this macro call transfers information from the system's error logging table to the user's, about (1) the current status of the system's logging table up to the time of the macro call, and about (2) the last error that occurred. (See Table 2-3).

In addition, this macro call transfers to the system's error logging table new values for the first six items (words seven through twelve) of the user's error logging table. New values are those entered by the user since the last execution of Error Logging, Start for the named device.

During execution of this macro call, the system (1) checks the new values of the user's error logging table for errors, and if they are correct, saves those values; (2) executes an Error Logging Information, Get macro call to move current values from the system's logging table to the user's logging table; and (3) moves and stores in the system's logging table the new logging values verified and saved from the user's logging table, thus replacing the previous values in the system's logging table. History counters in the system's logging table are reset to zero.

The device name specified by argument 1 must have been previously specified with the Error Logging, Start macro call that activated error logging for that device.

NOTES

1. When argument 1 is specified, the system places the location of the device name in $B2. If the argument is omitted, the system assumes that $B2 contains a pointer to the device-name.

2. When argument 2 is specified, the system places the address of the user's logging table in $B4. When the argument is omitted, the system assumes that $B4 contains a pointer to that table.

3. On return, $R1 contains one of the following status codes:

   0000 - Error logging information successfully exchanged

   3B01 - Invalid argument (1 or 2)

   3B02 - Named device is nonexistent

   3B03 - Invalid value specified for minimum number of I/O orders

   3B05 - Logging function for this device is not active

   3B06 - Invalid value specified for threshold

   3B07 - Invalid initial value for I/O order counter or device error counter

   3B08 - Invalid function code

   3B0A - Device name refers to communications device; macro call cannot be executed.

ERROR LOGGING INFORMATION, GET

ERROR LOGGING INFORMATION, GET ($ELGT)

Function Code: 02/08

Equivalent Command: None

Retrieve current logging information values for the named device from the system's error logging table; place them in the user's error logging table.

FORMAT:

```
[label]    $ELGT    [location of device-name],
                    [address of user's error logging table]
```

ARGUMENTS:

location of device-name

Any address form valid for an address register; provides the address of the device-name (previously coded in an Error Logging, Start macro call for this device) for the device whose error logging error information is to be transferred.

address of user's logging table

Any address form valid for an address register; provides the address of the previously generated user's error logging table (see Table 2-3 in the discussion of the Error Logging, Start macro call).

DESCRIPTION:

This macro call transfers current error logging information values for the named device from the system's error logging table to the user's error logging table. Error logging must have been previously activated for the device. Only those items in the system's logging table that have corresponding entries in the user's logging table are transferred.

The device name specified by argument 1 must have been previously specified with the Error Logging, Start macro call that activated error logging for that device.

NOTES

1. When argument 1 is specified, the system places the location of the device-name in $B2. If the argument is omitted, the system assumes that $B2 contains a pointer to the device-name.

2. When argument 2 is specified, the system places the address of the user's logging table in $B4. When the argument is omitted, the system assumes that $B4 contains a pointer to that table.

3. On return, $R1 contains one of the following status codes:

   0000 - Error logging values successfully transferred

   3B01 - Invalid argument (1 or 2)

   3B02 - Named device is nonexistent

   3B05 - Logging function for this device is not active

   3B08 - Invalid function code

   3B0A - Device-name refers to a communications device; macro call cannot be executed.

ERROR LOGGING, START

ERROR LOGGING, START ($ELST)

Function Code: 02/05

Equivalent Command: START_ELOG

Activate error logging for the named device.

FORMAT:

```
[label]   $ELST   [location of device-name],
                  [address of user's error logging table]
```

ARGUMENTS:

location of device-name

Any address form valid for an address register; provides the address of the device-name (designated at system building) for the peripheral (noncommunications) device to be monitored. Device name can have up to 12 ASCII characters.

address of user's logging table

Any address form valid for an address register; provides the address of the user's error logging table, which must have been previously generated.

DESCRIPTION:

This macro call starts error logging for the named device and maintains error logging information in memory. The call (1) allocates a block of system memory for the system's logging table and (2) checks parameters in words 7 through 12 of the user's logging table and stores the values in the system's logging table in memory.

Before this macro call is issued, the user must build and initialize a user logging table, either hand coding it or generating it by means of the Error Logging Table ($ELOG) macro call. In this table, the user supplies values for threshold ratio, minimum orders, initial orders, and initial errors. To contain these values, the table must be 14 words long, which is the minimum length required by Error Logging, Start. If, however, the table is to recieve error logging data (returned from the system erorr logging table by subsequent macro calls), the table must be 59 words long. The format of the user error logging table is shown in Table 2-3.

After execution of this call, system increments the I/O counter whenver an I/O order is issued. When there is a device error, the system increments the device error counter. When the specified number of I/O orders is processed, the system checks the error threshold ratio. If the value is exceeded, the system sends a message to the operator and resets the system's error logging table for this device.

The logging table is reset under any of the following conditions:

- Designated error threshold ratio exceeded.
- Either the I/O order counter or device error counter overflowed.
- Error Logging Information, Exchange macro call is executed.

When either of the first two occurs, the current value of the I/O order and device error counters are added to the history values in the system's error logging table. (These history values may be later delivered to corresponding history areas in the user's logging table (see Table 2-3)). If there is overflow in the addition, these counters are reset to zero, but the error threshold and I/O order minimum values are retained. When Error Logging Information, Exchange executed occurs, the items in the system's logging table are reinitialized from the new values supplied in the user's logging table.

Once initiated by the user, the error logging routine is called by the system for (1) incorrectable hardware errors, (2) correctable hardware errors, (3) unsuccessful I/O operations, and (4) I/O operations that were successful only after retries. For tape and all disk devices, retries may be software-initiated. For the mass storage unit only, retries may be hardware-initiated. The right byte of the I/O status word (words 46 and 54 of the table) contains the number of software retries for a successful I/O operation or the number eight, indicating an unsuccessful I/O operation after the maximum of eight retries. For some devices, the left byte of the I/O status word contains the number of hardware retries that preceded a successful I/O operation.

NOTES

1. When argument 1 is specified, the system places the location of the device-name in $B2. If the argument is omitted, the system assumes that $B2 contains a pointer to the device-name.

2. When argument 2 is specified, the system places the address of the user's logging table in \$B4. When the argument is omitted, the system assumes that \$B4 contains a pointer to that table.

3. The device-name must be that of a noncommunications peripheral device (which cannot be connected to an MLCP).

4. On return, \$R1 contains one of the following status codes:

   0000 - Error logging activated successfully

   3B01 - Invalid argument 1 or 2

   3B02 - Named device is nonexistent

   3B03 - Invalid value specified for minimum number of I/O orders

   3B06 - Invalid value specified for threshold

   3B07 - Invalid initial value for I/O order counter or device error counter

   3B08 - Invalid function code

   3B09 - Insufficient system memory for logging table

   3B0A - Device-name refers to communications device; logging cannot be activated.

5. The user can move the latest error logging information values from the system's logging table to the user's logging table with one of the following macro calls: Error Logging Information, Get; Error Logging Information, Exchange; or Error Logging, End macro call.

Table 2-3. User Error Logging Table

ERROR LOGGING TABLE VALUES		
User-Specified in $ELST and $ELEX Macro Calls		
Word(s)	Value (Signed Binary)	Function
0-6	Reserved for system	N/A
7, 8	2-word integer ≥ 0; normally initialized; to 0	Counter for number of incorrectable I/O order errors
9, 10	2-word integer ≥ 0; normally initialized to 0	Counter for number of correctable I/O order errors
11	1-word integer ≥ 0; normally initialized to 0	Counter for number of incorrectable device read errors
12	1-word integer ≥ 0; normally initialized to 0	Counter for number of correctable device read errors
13	1-word integer, from 0 through 1000, represented as a fraction in thousandths; i.e., DC 500 means .500	Error threshold ratio; ratio of DC 10 (i.e., 1% suggested for magnetic tape)
14	1-word integer ≥ 0	Minimum number of I/O orders processed before this threshold is checked
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls		
0	Two characters: a D followed by a blank	D signifies that this refers to a device (as contrasted with a volume)
1-6	12 characters	Device name
7, 8	2-word integer	Number of incorrectable I/O order errors
9, 10	2-word integer	Number of correctable I/O order errors

Table 2-3 (cont). User Error Logging Table

ERROR LOGGING TABLE VALUES		
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls (cont)		
Word(s)	Value (Signed Binary)	Function
11	1-word integer	Number of incorrectable device read errors
12	1-word integer	Number of correctable device read errors
13	1-word integer	Error threshold ratio
14	1-word integer	Number of I/O orders processed before threshold ratio is checked
15	Not used	N/A
16	1-word integer, either 0 or 1; index to words 43-50 and 51-58	Indicator to information about the two most recent errors, which is shown below in words 43-50 and words 51-58 0 = Most recent information is in words 43-50 1 = Most recent information is in words 51-58
17	Overflow indicator, ASCII 0 or 1	
	Left byte:	0 = No overflow 1 = Overflow, exceeds counter capacity of words 7, 8 (number of I/O orders)
	Right byte:	0 = No overflow 1 = Overflow, exceeds counter capacity of word 9 (read/write errors)

Table 2-3 (cont). User Error Logging Table

ERROR LOGGING TABLE VALUES		
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls (cont)		
Word(s)	Value (Signed Binary)	Function
18-20	3-word integer	Internal date/time (in milliseconds) of error logging startup for this device
21	1-word integer	Number of minutes since error logging started for this device
22, 23	2-word integer; history counter value of number of I/O orders	Number of I/O orders issued up to time of last counter reset; if overflow indicator (word 17) was set, this value not meaningful
24	History counter value of number of errors	Shows number of errors up to time of last reset of the counter; if overflow indicator (word 17) was set, this number not meaningful
25	History counter value of error threshold ratio	Error threshold ratio when counter was last reset
26-28	3-word integer; history counter value of data/time	Internal date/time when history counters were last reset
29-34	Six words	Device name
35-40	Six words	Volume name, if volume mounted
41	1-word integer	LRN for this device
42	1-word integer	Device type

Table 2-3 (cont). User Error Logging Table

ERROR LOGGING TABLE VALUES		
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls (cont)		
Word(s)	Value (Signed Binary)	Function
The next 16 words constitute an array containing two 8-word entries (words 43-50 and 51-58), each with similar information about the two most recent errors. See word 16.		
41-45	3-word integer	Internal date/time (milliseconds) of the error
46	One word	I/O status word
47-49	Device-specific words	<u>Word</u> <u>Meaning</u> <u>For disk devices</u>: 47 Status word 2 (for storage module unit only); bits 13-17 indicate which problem type corrected by retry 48 Cylinder number 49 Sector and track number 50 Range or number of software_initiated retries <u>For magnetic tape</u>: 47 Status word 2 48 Configuration word A 49 Task word (operation code) 50 Range or number of software_initiated retries

Table 2-3 (cont). User Error Logging Table

ERROR LOGGING TABLE VALUES		
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls (cont)		
Word(s)	Value (Signed Binary)	Function
47-50 (cont)	Device-specific words (cont)	<u>Word</u> <u>Meaning</u> <u>For printer</u>: 47 Not used 48 Task word (configuration word A) 49 Not used 50 Range or number of software_initiated retries <u>For card reader</u>: 47 Not used 48 Not used 49 Not used 50 Range or number of software-initiated retries <u>For terminal (noncommunications)</u>: 47 Configuration word A 48 Configuration word B 49 Not used 50 Range or number of software-initiated retries

Table 2-3 (cont). User Error Logging Table

ERROR LOGGING TABLE VALUES		
Values Returned by $ELGT, $ELEX, $ELEND Macro Calls (cont)		
Word(s)	Value (Signed Binary)	Function
51-58		Same as for words 43 through 50 above: Words 51-53 same as 43-55 Word 54 same as word 46 Word 55 same as word 47 Word 56 same as word 48 Word 57 same as word 49 Word 58 same as word 50

ERROR LOGGING TABLE ($ELOG)

Function Code: None

Equivalent Command: None

Generate and initialize a 59-word error logging table

FORMAT:

```
[label]   $ELOG   [threshold], [min_orders],
                  [init_orders], [init_errors]
```

ARGUMENTS:

threshold

Specifies the error threshold (0 ≥ threshold ≤ 100). The integer represents a decimal fraction in thousandths. When error/orders exceed this fraction, a message is sent to the operator, and the in-memory logging table is reset. The default is zero.

min_orders

Specifies the minimum number of I/O orders before threshold checking begins. The entry must be ≥0; default is zero.

init_orders

Counter for orders, which is normally initialized to zero. The entry must be ≥0; default is zero.

init_errors

Counter for errors, which is normally initialized to zero. The entry must be ≥0; default is zero.

DESCRIPTION:

This macro call generates in line a 59-word error-logging table (see Table 2-3). The call initializes the first 14 words of the table with values supplied by the arguments. The error logging macro calls $ELGT, $ELEX, and $ELEND return error logging data to words 15 through 58 of the table generated by this call.

ERROR OUT

ERROR OUT ($EROUT)

Function Code: 08/03

Equivalent Command: None

Write the next record to the error-out file for the task group of the issuing task.

FORMAT:

```
[label]  $EROUT   [location of record area address],
                  [location of record size],
                  [byte offset from beginning of record area]
```

ARGUMENTS:

location of record area address

Any address form valid for an address register; provides the address of a record area containing the record to be written to the error-out file. The first byte of the record must be a slew byte (print file form control byte; see "Printer Driver" in Section 6, Volume I.) The record text begins in the second byte.

location of record size

Any address form valid for a data register; provides the size (in bytes) of the record whose address is given in argument 1. The output size value must include the slew byte.

byte offset from beginning of record area

Any address form valid for a data register; provides the byte offset of the beginning of the record area (from the address provided in argument 1). If argument 3 is L, the record begins in the left byte of the address specified in argument 1; if argument 3 is R, the record area begins in the right byte of this address. Any other value for argument 3 is taken to be the location of the byte offset. If argument 3 is omitted, the record area is assumed to begin at the left byte of the address specified in argument 1.

DESCRIPTION:

This macro call allows a task to write the next record (an error message record) to the current error-out file. The error-out file is the same as the initial user-out file defined in the Request Group macro call, and cannot be changed during execution of the request.

NOTES

1. The system places in $B4 the address of the record to be written, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the output record.

2. The system places in $R6 the output record size, supplied by argument 2. If this argument is omitted, the system assumes that $R6 contains the size of the record.

3. If argument 3 is L, $R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, $R7 is set to one to designate that the record area begins in the right byte of the specified address. Any other value is assumed to be the location of the byte offset to be used, and is placed in $R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address, and $R7 is set to zero.

4. On return, $R1, $R6, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0817 - Memory access violation

   All data management write-next-record error codes may also be returned. See the System Messages manual.

   $R6 - Residual range (number of bytes not transferred from record area).

   $B4 - Address of record area containing output record.

Example:

In this example, the issuing task is to write an error message record on the error-out file. The record length is 12 bytes (including the slew byte). The output record is located at the record area address RECAD. The record area begins at the leftmost byte of the indicated address.

```
    OUTRB     $EROUT     !RECAD,=12
              .
              .
              .
    RECAD     TEXT       'AFIELD  ERROR'
```

EXPAND PATHNAME ($XPATH)

Function Code: 10/D0

Equivalent Command: None

Develop a full pathname from a relative pathname.

FORMAT:

```
[label]   $XPATH   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of an argument structure that must contain the following entries in the order shown.

input pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a relative pathname (which must end with an ASCII space character) to be expanded.

output pathname pointer

A 4-byte address that may be any address form valid for an address register; identifies a 58-byte field into which the absolute (i.e., expanded) pathname is placed by the system.

pathname base

A 2-byte binary value that specifies the basis for expanding the relative path, as follows:

0000 - Working directory
0001 - System library-1
0002 - System library-2

DESCRIPTION:

This macro call will expand any relative pathname, regardless of the format in which it is supplied, into an absolute pathname. It is possible that the resulting pathname will point to a nonexistent file. The expanded pathname cannot exceed 58 characters.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - Successful completion

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0222 - Pathname cannot be expanded; no working directory.

Example:

In this example, the pathname of the working directory is ^VOL6>SUB1>SUB2>SUB3>SUB4. A fully expanded absolute pathname is to be developed from the relative pathname <<ADF. In the macro call, the relative pathname (<<ADF) and the basis (working directory) for developing the absolute pathname must be defined, as well as an area into which the system can place the fully expanded absolute pathname. The main memory area is defined as follows:

```
    X_NAME     RESV    29
```

The argument structure is built as follows:

```
    XPND_1     DC      <RELPTH
               RESV    2-$AF,0
               DC      <X_NAME
               RESV    2-$AF,0
               DC      0
```

The relative pathname is defined as follows:

```
RELPTH   DC   '<<ADF '
```

The fully expanded pathname ^VOL6>SUB1>SUB2>ADF is developed as a result of the following macro call.

```
$XPATH   !XPND_1
```

EXTERNAL DATE/TIME, CONVERT TO

EXTERNAL DATE/TIME, CONVERT TO ($EXTDT)

Function Code: 05/04

Equivalent Command: Time (TIME)

Convert an internal format date/time value to an external format date/time value.

FORMAT:

```
[label]  $EXTDT  [location of address of internal date/time],
                 [location of address of receiving field],
                 [location of size of receiving field]
```

ARGUMENTS:

location of address of internal date/time

Any address form valid for an address register; provides the address of the 3-word field containing the internal date/time value to be converted. This value must be in the format returned by the Get Date/Time macro call.

location of address of receiving field

Any address form valid for an address register; provides the address of a field in the issuing task that is to receive the external format date/time value.

location of size of receiving field

Any address form valid for a data register; provides the size of the receiving field identified by argument 2. The field size must be less than or equal to 22 bytes. If this argument is omitted, the size is set to 22 bytes (the date/time value is resolved to a thousandth of a second).

DESCRIPTION:

This macro call converts an internal date/time value (in the format supplied by the Get Date/Time macro call) to an external date/time format. The date/time value appears in the receiving field as a character string having the format:

Word	Contents
0	yy (two ASCII numeric characters)
1	yy (two ASCII numeric characters)
2	/m (two ASCII characters)
3	m/ (two ASCII characters)
4	dd (two ASCII numeric characters)
5	h (two ASCII characters)
6	hm (two ASCII numeric characters)
7	m: (two ASCII characters)
8	ss (two ASCII numeric characters)
9	.t (two ASCII characters)
10	tt (two ASCII numeric characters)

yyyy - year	mm - minute
mm - month	ss - seconds
dd - day	ttt - tenths, hundredths, thousandths of seconds
hh - hour	

The size of the receiving field cannot terminate with a punctuation character (/, :, or .). Therefore, argument 3 cannot specify a size of 5, 8, 16, or 19 bytes.

NOTES

1. The system loads the internal date/time value, whose address was supplied by argument 1, into $R2, $R6, and $R7. If argument 1 is omitted or is =$R7, the system assumes that $R2, $R6, and $R7 contain the value to be converted.

2. The system places in $B4 the address of the receiving field supplied by argument 2. If this argument is omitted, the system assumes that $B4 contains the correct address.

3. The system places in $R5 the size of the receiving field supplied by argument 3 in $R5. If this argument is given as =$R5, the system assumes that $R5 contains the correct size. If this argument is omitted, $R5 is set to 22 bytes (thousandth of a second resolution).

4. On return, $R1, $R2, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0402 - Invalid (negative) receiving field length

   040A - Improper access to external date/ time field

   0817 - Memory access violation

   $R2, $R6, $R7 - Internal date/time value supplied

   $B4 - Address of receiving field.

Example:

See the example given for the Get Date/Time macro call.

EXTERNAL TIME, CONVERT TO ($EXTIM)

Function Code: 05/05

Equivalent Command: None

Convert an internal format date/time value to an external format time value.

FORMAT:

```
[label]   $EXTIM   [location of address of internal date/time],
                   [location of address of receiving field],
                   [location of length of receiving field]
```

ARGUMENTS:

location of address of internal date/time

Any address form valid for a data register; provides the address of a 3-word field containing the internal date/time value to be converted. This value must be in the format returned by the Get Date/Time macro call.

location of address of receiving field

Any address form valid for an address register; provides the address of a field in the issuing task that is to receive the external format time value.

location of length of receiving field

Any address form valid for a data register; provides the size of the receiving field identified by argument 2. The field size must be less than or equal to 11 bytes. If this argument is omitted, the size is set to 9 bytes (the time is resolved to a tenth of a second).

DESCRIPTION:

This macro call converts an internal date/time value (in the format supplied by the Get Date/Time macro call) to an external time format. The time value appears in the receiving field as a character string having the format hhmm:ss.ttt (see below).

Word	Contents
0	hh (two ASCII numeric characters)
1	mm (two ASCII numeric characters)
2	:s (two ASCII characters)
3	s. (two ASCII characters)
4	tt (two ASCII numeric characters)
5	t (two ASCII characters)

hh - hours ss - seconds
mm - minutes ttt - tenths, hundredths, thousandths of seconds

The size of the receiving field cannot terminate with a punctuation character (: or .). Therefore, the third argument cannot specify a size of 5 or 8 bytes.

NOTES

1. The system loads the internal date/time value, whose address is supplied by argument 1, into $R2, $R6, and $R7. If argument 1 is omitted or is =$R7, the system assumes that $R2, $R6, and $R7 contain the internal value to be converted.

2. The system places in $B4 the address of the receiving field supplied by argument 2. If this argument is omitted, the system assumes that $B4 contains the correct address.

3. The system places in $R5 the size of the receiving field supplied by argument 3. If argument 3 is =$R5, the system assumes that $R5 contains the correct size. If this argument is omitted, $R5 is set to 9 bytes (tenth of a second resolution).

4. On return, $R1, $R2, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0402 - Invalid (negative) receiving field length

040A - Invalid access to external date/ time field

0817 - Memory access violation

$R2, $R6, $R7 - Internal date/time value supplied

$B4 - Address of receiving field.

Example:

In this example, the Get Date/Time macro call is used to get the current date/time, in internal format (leaving it in $R2, $R6, and $R7). The External Time, Convert To ($EXTIM) macro call is then used to format this internal date/time value into a displayable format with a resolution to milliseconds. A message containing the external format date/time is then written on the user-out file.

```
              *
              * GET THE CURRENT DATE/TIME.
              *
                       $GDTM
              *
              * FORMAT IT FOR DISPLAY.
              *
                 $EXTIM   ,!P1_TIM,=11
              *
              * OUTPUT THE MESSAGE.
              *
                $USOUT    !P1_MSG,=P1_MLN
                              .
                              .
                              .
    P1_MSG   TEXT     'APHASE 1 FINISHED ATΔ'
    P1_TIM   TEXT     'HHMM:SS.TTT'
    P1_MLN   EQU      2*($-P1_MSG)
```

FILE INFORMATION BLOCK ($FIB)

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, perform one of the following:

- Build a 16-word file information block (FIB) containing default values for the words.
- Alter the contents of an existing FIB.
- Call and expand the File Information Block Offsets macro call ($TFIB) to provide labels for the FIB entries.

FORMAT:

```
[label]   $FIB   [argument]
```

ARGUMENTS:

There are three types of arguments for this macro call:

- Keyword only
- Keyword with expression
- Keyword with option.

The keyword RESV generates a data structure. The File Information Block macro call without the keyword RESV generates executable code to modify an existing data structure.

When the macro call is coded with only the keyword RESV, a 16-word FIB containing default values is built. The entries have the following values:

```
DC    0
DC    B'0110010010000000'
DC    0,0
DC    80
DC    80
DC    0
DC    Z'FFFF'
DC    0
DC    0,0
DC    Z'0104'
DC    0,0
DC    0,0
```

The default values generated for this FIB allow access to a file for reading and writing, and access to a record by both primary and relative keys. The default input and output record lengths are 80 characters; the default key format for input records is primary; key length is 4 bytes.

When the keyword RESV is used with other arguments, it preserves all entries in the generated FIB that are not specifically changed by the other arguments.

The keywords listed in Table 2-4 apply to the words of the file information block. These keywords can be coded in any order. If a new FIB is to be built and a keyword is omitted, the default value (described above) for that word is used. If an existing FIB is to be modified and a keyword is omitted, the existing value for that word is used. The table below shows the keywords and possible expression values, but does not necessarily correspond to the FIB physical structure. For more information, see System Programmer's Guide, Vol.I, Tables 3-1 and 3-3.

Table 2-4. FIB Keywords

Keyword	Expression
RESV	Build a new FIB (as opposed to altering an existing FIB).
ADR=	Address of a FIB to be modified. Not used when building new FIBs (i.e., when RESV is specified).
LFN=	A value from 0 through 255, specifying the logical file number with which the file is referenced; or -1.
	For Data Management (Record Level) Access
URP=	Address of start of user record area
IRL=	Maximum size of input record.
ORL=	Actual size of output record.
IRT=	Input record type; currently, must be Z'FFFF'.
ORT=	Output record type; currently MBZ.
IKP=	Address of user key area.

Table 2-4 (cont). FIB Keywords

Keyword	Expression
	For Data Management (Record Level) Access (cont)
IKF=	Type of key: 00 - None specified 01 - Primary, relative, or CALC (random) 02 - Simple
IKL=	Value specifying the length of the user key area (IKP). 256 ASCII characters is the allowable maximum for primary and CALC keys. (Simple and relative keys are always assumed to be four bytes.) When used with the RESV keyword, the specified value for IKL must be a 1-byte hexadecimal number (e.g., 0A, 01, etc.).
	For Storage Management (Block Level) Access
UBP=	Address of start of user buffer area.
BFS=	Value specifying size of data transfer buffer size.
BKS=	Value specifying size of block. With BKN, this value identifies the start address of the data transfer.
BKN=	Value specifying the starting block number of the data transfer; an integer, from 0 to 65,535, relative to the beginning of the file.

The keywords listed below in Table 2-5 apply only to the program view of the FIB. The option values can appear in any order, and more than one option can be specified for a keyword. The bits in the program view word that are not explicitly assigned a value through a keyword retain their previously set values. Those values identified as "default" indicate initial settings when the keyword RESV is specified. Table 2-5 shows keywords and possible values first for data management (record level) access, followed by those for storage management (block level) access. See System Programmer's Guide, Vol.I, Tables 3-1 and 3-2 for more information.

Table 2-5 (cont). FIB Program View Keywords

Keyword	Option	Meaning
For Data Management (Record Level) Access (cont)		
Set Record Attributes		
SRA=	FL	Only fixed-length records allowed.
	DV	Deleted records are visible.
RRA=	FL	Fixed- and variable-length records allowed (default).
	DV	Deleted records are not visible (default).
Set Data Transfer Attributes		
SDT=	BT	Data is transferred in binary transcription mode.
RDT=	BT	Data is transferred in ASCII mode (default).
	OP	File is open for data transfer at the record level (default).
Set Area Boundary		
ODD=	KY	User key area begins at odd-byte boundary.
	RC	User record area begins at odd-byte boundary.
EVN=	KY	User key area begins at even-byte boundary (default).
	RC	User record area begins at even-byte boundary (default).
For Storage Management (Block Level) Access		
Set Function		
SFN=	RD	Blocks can be read by $RDBLK macro call (default).
	WR	Blocks can be written by $WRBLK macro call (default).
RFN=	RD	Blocks cannot be read by $RDBLK macro call.
	WR	Blocks cannot be written by $WRBLK macro call.

Table 2-5 (cont). FIB Program View Keywords

Keyword	Option	Meaning
For Storage Management (Block Level) Access (cont)		
Set Data Transfer Attributes		
SDT=	BT	Data is transferred in binary transcription mode.
	AS	$RDBLK and $WRBLK macro calls executed asynchronously.
	OP	File is open for data transfer at the block level; (must be specified).
RDT=	BT	Data is transferred in ASCII mode (default).
	AS	$RDBLK and $WRBLK macro calls executed synchronously (default).
Set Area Boundary		
ODD=	BF	User buffer area begins at odd-byte boundary.
EVN=	BF	User buffer area begins at even-byte boundary (default).

DESCRIPTION:

A FIB must exist for a file if that file is to be operated upon by one of the following macro calls:

- Open File ($OPFIL)
- Close File ($CLFIL)
- Test File ($TIFIL, $TOFIL)
- Read Record ($RDREC)
- Write Record ($WRREC)
- Rewrite Record ($RWREC)
- Delete Record ($DLREC)
- Read Block ($RDBLK)
- Write Block ($WRBLK)
- Wait Block ($WTBLK).

If an existing FIB is to be modified and the argument ADR= is not entered, $B4 is assumed to point to the FIB to be modified.

Registers R7 and B5 are altered when an existing FIB is modified.

Macro global variable GX is used to prevent duplicate expansion of the File Information Block Offsets macro call ($TFIB).

When the File Information Block macro call is used to alter an existing FIB, arguments that use an address follow the convention in which addresses preceded by the ! character cause an LAB instruction to be generated, and addresses not preceded by the ! character cause an LDB instruction to be generated. When values for arguments coded as keyword=expression (IRL=, ORL=, etc.) are supplied, the address of the value is distinguished by a preceding character. No special character is needed to indicate that the string following the = character is a value. The second example given below uses both values and addresses (IFL=128 and LFN= !GETPRM).

The expressions specified with each argument must be in a form suitable for the DC statement. IKF and IKL must specify a 1-byte hexadecimal number.

Example 1:

In this example, the File Information Block Offsets ($TFIB) macro call is expanded.

```
    $FIB
```

Example 2:

In this example, an existing FIB is modified. This example assumes that $B4 has been loaded with the address of the FIB to be modified.

```
    $FIB  URP=!REC1,RFN=WR,SRA=FL,ODD=RC,IRL=128,LRN=!GETPRM
```

Execution of the macro call generates the following set of instructions:

```
    LAB    $B5,REC1
    STB    $B5,$B4.F_URP
    LBF    $B4.F_PROV,B'0010000000000000'
    LBT    $B4.F_PROV,B'0000000000100100'
    LDR    $R7,=128
    STR    $R7,$B4.F_IRL
    LDR    $R7,GETPRM
    STR    $R7,$B4.F_LFN
```

Example 3:

This example generates a FIB so that the file can be accessed for reading, writing, rewriting, and deleting records by either primary or relative keys. The rewrite and delete bits (bits 3 and 4) of the program view word are altered from the original values (provided by the RESV parameter) by means of the SFN=RWDL argument.

```
EXTFIB   $FIB   LFN=3,IKF=01,RESV,SFN=RWDL,SKA=PR,IKP=<KEY
```

This macro call generates the following FIB:

```
    EXTFIB     DC     3
               DC     B'0111110010000000'
               DC     0,0
               DC     80
               DC     80
               DC     0
               DC     Z'FFFF'
               DC     0
               DC     <KEY
               RESV   2-$AF
               DC     Z'0104'
               DC     0,0
               DC     0,0
```

Example 4:

In this example, a 16-word FIB is generated with default values for all words.

```
    EXTFIB     $FIB     RESV
```

The following FIB is generated:

```
    EXTFIB     DC     0
               DC     B'0110010010000000'
               DC     0,0
               DC     80
               DC     80
               DC     0
               DC     Z'FFFF'
               DC     0
               DC     0,0
               DC     Z'0104'
               DC     0,0
               DC     0,0
```

FILE INFORMATION BLOCK OFFSETS (DATA AND STORAGE MANAGEMENT ACCESS)

FILE INFORMATION BLOCK OFFSETS (DATA AND STORAGE MANAGEMENT ACCESS) ($TFIB)

Associated Macro Calls:

Open File, Close File, Test File ($TIFIL), Test File ($TOFIL), Read Record, Write Record, Rewrite Record, Delete Record, Read Block, Write Block, Wait Block

FORMAT:

```
[label]  $TFIB  [first letter of tags]
```

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure for Data Management Access:

Word	Fields	
0	Logical File Number (LFN)	
1	Program View	
2 3	User Record Pointer	
4	Input Record Length	
5	Output Record Length	
6	Input Record Status	Output Record Status

Word	Fields	
7	Input Record Type	
8	Output Record Type	
9 10	Input Key Pointer	
11	Input Key Format	Input Key Length
12 13	Output Record Address	
14 15	Reserved	

NOTE

Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Structure for Storage Management Access:

Word	Fields
0	Logical File Number (LFN)
1	Program View
2 3	User Buffer Pointer
4	Buffer (Transfer) Size
5	Block Size
6 7	Block Number
8 9 10 11 12 13 14 15	Reserved

> **NOTE**
>
> Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
F_LFN	0	Logical file number (LFN)
F_PROV	+1	Program view
F_URP	+2	User record pointer*
F_IRL	+4	Input record length*
F_ORL	+5	Output record length*
F_IRS	+6	Input record status (first byte)*
F_ORS	+6	Output record status (second byte)*
F_IRT	+7	Input record type*
F_ORT	+8	Output record type*
F_IKP	+9	Input key pointer*
F_IKF	+11	Input key format (first byte)*
F_IKL	+11	Input key length (second byte)*
F_ORA	+12	Output record address*
F_ORA1	F_ORA(+12)	(left half of F_ORA)
F_ORA2	F_ORA+1(+13)	(right half of F_ORA)
F_UBP	+2	User buffer pointer**
F_BFSZ	+4	Buffer size**
F_BKSZ	+5	Block size**
F_BKNO	+6	Block number**
F_BKN1	F_BKNO(+6)	(left half of F_BKNO)
F_BKN2	F_BKNO+1(+7)	(right half of F_BKNO)
F_SZ	16	Size of structure (in words); not a field in the block

*Specific to $RDREC, $WRREC, $RWREC, and $DLREC macro call.
**Specific to $RDBLK, and $WRBLK macro calls.

In addition to the offsets tags listed above, the following program view (F_PROV tag, above) masks are defined:

Tag	Mask	Meaning
MF_OPS	Z'8000'	Open for storage management
MF_RDA	Z'4000'	Read operations allowed
MF_WRA	Z'2000'	Write operations allowed
MF_RWA	Z'1000'	Rewrite operations allowed
MF_DLA	Z'0800'	Delete operations allowed
MF_PKA	Z'0400'	Access through primary key
MF_CKA	Z'0200'	Access through CALC key
MF_RKA	Z'0080'	Access through relative key
MF_SKA	Z'0040'	Access through simple key
MF_FRA	Z'0020'	Fixed-length records allowed
MF_DLV	Z'0010'	Deleted records visible to program
MF_KOD	Z'0008'	Key area starts an odd-byte boundary
MF_ROD	Z'0004'	Record area starts at odd-byte boundary (data management specific)
MF_BOD	Z'0004'	Buffer area starts at odd-byte boundary (storage management specific)
MF_BTM	Z'0002'	Data transferred in binary transcription mode
MF_AIO	Z'0001'	Next block function is asynchronous (storage management specific)

FILE INFORMATION BLOCK OFFSETS (DATA MANAGEMENT ACCESS)

FILE INFORMATION BLOCK OFFSETS (DATA MANAGEMENT ACCESS ($FIBDM))

Associated Macro Calls:

Open File, Close File, Test File (input), Test File (output), Read Record, Write Record, Rewrite Record, Delete Record

FORMAT:

```
[label]  $FIBDM  [first letter of tags]
```

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Logical File Number (LFN)	
1	Program View	
2 3	User Record Pointer	
4	Input Record Length	
5	Output Record length	
6	Input Record Status	Output Record Status
7	Input Record Type	
8	Output Record Type	
9 10	Input Key Pointer	

11	Input Key Format	Input Key Length
12 13	Output Record Address	
14 15	Reserved	

NOTE

Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
F_LFN	0	Logical File Number (LFN)
F_PROV	+1	Program view
F_URP	+2	User record pointer
F_IRL	+4	Input record length
F_ORL	+5	Output record length
F_IRS	+6	Input record status (first byte)
F_ORS	+6	Output record status (second byte)
F_IRT	+7	Input record type
F_ORT	+8	Output record type
F_IKP	+9	Input key pointer
F_IKF	+11	Input key format (first byte)
F_IKL	+11	Input key length (second byte)
F_ORA	+12	Output record address
F_SZ	16	Size of FIB (in words); not a field in the FIB.

In addition to the offsets listed above, the following define the program view (F_PROV) masks:

Tag	Mask		Meaning When Bit Set to 1
MF_OPD	Z'8000'	Bit 0:	0 = Open for data management
MF_RDA	Z'4000'	Bit 1:	Read functions allowed
MF_WRA	Z'2000'	Bit 2:	Write functions allowed
MF_RWA	Z'1000'	Bit 3:	Rewrite functions allowed
MF_DLA	Z'0800'	Bit 4:	Delete functions allowed
MF_PKA	Z'0400'	Bit 5:	Access by primary key
MF_CKA	Z'0200'	Bit 6:	Access by CALC key
MF_RKA	Z'0080'	Bit 8:	Access by relative key
MF_SKA	Z'0040'	Bit 9:	Access by simple key

Tag	Mask		Meaning When Bit Set to 1
MF_FRA	Z'0020'	Bit 10:	Fixed-length records allowed
MF_DLV	Z'0010'	Bit 11:	Deleted records are visible
MF_KOD	Z'0008'	Bit 12:	Key area starts at an odd-byte boundary
MF_ROD	Z'0004'	Bit 13:	Record area starts at an odd-byte boundary
MF_BTM	Z'0002'	Bit 14:	Data is transferred in binary transcription mode

Examples:

$B4 contains the address of the FIB. The user record pointer field can be accessed with:

```
$B4.F_URP
```

The program view bit, which indicates that the user record area starts at an odd-byte boundary, can be set by the instruction:

```
LBT    $B4.F_PROV,MF_ROD
```

The FIB can be initially set to zero by the instructions:

```
      LDV     $R1,F_SZ-1     R1-- SIZE OF FIB MINUS 1
$A    CL      $B4.$R1        CLEAR ONE WORD
      BDEC    $R1,>-$A       LOOP UNTIL ALL WORDS CLEARED
```

FILE INFORMATION BLOCK OFFSETS (STORAGE MANAGEMENT ACCESS) ($FIBSM)

Associated Macro Calls:

Open File, Close File, Read Block, Write Block, Wait Block

FORMAT:

[label] $FIBSM [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields
0	Logical File Number (LFN)
1	Program View
2 3	User Buffer Pointer
4	Buffer (transfer) Size
5	Block Size
6 7	Block Number
8 9 10 11 12 13 14 15	Reserved

> **NOTE**
>
> Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
F_LFN	0	Logical file number (LFN)
F_PROV	+1	Program view
F_UBP	+2	User buffer pointer
F_BFSZ	+4	Buffer (transfer) size
F_BKSZ	+5	Block size
F_BKNO	+6	Block number
F_SZ	16	Size of FIB (in words); not a field in the FIB

In addition to the offsets listed above, the following define the program view (F_PROV) masks:

Tag	Mask		Meaning When Bit Set to 1
MF_OPS	Z'8000'	Bit 0:	Open for storage management
MF_RDA	Z'4000'	Bit 1:	Read functions allowed
MF_WRA	Z'2000'	Bit 2:	Write functions allowed
MF_BOD	Z'0004'	Bit 13:	Buffer area starts at an odd byte boundary
MF_BTM	Z'0002'	Bit 14:	Data is transferred in binary transcription mode
MF_AIO	Z'0001'	Bit 15:	Next block function is asynchronous

Example:

$B4 contains the address of the FIB; $R6 and $R7 contain the address of the next block to be accessed. The block number field F_BKNO is set by the instruction:

```
SDI    $B4.F_BKNO
```

The program view bit, which indicates that the next call will be asynchronous, is set by the instruction:

```
LBT    $B4.F_PROV,MF_AIO
```

GET DATE/TIME ($GDTM)

Function Code: 05/06

Equivalent Command: None

Supply the requesting task with the current internal date/time value maintained by the system.

FORMAT:

[label] $GDTM [location of address of receiving field]

ARGUMENT:

location of address of receiving field

Any address form valid for a data register; provides the address of a 3-word field in the issuing task that is to receive the current internal date/time value.

DESCRIPTION:

This macro call returns to the issuing task the current 3-word internal date/time value. The leftmost word contains the most significant 16 bits; the rightmost word contains the least significant 16 bits. The value supplied is a binary count of the milliseconds since 1 January 1901 at 00:00:00.000 hours.

NOTES

1. The internal date/time value is returned in $R2, $R6, and $R7 and stored in the receiving field specified by argument 1. If argument 1 is omitted or is =$R7, the value is returned only in $R2, $R6, and $R7.

2. On return, $R1, $R2, $R6, and $R7 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   040A - Invalid access to external date/time field

$R2, $R6, and $R7 - Current 3-word internal date/time value. $R2 contains the most significant 16 bits and $R7 the least significant 16 bits.

Example:

In this example the Get Date/Time macro call is used to get the starting date/time, in internal format, of a process and store it in the field ST_TIM. The Convert to External Date/Time ($EXTDT) macro call is then used to format this internal clock value, contained in $R2, $R6, and $R7, into a displayable date/time format with resolution to a tenth of a second. A startup message containing the external format date/time is then written on the user-out file. Later, the Get Date/Time macro call is used again to get the finishing date/time of the process without storing it in memory. The low order two words of the starting date/time are then subtracted from the corresponding words of the finishing date/time, leaving the elapsed time (in milliseconds) in $R6 and $R7. The subtraction is performed assuming a central processor that does not have the subtract integer double instruction. The high order word of the starting and finishing date/time values is ignored with the assumption that the elapsed time is less than 2 million seconds (about 24.855 days).

```
*
*   GET THE STARTING TIME.
*
         $GDTM   !ST_TIM
*
*   FORMAT IT FOR DISPLAY.
*
        $EXTDT   ,!GO_TIM,20
*
*   OUTPUT THE START UP MESSAGE.
*
      $USOUT   !GO_MSG,=GO_MLN
```

```
     *
     *    BEGIN PROCESSING.
     *
                        .
                        .
     *                  .
     *    GET THE FINISHING TIME.
     *
                        $GDTM
     *
     *    CALCULATE THE ELAPSED TIME.
     *
             SUB        $R7,ST_TIM+2
             BCT        >$+3
             ADV        $R6,-1
             SUB        $R6,ST_TIM+1
                        .
                        .
                        .
ST_TIM       RESV       3,0
GO_MSG       TEXT       'APROGX STARTED ATΔ'
GO_TIM       TEXT       'YYYY/MM/DD HHMM:SS.T'
GO_MLN       EQU        2*($-GO_MSG)
```

GET DEVICE INFORMATION ($GIDEV)

Function Code: 10/66

Equivalent Command: None

Retrieve information about a specified device.

FORMAT:

```
[label] $GIDEV [parameter structure address]
```

ARGUMENT:

parameter structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The parameter structure must contain the following entries in the order shown. (Entries marked with an asterisk (*) are provided by the system. The user must supply values in the other entries.) The size of each entry, whose description follows this list, is as follows:

Parameter Structure Entry	Size (in bytes)
logical resource number	2
logical component number	2
*device name	12
*hardware device type	2
*software device id	2
*channel number	2
*RCT indicators word	2
*timeout interval	2
*RCT status word	2
reserved for future use	4

logical resource number

A 2-byte LRN used to refer to the device; must be a binary number in the range 0 through 255. This number must be supplied.

logical component number

A 2-byte logical component number (LCN) (i.e., sub-LRN) that refers to a device subcomponent. This field should be set to zero if the device is not addressable as a subcomponent.

device name

The system sets this 2-byte field to the 1- to 12-character name of the specified device. The name supplied is that which the user gave to the device when the system was configured. This field is space filled.

hardware device type

The system sets this 2-byte field to the 4-digit, hexadecimal device code of the specified device. The device name and marketing identifier signified by each code are listed in this manual under the Get File Information ($GIFIL) description.

software device id

The system sets this 2-byte field to the 4-digit, hexadecimal device descriptor of the specified device. The first pair of digits identifies the device; the second pair identifies the driver controlling the device. The device and driver codes include but are not limited to the following:

ID	Device	ID	Device
00	undefined	40	VIP 7700
01	card reader	41	VIP 7760
02	card reader/punch	42	VIP 7804
03-04	RFU	43	VTS 7710
05	line printer	44	VTS 7740
06	serial printer	45	VIP 7100
07-09	RFU	46	VIP 7200
0A	7-channel magnetic tape	47	VIP 7207
0B	9-channel magnetic tape	48	RFU
0C	phase encoded magetic tape	49	VIP 7801
0D-0F	RFU	4A	VIP 7808
10	diskette	4B	VIP 7803
11	disk cartridge	4C	TTY (KSR)
12	storage module	4D	Terminet 0300
13	cartridge module	4E	Terminet 1200
14	Cynthia	4F	POLY 21
15	mini diskette	50	VIP 7398
16	Lark	51	ROSY 24
17	Winchester	52	ROSY 26
18-3F	RFU	53	RFU

ID	Device
54	Spinwriter 5518
55-56	RFU
57	Sara 22
58-5A	RFU
5B	ASPI 10
5C	ASPI 32
5D	ASPI 38
5E	RFU
5F	ASPI 30
60	ASPI 35
61	ASPI 77
62	RFU
63	VIP 7401
64	VIP 7301
65	VIP 7307
66	VIP 7303
67	VIP 7399
68-7F	RFU

ID	Driver
00	undefined
01	card
02-03	RFU
04	printer
05-06	RFU
07	magnetic tape
08-0A	RFU
0B	disk
0C-0F	RFU
10	KSR
11	console
12-3F	RFU
40	VIP
41	STD
46-47	RFU
48	BSC 2780
49	BSC 3270
4A	HASP
4B-4C	RFU
4D	PVE
4E	RCI
4F	LHDLC
50-51	RFU
52	HDLC
53	LLHA/LLHB
54-7F	RFU

channel number

The system sets this 2-byte field to channel number of the specified device.

resource control table (RCT) indicators

The system sets this 2-byte field to the current value of the R_FLGS word in the RCT of the specified device. The mask bits of this word and their significance are shown below.

Mask Bit	Meaning
z'8000'	tape recovery requested
z'8000'	double density
z'8000'	input attention in KSR
z'4000'	IBM type diskette
z'4000'	tape recovery successful
z'2000'	tape block count invalid
z'1000'	2-word disk address

Mask Bit	Meaning
z'0800'	not single character mode (KSR)
z'0800'	do automatic volume recognition (AVR) now
z'0400'	disk type device
z'0200'	communication connected device
z'0100'	line printer or KSR
z'0080'	attention has occurred
z'0040'	disable device on attention
z'0020'	device disabled
z'0010'	error log busy
z'0008'	corrected hardware error occured
z'0004'	not connected to file system
z'0002'	power failure recovery state

timeout interval

The system sets this 2-byte field to the timeout interval (in seconds) of the specified device.

RCT status word

The system sets this 2-byte field to the current value of the R_STTS word in the RCT of the specified device. The value of this word indicates the hardware status of the device's controller, and is of use in analysing controller malfunctions. Modified values of R_STTS appear in the I_ST entry of the IORB upon completion of an I/O operation requested of the device.

DESCRIPTION:

To access specific entries in the parameter structure, use the Device Information Parameter Structure offsets ($DIPSB) macro call.

NOTE

1. The system places in $B4 the address of the parameter structure supplied by the argument of this macro call. If the argument is omitted, the system assumes that $B4 contains the parameter structure address.

GET DEVICE INFORMATION PARAMETER BLOCK OFFSETS

GET DEVICE INFORMATION PARAMETER BLOCK OFFSETS ($DIPSB)

Associated Macro Call: Get Device Information

FORMAT:

```
[label]  $DIPSB  [first letter of tags]
```

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields
0	Logical resource number
1	Logical component number
2 3 4 5 6 7	Device name
8	Hardware device type
9	Software device id
10	Channel number
11	RCT indicators word
12	Timeout interval
13	RCT status word
14 15	Reserved for future use

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
D_LRN	0	Logical resource number (input)
D_LCN	+1	Logical component number (input)
D_NME	+2	Device name (1-12 characters)
D_TYP	+8	Hardware device type
D_SDID	+9	Software device id
D_CHNL	+10	Channel number
D_IND	+11	RCT indicator word
D_TINV	+12	Timeout interval (seconds)
D_STS1	+13	RCT status word
D_SZ	16	Size of the structure (in words); not a field in the block

GET FILE ($GTFIL)

Function Code: 10/20

Equivalent Command: Get File (GET)

Locate and reserve a file (tape or disk file, disk directory, card reader, printer, or terminal device) for processing with the specified access rights. The file is identified by supplying either a logical file number (LFN) or a pathname. If both an LFN and a pathname are supplied, the file is reserved and is assigned to the LFN. Subsequent macro calls (Open File, Read Record, etc.) can then be directed to the file through this LFN. If the file is tape-resident, the Get File macro call supplies the necessary tape definition arguments. This function is normally done outside program execution, to assign the LFN to a file that is not known until execution time.

FORMAT:

```
[label]   $GTFIL   [argument structure address]
```

ARGUMENT:

NOTE

Any tape-specific argument is bypassed if explicitly specified by a previous GET command. This allows the user to override tape arguments outside program execution.

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown. A description of each entry follows this list.

Argument Structure Entry	Size (in bytes)
logical file number	2
pathname pointer	4
disk concurrency control	1
disk mount option	1
tape block size	2
tape logical record size	2
number of buffers	1
tape file sequence number	1
tape label format	1
tape data type	1
tape data format	1
tape file options	1
tape file section number	2
tape retention period	2

logical file number

A 2-byte LFN used to refer to the file; must be a binary number in the range 0 through 255, ASCII blanks (X'2020') if an LFN is not specified, or -1 (X'FFFF') if the system is to assign an LFN from the pool of available LFNs.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that, when expanded, identifies the file to be reserved. Binary zeros in this entry indicate that a pathname is not specified.

disk concurrency control

A 1-byte code, applicable only to disk files, that specifies the concurrency control to be established for the file.

If record locking is requested, the records in the file will be locked in shared-read/exclusive-write mode when the file is accessed. Once a file is reserved with locking, it cannot be reserved by another user (task group) unless that user also specifies record locking.

The type of file concurrency chosen indicates the file access chosen by the user and the way in which the user is willing to share access to the file with other users (task groups). There are six types of concurrency control, as follows:

Type 5 - Write or read; others can write or read (read/write sharing).

Type 4 - Write or read; others can read but not write (read share, exclusive write).

Type 3 - Write or read; no others can write or read (exclusive).

Type 2 - Read; others can read and write.

Type 1 - Read; others can read but not write (read sharing).

Type 0 - If the file is already reserved, the last concurrency specified is used. If the file is not already reserved, type 3 concurrency control is used.

The value of the disk concurrency control byte is determined as follows:

Bit(s)	Meaning
0	Lock specification: 0 - Do not lock records 1 - Lock records
1	Index only option: 0 - Not specified 1 - For Unified File Access System (UFAS) indexed and alternate indexed files, the user can access the index as if it were a data file.

Bit(s)	Meaning
2	Foreign file option: 0 - Not specified 1 - Override the foreign file attribute, set by the modify file function, to allow processing of a file that contains data not native to GCOS 6.
3	No lock option: 0 - Not specified 1- Through the specified LFN, allow records to be read without being locked even though the file has the record locking attribute. The integrity of the data being read is not guaranteed and no updates are allowed through the specified LFN.
4	No wait option: 0 - Not specified 1 - Through the specified LFN, do not wait for records that are locked by other users. Either the record will be locked immediately or an 022B error will be returned, indicating that it is locked by another user.
5-7	Concurrency control specification: 000 - Type 0 001 - Type 1 010 - Type 2 011 - Type 3 100 - Type 4 101 - Type 5

tape file options

A 1-byte code, applicable only to tape files, that defines packing, EBCDIC/ANSI translation, parity, file section number, and block sequence number usage.

Bit(s)	Meaning
0, 1	0 = Not specified 1 = For 7-track tape: No Packing 2 = For 9-track tape: EBCDIC/ANSI translation
2, 3	0 = Parity not specified 1 = Odd parity 2 = Even parity

Packing and parity bits are meaningful only for 7-track tapes to be opened for storage management (block level) access.

4	0 = Not specified 1 = File section number supplied
5	Must be zero
6, 7	0 = No BSN specified 1 = BSN not supplied 2 = BSN supplied

If 2 is specified, a BSN is assumed to be present on input; on output, a BSN will be inserted.

If the file is not tape-resident, this entry is ignored.

tape file section number

A 2-byte field specifying the relative file section number. Applies only to multi-volume tape files.

tape retention period

A 2-byte value, applicable only to tape files, that specifies the tape retention period in days. Zeros in this field indicate that the retention period is not specified.

If the file is not tape-resident, this entry is ignored.

For files with variable-length records, the block size can be any value, but should be at least as large as the maximum record size plus the 4-character logical record header and the length of any block header information.

The block sequence number is specified by the tape file options argument (see below).

The block size entry is ignored if the file is not tape-resident.

If the file is not currently reserved and block size is not specified (i.e., the field contains all zeros), a value is computed based on the values for logical record size, tape data format, and tape block sequence number (BSN) indicator.

If the file is already reserved and block size is not specified, the previously specified or computed value is not changed.

tape logical record size

A 2-byte binary value, applicable only to tape files, that specifies the logical record size in bytes.

The logical record size is the size of the longest record in the block, excluding the logical record header (if any).

If this is not a tape file, the tape logical record size entry is ignored.

If the file is not currently reserved and logical record size is not specified (i.e., the field contains all zeros), a value is computed based on the values for block size, tape data format, and tape BSN indicator.

If the file is already reserved and logical record size is not specified, the previously specified or computed value is not changed.

number of buffers

A 1-byte binary value specifying the number of buffers to be allocated in a buffer pool specific to this file. Use this argument carefully and only when the system-generated or operator-defined public buffer pools or the user-defined private buffer pools (i.e., for this user or task group only) are insufficient to satisfy this file's buffering requirements.

Default: If the file cannot be assigned to a public or private pool, a file-specific pool is created. This pool contains two buffers for indexed sequential files and one buffer for all other types of disk files. In addition, if the file has any alternate indexes, one additional buffer is allocated for all the indexes. For tape files, the default is two buffers.

For more details on buffer pool concepts see the Create Buffer Pool command description in the Commands manual.

Buffer space is allocated at open-file time and returned at close-file time when the file is accessed through data management macro calls. Buffer space is not required if the file is accessed through storage management macro calls.

This entry does not apply to device files; buffers are allocated according to information specified in DEVICE directives at system building time.

tape file sequence number

A 1-byte binary code, applicable only to tape files, that indicates the position of the file on a tape volume set; can have the following values:

- 00 - The desired file is the next file on the volume
- FF - Search for the file in a forward direction
- nn - Relative sequence number of the file on the volume set.

If a pathname is specified, it is used with the tape file sequence number to perform a file search when an Open File macro call is issued. (The maximum file-name length is 17 characters.)

See the description of the Open File macro call for a discussion of tape search rules.

If FF is specified, the search is performed from the current position on the volume to the volume set end-of-data.

If the file is not tape-resident, this entry is ignored.

tape label format

A 1-byte code, applicable only to tape files, that indicates the tape label format.

0 - No label format specified
1 - Tape has standard EBCDIC/ANSI labels
2 - Tape is not labeled

If the file is not tape-resident, this entry is ignored.

tape data type

A 1-byte code, applicable only to tape files, that specifies the data type.

0 - No data type specified
1 - Honeywell
2 - ANSI Level 3
3 - EBCDIC (IBM-compatible)

If the file is not tape-resident, this entry is ignored.

tape data format

A 1-byte code, applicable only to tape files, that indicates the data format.

0 - No format specified
1 - Fixed-length records
2 - Variable-length records
3 - Undefined records
4   Spanned records

If the file is not tape-resident, this entry is ignored.

tape file options

A 1-byte code, applicable only to tape files, that defines packing, EBCDIC/ANSI translation, parity, file section number, and block sequence number usage.

Bit(s)	Meaning
0, 1	0 = Not specified 1 = For 7-track tape: No Packing 2 = For 9-track tape: EBCDIC/ANSI translation
2, 3	0 = Parity not specified 1 = Odd parity 2 = Even parity

Packing and parity bits are meaningful only for 7-track tapes to be opened for storage management (block level) access.

Bit(s)	Meaning
4	0 = Not specified 1 = File section number supplied
5	Must be zero
6, 7	0 = No BSN specified 1 = BSN not supplied 2 = BSN supplied

If 2 is specified, a BSN is assumed to be present on input; on output, a BSN will be inserted.

If the file is not tape-resident, this entry is ignored.

tape file section number

A 2-byte field specifying the relative file section number. Applies only to multi-volume tape files.

tape retention period

A 2-byte value, applicable only to tape files, that specifies the tape retention period in days. Zeros in this field indicate that the retention period is not specified.

If the file is not tape-resident, this entry is ignored.

DESCRIPTION:

This macro call reserves the file with proper access rights for use by the data management and storage managememt macro calls. It can also be used to alter concurrency or tape definition arguments established by a previous Get File macro call, provided the file is not already open (see the Open File macro call) in the task group in which you are executing.

The file can be specified (in the argument structure) by an LFN only, a pathname only, or both an LFN and a pathname.

- If specified only by an LFN, the LFN must have been previously associated with a pathname (see the Associate File macro call) or it must have been previously assigned to the file through the Get File or Create File function.

- If only a pathname is specified, the file is reserved without a unique LFN. The only requests that can use the file are those that can reference the file by a pathname only, e.g., Get File, Get File Information, Delete File, Remove File.

- If a pathname is specified and the LFN field contains a value of -1 (FFFF), the system assigns a unique LFN from the task's LFN pool. In this case, it is the user's responsibility to return the LFN to the pool (by a Remove File macro call) when the LFN is no longer needed. In assigning a unique LFN from the pool, the system selects the highest LFN available for assignment and sets it in the LFN entry in the argument structure, overlaying the previous contents (FFFF). The user must move this value to other structures (argument structures or FIBs) as required.

- If both an LFN and a pathname are specified, the file is reserved and assigned to the LFN. This LFN-to-file assignment remains in effect until the file is removed from the task group or another Get File macro call that specifies the same LFN is issued.

The Get File macro call allows the user to append ASCII characters to a previously associated pathname or to a partial pathname (see the Associate File macro call). This is done by prefixing the string of characters to be appended (i.e., pointed to by the pathname pointer entry) with a colon (:). The system replaces the colon with the previously associated pathname, as follows:

Previously Associated Pathname	Characters to be Appended	Result
none	:	Working directory
none	:ABC	ABC
^VOL1>UDD	:>FILE01	^VOL1>UDD>FILE01
^VOL2>	:FILE02	^VOL2>FILE02

As stated above, the Get File macro call can be used to alter concurrency control. In doing so, note the following:

- If type 0 concurrency control is specified the first time the file is reserved in a task group, the system reserves the file for exclusive use (type 3 concurrency).
- If type 0 concurrency control is specified and the file was previously reserved in this task group, the previous concurrency control does not change. This could occur if the user wanted to change the tape file definition argument or to address the file through a different LFN.
- A Get File macro call does not alter the concurrency control established through a previously issued Get File command. Only by issuing another Get File command can the concurrency established through a previous Get File command be altered.
- If device level access is desired (i.e., the pathname is in the form !dev_name[volid]), the following rules apply:
  - Type 3 exclusive concurrency control is set, regardless of the value specified in concurrency control entry, if the pathname is specified as:

        !dev_name

    No volume label validation is performed. Note that tapes are always reserved with type 3 concurrency.
  - For disk volumes, type 2 concurrency control is set, regardless of the value specified in the disk concurrency control entry, if the pathname is specified as:

        !dev_name>volid

    The volume label is read and validated; if a mismatch occurs, the action specified in the disk mount option argument occurs.

- To change disk device-level concurrency control, a Remove File macro call must first be issued, and then a new Get File macro call.

- The following rules apply to directories reserved through a Get File macro call:

  - If the directory is reserved exclusively (type 3 control), all subdirectories and files inferior to the directory are also held exclusively. For example, a Get File macro call having a pathname of volid (i.e., only the volume directory supplied) and a concurrency of 3, would reserve the entire volume for exclusive use through normal file, data, and storage management facilities. This is not the same as device level access (SPD dev_name), since it permits normal access by the user at the file level.

  - If the directory is not reserved exclusively, read/write share concurrency control (type 5) is set, regardless of the specified value.

  - Directory-level concurrency cannot be changed by issuing a new Get file macro call. To change directory-level concurrency, a Remove File macro call must first be issued, and then a Get File macro call.

The record lock option is a mechanism that provides temporary multi-user interference protection for shared file access. When a record is accessed by a task group, it is locked (by locking the control interval(s) in which the record is contained) on a first-come first-served basis. If another user is sharing the file, he will be denied access to the record (and other records contained in the same control interval) until the previous user unlocks the record (through the Clean Point macro call). Record locking can be set as a permanent file attribute through the Create File (CR) command, the Modify File Attribute (MFA) command, or the Modify File ($MDFIL) macro call. Record locking is then automatically initiated at each file reservation request. If record locking is not specified as a permanent file attribute, when set at get-file time it remains in effect only until the file is removed. The user should consider the following points when using record locking:

- An LFN within a task group uniquely identifies a user for record locking purposes and thus provides interference protection between task groups. Since tasks within a task group may agree to access a file through different LFNs, interference protection is provided when the cooperating tasks agree to respect the LFN assignments.

- Lock requests are valid only for disk-resident files (a request to lock any other file is ignored). Directories and entire disk volumes cannot be reserved with lock. The primary index of an indexed file is never locked (since once created, it is never updated).

- Files reserved with lock cannot be modified (written) through storage management access.

- Records are locked in "shared read/exclusive write" mode, which is explained as follows:

  - For purposes of record locking, file system users may be classified as "readers" and "updaters". Readers have opened the file without update permission, since they need only to read records. The are not concerned if other users are reading the same record, but do not want to read a record while it is being updated.

  - Updaters have opened a file with update permission. They want to be the only users of a specific record. The record lock facility makes sure that a given record is accessed by only one updater or by n readers at one time.

  - Accordingly, readers set read locks, updaters set write locks. A given record may have any number of read locks, or it may have only one write lock.

- Once specified, locking is automatic. Any access (read or write) will cause an appropriate lock. The number of locks that can exist at one time is limited only by the amount of memory dedicated to the lock pool (i.e., the area of memory where locked records are recorded). (This area is defined at system building; see the <u>Building and Administration</u> manual.)

When record contention occurs (reader attempts to lock a record already locked by a writer, or writer attempts to lock a record already locked by another reader or writer), the system normally performs a wait until the record is unlocked. However, the wait is not performed under the following conditons:

- When the lock request would cause a deadlock. For example, a deadlock would occur if a user wanted a record that a second user had already locked, and the second user was waiting for a record that the first user had locked. Return code 022B indicates record lock deadlock. Normal user response to this return code is to issue a Roll Back macro call to recover the updates done since the last Clean Point macro call was issued, and then to start over.

- When the user has specified the "no wait" option. There may be conditions under which an application does not want to wait for the records to be unlocked. The "no wait" option allows the user to receive an 022B return status (indicating that the record is locked) rather than be suspended.

- Record locking is initiated by the first user who reserves the file with the record lock option. File reservation is denied if the file is already reserved for writing without record locking.

  To initiate record locking, the file must be reserved for writing (concurrency type 3, 4, or 5).

- A user who reserves a disk file with the "no lock" option can read records without applying any record locks. The records can be read even if they are currently locked and are being updated by other users. Data integrity is not guaranteed, and the user is not allowed to do any updates through the specified LFN.

- The Clean Point macro call is used to unlock records. If records are not unlocked, lock pool overflow or a deadlock record condition will probably result. (See the Clean Point macro call for details.)

- The user must provide for all actions to be taken when notified of lock pool overflow or record lock concurrency conflict. When a record deadlock condition occurs, the user should restart the current phase by unlocking all records and recycling to the point where the interrupted sequence began. (In so doing, some records may be updated, thereby making a simple recycling unsatisfactory.) From a practical standpoint, all records to be updated or deleted should be read first to ensure access; all inserts should be done first to make the unwinding of a transaction easier to manage.

If an operator terminal is not included in the system, or if messages to the operator terminal have been suppressed (through a Console Message Suppression macro call), a Get File macro call issued to reserve a volume that is not mounted results in an 020C (volume not mounted) error return.

If a file is reserved through an LFN and a subsequent Get File macro call is issued specifying the same LFN, this LFN becomes associated with the new file. The previously reserved file will remain reserved for the task group until it is removed (through the Remove File macro call).

Since the Get File macro call performs so many functions, it should be used as infrequently as possible. A Get File followed by multiple Open File/Close File sequences is much more efficient than a Get File, Open File, Close File, Remove File, Get File, etc.

Offset tags for the argument structure block can be defined by the Get File Parameter Structure Block Offsets macro call.

Tape file arguments are meaningful only when (1) a labeled tape file is being created (opened) in RENEW mode or (2) an unlabeled tape file is being processed for input/output. For labeled tapes being opened for input (PRESERVE mode), the various tape parameters are taken from the file header labels.

For tape files, default block size (BKSZ) and logical record size (LRSZ) are computed as shown in Figure 2-1.

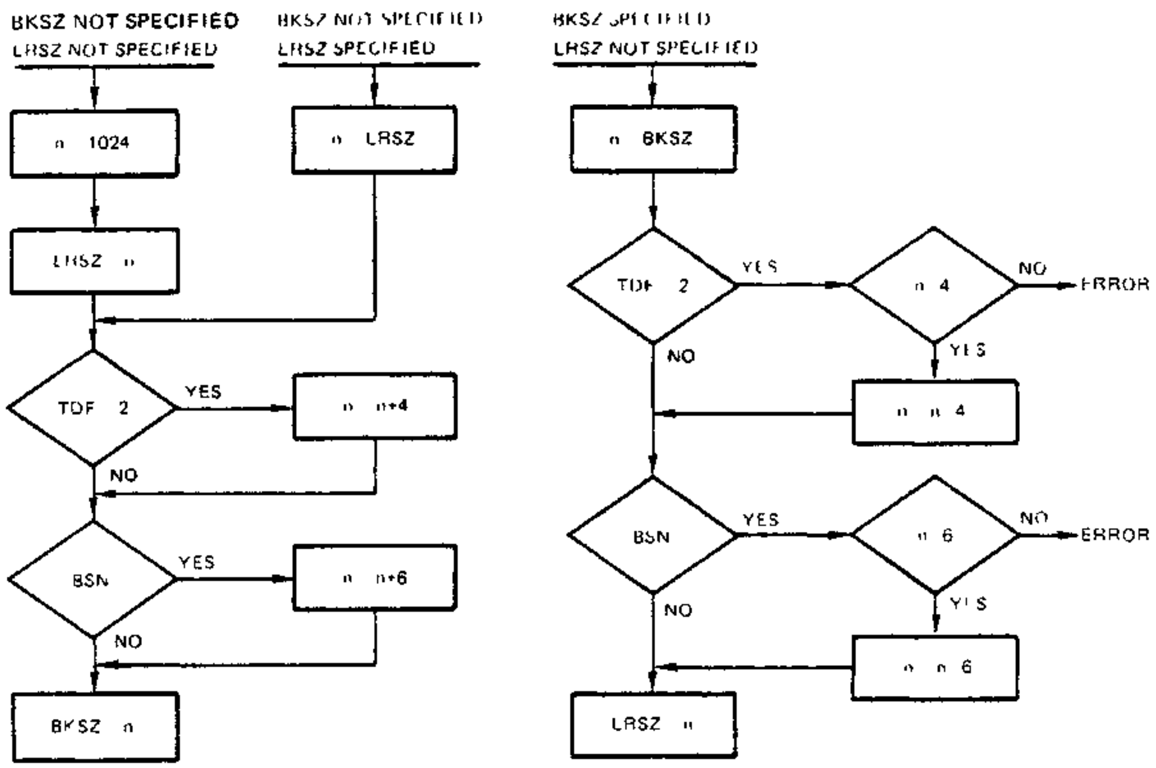


Figure 2-1. Calculating Block Size (BKSZ) and Logical Record Size (LRSZ) for Tape File

0205 - Invalid argument

0206 - Unknown or invalid LFN

0208 - LFN or file currently open in same task group

0209 - Named file or directory not found

020C - Volume not found

0210 - LFN conflict

0211 - Unable to establish a unique LFN

0213 - Cannot provide requested file concurrency

0222 - Pathname cannot be expanded; no working directory

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

022A - Record lock area overflow or not defined

022C - Access control list violation

022E - Record lock concurrency conflict

0238 - Invalid file description.

Example 1:

In the following example, the Get File macro call identifies an argument structure that contains the appropriate arguments to reserve the indexed file created in the example for the Create File ($CRFIL) macro call (i.e., FILE_A), with type 5 concurrency control (read/write share) and record locking. The argument structure was built as follows:

```
WRTFIL     DC      Z'0005'    See "Assumptions for File
                              System Examples" in Appendix A.
                              (The pathname is defined in
                              the example for the Create File
                              macro call.)

           DC      <IDX01
           RESV    2-$AF
           DC      Z'8051'    READ/WRITE SHARE; RECORD
                              LOCKING:  ISSUE MOUNT REQUEST
           RESV    2,0        INGORED
           DC      Z'0200'    BUFFERS=2
           RESV    4,0        IGNORED
```

It is assumed that the following macro calls have been issued before the Get File macro call is issued:

```
$CRDIR   !SUBDIR   (See Create Directory macro example)

$CRFIL   !FILE_A   (See "Assumption for File System
                   Examples" in Appendix A.)
```

The Get File macro call altering FILE_A concurrency from exclusive to share can be specified as follows:

```
$GTFIL   !WRTFIL
```

Example 2:

In this example, the Get File macro call is used to append characters to an incomplete pathname defined as follows:

```
DIRPTH   DC    '^VOL03 SUBINDEX.AΔ'   (See Create Direc-
                                      tory macro example)
```

This pathname has been associated with the LFN as follows:

```
$ASFIL   !FILE_X
```

where the argument structure labeled FILE_X has been defined as follows:

```
FILE_X   DC     Z'00A3'   LFN=163
         DC     <DIRPTH   PATHNAME   '^VOL03 SUBINDEX.AΔ'
         RESV   2-$AF
```

Assuming that the above definitions have been made, the following argument structure identifies the characters to be appended to the incomplete path (DIRPTH):

```
WTFIL2    DC      Z'00A3'    LFN=163
          DC      <IDX02     PATHNAME POINTER
          RESV    2-$AF
          DC      Z'0301'    EXCLUSIVE:  ISSUE MOUNT REQUEST
          RESV    2,0        UNSPECIFIED
          DC      Z'0200'    BUFFERS=2
          RESV    4,0        IGNORED
```

The pathname labeled IDX02 is defined as follows:

```
IDX02     DC      ':>FILE_C '
```

The result of specifying the above structure (WTFIL2) in the following Get File macro call is to reserve the file identified by the pathname VOL03>SUBINDEX.A>FILE_C with exclusive concurrency control:

```
$GTFIL    !WTFIL2
```

However, before FILE_C can be opened and accessed, it must exist in the file system hierarchy (i.e., it must have been created as defined in the Create File macro call example).

Alternate Index Specific Information

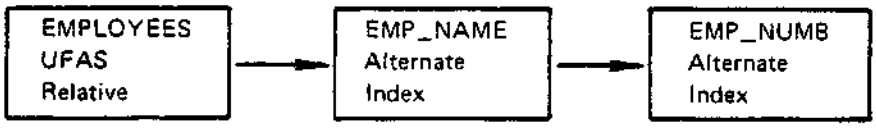


Figure 2-3. Example of Alternate Index Use

A UFAS data file, with one or more alternate indexes, can be used either as a standard data file or as an indexed file, depending on the pathname specified at Get File time.

1. Specifying the pathname of the data file:

   By reserving the data file directly, records in that file can be used sequentially through the key supported by the file organization or by an alternate key supported by one of its indexes. In the example above, by specifying the pathname "EMPLOYEES", records in EMPLOYEES can be used sequentially by a relative record number, by a "simple" key (CI and line number), or by one of the alternate keys employee name or employee number.

2. Specifying the pathname of an index:

   By reserving the file through an alternate index, the data file can be used as a standard "indexed" file. In the example above, by specifying the pathname "EMP_NAME", records in EMPLOYEES can be used sequentially (ordered by employee name), by a primary key of an employee name, or by an alternate key of an employee number.

GET FILE ACCESS RIGHTS ($GAFIL)

Function Code: 10/73

Equivalent Command: List Access (LAC)

List the access rights of a specified user to a specified disk file or directory.

FORMAT:

```
[label]   $GAFIL   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the address of the argument structure described below. The argument structure must contain the following entries in the order shown. (Entries marked by an asterisk are provided by the system.)

logical file number

A 2-byte logical file number (LFN) that refers to the file for which the function lists access. If specified, the LFN must be a binary number in the range 0 through 255. Two ASCII blanks (2020 in hexadecimal) indicate that the file's LFN is not supplied. If this entry contains blanks, the pathname pointer, described below, must be supplied.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the file or directory for which the function lists access. If zeros are entered in the pathname pointer field, the LFN must be supplied in the preceding field.

user id pointer

A 4-byte pointer to a field that identifies the user whose access rights the function retrieves. Zeros indicate that the user id pointer is not supplied.

The field that identifies the user comprises one to three of the following subfields:

person
person.account
person.account.mode

Each person and account subfields can be from 1 through 12 characters long; the mode subfield, from 1 through 3 characters long. The subfields must be separated from each other by a period; the last subfield must be followed by a space.

*access rights

A 2-byte field indicating the access rights of the specified user to the specified file, as shown below.

Bit	Meaning
0	1 - Access rights are for a directory
0	0 - Access rights are for a file
1	1 - Access rights returned result from an empty access control list
	0 - Access rights returned do not result from an empty access control list
2-11	Zeros; reserved for future use
12	1 - Create access for directories
	0 - Execute access for files
13	1 - Modify access for directories
	0 - Write access for files
14	1 - List access for directories
	0 - Read access for files
15	1 - Null access
	0 - Access, as specified by other bits in field.

DESCRIPTION:

This macro call retrieves the access rights of a user to a <u>disk</u> file or directory; the function does not apply to tape or device files.

A disk file or directory can be specified in the parameter structure block by either an LFN or a pathname. If a file is specified by an LFN, the file must have been previously assigned to that LFN by means of the Get File or Create File macro calls or the equivalent Execution Control Language (ECL) commands.

If a user id is not specified in the parameter structure block, the file access rights retrieved are those for the current user (i.e., the user issuing this monitor call).

NOTES

1. If the parameter structure address is coded, the system loads the address of the structure into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Media error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument (incorrect user id)

   0206 - Unknown or invalid LFN

   0209 - Named file or some superior directory not found

   0228 - Invalid file type

   022C - Access control list violation.

GET FILE ACCESS RIGHTS PARAMETER STRUCTURE BLOCK OFFSETS

GET FILE ACCESS RIGHTS PARAMETER STRUCTURE BLOCK OFFSETS ($GAPSB)

Associated Macro Call: $GAFIL

FORMAT:

[label] $GAPSB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Field
0	Logical File Number
1 2	Pathname Pointer
3 4	User id Pointer
5	Access Rights
6 7 8 9 10 11 12 13 14 15	Reserved

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
L_LFN	0	Logical File Number (LFN)
L_PTHP	+1	Pointer to pathname
L_UIDP	+3	Pointer to user id
L_IND	+5	Access rights indicator word
L_SZ	16	Size of structure (in words); not a field in block

GET FILE ACCOUNTING INFORMATION ($GTACT)

Function Code: 10/42

Equivalent Command: None

Retrieve the following information from the file accouning information record(s) of a specified file:

- Date/time created
- User id of creator
- Date/time last loaded
- Date/time last modified
- User id of modifier
- Date/time last accessed
- Retention period.

FORMAT:

```
[label]   $GTACT  [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown. A description of each entry follows this list:

Argument Structure Entry	Size (in bytes)
logical file number	2
pathname pointer	4
creation information block pointer	4
access information block pointer	4
reserved for future use; must be zeros	18

logical file number

A 2-byte logical file number (LFN) that refers to the specified file; must be a binary number in the range 0 through 255 or ASCII blanks (X'2020') if an LFN is not specified. If this entry contains blanks, the pathname-pointer entry (below) must point to a pathname.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the file to be reserved. Binary zeros in this entry indicate that a pathname is not specified.

creation information block pointer

A 4-byte address that may be any address form valid for an address register; points to the creation information block described in Table 2-6. Zeros in this entry indicate that the caller does not wish to retrieve information about the specified file's creation and retention.

access information block pointer

A 4-byte address that may be any address form valid for an address register; points to the access information block described in Table 2-7. Zeros in this entry indicate that the caller does not wish to retrieve information concerning access of the specified file.

DESCRIPTION:

This function reads information from a specified file's accounting information records into the creation and/or access information blocks pointed to by the function's argument structure. These records, which reside in a directory entry, are described in the <u>Data File Organizations and Formats</u> manual.

Accounting information records will exist for a file only if the creator specified the Accounting or Retention argument of the Create File command.

The file can be specified by either an LFN or a pathname. If an LFN is specified, that same LFN must have been previously specified when the file was reserved, by means of a Get File macro call.

The "creation date" field of the creation information block specifies the time at which a file's directory entry is created. This time does not necessarily correspond to the time at which the file is first loaded with data.

The "modification" date field of the access information block specifies the time at which the file was last opened for a write or load operation. This time does not necessarily correspond to the time at which the file was loaded or last written to, since a user who opens a file with the intention of performing either operation might not actually do so.

Values returned in the "modification date" and "access date" fields of the access information block can refer to either a write or a load operation. Both of these operations can involve access to or modification of a file. A value in the "access date" field can refer, additionally, to a read operation. To determine the signifiance of a value returned in these fields, see Figure 2-4. The symbols "a","b", and"c" each represent a different date/time.

File Operations	Date/Time Fields		
	Access	Modification	Load
Load	a	a	a
Write	b	b	-
Read	c	-	-

Figure 2-4. Interpreting Access Information

When a file is loaded, the date/time of that operation (represented by "a") is entered in all three date/time fields. The date/time of a write operation ("b") is entered in the "access date" and "modification date" fields. The date/time of a read operation ("c") is entered in the "access date" field. Thus, if the value of all three date/time fields are equal, those values refer to the last load operation. The values of the "access date" and "modification date" refer to the last write operation if they are equal but differ from the value of the "load date" field. Finally, if the value of the "access date" field is unique, it refers to the last read operation.

Table 2-6 shows the contents of the creation information block used by the Get File Accounting function.

Table 2-6. Creation Information Block for $GTACT

Field Name	Size (in bytes)	Meaning
RFU	2	Reserved for future use.
Creation Date	6	Date and time, in internal format, when the file was created.
User id	28	User identification of the file's creator. Person id is 12 characters, account id is 12 characters, and mode id is 3 characters.
Load Date	6	Date and time, in internal format, when the file was last loaded (i.e., opened in RENEW mode). Zeros indicate an unknown load date.
Retention date	6	Date and time, in internal format, when the retention period expires. Zeros indicate no retention period.

Table 2-7 shows the contents of the modification information block used by the Get File Accounting Information function.

Table 2-7. Access Information Block for $GTACT

Field Name	Size (in Bytes)	Meaning
RFU	2	Reserved for future use.
Modification Date	6	Date and time, in internal format, when the file was last modified. Zeros indicate an unknown modification date.

0226 - Not enough user memory for buffers or structures

0228 - Invalid file type

022C - Access Control List violation

0238 - Invalid file description information

023D - File does not have accounting information.

GET FILE INFORMATION ($GIFIL)

Function Code: 10/60

Equivalent Command: None

Retrieve information about the specified file. The file is identified by supplying either a logical file number (LFN) or a pathname. This macro call returns information such as file type, device type, and, optionally, other file attributes (logical record size, block or control interval size, space allocation, etc.). In addition, the user can receive a description of the keys of an indexed or random file.

FORMAT:

```
[label]   $GIFIL   [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown. (Entries marked with an asterisk (*) are provided by the system. The user must supply the other entries.) The size of each entry, whose descriptions follow this list, is as follows:

Argument Structure Entry	Size (in bytes)
logical file number	2
pathname pointer	4
*device type	2
*logical resource number	2
*file type	1
*data format	1
file attribute pointer	4
*terminal software device id or *file options	2
*data attributes	2
key or record descriptor pointer	4
*key or record descriptor size	2
*number of related files	2

logical file number

A 2-byte LFN used to refer to the file; must be a binary number in the range 0 through 255, or ASCII blanks (X'2020'), which indicate that an LFN is not specified. If this entry contains blanks, the pathname-pointer entry (below) must point to a pathname.

pathname pointer

A 4-byte address that may be any address form valid for an address register. If an LFN is specified in the first entry, this entry (optionally) points to a 58-byte field in main memory into which the system places the full absolute pathname associated with the LFN. If the LFN entry contains ASCII blanks, this entry points to the location where a pathname (which must end with an ASCII space character) is found. This pathname identifies the file for which the system is to retrieve information. Zeros in this entry indicate that the pathname is not to be returned. If zeros are specified, the LFN entry (above) must contain a nonblank value.

*device type

A 2-byte entry into which the system places the 4-digit hexadecimal device code of the device containing the file. The devices, their codes, and marketing identifiers include but are not limited to the following:

Peripheral Device	Device Type Code	Marketing Identifier
Card Reader	2008	CRU9101/9102/9103/9104
Card Reader/Punch	2088	CCU9101/PCU9101
Teleprinter	2018 2019	TTU9102 TTU9101
CRT Keyboard Console	2020	DKU9101
Keyboard Typewriter Console	2018	TWU9101

Peripheral Device	Device Type Code	Marketing Identifier
Mass Storage Unit	2360	MSU9101/9105 (40-megabyte)
	2361	MSU9102/9106 (80-megabyte)
	2362	MSU9103 (143/127-megabyte)
	2363	MSU9104 (288/256-megabyte)
Diskette	2010	DIU9101/9102
Cartridge Disk	2330	CDU9101
	2331	CDU9102
	2332	CDU9103
	2334	CDU9104
Cartridge Module Disk	2380	CDU9121 (Removable; 16-megabyte)
	2380	CDU9122 (Removable; 16-megabyte)
	2380	CDU9123 (Removable; 16-megabyte)
	2380	CDU9124 (Removable; 16-megabytye)
	2381	CDU9122 (Fixed; 16-megabyte)
	2383	CDU9123 (Fixed; 48-megabyte)
	2385	CDU9124 (Fixed; 80-megabyte)
	2388	CDU9125 (Removable; 8-megabyte)
	2389	CDU9125 (Fixed; 8-megabyte)
Serial Printer	2004	PRU9101
	2006	PRU9102
Line Printer	2000	PRU9104/9106
	2001	same as above but with Option PRF9102
	2002	PRU9103/9105
	2003	same as above but with Option PRF9102

Peripheral Device	Device Type Code	Marketing Identifier
Magnetic Tape	2045	MTU9104 - 9-track, 800 bpi, 45 ips
	2046	MTU9105 - 9-track, 800 bpi, 75 ips
	204D	MTU9109 - 9-track, 800/1600 bpi, 45 ips
	204E	MTU9110 - 9-track, 800/1600 bpi, 75 ips
	2049	MTU9114 - 9-track, 1600 bpi, 45 ips
	204A	MTU9115 - 9-track, 1600 bpi, 75 ips
	2079	MTU9112 - 7-track, 556/800 bpi, 45 ips
	207A	MTU9114 - 7-track, 556/800 bpi, 75 ips

*logical resource number

A 2-byte entry into which the system places the LRN that corresponds to the device on which the specified file is located.

*file type

A 1-byte entry into which the system places a code identifying the file organization of the specified file, as follows:

D - Directory file
S - UFAS sequential disk file
R - UFAS relative disk file
I - UFAS indexed disk file
C - UFAS random (CALC) disk file
V - UFAS dynamic disk file
X - UFAS alternate index
A - UFAS I-D-S/II data base area
T - Tape device
0 - Device file (see device type)
2 - Fixed-relative disk file without deletable records
5 - Fixed-relative disk file with deletable records
-1 - IBM diskette

*data format

A 1-byte entry into which the system places a code identifying the format of the data, as follows:

F - Fixed-length records
V - Variable-length records (binary count size)

file attribute pointer

A 4-byte address of a 32-byte block in main memory into which the system can place file-attribute information, as described below; may be any address form valid for an address register or zeros (which indicate that the information is not required). The file attribute block is described in Tables 2-8 through 2-11.

*terminal software device id or disk file options

For terminal files, this 2-byte field is set by the system to a 16-bit software device descriptor, which categorizes a device, both logically and physically, by major and minor codes. These codes are listed under the description of the Get Device Information ($GIDEV) macro call.

For disk files, this 2-byte field is set by the system to indicate options described in Table 2-13.

*data attributes

A 2-byte field for disk files set by the system to indicate the type of data recorded on the file, the type of terminal control information present in each record of the file, and the conformity of the file's data and format to GCOS 6 standards. The data attribute field is described in Table 2-14.

key or record descriptor pointer

A 4-byte address of an 18-byte field in main memory into which the system can place key-descriptor information, as described below; may be any address form valid for an address register, or zeros (which indicate that the information is not required).

*key or record descriptor size

A 2-byte field specifying the size (in words) of the user-declared area to receive record descriptor information.

If the record descriptor pointer above is null, then the system returns here the size required for record descriptor information. If the record descriptor is not null, this field should be set to define the size in words of the specified record descriptor area (if zero, a size of nine words is assumed).

*number of related files

A two-byte field indicating the number of alternate indexes associated with the specified file. The Get Name macro call can be used to retrieve the names of the alternate index files.

Table 2-8. File Attribute Information for Device Files

Field Name	Size (bytes)	Description
Logical Record Size	2	The maximum size of a logical record in bytes. This is a unit of data transfer for a device file.
Block Size	2	Same as logical record size.
File Indicators	2	Indicators that define how the device is currently being processed through the file system: Bit 0-2: input/output capabilities: 100 = Input only 010 = Output only 001 = Input and output
File Indicators		Bit 3-4: Detabbing option (i.e., whether or not spaces will be substituted for tabs on output): 10 = Detabbing done 01 = No detabbing done Bit 5-8: Asynchronous I/O option: 1000 = Asynchronous input (read ahead) 0100 = Asynchronous output (double buffered) 1100 = Asynchronous input and output 0010 = Synchronous input (no read ahead) 0001 = Synchronous output (single buffered) 0011 = Synchronous input and output Bit 9-10: System buffer option: 10 = Use system buffer for synchronous I/O 01 = Do not use system buffer (i.e., use the user's record area) Bit 11-12: Transfer mode option: 10 = Field transfer 01 = Block transfer

Table 2-8 (cont). File Attribute Information for Device Files

Field Name	Size (bytes)	Description
File Indicators (cont.)		Bit 13-14: Restart option: 10 = Automatic reconnect on powerfail or line-drop condition 01 = Return error to user on powerfail or line-drop condition Bit 15: Reserved for future use (zero)
Device Specific Word 1	2	The device-specific word to be used for connect/disconnect orders.
Device Specific Word 2	2	The device-specific word to be used during read/write orders.
Initial Device Specific Word 1	2	The initial setting of device-specific word 1 as specified at system generation time.
Initial Device Specific Word 2	2	The initial setting of device-specific word 2 as specified at system generation time.
Reserved for Future Use	18	Zeros.

Table 2-9. File Attribute Information for Tape Files

Field Name	Size (bytes)	Description
Logical Record Size	2	The maximum size of a logical record in bytes. This size does not include any logical record header information.
Block Size	2	The size of a block in bytes. This size includes logical record and block header information.
Tape Padding Character	1	Character to be used as padding to fill out the last block.
Tape File Sequence Number	1	The relative sequence number of the file.
Tape Label Format	1	X'01': Standard labels X'02': Unlabelled tape
Tape Data Types	1	X'01': Honeywell X'02': ANSI Level 3 X'03': EBCDIC (IBM)
Tape Data Format	1	X'01': Fixed-length records X'02': Variable-length records X'03': Undefined records X'04': Spanned records
Tape File Options	1	Defines packing, EBCDIC/ASCII translation, parity, file section number, and block sequence number options: Bit 0-1: 0 = Does not apply 1 = No packing for 7-track tapes; EBCDIC/ASCII translation for EBCDIC 9-track tapes 2 = Pack mode for 7-track tapes; no EBCDIC/ASCII translation for EBCDIC 9-track tapes Bit 2-3: 0 = Does not apply 1 = Odd parity 2 = Even parity Bit 4-5: Zeros Bit 6-7: 1 = Block sequence number (BSN) not supplied 2 = 6-character BSN supplied

Table 2-9 (cont). File Attribute Information for Tape Files

Field Name	Size (bytes)	Description
Tape File Section Number	2	The relative section number of the file.
Tape Retention Period	2	Tape file retention period in number of days.
Reserved for Future Use	18	Zeros.

Table 2-10. File Attribute Information for Disk files

Field Name	Size (bytes)	Description
Logical Record Size	2	The maximum size of a logical record in bytes. This size does not include the logical record header.
CI Size	2	For unified files, the size of a control interval (CI) in bytes. This size includes both CI and logical record header information. For fixed-relative files, the size of a physical sector.
Current Allocation in Size	2	The value of the highest numbered CI the file which contains data.
Allocation Growth Size	2	The number of additional CIs to be allocated to the file whenever it becomes necessary to do so. This is the size of an additional extent to be added to the file.
Maximum Allocation Size	2	The maximum number of CIs that can be allocated to the file. This is the limit to which the file can grow. Zeros returned for I-D-S/II areas.
Amount of Free Space per CI	2	For Unified File Access System (UFAS) indexed files, the number of bytes to be left free in each CI at file loading time. This supplies space for records to be inserted without causing overflow. For UFAS alternate indexes, the number of bytes to be left free in each index CI at index load time. This supplies space for new index entries without forcing the index to reorganize itself (i.e., without forcing a CI split to occur).

Table 2-10 (cont). File Attribute Information for Disk files

Field Name	Size (bytes)	Description
or Inventory Threshold		For UFAS dynamic and random files and I-D-S/II areas, the percent of space in a data CI which must be filled before inventory information is updated. Inventory information is used to determine the amount of free space available in data CIs (see the Create File macro call). For I-D-S/II areas, zeros indicate that the file has no inventory. This field is zero for other file formats.
Local Overflow,	2	For UFAS indexed files, a value that indicates how often a local overflow CI has been allocated when the file was last loaded.
Hash Results,		For UFAS random files, the number of possible hash results (see the Create File macro call).
or CALC Interval		For I-D-S/II areas, the number of records initially set aside for each CALC set. The number of possible hashing results is equal to the maximum number of data records in the area divided by the CALC interval. This field is zero for other file formats.
Number of Record Descriptors	2	For UFAS indexed files, random files, alternate indexes and I-D-S/II areas, the number of record descriptors in the file. This field is zero for other file formats.
Reserved for Future Use	4	Zeros.

Table 2-11. Additional File Attribute Information for I-D-S/II Areas Only

Field Name	Size (bytes)	Description
Number of Data Records	4	For I-D-S/II areas, the maximum number of data records which can exist in the file. This field is zero for other file formats.
Number of Records Per CI	2	For I-D-S/II areas, the maximum number of records per CI. This field is zero for other file formats.
I-D-S/II Options	1	For I-D-S/II areas, identifies the hashing algorithm: 0 = GOCS 66-compatible hashing algorithm. 1 = GCOS 6-MOD 600 Release 110 hashing algorithm.
Global Pointer Size	1	For UFAS I-D-S/II areas, the size of a global pointer (data base key) -- 2, 3, or 4 bytes. This field is zero for other file formats.
Global Pointer Base	4	For I-D-S/II areas, the global pointer value (data base relative record address) assigned to the first record in the area.

Table 2-12. Record Descriptor Information for UFAS Indexed Files, Random Files, Alternate Indexes, and I-D-S/II Areas

Field Name	Size (bytes)	Description
Record Descriptor Size	2	The actual size (in words) of all the record descriptor information for the file. This includes the size field, all the record descriptors (one per record type defined), as well as all the key components defined for each record descriptor.
Record Type	2	The record type that uniquely identifies the record described by this record descriptor. Bit 0: 1 = Duplicate keys allowed. 0 = Duplicate keys not allowed. Bits 1-3: Must be zero. Bits 4-15: Record type must be zero for random files, indexed files and alternate indexes.
Number of Key Components	1	If the record contains a key, the number of components in the key. This field is 1 for UFAS indexed files.
Reserved for Future Use	1	Zeros.
Record Address Range	8	For I-D-S/II areas, the minimum and maximum record numbers for storing records of this record type in the file. This field is zero for other file formats.

Table 2-12 (cont). Record Descriptor Information for UFAS Indexed Files, Random Files, Alternate Indexes, and I-D-S/II Areas

NOTES

1. Data type, size, and location constitute one key component descriptor that can be repeated as many times as the number of components per key. Only one component per key is currently supported for UFAS indexed files.

2. Record type, number of key components, number of non-CALC sets, record address range, key component data type, size, and location constitute one record descriptor, which can be repeated as many times as the number of record descriptors. Key component fields can also be repeated (as mentioned above) within a record descriptor.

Table 2-13. Disk File Options Field of $GIPSB

Bit	Option
0	Record Lock Option: 1 = Records are locked allowing n readers or one writer. 0 = Records are not locked.
1	Record Format Option: 1 = Supports both fixed and variable length records. 0 = Supports only fixed length records.
2	Immediate Update Option: 1 = The disk is updated whenever a logical record is updated. 0 = Updates are kept in memory until one of the following occurs: buffers are full, a cleanpoint is reached, or the file is closed.
3	File Recovery Option: 1 = "Before images" of updates are saved to recover the file to its last consistent state. 0 = "Before images" are not saved.
4-5	(MBZ)
6	Damaged File Indicator: 1 = File is not damaged. 0 = The file's data content is in a damaged or inconsistent state.
7	Write Protect Option: 1 = Write operations are allowed. 0 = Write operations are not allowed.
8	(MBZ)
9	File Restoration Option: 1 = "After images" of updates are recorded for later restoration of the file to its last consistent state. 0 = "After images" are not recorded.
10-15	(MBZ)

Table 2-14. Disk Data Attribute Field of $GIPSB

Bit	Data Attribute
0-3	Data Code Attribute: 0000 = Undefined data 0001 = Binary (noncharacter) data 0010 = ASCII (character) data
4-7	Must be Zero.
8-11	Terminal Control Attribute: 0000 = Unknown terminal control information 0001 = No terminal control information 0010 = GCOS 6 printer control information
12-14	Must be Zero
15	Foreign Data Attribute: 0 = GCOS 6 file data 1 = Non-native (non-GCOS 6) file data

DESCRIPTION:

Before this macro call is issued, tape-resident files must be open (see the Open File macro call) so that the system can retrieve the file attribute information. (File attribute information is stored in the tape labels.)

If neither the pathname nor the LFN is specified, a status code of 0205 is returned.

If an LFN is specified, the file must have been previously reserved through that LFN via a Get File or Create File macro call (or equivalent command).

To access specific entries in the argument structure, use the following macro calls: Get File Information, Parameter Structure Block Offsets; Get File Information Key Descriptor Block Offsets; and Get File Information, File Attribute Block Offsets.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0209 - Named file or directory not found

   020C - Volume not found

   0222 - Pathname cannot be expanded; no current working directory

   0225 - Not enough system memory for buffers or structures

   0228 - Invalid file type

   022C - Access control list (ACL) violation

   0238 - Invalid file description information.

Example:

In this example, the Get File Information ($GIFIL) macro call is used to obtain information about the file reserved in the example for the Get File macro call. The argument structure is defined as follows:

```
F_INFO    DC     5         LFN=5
          DC     <PATH5    POINTER TO PATHNAME
          RESV   2-$AF
          RESV   3,0       DEV. TYPE, LRN,
                           FILE/RECORD TYPE INFO AREA
          DC     <FILATT   POINTER TO FILE ATTRIBUTE AREA
          RESV   2-$AF
          RESV   6,0       RESERVED
PATH5     RESV   29,0      FIELD TO RECEIVE PATH
FILATT    RESV   16,0      FIELD TO RECEIVE FILE ATTRIBUTE INFO
```

Since, as stated under "Assumptions for File System Examples" Appendix A, the Get File Information, Parameter Structure Block Offsets, and Get File Information, File Attribute Block Offsets macro calls have been included in the procedure, any entry in F_INFO and FILATT can be referenced after executing the following macro call:

```
   $GIFIL    !F_INFO
```

The following instructions allow the reference to be made:

```
   LAB    $B6,F_INFO
   LAB    $B7,FILATT
```

Then, for example, to reference the system-supplied logical resource number and control interval size, respectively, the following address syllables would be specified in the instructions:

```
   $B6.I_LRN      SYSTEM-SUPPLIED LRN
   $B7.K_CISZ     SYSTEM-SUPPLIED CI SIZE
```

GET FILE INFORMATION, FILE ATTRIBUTE BLOCK OFFSETS ($GIFAB)

Associated Macro Calls:

Get File Information; Get File Information, Parameter Structure Block Offsets

FORMAT:

```
[label]  $GIFAB  [first letter of tags]
```

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure for Tape-Resident Files:

Word	Fields
0	Logical Record (transfer) Size
1	Block size
2	File Sequence Number
3	Label Format and Data Type
4	Data Format and Options
5	File Section Number
6	Retention Period
7 8 9 10 11 12 13 14 15	Reserved

Structure for Device Files:

Word	Fields
0	Logical Record (transfer) Size
1	Block size
2	File Indicators
3	Device-Specific Word 1
4	Device-Specific Word 2
5	Initial Device-Specific Word 1
6	Initial Device-Specific Word 2
7 8 9 10 11 12 13 14 15	Reserved

Structure for Disk Files:

Word	Fields
0	Logical Record Size
1	Control Interval Size
2	Current Allocation Size
3	Allocation Growth Size
4	Maximum Allocation Size
5	File-Specific Field; see details below
6	File-Specific Field; see details below
7	File-Specific Field; see details below
8	File-Specific Field; see details below
9 10 11 12 13 14 15	Reserved

Generated Offset Tags:

<u>For tape-resident files</u>:

Tag	Corresponding Offsets (in words)	Entry Name
T_LRSZ	0	Logical record size
T_BKSZ	+1	Block size
T_TFSN	+2	File sequence number (second byte)
T_TLF	+3	Label format (first byte)
T_TDT	+3	Data type (second byte)
T_TDF	+4	Data format (first byte)
T_TOPT	+4	Options for Block Sequence Number (second byte)
T_TFCN	+5	File section number
T_TRTN	+6	Retention period
T_TRFU	+7	Reserved
T_SZ	16	Size of structure (in words); not a field in the block

For Device Files:

Tag	Corresponding Offsets (in words)	Entry Name
T_LRSZ	0	Logical record size
T_BKSZ	+1	Block size
T_FIND	+2	File indicator word
T_DSW1	+3	Device-specific word 1 (for connect/disconnect orders)
T_DSW2	+4	Device-specific word 2 (for read/write orders)
T_ISW1	+5	Initial (sysgen) device-specific word 1
T_ISW2	+6	Initial (sysgen) device-specific word 2
T_SZ	16	Size of structure (in words); not a field in the block

For disk-resident files:

Tag	Corresponding Offsets (in words)	Entry Name
K_KRSZ	0	Logical record size
K_CISZ	+1	Control interval/physical sector size
K_CRSZ	+2	Current allocation size
K_GRSZ	+3	Allocation increment size
K_MXSZ	+4	Maximum allocation size
K_SZ	+16	Size of structure (in words); not a field in the block

Specific to indexed files:

Tag	Corresponding Offsets (in words)	Entry Name
K_FPC	+5	Amount of free space per control interval (indexed files)
K_LOV	+6	Local overflow allocation increment (indexed files)
K_HASH	+6	Number of hash results (random files)
K_NKD	+7	Number of key descriptors

Specific to random and virtual files:

K_INVT	+5	Inventory threshold
K_HASH	+6	Number of hash results
K_NRD	+7	Number of record descriptors

Specific to alternate indexes:

K_FPC	+5	Amount of free space per index control interval
K_NRD	+7	Number of record descriptors

Specific to I-D-S/II data base areas:

K_INVT	+5	Inventory threshold
K_CINT	+6	CALC interval
K_NRD	+7	Number of record descriptors

GET FILE INFORMATION, KEY DESCRIPTOR BLOCK OFFSETS ($GIKDB)

Associated Macro Calls:

Get File Information; Create File; Get File Information, Parameter Structure Block Offsets; Create File, Parameters Structure Block Offsets

FORMAT:

[label] $GIKDB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Reserved	
1	Record Type	
2	No. of Key Components	Reserved
3 4 5 6	Reserved	
7	Key Type	Key Length
8	Key Offset	

NOTE

Reserved fields must be set to zeros to ensure compatibility with later versions of this structure.

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
Y_RT	+1	Record type
Y_NKC	+2	Number of key components
Y_KTYP	+7	Key type (first byte)
Y_KLEN	+7	Key length, in bytes (second byte)
Y_KOFF	+8	Key offset, in bytes
Y_SZ	9	Size of structure (in words); not a field in the block

NOTE

This macro call has the same effect as the Create File Key Descriptors Block Offsets

GET FILE INFORMATION, PARAMETER STRUCTURE BLOCK OFFSETS ($GIPSB)

Associated Macro Calls:

Get File Information; Get File Information, File Attribute Block Offsets; Get File Information, Key Descriptors Block Offsets

FORMAT:

```
[label]  $GIPSB  [first letter of tags]
```

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Logical File Number (LFN)	
1 2	Pathname Pointer	
3	Device Type	
4	Logical Resource Number	
5	File Type	Data Format
6 7	File Attribute Block Pointer	
8	Terminal Software Device id or Disk File Options	
9	Disk File Data Attributes	
10 11	Key Descriptors Block Pointer	
12	Size of Record Descriptor Information	
13	Number of Related Files	

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
I_LFN	0	Logical file number (LFN)
I_PTHP	+1	Pointer to pathname
I_DTYP	+3	Device type
I_LRN	+4	Logical resource number
I_FTYP	+5	File type (first byte)
I_RTYP	+5	Data format (second byte)
I_FABP	+6	Pointer to file attributes (see $GIFAB description)
I_SDID	+8	Terminal software device descriptor id
I_OPT	+8	Disk file options
I_ATTR	+9	Disk file data attributes
I_KDP	+10	Pointer to key descriptors (see $GIKDB description)
I_RDSZ	+12	Size of record descriptor information
I_NRF	+13	Number of related files
I_SZ	14	Size of structure (in words); not a field in the block

GET FILE PARAMETER STRUCTURE BLOCK OFFSETS ($GTPSB)

Associated Macro Call: Get File

FORMAT:

[label] $GTPSB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Logical File Number (LFN)	
1 2	Pathname Pointer	
3	Disk Concurrency	Disk Mount Option
4	Tape Block Size	
5	Tape Logical Record Size	
6	No. of Buffers	Tape File Sequence No.
7	Tape Label Format	Tape Data Type
8	Tape Data Format	Tape (File Options)
9	Tape File Section Number	
10	Tape Retention Period	

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
G_LFN	0	Logical file number
G_PTHP	+1	Pointer to pathname
G_CONC	+3	Concurrency control (first byte)
G_MNT	+3	Mount option (second byte)
G_BKSZ	+4	Tape block size
G_LRSZ	+5	Tape logical record size
G_NBF	+6	Number of buffers (first byte)
G_TFSN	+6	Tape file sequence number (second byte)
G_TLF	+7	Tape label format (first byte)
G_TDT	+7	Tape data types (second byte)
G_TDF	+8	Tape data format (first byte)
G_TOPT	+8	Tape file options (second byte)
G_TFCN	+9	Tape file section number
G_TRP	+10	Tape retention period
G_SZ	11	Size of structure (in words); not a field in the block

GET FILE RECORD DESCRIPTOR BLOCK OFFSETS ($GIRDB)

Associated Macro Calls:

Create File; Get File Information; Create File Parameter Structure Block Offsets; Get File Information, Parameter Structure Block Offsets

FORMAT:

[label] $GIRDB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields	
0	Size of Record Descriptor Block (including this field)	
1	Record Type	
2	Number of Key components	Reserved
3 4	Low Record Number	
5 6	High Record Number	

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
Y_RDSZ	0	Size of record descriptor block (including this field)
Y_RT	+1	Record type
Y_NRC	+2	Number of key components
Y_LRNG	+3	Low record number
Y_HRNG	+4	High record number

NOTE

This macro call has the same effect as the Create File Record Descriptor Block Offsets macro call.

GET MEMORY/GET AVAILABLE MEMORY ($GMEM)

Function Code: 04/02 (Get Memory), 04/03 (Get Available Memory)

Equivalent Command: None

Allocate to the issuing task the requested amount of contiguous memory. The memory is allocated as a block from the memory pool of the task group to which the issuing task belongs. If the specified amount of contiguous memory is not available, perform one of the following actions:

- Return immediately to the issuing task without performing any allocation (Get Memory with DENY specified).
- Suspend the issuing task until the required memory becomes available (Get Memory with WAIT specified).
- Allocate the largest contiguous block of memory currently available in the memory pool and return to the issuing task (Get Available Memory with AVAIL specified).

FORMAT:

```
$GMEM  [location of maximum number of words required],
       [{DENY }]
       [{WAIT }]
       [{AVAIL}]
```

ARGUMENTS:

location of maximum number of words required

Any address form valid for a data register; provides the maximum number of words of memory to be allocated as a block to the issuing task. The value used cannot exceed the size of the pool minus the memory block header. (Each bit in the bit map represents a 32-word allocation.) The value for the number of words cannot exceed 1,048,575 (minus the memory block header).

DENY

If the number of words of memory specified in argument 1 is not available either in the task group's memory pool or, if the task group can extend into it, in the batch group's memory pool, return immediately to the issuing task. If argument 2 is omitted, DENY is the default value.

WAIT

If the number of words of memory specified in argument 1 is not available either in the task group's memory pool or, if the task group can extend into it, in the batch group's memory pool, suspend the issuing task until the memory becomes available. Activate the task, allocate the memory, and return to the task.

AVAIL

If the number of words of memory specified in argument 1 is not available either in the task group's memory pool or, if the task group can extend into it, in the batch group's memory pool, allocate to the issuing task the largest contiguous block of memory currently available.

DESCRIPTION:

This call allows the issuing task to dynamically obtain a block of memory from the task group's memory pool. If argument 2 is DENY, the task obtains a block of the specified size or no block at all. If argument 2 is WAIT, the task is suspended until the requested amount of memory becomes available. If the online pool extended into the batch pool, the largest amount of memory available is allocated from the batch pool.

If argument 2 is AVAIL, the task obtains a block of the specified size or the largest block (less than the specified size) that is currently available.

When AVAIL (Get Available Memory) is specified, the actual size of the memory block allocated may be much smaller than the desired size. This situation occurs because the Memory Manager does not wait for memory to become available. Rather, it checks for contiguous memory of the specified size and if none is available, allocates the largest contiguous block of memory that is available. If no memory is available, the system returns a status code of 0602.

NOTE

When AVAIL is specified, all of available memory may be removed from the pool. Other functions (including the command processor) that require memory from that pool then will not be able to execute until memory becomes available.

When a return is made to the issuing task, the actual size of the supplied contiguous memory block is placed in $R6 and $R7. "Actual size" has the following meaning. Memory is allocated in 32-word units. A block of memory contains an integral number of 32-word allocation units. A memory block also contains a header whose size is three words. The value returned in $R6 and $R7 is the specified number of words rounded up to the next higher allocation unit, minus the size of the memory block header.

NOTE

> If AVAIL is specified and a block of the requested size could not be found, the actual size of the block is that of the largest contiguous memory block available, minus the size of the header.

The maximum size of a memory block that can be obtained is 1,048,575 words, minus the memory block header. The block size cannot exceed the pool size.

On return to the issuing task, $B4 contains the address of the first usable word in the block (first word after the block header).

The Get Memory/Get Available Memory functions enable the task to dynamically acquire additional memory in response to processing needs. When a memory block is no longer required, it must be returned to the task group's memory pool (by a Return Memory or Return Partial Block of Memory macro call). If a task repeatedly acquires memory blocks and does not return them, the task group memory area will become empty (or nearly so), denying other tasks the opportunity to obtain memory blocks.

NOTES

1. The system places the number of contiguous words of memory required, supplied by argument 1, in $R6 and $R7. If this argument is =$R7, the system assumes that $R6 and $R7 contain the number of words desired.

2. When argument 2 is DENY, $R2 is set to zero.
   When argument 2 is WAIT, $R2 is set to -1.
   When argument 2 is AVAIL, $R2 is not set.
   When argument 2 is omitted, $R2 is set to zero (DENY).

3. On return to the issuing task, $R1, $R6, $R7, and $B4 contain the following information:

$R1 - Return status; one of the following:

0000 - If the call specified WAIT or DENY, memory allocation was successful. If the call specified AVAIL, at least one memory unit was allocated.

0601 - If the call specified WAIT or DENY, requested contiguous memory exceeds defined pool size; not applicable if the call specified AVAIL.

0602 - If the call specified WAIT or DENY, the requested contiguous memory was not obtained. If the call specified AVAIL, no memory allocation units were available.

The following codes could be returned if WAIT or DENY was specified.

0818 - No task group with specified group identifier exists (system software error).

081A - Suspend in progress (system software error).

081B - Rollout of online task group attempted (system software error).

081D - Batch task group already rolled out (system software error).

081E - Unrecoverable media error during rollout.

$R6, $R7 - Actual size of contiguous memory block supplied, rounded up to the nearest multiple of 32 words minus 3-word block header.

$B4 - If $R1 was 0000, address of first usable word in memory block.

Examples:

In this example, the Get Memory/Get Available Memory macro call is used to obtain 2500 words of memory from the issuing task group's memory area. If the memory is available, the system returns with a status of 0000 in $R1, the actual size of the memory area obtained in $R6 and $R7, and the address of the first usable word of the area in $B4. The example saves the address of the memory area in the field labeled M_PTR and continues processing. If 2500 contiguous words of memory are not available, the system returns with a status of 0602 in $R1. If the pool size is less than 2500 words, the system returns error code 0601 in $R1.

```
              $GMEM    =2500
              BNEZ     $R1,NO_MEM
              STB      $B4,M_PTR
                .
                .
                .
    M_PTR     DC       <$
```

In this example, the Get Memory/Get Available Memory macro call is used to obtain the largest contiguous area of memory, not exceeding 5000 words, available in the issuing task group's memory area. If any memory is available, the system returns with a status of 0000 in $R1, the actual size of the memory area obtained in $R6 and $R7, and the address of the first usable word of the area in $B4. If all of the memory in the task group's memory area is in use at the time, the system returns with a status of 0602 in $R1.

```
              $GMEM   =5000,AVAIL
```

GET NAME ($GNFIL)

Function Code: 10/3C

Equivalent Command: None

Retrieve the names of alternate index files associated with a specified file.

FORMAT:

[label] $GNFIL [argument structure address]

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the address of the argument structure provided below. The argument stucture must contain the following entries in the order shown. (Entries marked with an asterisk are provided by the system.)

logical file number

A 2-byte logical file number (LFN) used to refer to the file; must be a binary number from 0 to 255 or ASCII blanks (X'2020'), which indicate that an LFN is not specified.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the directory in the file hierarchy in which the file is found. Binary zeros in this entry indicate that a pathname is not specified.

file number

The number of the alternate index file for which the information is to be retrieved.

*name

A 12-byte field into which the system places the alternate index file name that is related to the given file number.

*file type

A 2-byte field into which the system places the type of file (X = alternate index).

*index id

A 2-byte field into which the system places a number that uniquely identifies the index and that can later be specified when accessing a data file.

*total number of related files

A 2-byte field into which the system places the total number of related alternate index files associated with a specified file. When the file number entry above is equal to the total number of related files, there are no more related files.

DESCRIPTION:

This macro call retrieves the names of the alternate index files associated with a data file. The data file in the argument structure can be specified by either the LFN or pathname. If an LFN is specified, the data file must have been previously reserved through that LFN with a Get File function.

If the names of all the alternate index files associated with a data file are required, multiple calls must be made with the file number commencing at one and increasing by one until the total number is reached. If the file number specified exceeds the total number of alternate index files, an error code (020F) is returned.

NOTES

1. If the argument structure address is coded, the system loads the address of the argument structure into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No Error

   01XX - Media Error

0201 - Invalid pathname

0202 - Pathname not specified

0206 - Unknown or invalid LFN

0209 - Named file or directory not found

020C - Volume not found

020F - Link or file number not found

0222 - Pathname cannot be expanded; no working directory

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

022C - Access control list (ACL) violation.

GET NAMES PARAMETER STRUCTURE BLOCK OFFSETS ($GNSPB)

Associated Macro Call: Get Name

FORMAT:

[label] $GNPSB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields
0	Logical File Number
1 2	Pointer to Pathname
3	File Number
4 5 6 7 8 9	File Name
10	File Type
11	Index id
12	Total Number of Related Files
13	Reserved

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
N_LFN	0	Logical File Number
N_PTHP	+1	Pointer to pathname
N_FNUM	+3	File number
N_FNME	+4	File name
N_FTYP	+10	File type
N_DDID	+11	Index id
N_NRF	+12	Total number of related files
N_RFU	+13	Reserved
N_SZ	16	Size of structure (in words); not a field in block

GET WORKING DIRECTORY ($GWDIR)

Function Code: 10/C0

Equivalent Command: List Working Directory (LWD)

Returns the name of the current working directory. This function is usually done outside program execution.

FORMAT:

```
[label]  $GWDIR  [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entry.

working directory pathname

A 45-byte field, in main memory, into which the system can place the full absolute pathname of the current working directory.

DESCRIPTION:

This macro call returns the full absolute pathname of your current working directory. Although the pathname may be shorter than the maximum 45 characters, the argument structure must be large enough to accommodate the maximum number of characters.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

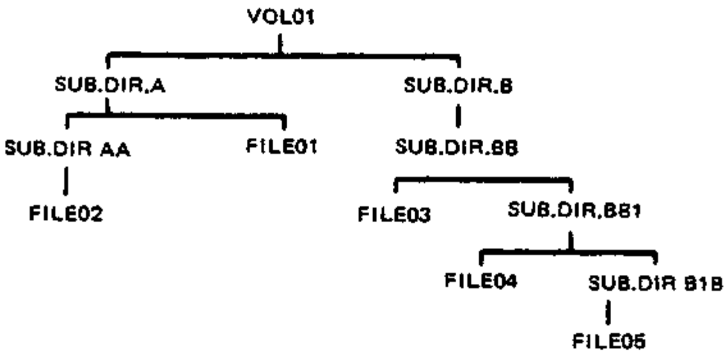
   0000 - No error

   0205 - Invalid argument

   0222 - Pathname cannot be expanded; no working directory.

Example:

This example assumes the following file system hierarchy (see the System Concepts manual) and that the working directory is SUB.DIR.BB1.



Coding the Get Working Directory macro call causes the system to place the full absolute pathname of the working directory, defined below, into the specified argument structure:

```
$GWDIR    !CURDIR

CURDIR    RESV 29
```

The path placed in the main memory field labeled CURDIR is:

```
^VOL01>SUB.DIR.B>SUB.DIR.BB>SUB.DIR.BB1ΔΔΔΔΔΔ
```

GROUP IDENTIFICATION ($GRPID)

Function Code: 14/08

Equivalent Command: USER TGID

Returns the 2-character task group id for the current group or for the group designated by the user identification specified in the macro call.

FORMAT:

```
[label] $GRPID [location of user id field address]
```

ARGUMENT:

location of user id field address

Any address form valid for an address register; provides the address of a field containing the user id of the task group whose group is to be returned. When the argument is null, the system returns the group id of the task group where the issuing task is running. The user id value, when specified, should be expressed as person.account.mode followed by a space.

DESCRIPTION:

This macro call returns in $R6 the group id of the designated task group. When the argument is null, the system returns the id of the task group where the issuing task is running. For any other group id to be returned, the user must know the user id of that group. (The format of the user id is described under the Login command in the *Commands* manual.) Note that the User Identification macro call returns the user id of the task group of the task that issues the call.

NOTES

1. The system placed in $B4 the address of the user id field supplied by the argument. When the argument is omitted, the system assumes that $B4 contains the address of the user id field.

2. On return, $R1 and $R6 contain the following:

$R1 - Return status code; one of:

0000 - No error
0817 - Memory access violation
0837 - User not logged in
0838 - Invalid user id format

$R6 - Task group id of the designated task group.

Example:

The macro call requests the group id of its own task group. The id will be returned in $R6.

```
$GRPID    =Null
```

GROW FILE ($GRFIL)

Function Code: 10/38

Equivalent Command: Grow File

Expand disk space allocated to a file and/or modify current values for the file's maximum growth and maximum size.

FORMAT:

[label] $GRFIL [parameter structure address]

ARGUMENT:

parameter structure address

Any address form valid for an address register; provides the address of the parameter structure described below, which must contain the following entries in the order shown.

Parameter Structure Entry	Size (in bytes)
logical file number	2
pathname pointer	4
space allocation options	2
expansion size	2
growth size	2
maximum size	2
(reserved for future use)	18

logical file number

A 2-byte LFN specifying the file to be expanded; must be a binary number in the range 0 through 255, or ASCII blanks (which indicate that an LFN is not specified). Either a LFN or pathname pointer (below) must be specified.

pathname pointer

A 4-byte entry that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the file to be expanded. Binary zeros indicate that a pathname is not specified.

space allocation options

A 2-byte entry indicating where on a multivolume set additional space is to be allocated. The bits of this word have the following significance:

bits 0-10: MBZ

bits 11-15: 00000 = Allocate space on the volume having the most available space

nnnnn = Allocate space on the 'nnnnn'th volume in the set.

Bits 11-15 are meaningful only if an expansion size (below) is specified.

expansion size

A 2-byte entry specifying the number of control intervals by which the file is to be expanded by execution of this call.

growth size

A 2-byte entry specifying, in control intervals, the smallest increment by which the file's space can be expanded. The value specified in this entry modifies the value already specified or defaulted to by the Growth Size argument of the Create File function, or already specified by a previous invocation of this macro call (The default growth size is 40 physical sectors.) Zeros signify that the current growth size is to be retained.

maximum size

A 2 byte entry specifying, in control intervals, the maximum size to which the file can grow. The value specified in this entry modifies the value, if any, already specified by the Maximum Size argument of the Create File function, or already specified by a previous invocation of this macro call. The value specified by this entry cannot be less than the current, logical end-of-data. Zeros signify that the current maximum size is to be retained. A value of -1 (FFFF) means that the new maximum growth size, established by this call, is unlimited.

Reserved for future use

This 18-byte field must be zero.

DESCRIPTION

This macro call expands the disk space allocated to a file; at the same time, or alternatively, it modifies the current value of the increment by which the file can grow and of the the limit to which the file can grow. The current growth size and maximum size values are established by the Create File function or by an earlier invocation of this function. Normally, this function is performed outside of program execution by means of the Grow File command.

The space allocation options, described above, allow the user to specify a member of a multivolume set on which additional space is to be allocated. By specifying, in a sequence of calls, different volumes for the same file, the user can spread a file's space over several volumes in order to reduce disk arm movement.

The function attempts to allocate the specified expansion size in a single extent. If this is not possible, the function allocates the largest available extents. The segments allocated must be as large as the current maximum growth size. If the full expansion size cannot be allocated in segments of allowable size, the function allocates part of the expansion size, returning an 0215 status (not enough contiguous disk space available).

When expanding an online, multivolume file, the function seeks contiguous space on the starting member. (This is the member specified in the space allocation option parameter or, if no volume was specified, the volume having the most available space.) If segments equal to the specified expansion size and of allowable size are not available on the starting member, the function seeks contiguous space on other members of the set.

The disk file to be grown can be specified in the parameter structure by either an LFN or pathname. If specified by an LFN, the file must have been previously reserved through that LFN by the Create File or Get File function.

To expand a file beyond the current maximum file size, the caller must modify the maximum file size by the same call that expands the file.

When an alternate index is specified as the file to be grown, space is allocated only for that index -- not for the associated data file.

A restorable file (i.e., one created or modified with the -RESTORE attribute) cannot be expanded unles the system's image journal is open.

This macro call cannot be used to expand the following types of file:

- Non-expandable files (i.e., files whose specified initial size is the same as the specified maximum size)

- Temporary disk files

- Directories.

NOTES

1. The system places in $B4 the address of the parameter structure supplied by the argument. If the argument is omitted, $B4 is assumed to contain the parameter structure address.

2. On return, $R1 contains one of the following return codes:

   0000 - No error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invlaid LFN

   0209 - Named file or some superior directory not found

   020C - Volume not found

   0213 - Cannot provide requested file concurrency

   0215 - Not enough contiguous disk space available

   0222 - Pathname connot be expanded; no working directory

   0225 - Not enough system memroy for buffers or structures

0226 - Invalid file type (a device or directory)

022C - Access control list (ACL) violation

0260 - Journal file not open.

GROW FILE PARAMETER BLOCK OFFSETS ($GRPSB)

Associated Macro Call: Grow File

FORMAT:

[label] $GRPSB [first letter of tags]

ARGUMENT:

first letter of tag

Allows the user to rename the tags to avoid conflicts with other labels in the same program.

Structure:

Word	Fields
0	Logical resource number
1 2	Pathname pointer
3	Space allocation options
4	Expansion size
5	Growth size
6	Maximum size
7 8 9 10 11 12 13 14 15	Reserved for future use

Generated Offset Tags:

Tag	Corresponding Offsets (in words)	Entry Name
G_LRN	0	Logical resource number
G_PTHP	+1	Pathname pointer
G_OPT	+3	Allocation options word
G_EXSZ	+4	Expansion size (in CIs)
G_GRSZ	+5	New growth size (in CIs)
G_MXSZ	+6	New maximum size (in CIs)
G_SZ	16	Size of structure (in words); not a field in the block.

HOME DIRECTORY ($HDIR)

Function Code: 14/0B

Equivalent Command: List Home Directory (LHD)

Return the pathname of the initial working directory of the calling task group to a 45-character receiving field.

FORMAT:

```
[label] $HDIR [location of home directory field address]
```

ARGUMENT:

location of home directory field address

Any address form valid for an address register; provides the address of a 45-character, aligned, nonvarying field into which the system places the pathname of the default working directory of the calling task group.

DESCRIPTION:

This macro call returns the pathname of the initial working directory to a field in the issuing task. The pathname returned is that specified in the -HD argument of the LOGIN command. If the -HD argument was not specified, the pathname returned is that set according to user registration arguments or system defaults.

NOTES

1. The system places the address of the receiving home directory field, supplied by argument 1, in $B4; if this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0817 - Memory access violation

   On return, $B4 contains the address of the receiving field.

Example:

In this example, the pathname of the initial working directory of the calling task group is stored in the 45-character field labeled DEF_WD.

```
              $HDIR    !DEF_WD
                .
                .
                .
    DEF_WD    RESV     22,0
              DC       0
```

INPUT/OUTPUT REQUEST BLOCK ($IORB)

Function Code: None

Equivalent Command: None

Generates an input/output request block (IORB). The length of the IORB is 11 to 12 words, unless extended (see extension indicator argument).

FORMAT:

```
[label]  $IORB  [logical resource number],
                ⎧,                                          ⎫
                ⎨WAIT                                       ⎬
                ⎩NWAIT, [issuing task termination action],  ⎭
                [buffer address],
                [buffer byte alignment],
                [buffer range],
                [extension indicator]
```

ARGUMENTS:

logical resource number

A value from 0 through 252 specifying the logical resource number (LRN) of the device involved in the request. The value specified must be that of a system LRN. If this argument is omitted, the left byte of the I_CT1 word (see Appendix C) is set to zero.

[WAIT] ,
[NWAIT]

One of the following values is specified to indicate whether the requesting task is to be suspended until the completion of the request:

WAIT - Suspend the issuing task until the request is complete (set the W-bit to zero)

NWAIT - Do not suspend the issuing task (set the W-bit to one)

If this argument is omitted, the value NWAIT is assumed.

If WAIT is specified, argument 3 (issuing task termination action) must be omitted.

issuing task termination action

One of the following values is specified to indicate the action to be taken upon the completion of the request.

SM=aa - Do not suspend the issuing task; release (V-op) the semaphore identified by aa (two ASCII characters), when requested task is completed.

RB=label - Do not suspend the issuing task; issue a request for the request block identified by label, when requested task is completed.

Note that the requesting task must be asynchronous, may not wait on the requested task later on, and can only point to a task request block (TRB). The requested task must have already been created (not spawned), be asynchronous, and have a valid LRN. When the requesting task terminates, the TRB pointed to by "label" must be inactive.

If this argument is omitted (or argument 2 is WAIT), the generated IORB contains no termination option.

buffer address

Address of a buffer area to be used for input/output transfers involving the specified device. If this argument is omitted, the buffer address field in the generated IORB is initialized to zeros.

buffer byte alignment

A value specifying the beginning byte of the buffer, as follows:

R - Buffer begins in right byte of word address specified by argument 4

L - Buffer begins in left byte of word address specified by argument 4

If this argument is omitted, a value of L is assumed.

buffer range

A value specifying the length, in bytes, of the buffer. If this argument is omitted, the generated IORB's range value is initialized to zero.

extension indicator

The following value, when specified, indicates that the IORB is to be extended beyond the standard IORB. The argument causes space for the IORB extension to be generated, resulting in an extended IORB (see Appendix C). When the argument is omitted, the system generates a standard-length IORB.

EXT - Generate an extended IORB

DESCRIPTION:

The IORB is usd as the standard means of requesting a physical I/O service. The IORB contains an LRN that identifies the I/O device being addressed. The IORB also identifies the location and size of the buffer to be used for physical I/O transfers as well as the specific function requested.

Example:

In this example, the Input/Output Request Block macro call generates a standard IORB having an LRN of zero, a WAIT status indicating that the requesting task will wait for I/O completion, and a label (DBUF) that gives the location of the 140-byte buffer area.

```
CONIO   $IORB   0,WAIT,,DBUF,,140
```

INPUT/OUTPUT REQUEST BLOCK OFFSETS ($IORBD)

Counterpart: $IORB (see Input/Output Request Block macro call).

Generated Label Prefixes:

```
                 I_RRB/I_SEM
IORB label       offset 0 (set, used by system)
                 I_CT1
                 I_CT2
                 I_ADR
                 I_RNG
                 I_DVS
                 I_RSR
                 I_ST
```

Extended words are:

```
                 I_DV2
                 I_FCS
                 I_HDR
                 I_ST2
                 I_QDP
                 I_TAB
                 I_CON
                 I_LOG
```

See Appendix C for the format of the input/output request block.

INTERNAL DATE/TIME, CONVERT TO ($INDTM)

Function Code: 05/07

Equivalent Command: None

Convert the external format date/time value to an internal format date/time value.

FORMAT:

```
[label]  $INDTM  [location of address of external date/time],
                 [location of address of receiving field],
                 [location of size of external date/time]
```

ARGUMENTS:

location of address of external date/time

Any address form valid for an address register; provides the address of a field containing an external date/time value. This value must be in the format returned by the Convert to External Date/Time macro call.

location of address of receiving field

Any address form valid for a data register; provides the address of a 3-word field into which the system places the internal format date/time value.

location of size of external date/time

Any address form valid for a data register; provides the size of the external date/time value identified by argument 1. The size must be less than or equal to 22 bytes. If this argument is omitted, the size is set to 20 bytes (tenth of a second resolution).

The size must be such that the date/time value does not end with the characters : (colon) or . (period).

DESCRIPTION:

This macro call converts an external date/time value (as supplied by the Convert to External Date/Time macro call) to internal format (as supplied by the Get Date/Time macro call). The internal date/time value appears in the receiving field as a binary count of the milliseconds that have elapsed from 1 January 1901 at 00:00:00.0000 hours.

NOTES

1. The system places in $B4 the address of the external date/time value supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct external value.

2. The internal date/time value returned is loaded into $R2, $R6, and $R7, and is placed in the receiving field specified by argument 2. If argument 2 is omitted, or is =$R7, the internal date/time value is returned only in $R2, $R6, and $R7.

3. The system places in $R5 the size of the external date/time value supplied by argument 3. If this argument is =$R5, the system assumes that $R5 contains the correct size. If this argument is omitted, $R5 is set to a value of 20 (tenth of a second resolution).

4. On return, $R1, $R2, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0407 - Invalid external date
   0408 - Invalid external time
   040A - Invalid access to external date/time field

   $R2, $R6, $R7 - Generated internal date/time value

   $B4 - Address of supplied external date/time value.

Example:

In this example, the Get Date/Time macro call is used to get the current date/time, in internal format, leaving it in registers $R2, $R6, and $R7. The External Date/Time, Convert To macro call is then used to convert this internal format to an external format, replacing the date portion (first 10 characters) of the field labeled TODAY. The TODAY field now contains the external format date/time for 0800 hours of today. The Internal Date/Time Convert To macro call then converts this date/time value back to an internal format contained in $R2, $R6, and $R7. One day (86,400,000 milliseconds) is then added to this internal date/time giving the internal date/time for 0800 hours tomorrow, which is stored in the 3-word field labeled MORROW. The addition is programmed with the assumption that the central processor does not have the add integer double instruction.

```
*
*  GET THE CURRENT DATE/TIME VALUE.
*
           $GDTM
*
*  CONVERT IT TO AN EXTERNAL FORMAT DATE.
*
          $EXTDT   ,!TODAY,=10
*
*  NOW CONVERT THE EXTERNAL DATE/TIME
*  BACK TO THE INTERNAL FORMAT.
*
           $INDTM   !TODAY,,=15
*
*  ADD IN ONE DAY.
*
           ADD    $R7,A_DAY+1
           CAD    =$R6
           CAD    =$R2
           ADD    $R6,A_DAY
           CAD    =$R2
*
*  NOW STORE THE RESULT.
*
           STR    $R2,MORROW
           SDI    MORROW+1
           .
           .
           .
   TODAY   TEXT    'YYYY/MM/DD 0800'
   A_DAY   DC      86400000B(31,0)
   MORROW  RESV    3,0
```

KILL (ABORT) TASK ($KILLT)

Function Code: 0C/11

Equivalent Command: Kill

Terminate the execution of the specified task and activate its cleanup trap handling routine.

FORMAT:

```
[label]  $KILLT  [location of logical resource number]
```

ARGUMENT:

location of logical resource number

Any address form valid for a data register; provides the logical resouce number (LRN), a value from 0 through 255 (decimal), of the task to be aborted.

DESCRIPTION:

This call causes trap condition 49 (Unwind) to be signalled to the task specified by its LRN. The system assumes that the specified task has enabled trap 49 (using $ENTRP) and that it includes a cleanup trap handling routine that releases any resources private to the task. If the task has not enabled trap 49, it is terminated.

NOTES

1. The system places the LRN of the task to be aborted, supplied by the argument, in $R2. When the argument is omitted, the system assumes that $R2 contains the correct LRN.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0802 - Invalid LRN.

Example:

The issuing task issues a Kill (Abort) Task macro call to abort another task (whose LRN is 34) in the same task group.

```
ABT34     $KILLT   =34
```

MESSAGE GROUP, ACCEPT ($MACPT)

Function Code: 15/01

Equivalent Command: None

Start a process of receiving a message group from the previously created mailbox.

FORMAT:

```
[label]   $MACPT   [location of MGIRB address]
```

ARGUMENT:

location of MGIRB address

Any address form valid for an address register; provides the address of the message group initialization request block (MGIRB), which must have been previously generated.

DESCRIPTION:

The acceptor task group issues this macro call in order to accept a message. The Message Group, Accept macro call validates access to the mailbox. It returns a message id to identify an accepted message. (See the System Concepts manual for a discussion of the Message Facility.)

Deferred messages may be accepted on the following selection criteria:

- First available message
- Sequence number
- Initiator (submittor) name
- Submittor name and sequence number.

Define selection criteria by supplying input arguments in the following MGIRB fields:

- Message group id field (MI_MGI)
- Residual range field (MI_RSR)
- Initiator mailbox name (MI_MBI).

To accept a message by sequence number, specify -1 in MI_MGI and a sequence number in MI_RSR. Local mail searches for a message from the specified sequence number. If no message exists for the specified sequence number, the first available message after the sequence number is received. If no message exists, an error is returned.

To accept a message by initiator mailbox name, specify zero in MI_MGI and the initiator name in MI_MBI. If MI_MBI contains null bytes, the first available message is accepted. If both a sequence number and an initiator mailbox name are specified, local mail searches for a message with the specified initiator name from the specified sequence number.

A message can be accepted when the user has received access (read access to the mailbox file ($MBX)) to the mailbox. (See the System Concepts manual about access to the mailbox.) However, if an acceptor specifies an initiator name, send access (list access on mailbox directory) is enough to accept a message.

Before the Message Group, Accept macro call is executed, the user must generate a MGIRB with values in the following fields:

MI_MAJ, bit 9 (wait bit)
MI_MPD
MI_MBI
MI_ADT
MI_MBA

MGIRB fields are described in Appendix C.

NOTES

1. A mailbox must have been created before the macro call is issued. (See the Create Mailbox (CMBX) command in the Commands manual.) Reference to mailbox fields when no mailbox has been created results in an error return.

2. The system places the address of the MGIRB in $B4. If the argument is omitted, the system assumes that $B4 contains a pointer to the MGIRB.

3. Local mail returns the maturity date/time of deferred messages in the MI_DV2 field of the acceptor's MGIRB in a standard 3-word date/time format. A deferred message group can be accepted before the specified maturity date/time by using the sequence number.

7. On return, $R1 contains the following status codes:

   0000 - No error

   0C02 - Invalid message id

   0C03 - Abnormal termination. This error is returned when a user tries to accept a message with an initiator name and no message is available.

   0C17 - Invalid message path description identifier

   0C19 - Acceptor mailbox may not be accessed by initiator

   0C1A - Acceptor mailbox not known.

8. On return, $B4 will point to the application's MGIRB, which is updated according to the specifications in the macro call.

MESSAGE GROUP, CANCEL ENCLOSURE ($MCME)

Function Code: 15/06

Equivalent Command: None

Delete the last record in the current incomplete quarantine unit or the entire last incomplete quarantine unit.

FORMAT:

```
[label]   $MCME   [location of MGCRB address]
```

ARGUMENT:

location of MGCRB address

Any address form valid for an address register; provides the address of the message group control request block (MGCRB), which must have been previously generated.

DESCRIPTION:

This macro call may be issued only by a sending task. It allows editing (delete a record) of the last quarantine unit before it becomes available to the receiving task. The sender can delete a record or delete the entire last incomplete quarantine unit by specifying the appropriate values in the MC_LVL field of the MGCRB.

Before the Message Group, Cancel Enclosure macro call is executed, the user must generate the MGCRB (by means of the $MGCRB macro calll) with values in the following fields

MC_MAJ, bit 9 (wait bit)
MC_MGI
MC_LVL

MGCRB fields are described in Appendix C.

NOTES

1. The system places the address of the MGCRB in $B4. If the argument is omitted, the system assumes that $B4 contains a pointer to the MGCRB.

3. On return, $R1 contains the following return status codes:

   0000 - No error

   0C03 - Abnormal termination

   0C17 - Invalid mesasge-path identifier

   0C19 - Acceptor mailbox may not be accessed by initiator

   0C1A - Acceptor mailbox not known

   0C22 through 0C2C - User-coded reason for abnormal message group.

MESSAGE GROUP, CONTROL REQUEST BLOCK ($MGCRB)

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, perform one of the following:

- Build a mesage group control request block (MGCRB) of 29 words that contains default values for all fields not explicitly specified in the call. See Appendix C.
- Generate instructions to alter the partial contents of an existing MGCRB.

FORMAT:

```
[label]   $MGCRB,[arguments]
```

ARGUMENTS:

There are three types of arguments for this macro call:

- Keyword only (i.e., RESV)
- Keyword with expression (expression is a user-selected variable whose literal value is used by the system)
- Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGCRB. When the macro call is issued with RESV as its only argument, an MGCRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGCRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGCRB, in which case the keyword with expression argument ADR=address is used to specify the address of the MGCRB to be changed. When ADR=address is omitted, the system assumes that $B4 points to that MGCRB. The argument ADR=address is not used in building a new MGCRB; that is, when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 2-15.

The first argument position is reserved for system use, and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by a comma.

Table 2-15 describes the arguments for the Mesage Group, Control Request Block macro call and indicates the fields in the MGCRB into which the system inserts the argument values.

Table 2-15. Argument Values for $MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only				
1	None	None	Reserved by system; must be a comma.	N/A
2			Issuing task suspension option:	MC_MAJ
	WAIT*	None	Suspend the issuing task until the request is completed (set W-bit (wait) to zero).	
	NWAIT (default)	None	Do not suspend the issuing task (set W-bit to one).	
Any	RESV	None	Generates MGCRB.	

Table 2-15 (cont). Argument Values for $MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword with expression				
3**			Issuing task termination option:	N/A
	SM=	aa	When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.	
	RB=	label	When requested task is complete, do not suspend the issuing task; issue a request for the request block identified by label.	
			Note that the requesting task must be asynchronous, may not wait on the requested task later on, and can only point to a task request block (TRB). The requested task must have already been created (not spawned), be asynchronous, and have a valid LRN. When the requesting task terminates, the TRB pointed to by "label" must be inactive.	

*When WAIT is specified, argument 3 must be omitted.

**When this argument is omitted, or argument 2 is WAIT, the generated MGCRB contains no termination option. In that case, the user must issue a Wait, Wait On Request List, or Test Completion Status macro call.

Table 2-15 (cont). Argument Values for $MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
		Keyword with expression (cont)		
Any	ADR=	address	When existing MGCRB is to be changed (RESV omitted), specifies address of MGCRB to be changed.	N/A
Any	BUF=	buffer address (default is 0)	Address of the buffer location in the task where sent or received record is to be placed.	MC_BUF
Any	RANGE=	number of bytes (default is 0)	Length, in bytes, of the buffer.	MC_BSZ
		Keyword with option		
Any	ALIGN=		Buffer byte alignment:	MC_OPT
		R	Buffer begins in right-most byte of address specified by BUF= argument.	
		L (default value)	Buffer begins in left-most byte of address specified by BUF= argument.	
Any	WTI=		Wait test indicator ($MRECV only):	MC_WTI
		WAIT	Do not process request until data is available.	
		DENY (default value)	Return error status when there is no data available.	

Table 2-15 (cont). Argument Values for $MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Any	ENC=		Enclosure level that delimits send or receive message unit.	MC_LVL
		EOR	End-of-record.	
		EOQ (default value)	End-of-quarantine-unit.	
		EOM	End-of-message.	

DESCRIPTION:

The message group control request block (MGCRB) is used for communication between task groups, and is the means for passing arguments among task groups in connection with the Message Group Send and Message Group Receive macro calls of the message facility. This macro call makes it possible to modify an existing MGCRB by generating executable instructions that use registers R6, R7, and B5 (as appropriate). The modifying process always uses $B4 to point to the MGCRB.

MESSAGE GROUP, COUNT ($MCMG)

Function Code: 15/07

Equivalent Command: None

Provide a count of the number of completed message groups not yet "accepted" by previous Message Group, Accept macro calls, including deferred message groups that are available for processing by subsequent macro calls.

FORMAT:

```
[label]   $MCMG   [location of MGIRB address]
```

ARGUMENT:

location of MGIRB address

Any address form valid for an address register; provides the address of the message group initialization request block (MGIRB), which must have been previously created.

DESCRIPTION:

The sending or receiving task group may issue this macro call to ascertain the number of completed groups currently in the mailbox not yet "accepted" by earlier Message Group, Accept macro calls, and available to subsequent Message Group, Accept macro calls. Note that a nonzero count may not necessarily reflect messages presently available; the nonzero count may indicate deferred messages.

Before execution of this call, the user must generate a MGIRB (by means of the $MGIRB macro call) with values for the following fields:

MI_MAJ, bit 9 (wait bit)
MI_MPD
MI_ADT, right byte
MI_MBA

MI_MBA specifies the name of the mailbox, which must have been created before execution of this call by means of the Create Mailbox (CMBX) command (see the <u>Commands</u> manual).

All fields of the MGIRB are described in Appendix C.

NOTES

1. The system places the address of the MGIRB in $B4. If this argument is omitted, the system assumes that $B4 contains a pointer to the MGIRB.

2. At successful macro execution, MI_CNT will contain the count of "unaccepted" completed message groups remaining in the mailbox.

3. On return, $R1 contains the following return status codes:

   0000 - No error

   0C02 - Invalid messge group id

   0C03 - Abnormal termination received

   0C19 - Acceptor mailbox may not be accessed by the initiator

   0C1A - Acceptor mailbox or acceptor mailbox node not known

   0C22 through 0C2C - User-coded reason for abnormal message group termination.

6. On return, $B4 will point to the application's MGIRB, which is updated according to the specifications in the macro call.

MESSAGE GROUP, INITIATE ($MINIT)

Function Code: 15/02

Equivalent Command: None

Start the process of sending a mesasge group to a previously created mailbox.

FORMAT:

```
[label]   $MINIT   [location of MGIRB address]
```

ARGUMENT:

location of MGIRB address

Any address form valid for an address register; provides the address of the message group initialization request block (MGIRB), which must have been previously generated.

DESCRIPTION:

The sender task group issues this call in order to send a message. The call is effective only for a one-way connection to another task group's mailbox. For the other task group to send messages, it must create its own initiator mailbox and issue its own Message Group, Initiate macro call.

The message may be deferred by specifying a maturity date/time in MI_DV2 of the MGIRB.

Successful macro call execution requires send access to the mailbox (list access on mailbox directory). See the System Concepts manual for a discussion of mailbox access.

Before execution of this call, the user must generate a MGIRB (by means of the $MGIRB macro call) with values in the following fields:

MI_MAJ, bit 9 (wait bit)
MI_MPD
MI_DV2 (optional)
MI_ADT, right byte
MI_MBI
MI_MBA

MESSAGE GROUP, INITIALIZATION REQUEST BLOCK ($MGIRB)

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, perform one of the following:

- Build a message group initialization request block (MGIRB) of 41 words that contains default values for all fields not explicitly specified in the call. See Appendix C.
- Generate instructions to alter the partial contents of an existing MGIRB.
- When modifying an existing MGIRB, call and expand the corresponding Message Group Initialization Request Block Offsets macro call to provide labels for the MGIRB's fields.

FORMAT:

```
[label]   $MGIRB , [arguments]
```

ARGUMENTS:

There are three types of arguments for this macro call:

- Keyword only (i.e., RESV)
- Keyword with expression (expression is a user-selected variable whose literal value is used by the system)
- Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGIRB. When the macro call is issued with RESV as its only argument, an MGIRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGIRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGIRB, in which case the keyword-with-expression argument ADR=address is used to specify the address of the MGIRB to be changed. When ADR=address is omitted, the system assumes that $B4 points to that MGIRB. The argument ADR=address is not used in building a new MGIRB; that is, when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 2-16 below.

The first argument position is reserved for system use, and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by a comma.

Table 2-16 describes the arguments for the Message Group, Initialization Request Block macro call and indicates the fields in the MGIRB into which the system inserts the argument values.

DESCRIPTION:

The message group initialization request block (MGIRB) is used for communication among task groups, and is the means for passing arguments among task groups in connection with the Message Group Accept, Message Group Initiate, and Message Group Count macro calls of the Message facility. This macro call makes it possible to modify an existing MGIRB by generating executable instructions that use registers R6, R7, and B5 (as appropriate). The modifying process always uses $B4 to point to the MGIRB.

Table 2-16. Argument Values for $MGIRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only				
1	None	None	Reserved by system; must be a comma.	N/A
2			Issuing task suspension option:	MI_MAJ
	WAIT*	None	Suspend the issuing task until the request is completed (set W-bit (WAIT) to zero).	
	NWAIT (default)	None	Do not suspend the issuing task (set W-bit to one).	
Any	RESV	None	Generates the MGIRB.	
Keyword with expression				
3**			Issuing task termination option	N/A
	SM=	aa	When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.	
	RB=	label	When requested task is completed, do not suspend the issuing task; issue a request for the request block identified by label. Note that the requesting task must be asynchronous, may not wait on the requested task later on, and can only point to a task request.	

Table 2-16 (cont). Argument Values for $MGIRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword with expression				
3 (cont)			block (TRB). The requested task must have already been created (not spawned), be asynchronous, and have a valid LRN. When the requesting task terminates, the TRB pointed to by "label" must be inactive.	
Any	ADR=	address	When existing MGIRB is to be changed (RESV omitted), specifies address of MGIRB to be changed.	N/A
Any	MBI=	Initiator mailbox name: From 1 to 12 ASCII characters, blank-filled, left-justified. Default is 12 blanks.		MI_MBI
Any	MBA=	Acceptor mailbox name. From 1 to 12 ASCII characters, blank-filled, left-justified. Default is 12 blanks.		MI_MBA

*When WAIT is specified, argument 3 must be omitted.

**When this argument is omitted, or argument 2 is WAIT, the generated MGIRB contains no termination option. In that case, the user must issue a Wait, Wait on Request List, or Test Completion Status macro call.

MESSAGE GROUP, RECEIVE ($MRECV)

Function Code: 15/03

Equivalent Command: None

Request that this task group receive a message group through a named mailbox, from another task group; specify how much message data is to be received; detect when there is no more data to be received.

FORMAT:

```
[label]   $MRECV   [location of MGCRB address]
```

ARGUMENT:

location of MGCRB address

Any address form valid for an address register; provides the address of the message group control request block (MGCRB), which must have been previously generated.

DESCRIPTION:

The task group that issued the Message Group Accept macro call to open the receive function of the Message Facility can issue one or more Message Group, Receive macro calls to receive message data, from the sending task group, through a named mailbox. The message group id returned in the Message Group Accept macro call is used by the Message Group, Receive macro call to identify the message group of the receiving task group. A receive message can be any unit, not necessarily exactly as defined by the sender. A portion of a message group cannot be available to the receiving task group until designated as a quarantine unit by the sender. The Message Group, Receive macro call can request that the message be received in record sizes other than those with which it was sent. It can specify how much data is to be received in terms of numbers of bytes (range) and by "enclosure level" (see below). Every receive unit is an enclosure. The receiving task group can delimit the amount of received data as end-of-record, end-of-quarantine-unit (see description of quarantine unit under the Message Group, Send macro call) or as end-of-message. Upon receipt of a quarantine unit, the previous quarantine unit is deleted.

Mailboxes must have been created before this macro call is issued. (See the Create Mailbox (CMBX) command in the Commands manual.)

Before issuing the macro call, the user must generate the MGCRB (see the Message Group Control Request Block macro call) with the argument values shown in Table 2-17.

At successful macro execution, the system returns the following MGCRB output argument values:

text residual range

MC_RSR field reports the number of bytes of text not transferred into the buffer area. When a record has no text associated with it, the value will equal buffer size.

text length

MC_LEN field reports the number of bytes of text transferred into the buffer if the revision-1 id was specified in the MC_REV field of the MGCRB.

detected user enclosure level:

MC_LVL field (bits 8-F) reports the enclosure level detected at end of transfer. Possible values (ASCII):

0 - No enclosure detected
1 - End-of-record
2 - End-of-quarantine-unit
5 - End-of-message

After successful receipt of a complete message (i.e., value of detected enclosure level in bits 8-F in MC_LVL is ASCII 5), the receiving task group must issue a Message Group Terminate macro call to terminate the message group. (See Message Group, Terminate macro call for a discussion of normal and abnormal termination.)

NOTES

1. The system places the address of the MGCRB in $B4. If the argument is omitted, the system assumes that $B4 contains a pointer to the MGCRB.

2. On return, $R1 contains the following status codes:

   0000 - No error

   000F - No data available

   021A - Record length error

   0C02 - Invalid message group id

   0C03 - Abnormal termination received

   0C09 - Invalid enclosure level specified

   0C10 - Message quarantine unit exceeded capacity

   0C22 through 0C2C - User-coded reason for abnormal message group termination.

3. On return, $B4 will point to the application's MGCRB, which is updated according to the specifications in the macro call.

Table 2-17. MGCRB Argument Values for $MRECV Macro Call

Argument Name and Description	Field in MGCRB	Argument Value
message group id Identifies the message group within whose enclosures the record is to be received.	MC_MGI	Value returned in $MACPT macro call.
buffer area id Defines the location within the task where the received record is to be placed.	MC_BUF	Buffer pointer.
range Defines the maximum number of bytes to be placed into the buffer area in one execution of the macro call. When the specified range is exceeded, the transfer of message text is terminated.	MC_BSZ	User-specified.
requested enclosure level Amount of data, in text units, that the receiving task group is to receive. When the buffer range is exceeded, text transfer terminates.	MC_LVR (bits 0-7)	ASCII values: 1 - End-of-record, but not last record in quarantine unit. 2 - End-of-quarantine-unit. 5 - End-of-message.
wait test indicator Specifies whether user waits for data, even if none now available; or whether request is terminated when there is no data.	MC_WTI (bits 8-F)	0 - Terminate the request. 1 - Wait for data to become available.

Table 2-17 (cont). MGCRB Argument Values for $MRECV Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
synchronous/asynchronous indicator Indicates whether macro call execution is to be synchronous or asynchronous.	MC_MAJ (bit 9)	0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed. 1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.

MESSAGE GROUP, RECOVERY REQUEST BLOCK ($MGRRB)

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, perform one of the following:

- Build a message group recovery request block (MGRRB) of 27 words that contains default values for all fields not explicitly specified in the call. MGRRB fields are described in Appendix C.
- Generate instructions to alter the partial contents of an existing MGRRB.
- When modifying an existing MGRRB, call and expand the corresponding Message Group, Recovery Request Block Offsets macro call to provide labels for the MGRRB's fields.

FORMAT:

```
[label]   $MGRRB ,[arguments]
```

ARGUMENTS:

There are three types of arguments for this macro call:

- Keyword only (i.e., RESV)
- Keyword with expression (expression is a user-selected variable whose literal value is used by the system)
- Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGRRB. When the macro call is issued with RESV as its only argument, an MGRRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGRRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGRRB, in which case the keyword-with-expression argument ADR=address is used to specify the address of the MGRRB to be changed. When ADR=address is omitted, the system assumes that $B4 points to that MGRRB. The argument ADR=address is not used in building a new MGRRB; that is, when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 2-18.

The first argument position is reserved for system use and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by a comma.

Table 2-18 describes the arguments for the Message Group, Recovery Request Block macro call, and indicates the fields in the MGRRB into which the system inserts the argument values.

DESCRIPTION:

The Message Group Recovery Request Block macro call is used for communication between task groups and is the means for passing arguments between task groups in connection with the Message Group, Terminate macro call of the Message Facility. This macro call makes it possible to modify an existing MGRRB by generating executable instructions that use registers R7 and B5 (as appropriate). The modifying process always uses $B4 to point to the MGRRB.

Table 2-18. Argument Values for $MGRRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only				
1	None	None	Reserved by system; must be a comma.	N/A
2			Issuing task suspension option:	MR_MAJ
	WAIT*	None	Suspend the issuing task until the request is completed (set W-bit (wait) to zero).	

Table 2-18 (cont). Argument Values for $MGRRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only (cont).				
	NWAIT (default)	None	Do not suspend the issuing task. (Set W-bit to one).	
Any	RESV	None	Generates the MGRRB.	
Keyword with expression				
3**			Issuing task termination option:	N/A
	SM=	aa	When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.	
	RB=	label	When requested task is complete, do not suspend the issuing task; issue a request for the request block identified by label.	
Any	ADR=	address	When existing MGRRB is to be changed (RESV omitted), specifies address of MGRRB to be changed.	N/A

*When WAIT is specified, argument 3 must be omitted.

**When this argument is omitted, or argument 2 is WAIT, the generated MGRRB contains no termination option. In that case, the user must issue a Wait, Wait on Request List, or Test Completion Status macro call.

Table 2-18 (cont). Argument Values for $MGRRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only (cont).				
Any	TERM=	0, or 22 through 2C	Message group termination code: 0 - Indicates normal termination of this message group. 22 through 2C - User-coded reason for abnormal termination.	MR_RSN

MESSAGE GROUP, RECOVERY REQUEST BLOCK OFFSETS ($MGRRT)

Generated Label Prefixes:

MR_OS
MR_MAJ
MR_OPT
MR_BUF
MR_BSZ
MR_ITP
MR_RES
MR_RSN
MR_EXT
MR_FNC/MR_REV
MR_MGI
MR_CNC
MR_FMT
MR_MRU
MR_AMU

Appendix C describes the contents of the message group recovery request block (MGRRB).

MESSAGE GROUP, SEND ($MSEND)

Function Code: 15/05

Equivalent Command: None

Send a specified amount of message text from the initiator task group. Optionally, make this record and any previously sent records available to the receiver by declaring this message text as a quarantine unit.

FORMAT:

```
[label]   $MSEND   [location of MGCRB address]
```

ARGUMENT:

location of MGCRB address

Any address form valid for an address register; provides the address of the message group control request block (MGCRB), which must have been previously generated.

DESCRIPTION:

The task group that issued a Message Group, Inititiate macro call to initiate a message connection, issues one or more Message Group, Send macro calls to send message data through that connection. A task group sends a message through a named mailbox, from which the receiving task group obtains the message. The Message Group, Send macro call uses the same message group id, returned in the Message Group, Initiate macro call, to identify the message group.

Text units of information sent by the sending task group (initiator) are in the form of records. A message is one or more records. Each Message Group, Send call sends one record, which is the basic unit of data exchange. Each Message Group, Send transmission points to an MGCRB that describes the buffer of message data.

Associated with each message group is the concept of a nested set of enclosures. All message group text is contained within a hierarchy of enclosures, which from the lowest to the highest are:

- Record
- Quarantine unit
- Message.

Intermediate or last records in the message have an enclosure level that defines sent data as end-of-record, end-of-quarantine unit, or end-of-message. Terminating any of these enclosure levels forces termination of a lower level enclosure; that is, end-of-message implies end-of-quarantine unit, and end-of-quarantine unit implies end-of-record.

A record enclosure consists of all message text transferred by one or more Message Group, Send macro calls. The Message Facility accepts text sent by a Message Group, Send macro call as part of the same record, until another Message Group, Send includes an end-of-record indicator, signalling the end of the record and beginning of a new one.

A quarantine unit enclosure, which is terminated by an end-of-quarantine indicator, consists of all the records transmitted since the last end-of-quarantine indicator was sent. Not until an end-of-quarantine indicator is included in the enclosure level (see Table 2-19) of a Message Group, Send macro call, is the group of records, sent since the last end-of-quarantine indicator, made available to the receiving task group. The end-of-quarantine indicator also terminates the current record enclosure. A quarantine unit is the smallest amount of transmitted data that is available to the receiver.

Before execution of this macro call, the user must have done the following:

- Created mailboxes by means of the Create Mailbox (CMBX) command (see the Commands manual)
- Generated a MGCRB by means of the $MGCRB macro call, with the values shown in Table 2-19.

To complete sending a message group, the sending task group must terminate the message group by either:

1. Specifying an ASCII 5 (end-of-message) enclosure level in MC_LVL of the MGCRB (see Table 2-19) supplied on a Message Group, Send macro call

or

2. Issuing the macro call Message Group Terminate, with the value zero in MR_CNC of the MGRRB. (See the Message Group, Terminate macro call for a discussion of normal and abnormal termination.)

NOTES

1. The system places the address of the MGCRB in $B4. If the argument is omitted, the system assumes that $B4 contains a pointer to the MGCRB.

2. On return, $R1 contains the following return status codes:

   0000 - No error

   0C02 - Invalid message group id

   0C03 - Abnormal termination received

   0C09 - Invalid enclosure level specified

   0C10 - Message/quarantine unit exceeded capacity

   0C22 through 0C3E - User-coded reason for abnormal message group termination.

3. On return, $B4 will point to the application's MGCRB, which is updated according to the specification in the macro call.

Table 2-19. MGCRB Argument Values for $MSEND Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
synchronous/asynchronous indicator Indicates whether macro call execution is to be synchronous or asynchronous.	MC_MAJ (bit 9)	0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed. 1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.

Table 2-19 (cont). MGCRB Argument Values for $MSEND Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
message group id Identifies the message group within whose enclosure the record is to be sent.	MC_MGI	Value returned in Message Group, Initiate macro call.
buffer area id A pointer to the buffer where message text is located before it is transmitted.	MC_BUF	Buffer pointer.
user-requested enclosure level Defines the unit of text; that is, how much data is contained in an "enclosure level."	MC_LVL (bits 0-7)	ASCII values: 1 - End-of-record, but not last record in a quarantine unit. 2 - End-of-quarantine unit. 5 - End-of-message.
range Indicates the number of bytes of message text to be sent from the buffer area. A zero value indicates no text is to be sent; even then, the other argument values are examined and a record enclosure is opened, if not already open.	MC_BSZ	User-specified.

MESSAGE GROUP, TERMINATE ($MTMG)

Function Code: 15/04

Equivalent Command: None

Terminate a message group, either normally or abnormally.

FORMAT:

```
[label]  $MTMG  [location of MGRRB address]
```

ARGUMENT:

location of MGRRB address

Any address form valid for an address register; provides the address of the message group recovery request block (MGRRB), which must have been previously generated.

DESCRIPTION:

This macro call, issued by a sending or receiving task group, causes normal or abnormal termination of a message group. A sending task group, after normal transmission of a message, must terminate the message group with either a Message Group, Send macro call that specifies an end-of-message enclosure level (5 in MC_LVL of the MGCRB), or with a Message Group Terminate macro call having a termination value of zero in bits 0 through 7 of MR_RSN. The sending task group can specify abnormal termination of the message group by inserting a user-coded value from 22 through 2C in bits 0-7 of MR_RSN. This code informs the receiving task group of the reason for abnormal termination.

Normal termination of the receive message process causes the message to be deleted. Abnormal termination of the receive message process terminates the message without destroying it.

When the message group is terminated, its message group id is available for reuse.

Before execution of this macro call, the user must generate a MGRRB (by means of the $MGRRB macro call) with values in the following fields:

MR_MAJ, bit 9 (wait bit)
MR_MGI
MR_RSN, left byte

MGRRB fields are described in Appendix C.

NOTES

1. The system places the address of the MGRRB in $B4. If the argument is omitted, the system assumes that $B4 contains a pointer to the MGRRB.

2. On return, $R1 contains the following status codes:

   0000 - No error

   0C02 - Invalid message group id

   0C21 - Invalid user-coded abnormal termination

   0C22 through 0C2C - User-coded reason for abnormal message group termination.

3. On return, $B4 will point to the application's MGRRB, which is updated according to the specifications in the macro call.

MODE IDENTIFICATION ($MODID)

Function Code: 14/03

Equivalent Command: None

Returns the mode component of the calling task group's user id to a 3-character receiving field.

FORMAT:

```
[label]   $MODID   [location of mode id field address]
```

ARGUMENT:

location of mode id field address

Any address form valid for an address register; provides the address of a 3-character, aligned, nonvarying field into which the system places the mode component of the user id associated with the issuing task group.

DESCRIPTION:

This call returns the mode component of the task group's user id to a field in the issuing task. The mode id returned is that entered as part of the Login command that established the user as a primary or secondary user of this task group. See the Commands manual for details.

The entire user id is returned by the User Identification macro call.

NOTES

1. The system places in $B4 the address of the receiving mode id field, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the receiving mode id field.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0817 - Memory access violation.

3. On return, $B4 contains the address of the receiving field.

Example:

In this example, $B4 is loaded with the address (MODFL) of a 3-character field, and the Mode Identification macro call is issued to place the mode id of the task group in that field.

```
MODFL     RESV     2
          LAB      $B4,MODFL
          .
          .
          .
          $MODID
```

MODIFY FILE ($MDFIL)

Function Code: 10/41

Equivalent Command: Modify File Attributes (MFA)

Modify the attributes of a disk file.

FORMAT:

```
[label]   $MDFIL   [parameter structure address]
```

ARGUMENT:

parameter structure address

Any address form valid for an address register; provides the location of the parameter structure defined below. The parameter structure must contain the following entries, in the order shown.

logical file number

A 2-byte logical file number (LFN) used to refer to the file. It must be a binary number in the range 0 through 255, ASCII blanks (X'2020') (an LFN is not specified), or -1 (X'FFFF') (the system should assign an LFN from the pool of available LFNs).

pathname pointer

A 4-byte address of the pathname that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that, when expanded, identifies the file to be modified. Binary zeros (null pointer) in this entry indicate that a path is not specified; if the path identified is a single ASCII space (20) character, the file being created is a temporary file.

file options

A 2-byte field that is used in conjunction with the file options mask (see below) to turn on (1) or turn off (0) the various disk file options:

Bit	Meaning
0	Record lock option: 1 - Lock records allowing n readers or one writer. 0 - Do not lock records.
1	Record format option: 1 - Support fixed- and variable-length records. 0 - Support only fixed-length records.
2	Immediate update option: 1 - Immediately update the disk when a record is updated. 0 - Delay updating the disk until the buffers are full, a "cleanpoint/checkpoint" is issued, or the file is closed.
3	Recovery option: 1 - Journalize on disk the "before" images of all updates. These images can be used in the event of a program or system failure to rollback the file. 0 - Do not journalize "before" images.
4-5	Reserved; must be zero.
6	Damaged File Indicator: 1 - File is not damaged. 0 - The file's data content is in a damaged or inconsistent state. The file cannot be opened until either this indicator is reset or the file is restored.

Bit	Meaning
7	Write protect option: 1 - Place the disk file in "write protect", allowing only read access. 0 - Allow write operations to the disk file.
8	Reserved; must be zero.
9	Restoration option: 1 - Journalize on tape the "after" images of all updates. These images can be used in the event of a disk failure or corruption to restore or roll forward the file to its latest good state. 0 - Do not journalize "after" images.
A through F	Reserved; must be zero.

file options mask

A 2-byte field that indicates which file options are to be set or reset. If a bit in the mask is one, the corresponding file option is set or reset according to the value specified in the previous file options field. If a bit in the mask is zero, the corresponding file option is not modified.

Bit	Meaning
0	Record lock
1	Record format
2	Immediate update
3	Recovery
4-5	Must be zero
6	Damaged file indicator
7	Write protect
8	Must be zero
9	Restoration
A through F	Must be zero

new attributes

A 2-byte field that is used in conjunction with the attributes mask (see below) to specify new file attributes.

Bit	Meaning
0-3	Data Code Attribute: 0000 - Undefined data 0001 - Binary (non-character) data 0010 - ASCII (character) data
4-7	Must be zero
8-11	Terminal control attribute: 0000 - Unknown terminal control information 0001 - No terminal control information 0010 - GCOS 6 printer control information
12-14	Must be zero
15	Foreign data attribute: 0 - GCOS file data 1 - Non-native (non-GCOS 6) file data

attributes mask

A 2-byte field that indicates which file attribute is to be modified or left unchanged. If a bit in the mask is one, the corresponding attribute is changed according to the value specified in the previous new attributes field. If the mask is zero, the corresponding file attribute is not modified.

Bit	Meaning
0-3	Data Code attribute
4-7	Must be zero
8-11	Terminal control attribute
12-14	Must be zero
15	Foreign data attribute

free space per control interval

A 2-byte field that applies only to indexed files and specifies the number of bytes to be left free in each control interval at file loading time.

reserved

A 16-byte field that is reserved for future use.

DESCRIPTION:

This function is normally performed outside program execution by means of the Modify File Attribute (MFA) command.

The file to be modified can be specified in the argument structure by a logical file number (LFN) or a pathname. If an LFN is specified, the file must have been previously assigned to that LFN by means of the Get File or Create File function or the equivalent command.

Modify File cannot be issued if the file is currently open or reserved by a task group other than the user's.

The function modifies only disk files; it does not apply to directories or device files.

The record lock, file recovery, and file restoration options apply only to disk files organized in the following UFAS formats: sequential, relative, indexed, dynamic, and random.

If an alternate index is modified with the write protect option of this function, any future updates applied to the data file will not be reflected in the index. This option, together with the index only option of the Get File function, can be used to build an index that refers only to a subset of the data file.

A restorable file (i.e., one created or modified with the -RESTORE attribute) can be modified only if the system's journal file is open.

NOTE

On return, $R1 contains one of the following status codes:

0000	No error
01xx	Physical I/O error
0201	Invalid pathname
0202	Pathname not specified
0205	Invalid argument
0206	Unknown or invalid logical file number (LFN)
0208	LFN or file open
0209	File or directory not found
0213	Cannot provide requested file concurrency
0217	Access violation
0220	File not empty
0228	Invalid file type
022C	Access control list (ACL) violation
0260	Journal file not open.

MODIFY FILE PARAMETER STRUCTURE BLOCK OFFSETS ($MDPSB)

Associated Macro Call: Modify File

Structure:

Word	Fields
0	Logical File Number
1 2	Pathname Pointer
3	File Option
4	File Options Mask
5	New Attributes
6	Attributes Mask
7	Free Space Per Control Interval
8 9 10 11 12 13 14 15	Reserved

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
M_LFN	0	Logical file number
M_PTHP	+1	Pathname pointer
M_OPT	+3	File options
M_MSK	+4	File options mask
M_ATTR	+5	Data attributes
M_MASK2	+6	Data attributes mask
M_FPC	+7	Free space per control interval
M_LOV	+8	Reserved
M_SZ	18	Size of structure (in words); not a field in the block.

MODIFY REBOOT PARAMETERS ($RBPRM)

Function Code: 20/05

Eqivalent Command: Modify Reboot Parameters (RBPRM)

Specify the reboot volume and/or configuration file to be used by the Software Reboot Facility (SRF) when reinitializing the system. Validate preconditions for a dump to be taken immediately prior to reinitialization; alternatively, specify that a dump is not desired.

FORMAT:

```
[label]  $RBPRM  [location of reboot volume identification],
                 [location of  configuration file pathname],
                 [location of dump condition]
```

ARGUMENTS:

location of reboot volume identification

Identifies the volume to be used for reinitializing the system; must begin with one of the following key words.

PN=

Signifies "pathname"; must be followed by an address form valid for an address register; provides the address of a volume name (e.g., ^ABC) or of a device name (e g., !FCD00). A null address specifies that the current reboot volume/device is to be used for reinitializing the system. The current volume/device is that which was used to initiate the current session, or one subsequently specified by a $RBPRM call.

CH=

Signifies "channel number"; must be followed by an address form valid for a data register; provides the channel number of the device on which the desired reboot volume is mounted. A value of zero for the channel number specifies that the current reboot volume is to be used.

=$B2

Signifies that $B2 already contains the address of a volume/device name.

=$R2

Signifies that $R2 already contains the channel number of the device on which the desired reboot volume is mounted.

location of configuration file pathname

Identifies the configuration file to be used when the SRF reinitializes the system. This argument must take one of the following two forms:

Any address form valid for an address register; provides the address of the pathname of the configuration file to be used. A blank pathname, consisting of one or more spaces (e.g., ' '), specifies that the configuration file >SID>CLM_USER is to be used. An invalid pathname consisting of an asterisk followed by one or more spaces (e.g., '* ') specifies one of two configuration files supplied by the manufacturer: If the operator's console is connected to a Multiline Communications Processor (MLCP), an invalid pathname specifies >SID>CLM_MCP; if the operator's console is connected to a Multiple Device Controller (MDC), an invalid pathname specifies file >SID>CLM_MDC.

=B4

Specifies that $B4 already contains the address of the pathname of the desired configuration file.

Either form of argument 2 can supply a null address. The significance of a null address depends on the value of argument 1, as follows:

If argument 1 supplies a non-null address and argument 2 supplies a null address, the configuration file to be used is >SID>CLM_USER. If argument 1 and argument 2 both supply null addresses, the configuration file to be used is the current one. The current configuration file is that which was used by the Configuration Load Manager (CLM) when the current session was initialized, or one subsequently specified by a $RBPRM call.

location of dump condition

Specifies whether a dump is to taken immediately before reinitialization; must be one of the following keywords:

DUMP

Take a dump.

NDUMP

Do not take a dump.

=$R6

$R6 contains the value 0 or 1, indicating DUMP or NDUMP, respectively.

NOTE

The PATH argument of the CLM directive REBOOT implicity instructs the SRF to take a dump before reinitializing the system; omitting the PATH argument implicity instructs the SRF not to take a dump. Specifying the DUMP keyword of argument 3 does not override a REBOOT directive whose PATH argument is omitted. A user who omits the PATH argument can later direct the SRF to take a dump only by modifying the REBOOT directive so that it provides a value for the PATH argument.

On the other hand, specifying the keyword NDUMP of argument 3 does override a REBOOT directive that provides a value for the PATH argument.

DESCRIPTION:

$RBPRM can modify parameters currently entered in the system control block (SCB) fields S_BTD1, S_CF, and S_DMP. These fields identify respectively the reboot volume, configuration file, and dumpfile (if any), to be used by the SRF.

The current values of S_BTD1 and S_CF are established in one of two ways:

- By the initialization of the current session. Assume for example, that the operator initiated the current session by mounting the reboot volume ^ZSYS71, and that the Configuration Load Manager (CLM) used the configuration file >SID>CLM_USER residing on this volume. Assume further that after the current session was initialized, a Modify Reboot Paramaters macro call/command has not specified a different reboot volume or configuration file. The SCB currently indicates that volume ^ZSYS71 and configuration file >SID>CLM_USER are to be used by the SRF for reinitializing the system.

- By the last Modify Reboot Parameters macro call/command issued in the current session.

$RBPRM modifies SCB entries S_BDT and S_CF only after validating that:

- The reboot volume specified by argument 1 exists, is mounted, and is a valid reboot volume.

- The configiruation file specified by argument 2 resides on the reboot volume specified by argument 1, and is of the proper file type.

- The directory containing the configuration file specified by argument 2 also contains a START_UP.EC file.

After validating the values entered for arguments 1 and 2, $RBPRM modifies the current reboot parameters in S_BDT and S_CF.

As explained above, the current dumpfile parameter is established by either a CLM REBOOT directive or a Modify Reboot Parameters macro call/command. The REBOOT directive initially establishes the dumpfile parameter; a Modify Reboot Paramters macro call alters the inital dumpfile parameter only if:

- The call is issued in the current session.
- The REBOOT directive specified that a dump be taken.
- Argument 3 of the call specifies NDUMP.

If the REBOOT directive specified that a dump be taken, specifying DUMP for argument 3 validates that the conditions necessary for a dump exist (i.e., that the dumpfile is configured, is unlocked, and is not write-protected). If any of these conditions do not exist, an error message is returned in $R1.

Argument 3 provides an easy way of determining whether or not the CLM REBOOT directive specified that a dump be taken: entering DUMP for argument 3 results in the error message "Dumpfile not configured" if the REBOOT directive did not specify a dump.

$RBPRM is a privileged call (i.e., it can be issued only by a task running in a memory pool configured as privileged).

NOTES

1. If argument 1 specifies the keyword "PN=" followed by the address of a pathname, the system places the address in $B2 and sets $R2 to 0. Alternatively, if argument 1 specifies the keyword "CH=" followed by a channel number, the system places the channel number in $R2. If argument 1 is omitted, the system:

   - Assumes that $B2 contains the address of a new volume/device name or a null address (which signifies the current volume/device name). Note that the system does not generate a default value in $B2; the user, if omitting argument 1, is responsible for placing the correct value in $B2.

   - Sets $R2 to 0.

2. If argument 2 specifies the address of a configuration pathname, the system places the address in $B4. If argument 2 specifies =B4, or if argument 2 is omitted, the system assumes that $B4 contains the pathname of the configuration file desired. The system does not generate a default value in $B4; the user is responsible for placing the correct value in that register.

3. If argument 3 specifies the keyword "DUMP", the system sets $R6 to 1; if "NDUMP" is specified, the system sets $R6 to 0. If argument 3 is omitted, the system assumes that $R6 contains the correct value; the system does not generate a default value in that register.

4. On return, $R1 contains one of the following status codes:

0000 - Reboot parameters successfully modified

083A - Function illegal for unprivileged task group

0865 - Dumpfile not configured

0866 - Cannot reserve reboot volume

0867 - Attributes of specified volume incorrect for reboot volume

0868 - Cannot reserve configuration file

0869 - Error returned by file system when attempting to reference dumpfile

086A - Attributes of specified file incorrect for configuration file

086E - Dumpfile write-protected

086F - Cannot reserve START_UP.EC file.

Example:

In this example, $RBPRM is issued to modify the current reboot volume and configuration file parameters, and to validate the current dump parameter. On return, $R1 will indicate whether the dumpfile, previously specified by the CLM REBOOT directive, is ready to receive a dump. When next activated, the SRF will reinitialize the system using the new reboot volume ^MYVOL and the new configuration file MYDIR>MYCLM.

```
         $RBPRM         PN=VOL,CF,DUMP
         .
         .
         .
VOL      DC             '^MYVOLΔ'
CF       DC             'MYDIR>MYCLM '
```

3. On return, $R1, $R6, $R7, $B2, and $B4 contain the following information:

   $R1 - Return status; contains the following:

   0000 - No error

   All file management get-file and open-file error codes may also be returned

   $R6 - Record length of the redefined file

   $R7 - File status/type of the redefined file

   $B2 - Address of the argument list (if supplied)

   $B4 - Address of the pathname of the new command-in file (if supplied).

NEW MESSAGE LIBRARY ($NMLF)

Function Code: 08/08

Equivalent Command: None

Redefine or set the message library for the issuing task.

FORMAT:

```
[label]  $NMLF  [location of pathname address]
```

ARGUMENT:

location of pathname address

Any address form valid for an address register; provides the address of the pathname of the new message library file.

DESCRIPTION:

$NMLF allows a task to redefine or set its message library. A task's message library is initially defined when its task group is requested (or spawned). At that time, the requestor may specify a message library file, which is assigned to the lead task of the requested group. If the requestor does not specify a message library, the lead task's message library defaults to that of the requestor. If no message library is defined for the requestor (e.g., because no system message library is configured), none is assigned to the lead task. All tasks created by the lead task share the lead task's message library (if one exists). $NMLF allows a task to use a message library other than one inherited from its parent task. Alternatively, should no message library be defined for the parent task, $NMLF allows the created task to establish one.

$NMLF opens the file specified by the argument, and sets the file's usage count to one. If a different message library file was previously assigned to the calling task, $NMLF decrements the usage count of that file. If the usage count of the old file reaches zero, the old file is closed and removed.

NOTES

1. The address of the pathname of the new message library file supplied by argument 1 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the pathanme.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0602 - Memory unavailable for message library file control structure

   All file management Get File, Open File, Close File, and Remove File error codes may also be returned in $R1. These codes are listed in the System Messages manual.

   $B4 - Address of pathname string of new message.

Example:

In this example, $NMFL defines the message library of the issuing task as ^V1124>UDD>JONES>MYERR.

```
INML    $NMLF     !NEWML
        .
        .
        .
NEWML   DC        '^V1124>UDD>JONES>MYERRΔ'
```

NEW PROCESS ($NPROC)

Function Code: 0D/0B

Equivalent Command: New Process (NEW_PROC)

Terminate the current task group request and restart the task group request with the same parameters as the original invocation of the task group for this request.

FORMAT:

```
[label]   $NPROC
```

ARGUMENT:

There are no arguments for this macro call.

DESCRIPTION:

This macro call terminates the current request for the issuing task group, then restarts the request using the same parameters as in the original request.

Example:

In this example, the New Process macro call is used to terminate and restart the task group request.

```
AGAIN    $NPROC
```

NEW USER INPUT ($NUIN)

Function Code: 08/04

Equivalent Command: None

Redefine, reset, or set the user-in file for the issuing task. The user-in file can be redefined by a new pathname, reset to the initial user-in file, or set to the file currently defined as the command-in file. The action taken applies only to the task that issues the macro call.

FORMAT:

```
[label]   $NUIN   [location of pathname address]
```

ARGUMENT:

location of pathname address

Any address form valid for an address register; provides the pathname of the file that is to be used as the new user-in file for the issuing task. If $CIN is specified for this argument, the file currently defined as the task's command-in file is also used as the new user-in file. If this argument is omitted, the file defined by the Reqeust Group macro call as the user-in file for tasks in this task group is again used for this task.

DESCRIPTION:

This call allows the issuing task to use another file as the user-in file.

If a pathname is specified in the macro call, input is read from the file identified by the pathname.

If $CIN is specified for this argument, the file that is currently the task's command-in file is used as the source of input for the task.

If the call is written without an argument, the user-in file is identified as the initial user-in file for this task. (The Request Group macro call identifies this user-in file.)

The New User Input call also performs a Close File and Remove File of the existing user-in, if one exists, and a Get File and Open File of the new user-in file.

When the macro call has been executed, $R6 contains the record length of the new user-in file, and $R7 contains the file status.

NOTES

1. If argument 1 is a pathname address, $R2 is set to zero and the pathname supplied by argument 1 is placed in $B4. If argument 1 is $CIN, $R2 is set to two. If argument 1 is omitted, $R2 is set to one.

2. On return, $R1, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0817 - Memory access violation

   All file management get-file and open-file error codes may also be returned. See the System Messages manual.

   $R6 - Record length of redefined file

   $R7 - File type of redefined file (see the Command In macro call)

   $B4 - Address of pathname string of new user-in file (if pathname was supplied in argument 1).

Example:

In this example, the issuing task is to read its input from a new user-in file name, ^V1124>UDD>TEST>JONES.

```
    INAA     $NUIN    !NEWIN
             .
             .
             .
    NEWIN    DC       '^V1124>UDD>TEST>JONESΔ'
```

NEW USER OUTPUT ($NUOUT)

Function Code: 08/05

Equivalent Command: File Out (FO)

Redefine or reset the user-out file for the task group of the issuing task. The user-out file can be redefined by a new pathname or reset to the user-out file initially defined for the issuing task group. The action taken applies to all tasks in the task group from which the command is issued.

FORMAT:

```
[label]   $NUOUT   [location of pathname address]
```

ARGUMENT:

location of pathname address

Any address form valid for an address register; provides the pathname of the file to be used as the new user-out file for the issuing task group. If this argument is omitted, the file defined by the Request Group macro call is used as the user-out file for tasks in this task group.

DESCRIPTION:

This call allows the issuing task group to use another file as the user-out file.

If a pathname is specified in the macro call, the tasks in this task group write their output to the file identified by the pathname.

If the call is written without an argument, the user output file identified as the initial output file for this task group is used for task output. (The Request Group macro call identifies the initial user-out file.)

$NUOUT also performs a Close File and Remove File for the existing user-out file (if one exists) and a Create File, Get File, and Open File for the new user-out file. If the user-out file already exists, the Create File is ignored.

When the macro call has been executed, $R6 contains the record length of the new user-out file, and $R7 contains its file status.

NOTES

1. The address of the pathname supplied by argument 1 is placed in $B4, and $R2 is set to zero. If this argument is omitted, $R2 is set to one.

2. On return, $R1, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0817 - Memory Access Violation

   All file management get-file, create-file, and open-file error codes may also be returned. See the System Messages manual.

   $R6 - Record length of redefined file

   $R7 - File type of redefined file (see the Command In macro call)

   $B4 - Address of pathname string of new user-out file (if a pathname was specified in argument 1).

Example:

In this example, the user-out file is reset to its initial definition.

```
    OUTBK    $NUOUT
```

OPEN FILE ($OPFIL)

Function Code: 10/50 (preserve), 10/51 (renew)

Equivalent Command: None

Initialize and establish addressability to a file (which can be used by any task in the group). The file to be opened is identified by supplying its logical file number (LFN).

FORMAT:

```
[label]  $OPFIL  [FIB address]  [{,PRESERVE}]
                                [{,RENEW   }]
```

ARGUMENTS:

fib address

Any address form valid for an address register; provides the location of the file information block (FIB). The FIB must contain a valid LFN and program view.

{PRESERVE}
{PRE}

Specifies that this is an existing data file, and that labels and data already in the file are to be preserved. Reading starts from the first logical record; writing starts at the current logical end-of-file. PRESERVE is the default value for this macro call.

For indexed files only, specifying PRESERVE means that a file, when opened, cannot be opened by anyone else in RENEW mode.

{RENEW}
{REN}

Specifies that this is a new file that is considered empty until data is written to it.

For disk files, both writing and reading start from the first logical record.

For tape files, RENEW is used to rewrite an existing file or add a new file to a volume. Write permission must be granted in the FIB program view word.

For indexed files, RENEW requires that, after the file is opened, the user write records in ascending sequence by key value. This is a special "load mode" that generates the index.

DESCRIPTION:

Before this macro call can be issued, the following actions must have occurred:

1. The specified file must physically exist (i.e., it must have been created through a Create File macro call).

2. The LFN must have been associated with the external file through an Associate File, Get File, or Create File macro call (or through an equivalent command).

If a file is currently opened elsewhere in the system (by any LFN in the requesting task group or any other task group), the following rules apply:

- Opening the file in RENEW mode is not allowed.

- Opening an indexed file in PRESERVE mode is not allowed if the file is currently open in RENEW mode.

- Opening a tape file in any mode is not allowed.

If an Associate File macro call was executed, but a Get File macro call was not, the Open File macro call will attempt to reserve the file with exclusive concurrency control. (This method of opening a file is not recommended.)

A file cannot be opened directly through its pathname. If the user issues a Get File or Create File macro call with only a pathname (no LFN specified), the system will assign an LFN, which the user can then use to open the file.

If an indexed sequential file is empty (i.e., has been created but never opened in RENEW mode), and this file is opened in PRESERVE mode, the system converts the open to an open in RENEW mode and provides no notification of this change. Subsequent reads, writes, and rewrites will operate as though the file were empty.

The following discussion and rules apply only to magnetic tape files.

1. Certain tape search rules are used when the file is opened to locate the required tape file. These rules are applied when the tape is opened for data management (record-level) access, or when a file name is specified and the tape is opened for storage management (block level) access. Table 2-20 defines these rules.

Table 2-20. Tape File Search Rules for $OPFIL Macro Call

File Label Type and Open Mode	FSN* Value in $GTFIL Call	Result
Labeled tapes opened in PRESERVE mode:		
File name not specified	0/FF	Tape positioned to next file.
	n	Tape positioned to nth file.
File name is specified	0	Tape positioned to next file; file name in header label is compared to specified file name.
	n	Tape positioned to nth file; file name in header label is compared to specified file name.
	FF	Tape searched, in <u>forward</u> direction only, for a header label with a matching file name.
Labeled tape opened in RENEW mode	0	Tape positioned to next file.
	n	Tape positioned to nth file.
	FF	Tape positioned, in <u>forward</u> direction only, to a file with a matching file name. If no match is found, the new file is appended after the end of all existing files on the last tape volume.

Table 2-20 (cont). Tape File Search Rules for $OPFIL Macro Call

File Label Type and Open Mode	FSN Value in $GTFIL Call	Result
Unlabeled tapes opened in PRESERVE mode (file or volume name cannot be specified)	0/FF	Tape positioned to the next file (past the next tape mark).
	n	Tape positioned to the nth file (past the nth tape mark).
Unlabeled tapes opened in RENEW mode (file or volume name cannot be specified)	0	Tape positioned to the next file (past the next tape mark).
	n	Tape positioned to the nth file (past the nth tape mark).
	FF	Tape positioned forward only to the end of existing data; the new file is appended after the end of all existing files on the tape.

*FSN = Tape file sequence number argument in $GTFIL macro call.

2. For tapes opened in PRESERVE mode, the position of data within the file is determined as follows:

   a. If only read permission is granted (FIB program view word allows read but not write), the header label group is processed and the file is positioned directly in front of the first data record.

   b. If only write permission is granted (FIB program view word allows write but not read), the header label group is processed and the file is positioned directly after the last data record. This, in effect, is "append" mode, a way for the user to add records to the end of a file without having to read past all the existing data records.

      Trailer labels and an end-of-data tape mark are written when the file is closed. Files following the current file are lost.

c. If read and write permissions are granted (FIB program view word allows both read and write), the header label group is processed and the file is positioned directly in front of the first data record. Any write request issued after the file is opened will cause all data records that were read to be preserved, and those records that were not read to be lost. This procedure can be used to preserve part of the file while renewing the rest.

   If no write operations are done and the file is closed, no trailer labels are written. Thus, files located after the current file are preserved.

   If write operations are done, trailer labels and an end-of-data tape mark are written when the file is closed. Files that follow the current file are lost.

3. For tapes opened in RENEW mode, the position of data within the file is determined as follows:

   a. Creation of the new file is initiated at the current tape position. (If the tape is positioned at beginning of tape (BOT), the volume header label is bypassed.) The header label group is written as specified in the preceding Get File macro call. After these actions, the tape is positioned at the end of the header label group.

   b. Data and/or files following the current tape position are destroyed when the file is opened.

As part of the initialization process, this macro call verifies that sufficient space is available for buffers and control structures.

This macro call must be issued before any of the data management or storage management macro calls can be executed.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined through the File Information Block Offsets macro call.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0208 - LFN or file already open

   0209 - Named file or directory not found

   020C - Named volume not found

   0214 - Bad program view of file

   0217 - Access violation

   0218 - Damaged file

   0225 - Not enough system memory for buffers or structures

   0226 - Not enough user memory for buffers or structures

   022C - Access control list (ACL) violation

   0232 - Invalid tape file header or tape file trailer label

   0237 - Invalid record or control interval format

   0238 - Invalid file description information.

Example:

This Open File example opens a new file, in which records are to be written through the data management macro call(s) that follow this macro call.

Following is a sample sequence of macro calls and FIB used to open FILE_A for processing.

```
FILE_A   DC       Z'0005'
         .        .
         .        .                          (See "Assumptions
         .        .                          for File System
MYFIB    DC       Z'0005'                    Examples" in
         .        .                          Appendix A)
         .        .
         .        .
KEY      DC       Z'0000FFFF'
         .        .
         .        .
         .        .
IDX01    DC       '^VOL03>SUBINDEX.A>FILE_AΔ'  (See Create File
                                             macro call)
WRTFIL   DC       Z'0005'                    (See Get File
         .        .                          macro call)
         .        .
         .        .
         $CRFIL   !FILE_A or $GTFIL !WRTFIL

         $OPFIL   !MYFIB,RENEW

         $WRREC   !MYFIB                     (See Write Record
                                             macro call)
```

OPERATOR INFORMATION MESSAGE ($OPMSG)

Function Code: 09/00

Equivalent Command: Message (MSG)

Display an information message on the terminal designated as the operator terminal.

FORMAT:

```
[label] $OPMSG [location of IORB address]
```

ARGUMENT:

location of IORB address

Any address form valid for an address register; provides the address of the Input/Output Request Block (IORB) that describes the location and range of the output information message. See Appendix C for a description of the IORB.

DESCRIPTION:

This macro call enables the issuing task to send a message to the system operator. The location of the message and its range are specified in the IORB (which is generated by the Input/Output Request Block macro call or coded by the user). The IORB also specifies whether control is to be returned to the issuing task immediately or the task is to wait until the message is displayed.

NOTES

1. The system places in $B4 the address of the IORB supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0801 - IORB in use (T-bit on)

0802 - Invalid LRN, or console message suppression in effect
0803 - Invalid wait, or the R, S, D bit in the IORB is still on
0817 - Memory access violation

The following could occur if the IORB specified the issuing task was to wait for the message to be displayed.

0104 - Invalid arguments

0105 - Device not ready

0106 - Device timeout

0107 - Hardware error (check IORB status word)

0108 - Device disabled

0109 - File mark encountered

010A - Controller unavailable

010B - Device unavailable

010C - Inconsistent request

$B4 - Address of IORB.

Example:

In this example, the Operator Information Message macro call is used to write the message labeled OP_MSG on the operator terminal. The Wait macro call ($WAIT) is later used to block the task until the message has been received by the accept message facility in the operator's task group.

```
        $OPMSG  !IORB           THIS CODE EXECUTES WHETHER OR NOT
           .                    OPERATOR'S MESSAGE WAS PHYSICALLY
           .                    WRITTEN TO THE TERMINAL.
           .
        $WAIT   !IORB           THIS CODE EXECUTES ONLY AFTER THE
           .                    MESSAGE IS PHYSICALLY WRITTEN.
           .
           .
*
*      DEFINE THE IORB.
*
IORB    RESV    $AF, 0          RSU

        TEXT    Z'00';          RETURN STATUS
                B'0';           T (IN USE) BIT
                B'1';           W (DON'T WAIT) BIT
                B'0';           U (USER) BIT                    I_CT1
                B'0';           MBZ
                B'0';           MBZ
                B'0';           MBZ
                B'01'           MUST BE 1

        TEXT    Z'00';          LRN
                B'0';           MBZ
                B'0';           B (BYTE INDEX) BIT              I_CT2
                B'0';           MBZ
                B'0';           MBZ
                Z'1'            FUNCTION CODE

        DC      <OP_MSG         BUFFER ADDRESS
        DC      OP_MLN          RANGE (IN BYTES)

        TEXT    B'0000000';
                B'0';           B (BREAK) BIT
                B'0';           D BIT (MBZ)
                B'0';           K BIT (MBZ)
                B'0';           E (KEYBOARD ECHO) BIT           I_DVS
                B'1';           L (LF) BIT
                B'0';           C (NO CR) BIT
                B'000'          MODE

        DC      0               RESIDUAL RANGE
        DC      0               STATUS WORD
*
*      END OF THE IORB.
*
OP_MSG  TEXT    'A MESSAGE TO THE OPERATOR.'
OP_MLN  EQU     2*($-OP_MSG)
```

OPERATOR RESPONSE MESSAGE

OPERATOR RESPONSE MESSAGE ($OPRSP)

Function Code: 09/01

Equivalent Command: None

Display a message on the operator terminal and place the operator's response to that message in a buffer specified by the input request block.

FORMAT:

```
[label]  $OPRSP  [location of IORB list address]
```

ARGUMENT:

location of IORB list address

Any address form valid for an address register; provides the address of a list specifying the input/output request blocks to be used. The format of the IORB list is as follows:

entry 1 - Address of IORB describing output message (to operator terminal)

entry 2 - Address of IORB describing input message (for operator response).

DESCRIPTION:

This macro call enables the issuing task to send a message to the system operator and to receive the operator's response to that message.

Two IORBs are needed: an IORB describing the output message and an IORB describing the input buffer for the response. Both IORBs are generated through an Input/Output Request Block macro call or are coded by the user.

The output message IORB describes the location and size of the output message.

The input IORB describes the location of the input buffer for the response, the size of the buffer, and whether control is to be returned to the issuing task immediately or after the response has been received (by setting the W-bit of the input IORB).

NOTES

1. The system places in $B4 the address of the IORB list supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0801 - IORB in use (T-bit on)

   0802 - Invalid logical resource number (LRN), or console message suppression in effect

   0803 - Invalid wait, or the R, S, D bit in the IORB is not zero

   0817 - Memory access violation

   The following could occur if the IORB describing the input buffer specified that the issuing task was to wait for the response.

   0104 - Invalid argument

   0105 - Device not ready

   0106 - Device timeout

   0107 - Hardware error (check IORB status word)

   0108 - Device disabled

   0109 - File mark encountered

   010A - Controller unavailable

   010B - Device unavailable

   010C - Inconsistent request

   010D - EOT on magnetic tape detected

   $B4 - Address of input IORB.

Example:

In this example, the Operator Response Message macro call causes the message labeled OP_QRY to be written on the operator terminal. A reply from the operator terminal is then read into the buffer labeled OP_ANS. The issuing task remains blocked until the above actions have been completed.

```
          $OPRSP  !IORB_L
              .
              .
              .
*
*     DEFINE THE IORB LIST.
*
IORB_L  DC      <OUT_RB,<IN_RB
*
*     DEFINE THE IORBS.
*       OUTPUT IORB:
*
OUT_RB  RESV    $AF, 0
        TEXT    Z'00', B'00000001'
        TEXT    Z'00', B'0000', Z'1'
        DC      <OP_QRY
        DC      OP_QLN
        TEXT    B'0000000000010000'
        DC      0, 0
*
*       INPUT IORB:
*
IN_RB   RESV    $AF, 0
        TEXT    Z'00', B'00000001'
        TEXT    Z'00', B'0000', Z'2'
        DC      <OP_ANS
        DC      OP_ALN
        TEXT    B'0000000000110000'
        DC      0, 0
*
*     END OF IORBS.
*
OP_QRY  TEXT    'A QUERY TO THE OPERATOR?'
OP_QLN  EQU     2*($-OP_QRY)
OP_ANS  RESV    40, 0
OP_ALN  EQU     2*($-OP_ANS)
```

```
FILE_A   DC       Z'0005'
         .        .
         .        .                          (See "Assumptions
         .        .                          for File System
MYFIB    DC       Z'0005'                    Examples" in
         .        .                          Appendix A)
         .        .
         .        .
KEY      DC       Z'0000FFFF'
         .        .
         .        .
         .        .
IDX01    DC       '^VOL03>SUBINDEX.A>FILE_AΔ'  (See Create File
                                             macro call)
WRTFIL   DC       Z'0005'                    (See Get File
         .        .                          macro call)
         .        .
         .        .
         $CRFIL   !FILE_A or $GTFIL !WRTFIL

         $OPFIL   !MYFIB,RENEW

         $WRREC   !MYFIB                     (See Write Record
                                             macro call)
```

OPERATOR INFORMATION MESSAGE ($OPMSG)

Function Code: 09/00

Equivalent Command: Message (MSG)

Display an information message on the terminal designated as the operator terminal.

FORMAT:

[label] $OPMSG [location of IORB address]

ARGUMENT:

location of IORB address

Any address form valid for an address register; provides the address of the Input/Output Request Block (IORB) that describes the location and range of the output information message. See Appendix C for a description of the IORB.

DESCRIPTION:

This macro call enables the issuing task to send a message to the system operator. The location of the message and its range are specified in the IORB (which is generated by the Input/Output Request Block macro call or coded by the user). The IORB also specifies whether control is to be returned to the issuing task immediately or the task is to wait until the message is displayed.

NOTES

1. The system places in $B4 the address of the IORB supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0801 - IORB in use (T-bit on)

0802 - Invalid LRN, or console message suppression in effect
0803 - Invalid wait, or the R, S, D bit in the IORB is still on
0817 - Memory access violation

The following could occur if the IORB specified the issuing task was to wait for the message to be displayed.

0104 - Invalid arguments

0105 - Device not ready

0106 - Device timeout

0107 - Hardware error (check IORB status word)

0108 - Device disabled

0109 - File mark encountered

010A - Controller unavailable

010B - Device unavailable

010C - Inconsistent request

$B4 - Address of IORB.

Example:

In this example, the Operator Information Message macro call is used to write the message labeled OP_MSG on the operator terminal. The Wait macro call ($WAIT) is later used to block the task until the message has been received by the accept message facility in the operator's task group.

```
        $OPMSG  !IORB          THIS CODE EXECUTES WHETHER OR NOT
           .                   OPERATOR'S MESSAGE WAS PHYSICALLY
           .                   WRITTEN TO THE TERMINAL.
           .
        $WAIT   !IORB          THIS CODE EXECUTES ONLY AFTER THE
           .                   MESSAGE IS PHYSICALLY WRITTEN.
           .
           .
*
*     DEFINE THE IORB.
*
IORB    RESV    $AF, 0         RSU

        TEXT    Z'00';         RETURN STATUS              )
                B'0';          T (IN USE) BIT             )
                B'1';          W (DON'T WAIT) BIT         )
                B'0';          U (USER) BIT               )   I_CT1
                B'0';          MBZ                        )
                B'0';          MBZ                        )
                B'0';          MBZ                        )
                B'01'          MUST BE 1                  )

        TEXT    Z'00';         LRN                        )
                B'0';          MBZ                        )
                B'0';          B (BYTE INDEX) BIT         )   I_CT2
                B'0';          MBZ                        )
                B'0';          MBZ                        )
                Z'1'           FUNCTION CODE              )

        DC      <OP_MSG        BUFFER ADDRESS             )
        DC      OP_MLN         RANGE (IN BYTES)           )

        TEXT    B'0000000';                               )
                B'0';          B (BREAK) BIT              )
                B'0';          D BIT (MBZ)                )
                B'0';          K BIT (MBZ)                )
                B'0';          E (KEYBOARD ECHO) BIT      )   I_DVS
                B'1';          L (LF) BIT                 )
                B'0';          C (NO CR) BIT              )
                B'000'         MODE                       )

        DC      0              RESIDUAL RANGE
        DC      0              STATUS WORD
*
*     END OF THE IORB.
*
OP_MSG  TEXT    'A MESSAGE TO THE OPERATOR.'
OP_MLN  EQU     2*($-OP_MSG)
```

OPERATOR RESPONSE MESSAGE

OPERATOR RESPONSE MESSAGE ($OPRSP)

Function Code: 09/01

Equivalent Command: None

Display a message on the operator terminal and place the operator's response to that message in a buffer specified by the input request block.

FORMAT:

```
[label]  $OPRSP  [location of IORB list address]
```

ARGUMENT:

location of IORB list address

Any address form valid for an address register; provides the address of a list specifying the input/output request blocks to be used. The format of the IORB list is as follows:

entry 1 - Address of IORB describing output message (to operator terminal)

entry 2 - Address of IORB describing input message (for operator response).

DESCRIPTION:

This macro call enables the issuing task to send a message to the system operator and to receive the operator's response to that message.

Two IORBs are needed: an IORB describing the output message and an IORB describing the input buffer for the response. Both IORBs are generated through an Input/Output Request Block macro call or are coded by the user.

The output message IORB describes the location and size of the output message.

The input IORB describes the location of the input buffer for the response, the size of the buffer, and whether control is to be returned to the issuing task immediately or after the response has been received (by setting the W-bit of the input IORB).

NOTES

1. The system places in $B4 the address of the IORB list supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0801 - IORB in use (T-bit on)

   0802 - Invalid logical resource number (LRN), or console message suppression in effect

   0803 - Invalid wait, or the R, S, D bit in the IORB is not zero

   0817 - Memory access violation

   The following could occur if the IORB describing the input buffer specified that the issuing task was to wait for the response.

   0104 - Invalid argument

   0105 - Device not ready

   0106 - Device timeout

   0107 - Hardware error (check IORB status word)

   0108 - Device disabled

   0109 - File mark encountered

   010A - Controller unavailable

   010B - Device unavailable

   010C - Inconsistent request

   010D - EOT on magnetic tape detected

   $B4 - Address of input IORB.

Example:

In this example, the Operator Response Message macro call causes the message labeled OP_QRY to be written on the operator terminal. A reply from the operator terminal is then read into the buffer labeled OP_ANS. The issuing task remains blocked until the above actions have been completed.

```
          $OPRSP  !IORB_L
              .
              .
              .
*
*     DEFINE THE IORB LIST.
*
IORB_L  DC      <OUT_RB,<IN_RB
*
*     DEFINE THE IORBS.
*       OUTPUT IORB:
*
OUT_RB  RESV    $AF, 0
        TEXT    Z'00', B'00000001'
        TEXT    Z'00', B'0000', Z'1'
        DC      <OP_QRY
        DC      OP_QLN
        TEXT    B'0000000000010000'
        DC      0, 0
*
*       INPUT IORB:
*
IN_RB   RESV    $AF, 0
        TEXT    Z'00', B'00000001'
        TEXT    Z'00', B'0000', Z'2'
        DC      <OP_ANS
        DC      OP_ALN
        TEXT    B'0000000000110000'
        DC      0, 0
*
*     END OF IORBS.
*
OP_QRY  TEXT    'A QUERY TO THE OPERATOR?'
OP_QLN  EQU     2*($-OP_QRY)
OP_ANS  RESV    40, 0
OP_ALN  EQU     2*($-OP_ANS)
```

OVERLAY AREA, RELEASE ($OVRLS)

Function Code: 07/06

Equivalent Command: None

Exit from the calling floatable overlay, decrement the count of users maintained for this overlay, and transfer control to the supplied return point. (The overlay must have been requested through an Overlay Area Reserve, and Execute Overlay macro call.)

FORMAT:

[label] $OVRLS [location of return point address]

ARGUMENT:

location of return point address

Any address form valid for an address register; provides the address of the return point to which control is to be transferred.

DESCRIPTION:

This macro call causes an exit from the calling overlay and a return to a specified point. The identity of the overlay area table (OAT) controlling the floatable overlay is extracted from the task control block (TCB) of the issuing task. The identity of the OAT is cleared from the TCB and the count of the number of users of this overlay is decremented in the defining OAT. When the count drops to zero (i.e., the task is the last to use the reserved area), the overlay area is marked as available (i.e., released) and can be reused by an Overlay Area Reserve, and Execute function. Control is transferred to the return point supplied by argument 1.

NOTES

1. The system places in $B5 the return point address, supplied by argument 1. If this argument is omitted, the system assumes that $B5 contains the correct return point address.

2. No return is made to the caller; control is returned to the address supplied in $B5. All registers except $R1 are preserved as they existed when the function was executed.

Example:

In this example, the calling overlay uses the Overlay Area, Release macro call to release its overlay area and return to the caller at the return point named OV2_RA. The calling overlay is assumed to be the overlay (OVLY2) that was loaded and executed as shown in the example for the Overlay Area Reserve, and Execute Overlay macro call.

```
XLOC     OV2_RA
$OVRLS   !<OV2_RA
```

OVERLAY AREA RESERVE, AND EXECUTE OVERLAY ($OVRSV)

Function Code: 07/05

Equivalent Command: None

Reserve the overlay area defined by the specified overlay area table (OAT), increment the user count for that overlay area, load the specified floatable overlay, and transfer control to the overlay at the specified or default entry point. (The overlay area must have been previously created by a Create Overlay Area Table ($CROAT) macro call.)

FORMAT:

```
[label]  $OVSRV  [location of overlay id],
                 [location of entry point offset],
                 [location of OAT address],
                 [location of bound unit index id]
```

location of overlay id

Any address form valid for a data register; provides the overlay id of the floatable overlay to be loaded and executed. (The overlay id is a binary value generated by the Linker.)

location of entry point offset

Any address form valid for a data register; provides the offset (from the overlay load base) of the overlay entry point to which control is to be transferred. If this argument is omitted, control is transferred to the start address declared to the language processor or the Linker.

location of OAT address

Any address form valid for an address register; provides the address of the OAT that defines this overlay area. This address was returned by the system when the OAT was created through the Create Overlay Area Table ($CROAT) macro call. If the issuing task is a multi-bound unit task (see below), a null address specifies an OAT created by the primary bound unit.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose overlay is to be loaded; required only if the issuing task is a multi-bound unit task (i.e.,has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). These two calls return in $R6 the index id of the attached bound unit. The index id of the intial bound unit is 0.

DESCRIPTION:

This macro call causes the system to perform the following:

1. Determine if the issuing task already has an area reserved. If so, an illegal overlay nesting error code (160B) is returned.

2. If the issuing task has no area reserved, determine whether the specified overlay of the bound unit being executed by the issuing task is currently resident in any entry of the overlay area defined by the OAT pointed to by argument 3.

3. If the overlay is resident, increment the area's user count and transfer control to the overlay at the specified (or default) entry point.

4. If the overlay is not resident, attempt to reserve an overlay area within the specified OAT. If the overlay area is successfully reserved, increment the user count for the area, load the specified overlay, and transfer control to its specified (or default) entry point.

5. When control is transferred to the overlay, record the identity of the defining OAT in the task control block of the issuing task.

Argument 4 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id number of the initial bound unit is 0. If not applicable, this argument is bypassed.

Generally, an OAT manages only the overlays belonging to the bound unit that created the OAT (by means of the $CROAT macro call). In the case of a multi-bound unit task, however, the OAT created by the primary bound unit can manage the overlays of attached bound units. If the value specified for argument 4 is from 1 to 7, and argument 3 specifies a null address, then the overlay to be loaded belongs to an attached bound unit but will be managed by an OAT created by the primary bound unit. If the user wishes an overlay belonging to an attached bound unit to be managed by an OAT created by the same atached bound unit, then argument 3 should specify the address of that OAT. (The address is returned in $B4 by the $CROAT call that created the OAT.)

NOTES

1. The overlay id supplied by argument 1 is placed in $R2; if this argument is omitted, $R2 is assumed to contain the overlay id.

2. The relative displacement of the entry point from the overlay base, supplied by argument 2, is placed in $R6; if this argument is omitted, a value of -1 is placed in $R6 to indicate that the default entry point established through the language processor or the Linker is to be used.

3. The address of the overlay area table (OAT), supplied by argument 3, is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the OAT to be used. When $OVRSV is issued by a multi-bound unit task, a null address signifies an OAT created by the primary bound unit.

4. The bound unit index id supplied by argument 4 is placed in $R7; if this argument is omitted, $R7 is assumed to contain the bound unit index id.

5. On return, $R2, $B4, and $R1 contain the following:

   $R2 - Overlay id (as supplied)

   $B4 - OAT address (as supplied)

   $R1 - Return status; one of the following:

      01xx - Media error

0602 - Memory unavailable

1601 - Invalid overlay id

1604 - Invalid start address specification

1607 - Unrecoverable media error

1608 - Symbol resolution error

160A - Insufficient memory

160B - Invalid overlay nesting

160C - Invalid overlay size for area managment

1610 - Named OAT cannot be found

1612 - Overlay not a user segment

1614 - Access violation:

- Root of sharable bound unit
- No access rights

1617 - Zero-length overlay.

Example:

In this example, the Create Overlay Area Table ($CROAT) macro call is used to create an overlay area table of three 512-word entries. (It is assumed that no existing OAT controls 512-word entries.) The address of the controlling OAT is stored in the field OAT_A. Later, the Overlay Area Reserve, and Execute Overlay macro call is used to cause the floatable overlay named OVLY2 to be loaded into one of the areas controlled by the OAT (if it is not already available in one of the OAT areas) and then to be executed at its default entry point.

```
            XVAL      OVLY2
  *
  *   CREATE AN OAT IF ONE DOES NOT ALREADY EXIST
  *
            $CROAT    OAT_A, =512, =3
  *
  *   CHECK FOR ERRORS
  *
            BNEZ      $R1, ERROR1
            .
            .
            .
```

```
*
*   LOAD OVLY2 (IF NECESSARY) AND EXECUTE IT
*
          $OVRSV       =OVLY2,, OAT_A
*
*   CHECK FOR ERRORS
*
          BNEZ         $R1, ERROR2
          .
          .
          .
*
*   DEFINE NORMAL RETURN ADDRESS FOR OVERLAY
*
          XDEF         (OV2_RA, $)
          .
          .
          .
OAT_A     DC           <$
```

OVERLAY, EXECUTE ($OVEXC)

Function Code: 07/00

Equivalent Command: None

Load the specified overlay of the bound unit being executed by the issuing task. Transfer control to the overlay at the specified entry point or at the start address declared to the language processor or Linker.

FORMAT:

```
[label]  $OVEXC  [location of overlay id],
                 [location of entry point offset],
                 [location of overlay base address],
                 [location of bound unit index id]
```

ARGUMENTS:

location of overlay id

Any address form valid for a data register; provides the overlay id of the overlay to be executed. (The overlay id is a binary value generated by the Linker.)

location of entry point offset

Any address form valid for a data register; provides the offset (from the overlay load base) of the overlay entry point to which control is to be transferred. If this argument is omitted, control is transferred to the start address declared to the language processor or the Linker.

location of overlay base address

Any address form valid for an address register; provides the base address of the overlay to be loaded and executed; used only for floatable overlays. (Fixed overlays are loaded by $OVEXC at an address established at link time.) A null address specifies that the floatable overlay is to be loaded into a memory block dynamically allocated by the loader.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose overlay is to be loaded and executed; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

This macro call causes the loading and execution of the overlay specified by argument 1 at a base address returned in $B4.

If the specified overlay is fixed, it is loaded at the base address established at link time.

If the specified overlay is floatable, the user must use argument 2 to specify either (1) the base address at which the overlay is to be loaded, or (2) a null address, which directs the loader to allocate memory for the overlay. In the first case, the address specified is that of the first word in a memory block or segment previously created by means of the Get Memory or Create Segment macro calls, respectively. (Both calls return the start address of the memory obtained.) The user would have created a segment only if running in a swap pool. In the second case, the loader allocates a memory block from the user's memory pool or (in an Improved Memory Manager environment) from the user's group work segment (GWS).

Argument 3 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id of the initial bound unit is 0. If not applicable, this argument is bypassed.

If the overlay to be loaded resides in a segment, that segment must have proper access rights.

NOTES

1. The system places in $R2 the overlay id supplied by argument 1. If this argument is omitted, the system assumes that $R2 contains the overlay id.

2. The system places in $R6 the relative displacement of the entry point from the overlay load base, supplied by argument 2. If this argument is omitted, the system places a value of -1 in $R6 to designate that the default entry point is to be used.

3. The system places in $B4 the overlay base address supplied by argument 3. If this argument is omitted, the system assumes that $B4 contains the address.

4 The location of the bound unit index id, supplied by argument 3, is placed in $R7; if this argument is omitted, $R7 is assumed to contain the bound unit index id.

5. On overlay entry, $R1, $R2, $R6, and $B4 contain the following:

   $R1 - 000 (No error)
   $R2 - Overlay id
   $R6 - Entry point offset
   $B4 - Overlay load address.

6. If an error is made in the calling sequence, return is to the caller. $R1 contains one of the following status codes:

   01xx - Media error

   0602 - Memory unavailable

   0817 - Memory access violation

   0E02 - No memory available for nonswappable task

   1601 - Invalid overlay id

   1604 - Invalid start address (offset greater than or equal to overlay size)

   160A - Insufficient memory

   1611 - Zero-length overlay

   1614 - Access violation:

   - Root of sharable bound unit
   - No access.

Example:

In this example, the Overlay, Execute macro call causes the overlay named DPOSIT (of the bound unit being executed) to be loaded and started at the entry point whose offset is named ENTRY2. The example assumes that ENTRY2 was defined as an external value when the bound unit was linked (or, possibly, when its source unit was assembled or compiled).

```
    XVAL     DPOSIT,   ENTRY2
    $OVEXC  =DPOSIT,  =ENTRY2
```

OVERLAY, LOAD ($OVLD)

Function Code: 07/01

Equivalent Command: None

Load the specified overlay of the bound unit being executed by the issuing task. Return control to the issuing task.

FORMAT:

```
[label]  $OVLD  [location of overlay id],
                [location of overlay base address],
                [location of bound unit index id]
```

ARGUMENTS:

location of overlay id

Any address form valid for a data register; provides the overlay id of the overlay to be loaded. (The overlay id is a binary value generated by the Linker.)

location of overlay base address

Any address form valid for an address register; provides the base address of the overlay to be loaded; used only for floatable overlays. (Fixed overlays are loaded by $OVLD at an address established at link time.) A null address specifies that the floatable overlay is to be loaded into a memory block dynamically allocated by the loader.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose overlay is to be loaded; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

This macro call causes the loading of the overlay specified by argument 1 at a base address returned in $B4.

If the specified overlay is fixed, it is loaded at the base address established at link time.

If the specified overlay is floatable, the user must use argument 2 to specify either (1) the base address at which the overlay is to be loaded, or (2) a null address, which directs the loader to allocate memory for the overlay. In the first case, the address specified is that of the first word in a memory block or segment previously created by means of the Get Memory or Create Segment macro calls, respectively. (Both calls return the start address of the memory obtained.) The user would have created a segment only if running in a swap pool. In the second case, the loader allocates a memory block from the user's memory pool or (in an Improved Memory Manager environment) from the user's group work segment (GWS).

Argument 3 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id number of the initial bound unit is 0. If not applicable, this argument is bypassed.

If the overlay to be loaded resides in a segment, that segment must have the proper access rights.

NOTES

1. The system places in $R2 the location of the overlay id, supplied by argument 1. If argument 1 is omitted, the system assumes $R2 contains the overlay id.

2. The system places in $B4 the location of the overlay base address, supplied by argument 2. If this argument is omitted, the system assumes that $B4 contains the address.

3. The location of the bound unit index id, supplied by argument 3, is placed in $R7; if this argument is omitted, $R7 is assumed to contain the bound unit index id.

4. On return, $R2, $R2, $R6, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0000 - Media error

   0601 - Requested contiguous memory exceeds defined pool size

   1603 - Invalid load address

   1607 - Unrecoverable media error

   160A - Insufficient memory

   0602 - Insufficient system memory

   0817 - Memory access violation

   0E02 - No memory availble for non-swappable task

   1601 - Invalid overlay id

   160A - Insufficient memory

   1611 - Zero-length overlay

   1612 - Overlay not a user segment

   1614 - Access violation:

   - Root of sharable bound unit
   - No access

   $R2 - Overlay id (on a successful return)

   $R6 - Overlay default start address offset (on a successful return)

   $B4 - Overlay base address.

Example:

In this example, the Overlay, Load macro call causes the floatable overlay named DPOSIT (of the bound unit being executed) to be loaded, but not executed. This overlay belongs to an attached bound unit whose bound unit index id is 3. Upon return from the system, $B4 contains the overlay base address. $R6 contains the offset from its base address to its default start address. The overlay base address and the offset to the default start address are saved in OVLY_A and OVLY_E, respectively. Thus, the overlay can be entered later, at is default start address, by an instruction sequence such as that shown in the middle of the example. When the overlay is no longer needed it is unloaded by the Overlay, Unload macro call.

```
*
*    LOAD THE DPOSIT OVERLAY
*

         XVAL     DPOSIT
         $OVLD    =DPOSIT,,=3
*
         BNEZ     $41, BAD_ID        CHECK FOR LOAD ERRORS
         .
         .
         .
*
*    SAVE THE BASE ADDRESS AND ENTRY POINT OFFSET
*
         STB      $B4, OVLY_A
         STR      $R6, OVLY_E
         .
         .
         .
*
*    JUMP TO DPOSIT'S DEFAULT ENTRY POINT
*
         LDB      $B4, OVLY_A
         LDR      $R1, OVLY_E
         JMP      $B4.$R1
         .
         .
         .
*
*    UNLOAD THE OVERLAY
*
         $OVUN    =DPOSIT, OVLY_A,,=3
         .
         .
         .
OVLY_A   DC       <$
OVLY_E   DC       00
```

OVERLAY RELEASE, WAIT, AND RECALL

OVERLAY RELEASE, WAIT, AND RECALL ($OVRCL)

Function Code: 07/07

Equivalent Command: None

Exit from the calling overlay. When completion status has been posted to the specified request block, perform an Overlay Area Reserve, and Execute Overlay function for the specified overlay. The calling overlay must have been loaded through the Overlay Area Reserve, and Execute Overlay macro call.

FORMAT:

```
[label]  $OVRCL  [location of overlay id],
                 [location of entry point offset],
                 [location of request block address],
                 [location of bound unit index id]
```

ARGUMENTS:

location of overlay id

Any address form valid for a data register; provides the overlay id of the overlay to be called when the specified request block has been posted as complete. (The overlay id is a binary value generated by the Linker.) If this argument is omitted, the overlay that issued this macro call is recalled.

location of entry point offset

Any address form valid for a data register; provides the offset (from the overlay load base) of the overlay entry point to which control is to be transferred. If this argument is omitted, control is transferred to the start address declared to the language processor or the Linker.

location of request block address

Any address form valid for an address register; provides the address of the request block whose completion status is to be awaited.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose overlay is to be called; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

This macro call enables the issuing task to exit from the currently executing overlay and then to load the same or another overlay when the specified request block is posted as complete.

After the issuing task exits from the executing overlay, the call releases the area occupied by the overlay, if no other task requests use of the overlay. The address of the OAT controlling the overlay is extracted from a field in the task's control block (TCB), and is saved.

The issuing task waits on the specified request block. When the request block is posted as completed, an Overlay Area, Reserve and Execute Overlay function attempts to call/recall the overlay specified by argument 1, using the saved OAT address. If this overlay is already resident in the overlay area, the area's user count is incremented and control is transferred to the overlay at the specified (or default) entry point (argument 2). If this overlay is not resident and space for it does not exist in the overlay area, the issuing task is suspended until space becomes available.

The calling overlay, from which $OVRCL is issued, was loaded into its overlay area by an Overlay Area Reserve, and Execute Overlay ($OVRSV) call. This $OVRSV call, in turn, was preceded by a Create Overlay Area ($CROAT) call that defined and created the overlay area. The overlay specified by argument 1 of $OVRCL is loaded into the same overaly area vacated by the calling overlay. Thus, the overlay specified by argument 1 must be no larger than the entry size specified by the $CROAT call that created the overlay area.

Argument 4 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id number of the initial bound unit is 0. If not applicable, this argument is bypassed.

6. If the calling sequence is in error, return is made to the calling overlay. $R1, $R2, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0802 - Invalid LRN
   0803 - Invalid wait
   1601 - Invalid overlay id
   1617 - OAT has no overlay to release

   $R2 - Overlay id value (as supplied)

   $R6 - Overlay entry point offset (as supplied)

   $B4 - Request block address (as supplied).

Example:

In the following example, the task is to exit from the current overlay and wait for the task request block named TRB1 to be marked as complete before loading overlay OVLY2 and executing it at its default entry point. Note that the overlay to be exited from and the overlay to be loaded and executed are controlled by the OAT whose identity was stored in the task control block of the issuing task by a previously issued Overlay Area Reserve, and Execute Overlay macro call.

```
        XVAL    OVLY2
        .
        .
        .
TRB1    $TRB    17
        .
        .
        .
        $OVRCL  =OVLY2,,  !TRB1
```

OVERLAY STATUS ($OVST)

Function Code: 07/03

Equivalent Command: None

Return the current status of the specified overlay. Among the items of status information returned are:

- Sharable or nonsharable bound unit
- Patched or nonpatched overlay.

FORMAT:

```
[label]  $OVST  [location of overlay id],
                [location of bound unit index id]
```

ARGUMENT:

location of overlay id

Any address form valid for a data register; provides the overlay id of the overlay whose status is desired. (The overlay id is a binary value generated by the Linker.)

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose status is to be returned; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

This macro call causes the system to return an overlay status word in $R2. The contents of this word are described below.

Argument 2 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. Index id numbers 1-7 refer to attached bound units; the index id of the initial bound unit is 0. If not applicable, this argument is bypassed.

When this call is executed, the overlay entry point is returned in $R6, the overlay size in $R7, and the overlay base address in $B4.

NOTES

1. The system places in $R2 the overlay id, supplied by argument 1. If this argument is omitted, $R2 is assumed to contain the required overlay id.

2. The system places in $R7 the bound unit index id, supplied by argumment 2. If this argument is omitted, $R7 is assumed to contain the bound unit index id.

3. On return, $R1, $R2, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   1601 - Invalid overlay id
   1611 - Zero-length overlay.

   $R2 - Overlay status indicator word:

   Bit 0 - Set to one if bound unit sharable; otherwise zero.

   Bit 1 - Set to one if overlay permanently loaded; otherwise zero.

   Bit 2 - Set to one if slow load section present; otherwise zero.

Bit 3 - Set to one if overlay floatable; otherwise zero.

Bit 4 - Set to one if bound unit can be executed in system task group; otherwise zero.

Bit 5 - Set to one if overlay resident in memory; otherwise zero; meaningful only to call/cancel/exit controller.

Bit 6 - Set to one if overlay has been called, but has not exited; otherwise zero; meaningful only to call/cancel/exit controller.

Bits 7 through 9 - Reserved for system use.

Bit 10 - Set to one if overlay contains initialized relocatable pointers; otherwise zero.

Bit 11 - Set to one if overlay contains symbolic references; otherwise zero.

Bit 12 - Set to one if overlay defines symbolic names; otherwise zero.

Bit 13 - Set to one if overlay is patched; otherwise zero.

Bit 14 - Set to zero, indicating Long Address Form (a memory address takes up two words).

Bit 15 - Set to one, indicating Long Address Form (a memory address takes up two words).

$R6 - Overlay default entry point (as specified by the language processor or Linker); given as a word offset from the overlay base address.

$R7 - Overlay size in words (0000 is returned if the size is 64K words).

$B4 - Base address of overlay if permanently loaded or nonfloatable.

Example:

In this example, the Overlay Status macro call is used to determine the status of the overlay named DPOSIT, which is an overlay of the bound unit being executed. If the overlay is floatable, the Get Memory macro call obtains memory for the overlay. The Overlay, Execute macro call ($OVEXC) then loads the overlay and starts it at its default entry point. To simplify the example, the return status is not checked for possible errors.

```
*
*       NAME THE STATUS INDICATORS TO BE USED.
*
FLOAT                EQU    B'0001000000000000'
*
*       DECLARE THE OVERLAY'S NAME.
*
                     XVAL   DPOSIT
*
*       GET THE OVERLAY'S STATUS.
*
                     $OVST  DPOSIT
*
*       GET MEMORY FOR IT IF IT IS FLOATABLE.
*
                     LB     =$R2,FLOAT
                     BBF    >LOAD
                     LDV    $R6,0
                     $GMEM  =$R7,WAIT
*
*       LOAD AND EXECUTE THE OVERLAY.
*
LOAD                 $OVEXC DPOSIT
```

OVERLAY, UNLOAD

OVERLAY, UNLOAD ($OVUN)

Function Code: 07/0C

Equivalent Command: None

Unload the specified overlay, which has previously been loaded by an Overlay, Load ($OVLD) macro call.

FORMAT:

```
[label]  $OVUN  [location of overlay id],
                [location of overlay base address],
                [location of return point address],
                [location of bound unit index id]
```

ARGUMENT:

location of overlay id

Any address form valid for a data register; provides the overlay id of the overlay to be unloaded. (The overlay id is a binary value generated by the Linker.)

location of overlay base address

Any address form valid for an address register; provides the base address at which a floatable overlay specified by argument 1 has been loaded. This is the address returned by Overlay, Load ($OVLD) in $B4.

location of return point address

Any address form valid for an address register; provides the address of the return point to which control will be returned after the macro call is executed. If this argument is omitted, the address of the first word following the generated monitor call sequence is assumed to be the return point address.

location of bound unit index id

Any address form valid for a data register; provides the index id (0-7) of the bound unit whose overlay is to be unloaded; required only if the issuing task has previously executed a Bound Unit, Attach ($BUAT) or Bound Unit, Load ($BULD) macro call. These two calls return in $R6 the index id of the attached bound unit. The index id of the initial bound unit is 0.

DESCRIPTION:

This macro call causes an overlay to be unloaded from an area not controlled by an overlay area table (OAT).

If the overlay to be unloaded is floatable, argument 2 must be used to provide the overlay's location in memory. That location is returned by Overlay, Load ($OVUN) in $B4. If the overlay specified by argument 1 is not floatable, argument 2 is bypassed.

Argument 4 is applicable if the issuing task is a multi-bound unit task (i.e., has previously executed a Bound Unit Attach ($BUAT) or Bound Unit, Load ($BULD) macro call). Even if the issuing task has detached (by means of the Bound Unit, Detach macro call) all tasks previously attached, the issuing task is still considered to be a multi-bound unit task, and a value must be specified for this argument. The index id numbers 1-7 refer to attached bound units; the index id number of the initial bound unit is 0. If not applicable, this argument is bypassed.

NOTES

1. The overlay id supplied by argument 1 is placed in $R2; if this argument is omitted, $R2 is assumed to contain the overlay id.

2. The overlay base address supplied by argument 2 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the base address.

3. The return point address supplied by argument 3 is placed in $B5; if this argument is omitted, the return point address is assumed to be the address of the first word following the generated monitor call sequence.

4. The bound unit index id supplied by argument 4 is placed in $R7; if this argument is omitted, $R7 is assumed to contain the bound unit index id.

5. On return, $R1 contains one of the following status codes:

   0000 - No error

   0602 - Insufficient system memory

   0603 - Block returned is not within its own memory pool

0817 - Memory access violation:

- System segment
- No access rights
- Root of sharable bound unit

0818 - No task group with specified id exists (system software error).

Example:

See the example given for the Overlay, Load ($OVLD) macro call.

PARAMETER BLOCK ($PRBLK)

Function Code: None

Equivalent Command: None

Generate a parameter block that is equivalent to the argument list portion of the task request block.

FORMAT:

```
[label]  $PRBLK  [user argument 1],
                 [user argument 2],
                          .
                          .
                          .
                 [user argument n]
```

ARGUMENTS:

user argument 1 ... user argument n

User argument values; a parameter block is generated containing the specified user argument values in the parameter positions that correspond to the argument positions in the Parameter Block macro call. Pathname arguments must include a trailing blank and must be enclosed in single or double (' or ") quotation marks.

If an argument value of zero is specified before the last argument, an argument pointer of zeros is generated in the corresponding position in the argument list.

DESCRIPTION:

The parameter block generated by $PRBLK is equivalent in format to the argument list portion of the task request block. This format is explained in the Task Request Block ($TRB) description and is illustrated in Appendix C under "Parameter Block Format."

$PRBLK is commonly used for entering requests against previously created task groups or for spawning task groups. The arguments listed by $PRBLK control execution of the created or spawned group's lead task. The format of arguments supplied by the user varies according to whether or not the requested lead task is the Command Processor. See the descriptions of the Request Batch Group, Request Group, and Spawn Group macro calls for further details and for examples.

PERSON IDENTIFICATION ($PERID)

Function Code: 14/01

Equivalent Command: None

Return the person component of the calling task group's user id to a 12-character receiving field.

FORMAT:

```
[label]  $PERID  [location of person id field address]
```

ARGUMENT:

location of person id field address

Any address form valid for an address register; provides the address of a 12-character, aligned, nonvarying field into which the system will place the person component of the user id associated with the issuing task group.

DESCRIPTION:

This macro call returns the person component (i.e., the user's personal id) of the task group's user id to a field in the issuing task. The person id returned is that entered as part of the LOGIN command that established the user as a primary or secondary user of this task group. See the Commands manual for details.

The entire user id is returned by the User Identification macro call.

NOTES

1. The system places in $B4 the address of the receiving person id field, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the field.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0817 - Memory access violation.

3. On return, $B4 contains the address of the receiving field.

Example:

In the following example, a 12-character field is set up in the issuing task, and the Person Identification macro call is issued to place the person id of the task group in that field.

```
ID02    $PERID  !PRIDFL
          .
          .
          .
PRIDFL  RESV    6,0
```

POSTPONE REQUEST ON TAIL ($PPNTL)

Function Code: 01/0E

Equivalent Command: None

Dequeue the currently dispatched task request and requeue it at the end of the queue for those requests previously deferred at the specified priority. Continue task execution at the instruction that immediately follows this call, without dispatching any other request.

FORMAT:

```
[label]  $PPNTL  [location of defer priority]
```

ARGUMENT:

location of defer priority

Any address form valid for a data register; specifies the frame priority number at which the currently dispatched, dequeued request is deferred. Must be a value between +1 and +32,767.

DESCRIPTION:

This macro call dequeues the currently dispatched request and requeues it at the tail of those requests previously deferred under the specified priority. Unlike the related function Defer_Request_On_Tail, this function does not dispatch another request.

NOTES

1. The system places in $R5 the defer priority supplied by the argument. If the argument is omitted, the system assumes that $R5 contains the defer priority.

2. On return, $R1 contains one of the following:

   0000 - Request was successfully dequeued and requeued

   0814 - No currently dispatched request exists.

PROFILE RECORD, ACCOUNTING UPDATE ($PRFAU)

Function Code: 24/42

Equivalent Command: None

Update the specified subsystem profile record with information supplied by the caller.

```
[label]  $PRFAU  [location of address of upadate buffer],
                 [location of record type],
                 [location of user id],
                 [location of buffer size]
```

ARGUMENTS:

location of address of update buffer

Any address form valid for an address register; provides the address of the buffer into which the caller places information used for updating a subsystem profile record. If the user to whom the profile record belongs is not currently logged into the calling subsystem as a primary or secondary user, the caller must supply in the buffer the id of that user. The formats of the user id and of the buffer are described below.

location of record type

Any address form valid for a data register; identifies, by two ASCII characters, the record type of the record to be updated. These two characters must be the same as those used to designate the calling subsystem, as a subsystem can access only its own records.

location of user id

Any address form valid for a data register; provides, along with argument 2, the key for locating the record to be updated. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user whose id this argument specifies is currently logged into the subystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in during the current session as a primary user.

B

The ASCII character 'B' (signifying "buffer") indicates that the user id specified by this argument is entered in bytes 0 through 29 of the buffer whose address is supplied by argument 1. The user id comprises three subfields delimited by periods: name.account.mode. These subfields must be blank-filled and aligned as follows: bytes 0-11, name; byte 12, "."; bytes 13-24, account; byte 25, "."; bytes 26-28, mode; byte 29, ASCII space character.

=$R6

Indicates that the correct value has already been loaded into register $R6.

Default: A null value for this argument defaults to 'P'.

location of buffer size

Any address valid for an address register; provides the size, in bytes, of the buffer pointed to by argument 1. The following diagram illustrates the buffer format.

user id	#fields	FD1	...	FDn	update field1	...	update fieldn

The size and contents of each buffer entry illustrated above are as follows:

user id

This 30-byte portion of the buffer exists only if argument 1 specifies that the user id is provided in the buffer. If this entry exists, it contains the id of the user whose record is to be updated. The format of the user id is described above under argument 3.

#fields

The number of field descriptors in this buffer. This number does not necessarily correspond to the number of fields to be updated, because the caller can instruct the Update function to 'skip' a field defined by a field descriptor. The caller must place in this 2-byte entry a decimal value in the range 1-24.

FD

Field descriptor; defines a field (in the specified profile record) to be updated and and the mode of updating. There must be one 2-byte field descriptor (FD) for each record field to be updated by this execution of $PRFAU. The field descriptor format is shown below. The caller supplies values for all items but the first.

Bit	Content
0	Accumulator bit. Set to 1 by sytem if update by accumulation causes overflow.
1-2	Update Code: 00 = Accumulate; i.e. add value in update field to current value in record field. 01 = Skip; i.e., do not update field defined by this FD. 10 = Overwrite current value in record field with value in update field. 11 = Overwrite only if value in update field exceeds current value in record field.
3-7	Field size. Size, in words, of field defined by this FD.
8-F	Offset, in words, from start of record to start of defined field.

update field

Information for updating the record field defined by the corresponding FD (e.g., information in update field1 is used for updating record field defined by FD1). An update field must be the same size as the record field to be updated, right justified, and zero-filled.

DESCRIPTION:

This macro call can be issued only by bound units declared by the system administrator to have access to profile records. If the caller has not been so declared, or if the specified record has not been declared as containing statistics, the call returns a zero in $R1 without performing an update.

$PRFUA updates a statistic in a profile record by means of overwriting or accumlation. When overwriting, the function replaces the profile record statistic with one supplied by the caller in the buffer. When accumulating, the function adds to the profile record statistic a value supplied by the caller in the buffer. The update code in the field descriptor (FD) allows the caller to specify which type of update operation is to be performed. By the same code, the caller can also limit overwriting to instances where the value supplied in the update field is greater than the current value in the corresponding record field.

$PRFUA is useful for updating a profile record after the record has been created (by means of the Profile Record, Create macro call) and initialized (by means of the Profile Record, Update macro call). Although the Profile Record, Update ($PRFUP) macro call can be used for all updating operations, it is not as efficient in some respects as Profile Record, Accounting Update ($PRFUA). $PRFUP writes in the caller's buffer all fields of a profile record that can be modified by the caller. The caller overwrites the record image in the buffer, which is then written out to the profile file. Thus, $PRFUP may require one more I/O operation than does $PRFUA; also, when using $PRFUP, the caller can increment a profile record statistic only by extracting the statistic from the buffer, adding a value to it, then storing the result back in the buffer.

The skip bit of the file descriptor (FD) allows the caller to use the same buffer for multiple update operations involving different record fields. The caller need not rebuild the buffer when the number and position of target fields change from one execution of $PRFUA to the next.

NOTES

1. The buffer address specified by argument 1 is placed in $B2; if the argument 1 is omitted, $B2 is assumed to contain the buffer address.

2. The record type specified by argument 2 is placed in $R2; if this argument is omitted, $R2 is assumed to contain the record type.

3. The user id specified by argument 3 is placed in $R6. If this argument is 'P'or omitted, $R6 is set to X'FF' indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer.

4. The buffer size specified by argument 4 is placed in $R7. If this argument is omitted, $R7 is assumed to to contain the buffer size.

5. On return, $R1 contains one of the following status codes:

   0000 - No error

   0817 - Memory access error

   0602 - Memory unavailable

   1709 - Invalid combination of arguments

   0811 - User registration is not configured

   020E - Record not found or not accessible to caller

   02xx - File system messages that may occur on reading or re-writing a record.

6. If $R6 is set by the caller to FF (argument 3 = 'P'), it contains on return the caller's LRN. Otherwise, return register remain unchanged.

PROFILE RECORD, CREATE ($PRFCR)

Function Code: 24/20

Equivalent Command: None

Create a skeletal subsystem profile record for a named user who is currently logged into the calling subsystem.

FORMAT:

```
[label]  $PRFCR  [location of record type],
                 [location of user id]
```

ARGUMENTS:

location of record type

Any address form valid for a data register; identifies, by two ASCII characters, the record type of the record to be created. These two characters must be the same as those used to designate the calling subsystem, since a subsystem can create and access only its own records.

location of user id

Any address form valid for a data register; provides the user id to be entered on the created record. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user, whose id this argument specifies, is currently logged into the subsystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in as a primary user during the current session.

=$R6

Indicates that the correct value has already been loaded into $R6.

Default: A null value for this argument defaults to 'P'.

DESCRIPTION:

This function enables the calling subsystem to create a record of its own type for a person currently logged into the system as either a primary or a secondary user. The function fills in three fields of the record: date/time updated (in this case, date/time of creation), user id, and record type. The rest of the 188-byte record is zero-filled. The function then writes this record image from a buffer in system memory to the profile file.

Before this function can be executed, the system administrator must have declared the bound unit issuing this call to have access to the subsystem profile record type. In the absence of this declaration, the function returns a 020E (record not found) error status. If the user is already registered under the calling subsystem, the function returns a 021B (duplicate record) error status.

NOTES

1. The system places in $R2 the record type specified by argument 1. If this argument is omitted, the system assumes that $R2 contains the record type.

2. The system places in $R6 the user id specified by argument 2. If this argument is omitted, $R6 is set to X'FF', indicating that the user id specified by argument 2 is the same user id supplied by the caller when logging in as a primary user during the current session.

3. On return, registers R1, R2, and R6 contain the following information:

   $R1 - Return status; one of the following:

      0000 - No error

      021B - Duplicate record

      020E - Record not found

   $R2 - Record type specified by argument 2

   $R6 - LRN specified by argument 3; or, if the value of argument 3 is ASCII 'P', LRN of the caller's terminal.

PROFILE RECORD, DELETE ($PRFDL)

Function Code: 24/30

Equivalent Command: None

Delete a subsystem profile record, thereby canceling a user's registration under that subsystem.

FORMAT:

```
[label]  $PRFDL  [location of buffer address],
                 [location of record type],
                 [location of user id]
```

ARGUMENTS:

location of buffer address

This argument should be specified only if the value of argument 3 (see below) is ASCII 'B'. It can take any address form valid for an address register; provides the address of a buffer that contains the 30-byte user id appearing on the record to be deleted. The buffer must be entirely within the caller's space.

location of record type

Any address form valid for a data register; identifies, by two ASCII characters, the record type of the record to be deleted. These two characters must be the same as those used to designate the calling subsystem, since a subsystem can access only its own records.

location of user id

Any address form valid for a data register; provides, with argument 2, the key for locating the record to be deleted. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user, whose id this argument specifies, is currently logged into the subsystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in as a primary user during the current session.

B

The ASCII character 'B' (signifying "buffer") indicates that the user id specified by this argument is entered in bytes 6 through 35 of the buffer whose address is supplied by argument 1. The user id comprises three subfields delimited by periods: name.account.mode. These subfields must be blank-filled and aligned as follows: bytes 6-17, name; byte 18, "."; bytes 19-30, account; byte 31, "."; bytes 32-34, mode; byte 35, ASCII space character.

=$R6

Indicates that the correct value has already been loaded into $R6.

Default: A null value for this argument defaults to 'P'.

DESCRIPTION:

This function allows a subsystem to delete a record of its own subsystem type. The person named by the user id field of the record to be deleted need not be currently logged into the calling subsystem when the call is made.

NOTES

1. The system places in $B2 the buffer address specified by argument 1. If the value of argument 3 is ASCII 'B' and argument 1 is omitted, the system assumes that $B2 contains the buffer address.

2. The system places in $R2 the record type specified by argument 2. If this argument is omitted, the system assumes that $R2 contains the record type.

3. The system places in $R6 the user id specified by argument 3. If this argument is 'P' or omitted, $R6 is set to X'FF', indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer pointed to by argument 1.

4. On return, registers R1, R2, R6, and B2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   020E - Record not found or not accessible to caller

   0817 - Memory access error; buffer address not in caller's space

   $R2 - Record type specified by argument 2

   $R6 - LRN specified by argument 3; or, if the value of argument 3 is ASCII 'P', LRN of the caller's terminal

   $B2 - Address of the buffer specified by argument 1.

PROFILE RECORD, GET ($PRFGT)

Function Code: 24/10

Equivalent Command: None

Enable the issuing subsystem to read the profile record of a person registered as a user of that subsystem.

FORMAT:

```
[label]  $PRFGT  [location of buffer address],
                 [location of record type],
                 [location of user id]
```

ARGUMENTS:

location of buffer address

Any address form valid for an address register; provides the address of the buffer into which the function returns the record specified by arguments 2 and 3. The buffer must be entirely within the caller's space and no smaller than 188 bytes (the length of a profile record).

location of record type

Any address form valid for a data register; identifies, by two ASCII characters, the record type of the record to be retrieved. These two characters must be the same as those used to designate the calling subsystem, since a subsystem can access only its own records.

location of user id

Any address form valid for a data register; provides the user id entered on the record to be retrieved. This user id and the record type supplied by argument 2 index a unique subsystem record in the profile record file. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user, whose id this argument specifies, is currently logged into the subsystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in during the current session as a primary user.

B

The ASCII character 'B' (signifying "buffer") indicates that the user id specified by this argument is entered in bytes 6 through 35 of the buffer whose address is supplied by argument 1. The user id comprises three subfields delimited by periods: name.account.mode. These subfields must be blank-filled and aligned as follows: bytes 6-17, name; byte 18, "."; bytes 19-30, account; byte 31, "."; bytes 32-34, mode; byte 35, ASCII space character.

=$R6

Indicates that the correct value has already been loaded into $R6.

Default: A null value for this argument defaults to 'P'.

DESCRIPTION:

This macro call enables a subsystem to read a record of a person who has been registered as a user of that subsystem. The user need not be logged in when the call is issued. The record to be retrieved must be of the same type as the subsystem. If, for example, the calling subsystem is the Network Terminal Manager (NT), the record-type field of the record to be retrieved must show "NT".

The function locates the record to be retrieved by means of the record type/user id key supplied by arguments 2 and 3. The function validates the record type and buffer address supplied and issues appropriate error messages, if any. It returns the specified record in the buffer pointed to by argument 1.

NOTES

1. The system places in $B2 the buffer address specified by argument 1. If this argument is omitted, the system assumes that $B2 contains the buffer address.

2. The system places in $R2 the record type specified by argument 2. If this argument is omitted, the system assumes that $R2 contains the record type.

3. The system places in $R6 the user id specified by argument 3. If this argument is 'P' or omitted, $R6 is set to X'FF', indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer pointed to by argument 1.

4. On return, registers R1, R2, R6, and B2 contain the following information:

   $R1 - Return status; one of the following:

      0000 - No error

      020E - Record not found or not accessible to caller

      0817 - Memory access error; buffer not in caller's space

   $R2 - Record type specified by argument 2

   $R6 - LRN specified by argument 3; or, if the value of argument 3 is 'P', LRN of the caller's terminal

   $B2 - Buffer address specified by argument 1.

PROFILE RECORD, GET USER INFORMATION ($PRFIF)

Function Code: 24/12

Equivalent Command: None

Provide the calling subsystem selected information from a specified registration profile record.

FORMAT:

```
[label]  $PRFRR  [location of buffer address],
                 [location of buffer size],
                 [location of user id]
```

ARGUMENTS:

location of buffer address

Any address form valid for an address register; provides the address of the buffer that may contain the user id and into which information from the specified registration record is read. The buffer must be entirely within the caller's space.

location of buffer size

Any address form valid for a data register; specifies the length, in bytes, of the buffer pointed to by argument 1. The size of the buffer determines the amount of information returned by the function. The function does not return part of the field if there is not space in the buffer for the entire field; that is, the last field cannot be truncated. The function places information into the buffer according to the following format:

Bytes	Contents
0-1	Logical resource number (LRN) of the terminal at which the user is logged in, if argument 3 specifies an LRN or ASCII 'P' (i.e., if the caller is logged in as a secondary or primary user); otherwise, if argument 3 specifies ASCII 'B', this field shows X'FE'
2-13	Symbolic device name of terminal at which user is logged in or was last logged in

NOTES

1. The system places in $B2 the buffer address specified by argument 1. If this argument is omitted, the system assumes that $B2 contains the buffer address.

2. The system places in $R2 the record type specified by argument 2. If this argument is omitted, the system assumes that $R2 contains the record type.

3. The system places in $R6 the user id specified by argument 3. If this argument is 'P' or omitted, $R6 is set to X'FF', indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer pointed to by argument 1.

4. On return, registers R1, R2, R6, and B2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   020E - Record not found or not accessible to caller

   0817 - Memory access error; buffer not in caller's space

   $R2 - Record type specified by argument 2

   $R6 - LRN specified by argument 3; or, if the value of argument 3 is 'P', LRN of the caller's terminal

   $B2 - Buffer address specified by argument 1.

PROFILE RECORD, GET USER INFORMATION ($PRFIF)

Function Code: 24/12

Equivalent Command: None

Provide the calling subsystem selected information from a specified registration profile record.

FORMAT:

```
[label]  $PRFRR  [location of buffer address],
                 [location of buffer size],
                 [location of user id]
```

ARGUMENTS:

location of buffer address

Any address form valid for an address register; provides the address of the buffer that may contain the user id and into which information from the specified registration record is read. The buffer must be entirely within the caller's space.

location of buffer size

Any address form valid for a data register; specifies the length, in bytes, of the buffer pointed to by argument 1. The size of the buffer determines the amount of information returned by the function. The function does not return part of the field if there is not space in the buffer for the entire field; that is, the last field cannot be truncated. The function places information into the buffer according to the following format:

Bytes	Contents
0-1	Logical resource number (LRN) of the terminal at which the user is logged in, if argument 3 specifies an LRN or ASCII 'P' (i.e., if the caller is logged in as a secondary or primary user); otherwise, if argument 3 specifies ASCII 'B', this field shows X'FE'
2-13	Symbolic device name of terminal at which user is logged in or was last logged in

Bytes	Contents
14-25	Person field of user id
26-37	Account field of user id
38-45	Encrypted password if caller is running in ring 0; otherwise, the field is filled with '1' bits
45-75	User id in person.account.mode format with trailing spaces
76-133	Message library pathname last specified

As mentioned above, the size of the buffer determines the amount of information returned. If, for example, the size specified for the buffer is 13 bytes, the function returns only the first two fields shown above.

location of user id

Any address form valid for a data register; identifies the registration record to be read from. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user, whose id this argument specifies, is currently logged into the subsystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in as a primary user during the current session.

B

The ASCII character 'B' (signifying "buffer") indicates that the user id specified by this argument is entered in bytes 6 through 35 of the buffer whose address is supplied by argument 1. The user id comprises three subfields delimited by periods: name.account.mode. These subfields must be blank-filled and aligned as follows: bytes 6-17, name; byte 18, "."; bytes 19-30, account; byte 31, "."; bytes 32-34, mode; byte 35, ASCII space character.

=$R6

Indicates that the correct value has already been loaded into $R6.

Default: A null value for this argument defaults to 'P'.

DESCRIPTION:

This call allows a subsystem to obtain information about a user from a registration profile record whether or not that user is currently logged into the calling subsystem. The function returns the information in a buffer whose address and size are specified by arguments 1 and 2, respectively. In addition, the function returns the user's language key, also obtained from the registration record, in $R6. (A language key is a 2-character code used as a suffix in designating a user's message library.)

NOTES

1. The system places in $B2 the buffer address specified by argument 1. If this argument is omitted, the system assumes that $B2 contains the buffer address.

2. The system places in $R2 the buffer size specified by argument 2. If argument 2 is omitted, $R2 contains the buffer size.

3. The system places in $R6 the user id specified by argument 3. If this argument is 'P' or omitted, $R6 is set to X'FF', indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer pointed to by argument 1.

4. On return, registers R1, R6, and B2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   020E - Record not found or inaccessible to caller

   0817 - Memory access error; buffer not in caller's space

$R6 - Language key of user specified by argument.

$B2 - Address of buffer into which the function places information from the specified registration record.

PROFILE RECORD, UPDATE ($PRFUP)

Function Code: 24/40

Equivalent Command: None

Modify the subsystem-defined portion of a record belonging to the calling subsystem.

FORMAT:

```
[label]  $PRFUP  [location of buffer address],
                 [location of record type],
                 [location of user id]
```

ARGUMENTS:

location of buffer address

Any address form valid for an address register; provides the address of a buffer that contains, in bytes 98-187, that portion of a record defined by the subsystem. Additionally, if the value of argument 3 is ASCII 'B', bytes 6-35 of the buffer contain a user id. The buffer must be entirely within the caller's space and no smaller than 188 bytes (the length of a profile file record).

location of record type

Any address form valid for a data register; identifies, by two ASCII characters, the record type of the record to be modified. These two characters must be the same as those used to designate the calling subsystem, since a subsystem can access only its own records.

location of user id

Any address form valid for a data register; provides, with argument 2, the key for locating the subsystem record to be modified. The value of this argument may be one of the following:

lrn

Logical resource number of the terminal on which the user, whose id this argument specifies, is currently logged into the subsystem as a secondary user. Must be a binary number in the range 0 through 255.

P

The ASCII character 'P' (signifying "primary user") indicates that the user id specified by this argument is the same user id supplied by the caller when logging in as a primary user during the current session.

B

The ASCII character 'B' (signifying "buffer") indicates that the user id specified by this argument is entered in bytes 6 through 35 of the buffer whose address is supplied by argument 1. The user id comprises three subfields delimited by periods: name.account.mode. These subfields must be blank-filled and aligned as follows: bytes 6-17, name; byte 18, "."; bytes 19-30, account; byte 31, "."; bytes 32-34, mode; byte 35, ASCII space character.

=$R6

Indicates that the correct value has already been loaded into $R6.

Default: A null value for this argument defaults to 'P'.

DESCRIPTION:

This call enables the calling subsystem to modify bytes 98-187 of a record whose type matches the subsystem's. The user named by the user id field of the modified record need not be logged into the calling subsystem when the call is made.

If the record type and buffer address specified are valid, the function reads the record to be modified from the profile file to a temporary buffer. The call performs this read operation by means of the Get Profile Record function. The call replaces bytes 98-187 of the record with data supplied in the buffer pointed to by argument 1. After updating the record's date field (bytes 0-5), the call rewrites the modified record into the profile file.

Note that neither this call nor the Create Profile Record function allows a subsystem to set the access level field of a subsystem record. Only the system administrator, using the Edit Profile utility, can rewrite into this field.

NOTES

1. The system places in $B2 the buffer address specified by argument 1. If this argument is omitted, the system assumes that $B2 contains the buffer address.

2. The system places in $R2 the record type specified by argument 2. If this argument is omitted, the system assumes that $R2 contains the record type.

3. The system places in $R6 the user id specified by argument 3. If this argument is 'P' or omitted, $R6 is set to X'FF', indicating the user id of the primary user. If this argument is 'B', $R6 is set to X'FE', indicating that the user id is provided in the buffer pointed to by argument 1.

4. On return, registers R1, R2, and R6 contain the following information:

   $R1 - Return status; one of the following:

   - 0000 - No error
   - 020E - Record not found or not accessible to user
   - 0817 - Memory access error; buffer not in caller's space

   $R2 - Record type was specified by argument 2

   $R6 - LRN specified by argument 3; or, if the value of argument 3 is 'P', LRN of the caller's terminal.

READ BLOCK ($RDBLK)

Function Codes: 12/00 (normal), 12/01 (tape mark), 12/02 beginning of tape), 12/03 (space), 12/04 (end of tape)

Equivalent Command: None

Read (i.e., transfer) a block from a file to a buffer in main memory. The user must supply a buffer and specify both the size of the block and its relative location in the file.

FORMAT:

```
                                     ┌ ┌,NORMAL┐ ┐
                                     │ │,TM    │ │
[label]  $RDBLK  [FIB address]       │ <,BOT   > │
                                     │ │,SPACE │ │
                                     └ └,EOT   ┘ ┘
```

ARGUMENTS:

FIB address

Any address form valid for an address register; provides the location of the file information block (FIB). The following FIB entries are required.

logical file number

program view

Should include buffer alignment and whether the next read operation is synchronous or asynchronous.

user buffer pointer

transfer size

The maximum transfer size is 32,767 bytes.

block size

Must be a multiple of the physical sector size.

block number

$\begin{Bmatrix} \text{NORMAL} \\ \text{NOR} \end{Bmatrix}$

For disk-resident files, this mode argument indicates that the block identified in the block number entry in the FIB is transferred from the file to the buffer area.

For nondisk files, this mode argument indicates that the next block is to be transferred from the file to the buffer.

NORMAL is the default value for this macro call.

TM

(For tape-resident files only.) This mode argument indicates that the tape is to be moved forward or backward the number of tape marks specified in the block number entry in the FIB. Positioning is to a point immediately following the nth tape mark. A positive value indicates forward movement; a negative value indicates backward movement.

BOT

(For tape-resident files only.) This mode argument causes the tape to be positioned to its physical beginning. A tape's physical beginning precedes (in the case of labeled tapes) any labels or (in the case of unlabeled tapes) any data.

$\begin{Bmatrix} \text{SPACE} \\ \text{SPA} \end{Bmatrix}$

(For tape-resident files only.) This mode argument indicates that the tape is to be moved forward or backward the number of blocks specified in the FIB block number entry. Positioning is to a point immediately following the nth block. A positive value in the block number entry indicates forward movement; a negative value indicates backward movement.

EOT

(For tape-resident files only.) This mode argument causes the tape to be positioned to its logical end, which is defined as the occurrence of two tape marks in succession. Positioning is to a point immediately following the second tape mark.

DESCRIPTION:

Before this macro call can be executed, the logical file number (LFN) must be opened (see Open File macro call) with a FIB program view word that allows access through storage management (bit 0 is one and allows read operations (bit 1 is one). In order to read the file sequentially, it is necessary only to issue successive Read Block macro calls in NORMAL mode, which causes the block-number entry to be incremented by 1 after each transfer. If there is not sufficient data in the block being transferred to fill the buffer, the transfer size entry in the FIB is set by the system to the number of bytes read and a return code of 0000 is delivered.

After completion of a TM, BOT, or EOT operation, the block-number entry in the FIB is automatically reset to zero; however, a SPACE operation causes the system to specify the actual relative number of the next block that would be read by a Read Block macro call. If a tape mark is encountered during a SPACE operation, the operation is terminated and a return-status code of 021F is delivered. In addition, if the end-of-reel is reached, a 0105 error code (device not ready) is delivered; however, if the end-of-tape is reached, it is treated like a normal operation and a return code of 0000 is delivered on successful completion.

Only one asynchronous I/O operation per file can be outstanding at any given time.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Storage Management Access) macro call.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. Upon return, $R1 contains one of the following return codes:

   0000 - No error
   01xx - Physical I/O error
   0203 - Invalid function
   0205 - Invalid argument
   0206 - Unknown or invalid LFN
   0207 - LFN not open
   020A - Address out of file

020B - Invalid extent description information
0217 - Access violation
021F - End of file.

Example:

In this example the FIB is defined as follows:

```
BLKFIB  DC      Z'0005'         LFN=5
        DC      Z'E000'         PROGRAM VIEW = ALLOW READ/WRITE
                                SYNCHRONOUS PROCESSING
        DC      <BLKBUF        BUFFER POINTER
        RESV    2-$AF
        DC      256             TRANSFER SIZE = 256
        DC      256             BLOCK SIZE = 256
        DC      Z'00000000'
```

Based on the above FIB, block zero, which is 256 bytes long, is transferred to a buffer, labeled BLKBUF, in main memory.

```
    $RDBLK   !BLKFIB,NORMAL
```

READ EXTERNAL SWITCHES ($RDSW)

Function Code: 0B/00

Equivalent Command: None

Return the current value of the specified switches in the task group's external switch word; return the inclusive logical OR of the current settings.

FORMAT:

```
[label]  $RDSW  [external switch name],
                [external switch name],
                        .
                        .
                        .
                [external switch name]
```

ARGUMENT:

external switch name ... external switch name

A single hexadecimal digit specifying the external switch in the task group's external switch word to be read. A maximum of 16 external switch names (0 through F) can be specified. If no arguments are supplied, $R2 is assumed to contain the switches to be read. If ALL is specified, all switches are read.

DESCRIPTION:

This macro call provides a mask by which the current setting of selected switches in the task group's external switch word can be read.

$R2 is the mask word. Each bit that is one in $R2 causes the corresponding bit in the external switch word to be read.

When the Read External Switches macro call is executed, $R2 contains the current value of the external switch word. Bit 11 (bit-test indicator) of the I-register provides an indication of the setting of the switches, as follows:

- If bit 11 is zero, none of the switches read was on.
- If bit 11 is one, at least one of the switches read was on.

NOTES

1. The bits corresponding to the external switches in the arguments are set on in $R2; if no arguments are supplied, $R2 is assumed to contain the mask to be used. If ALL is specified for any argument, all bits are set on in $R2.

2. On return, $R2 and the I-register contain the following information:

   $R2 - Current value of external switch word

   I-register (Bit 11) - Inclusive OR of switches read:

   0 - No switch read was on
   1 - At least one switch read was on.

Example:

In this example, the Read External Switches macro call is used to read the specified switches in the external switch word of the task group in which the issuing task is executing. The contents of $R2 (the mask word) are to be 2F4A so that switches 2, 4, 5, 6, 7, 9, C, and E will be read, inclusive ORed, and stored in the central processor's bit indicator. To illustrate:

```
Word:  2    F    4    A

Bits:  0123 4567 89AB CDEF
       0010 1111 0100 1010

Switches:  2  4567  9    C E
```

The BBT instruction is used to transfer control to the routine DO_IT if one or more of the switches is turned on.

```
RDSW_A  $RDSW  2,4,5,6,7,9,C,E
        BBT    DO_IT
```

READ RECORD ($RDREC)

Function Code: 11/10 (next), 11/11 (key), 11/19 (duplicate), 11/12 (position equal), 11/13 (position greater than), 11/14 (position greater than or equal), 11/15 (position forward), 11/16 (position backward)

Equivalent Command: None

Retrieve one logical record from a file to your record area or merely position the read pointer to a desired record. Whether to retrieve or position is specified by the second (i.e., mode) argument.

FORMAT:

```
                                    ┌ ⎧,NEXT    ⎫ ┐
                                    │ ⎪,KEY     ⎪ │
                                    │ ⎪,DUP     ⎪ │
                                    │ ⎨,POSEQ   ⎬ │
[label]  $RDREC  [FIB address]      │ ⎪,POSGR   ⎪ │
                                    │ ⎪,POSGREQ ⎪ │
                                    │ ⎪,POSFWD  ⎪ │
                                    └ ⎩,POSBWD  ⎭ ┘
```

ARGUMENTS:

FIB address

Any address form valid for an address register; provides the location of the file information block (FIB).

$\left\{\begin{array}{l}\text{NEXT}\\\text{NXT}\end{array}\right\}$

(For all files.) This mode argument indicates that the record pointed to by the read pointer is to be read next. The read pointer is set to the next logical record in the file after the read is complete. Only active records are read (i.e., deleted records are skipped unless bit 11 in the program view FIB entry is set to one). This is the default for this macro call. You must code the following FIB entries:

logical file number
program view (record area alignment)
user record pointer
input record length.

After the record is transferred to main memory, the system updates the following FIB entries:

output record length
output record address.

This mode is referred to as read next.

KEY

(For disk files accessed by key, only.) This mode argument indicates that the record identified by the key value pointed to by the FIB is to be read. The read pointer is set to the next logical record in the file after the read is complete. Only active records are read unless bit 11 in the program view FIB entry is set to one. You must code the following FIB entries:

logical file number
program view (record and key area alignment)
user record pointer
input record length
input key pointer
input key format
input key length.

If the file type used is simple or relative, the input key pointer points to the start of an input key area in which a key value has been placed. If the file type used is indexed or random, the input key pointer points to the start of a key value placed in the user record area. The offset of that value from the start of the user record area must be the same offset as that specified by the "key component location" field of the record descriptor. The record descriptor is an entry of the Create File ($CRFIL) argument structure.

After the record is transferred to main memory, the system updates the following FIB entries:

output record length
output record address.

This mode is referred to as read with key.

DUP

(For CALC (random) files.) Reads a record whose CALC key is the same as the last record read. The FIB input key pointer field must point to a CALC key value placed in the user record area. The offset of that value from the start of the user record area must be the same offset as that specified by the "key component location" field of the record descriptor. The record descriptor is an entry of the Create File ($CRFIL) argument structure.

$\left\{\begin{matrix}\text{POSEQ} \\ \text{PEQ}\end{matrix}\right\}$

(For disk files accessed by key, only.) This mode argument positions the read pointer to the first logical record in the file whose key is equal to the one specified in the FIB. It is not necessary for the record pointed to be active. The record can be read through the read next mode argument of the Read Record macro call (see above). You must code the following FIB entries:

logical file number
program view
input key pointer
input key format
input key length.

If the file type used is simple or relative, the input key pointer points to the start of an input key area in which a key value has been placed. If the file type used is indexed or random, the input key pointer points to the start of a key value placed in the user record area. The offset of that value from the start of the user record area must be the same offset as that specified by the "key component location" field of the record descriptor. The record descriptor is an entry of the Create File ($CRFIL) argument structure.

This mode is referred to as read position equal.

$\left\{\begin{matrix}\text{POSGR} \\ \text{PGR}\end{matrix}\right\}$

(For disk files accessed by key, only.) This mode argument positions the read pointer to the first logical record in the file whose key is greater than the one specified in the FIB. It is not necessary for the record pointed to to be active. The record can be read through the read next mode argument of the Read Record macro call (see above). The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as read position greater than.

$\left\{ \begin{array}{l} \text{POSGREQ} \\ \text{PGE} \end{array} \right\}$

(For disk files accessed by key, only.) This mode argument positions the read pointer to the first logical record in the file whose key is greater than or equal to the one specified in the FIB. It is not necessary for the record pointed to to be active. The record can be read through the read next mode argument of the Read Record macro call (see above). The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as read position greater than or equal.

$\left\{ \begin{array}{l} \text{POSFWD} \\ \text{PFD} \end{array} \right\}$

(For tape-resident, disk sequential, and relative files, only.) This mode argument moves the read pointer forward the number of record positions indicated by the FIB (but not beyond the end-of-file). The number of record positions is specified in the input key area pointed to by the FIB input key pointer. It is not necessary for the record pointed to to be active. The record can be read through the read next mode argument of the Read Record macro call (see above). The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as read position forward.

$\left\{ \begin{array}{l} \text{POSBWD} \\ \text{PBD} \end{array} \right\}$

(For tape-resident, disk sequential, and relative files, only.) This mode argument is the same as for POSFWD, above, except that the pointer is moved backward the number of record positions specified by the key value in the FIB (but not before the first record). This mode is referred to as read position backward.

DESCRIPTION:

Before this macro call can be executed, the logical file number (LFN) must have been opened (see the Open File macro call) with a program view word that allows access through data management (bit 0 is zero) and allows read operations (bit 1 is one). The read pointer is a logical pointer to the next record to be read; it is maintained separately from the write pointer. There is one read pointer per file, per user. At open-file time, the pointer is set to the first record in the file and is modified by each Read Record operation.

The FIB can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Data Management Access) macro call.

The following illustrate the effects of read actions according to file organizations.

File Organizations	Effects of Read Actions
Sequential	Read next causes sequential read. Read with key causes direct read.* A simple key is used.
Relative	Read next causes a sequential read. Read with key causes a direct read. A relative or simple key can be used.
Indexed	Read next causes a sequential read. The records returned are in ascending sequence according to primary key value. (This is not necessarily in the same time-dependent or physical sequence that the records were loaded into the file.) Read with key causes a direct read. A primary key or simple key can be used.
CALC (random)	Read next causes a sequential read. The records are returned in physical sequence. The file can be read directly with a CALC key or a simple key.
Fixed Relative	Read next causes a sequential read. Read with key causes a direct read. A relative key is used.
Device Files	Read next causes a sequential read, provided the device can be read and was defined as a readable device.
Tape Files	Read next causes a sequential read. The file can also be positioned n records forward or backward.

*A read, with any position mode, positions the read pointer to the desired record, so that a subsequent read next will retrieve that record.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0203 - Invalid function

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0207 - LFN not open

   020A - Address out of file

   020B - Invalid extent description information

   020E - Record not found

   0217 - Access violation

   0219 - No current record pointer

   021A - Record length error

   021E - Key length or location error

   021F - End of file

   022A - Record lock overflow

   022B - Record deadlock occurred

   022F - Unknown or invalid record type

   0236 - Tape BSN or trailer label block count error

   0237 - Invalid record or control interval format.

Example:

This example assumes that the address of the FIB (i.e., MYFIB) was loaded in $B4. In addition, the required entries in the FIB are those defined in "Assumptions for File System Examples" in Appendix A, with this exception: the value of the second word (program view) is Z'4000' (indicating read operation) rather than Z'2000' (indicating write operation). Also, it is assumed that the file was reserved (see Get File), and that the Open File macro call was coded with the LFN and program-view entries as defined in the example for the Open File macro call.

The macro call is then specified as follows:

```
    $RDREC  ,NEXT
```

After the record is read, the system updates the following entries, which the user can interrogate using the FIB offset tags:

```
    F_ORL  (Output record length)
    F_ORA  (Output record address)
```

REBOOT ($RBOOT)

Function Code: 20/06

Equivalent Command: Reboot (REBOOT)

Activate the Software Reboot Facility (SRF).

FORMAT:

```
[label]  $RBOOT  [location of dump condition],
                 [location of halt condition]
```

ARGUMENTS:

location of dump condition

Any address form valid for a data register; specifies whether or not the SRF is to take a dump of main memory before reinitializing the system. The dump condition desired is indicated by one of the following keywords:

DUMP

Take a dump.

NDUMP

Do not take a dump.

=$R6

$R6 contains the value 1 or 0, signifying DUMP or NDUMP, respectively.

Default: Take a dump.

location of halt condition

Any address form valid for an data register; specifies whether or not the SRF halts the system after taking a dump and before reinitializing the system. The halt condition desired is indicated by one of the following keywords:

HALT

Halt

=$R2

$R2 contains the value 1 or 0, indicating that the system is to halt or is not to halt, respectively.

Default: Do not halt.

DESCRIPTION:

$RBOOT explicitly activates the SRF. The SRF is activated dynamically by exhaustion of trap save areas (TSAs) and of indirect request blocks (IRBs), or by the expiration of a Watchdog Timer (WDT) timeout interval.

The PATH argument of the CLM directive REBOOT implicitly instructs the SRF to take a dump before reinitializing the system; omitting the PATH argument implicitly instructs the SRF not to take a dump. Specifying the DUMP keyword in argument 1 of $RBOOT does not override a REBOOT directive whose PATH argument is omitted. A user who omits the PATH argument can later direct the SRF to take a dump only by modifying the REBOOT directive so that it provides a value for the PATH argument.

On the other hand, specifying the keyword NDUMP in argument 1 of $RBOOT does override a REBOOT directive that provides a value for the PATH argument.

Specifying DUMP in argument 1 of $RBOOT will cause a dump to be taken if all of the following conditions exist:

- A REBOOT directive provides a value for its PATH argument
- The dumpfile and reboot volumes reside on the same device
- The device on which the dumpfile resides is available when the $RBOOT call is issued.

Specifying HALT in argument 2 allows the operator to perform some action before the system is reinitialized, such as taking a dump of Multiline Communications Processor (MLCP) memory. After a halt, the operator causes the SRF to reinitialize the system by pressing Ready and Execute on the control panel.

NOTES

1. Specifying DUMP or NDUMP for argument 1 sets $R6 to 1 or 0, respectively. Omitting argument 1 sets $R6 to 1.

2. Specifying HALT for argument 2 sets $R2 to 1. Omitting argument 2 sets $R2 to 0.

3. On return, $R1 contains one of the following status codes:

   083A - Function illegal for unprivileged task group

   086D - Illegal reboot options

   1613 - Error trying to take dump

   1614 - HALT option selected

   1615 - Error trying to reboot.

Example:

In this example, a $RBOOT call (issued without arguments) activates the SRF. The SRF takes a memory dump and reinitializes the system.

```
$REBOOT
```

RECALL FROM HEAD ($RCLHD)

Function Code: 01/0F

Equivalent Command: None

Dequeue any currently dispatched request and post the specified completion status. Recall task and dispatch the request at the head of the queue of those requests previously deferred at the specified priority.

FORMAT:

```
[label]  $RCLHD  [location of recall priority],
                 [location of completion status]
```

ARGUMENTS:

location of recall priority

Any address form valid for a data register; specifies the priority number from which the request is to be recalled. Must be a value between +1 and +32,767, or -1; -1 specifies that the request is to be recalled from the highest priority (lowest number) in the list.

location of completion status

Any address form valid for a data register; provides the status of the dequeued request. The user may select any status code as the value of this argument.

DESCRIPTION:

This function dequeues the currently dispatched request, if any, and posts its completion status. The function then recalls the request that is at the head of the specified priority. Execution of the issuing task continues at the next instruction after this call.

NOTES

1. The system places in $R5 the recall priority supplied by argument 1. If argument 1 is omitted, the system assumes that $R5 contains the recall priority.

2. The system places in $R2 the completion status specified by argument 2. If argument 2 is omitted, the system assumes that $R2 contains the completion status.

3. On return, registers R1, R5, and B4 contain the following information:

   $R1 - Return status code '0000'

   $R5 - Priority of recalled request (if $B4 is not null)

   $B4 - Address of request block of recalled request. A null address value means that there is no dispatched request at the specified priority.

RELEASE SEMAPHORE ($RLSM)

Function Code: 06/03

Equivalent Command: None

Release a resource controlled by the specified semaphore, and activate the first waiting task queued on that semaphore if the value of the semaphore is negative (both actions are known collectively as a V-op).

FORMAT:

```
[label] $RLSM [location of semaphore id]
```

ARGUMENT:

location of semaphore id

Any address form valid for a data register; provides the two ASCII characters that identify the semaphore controlling the resource to be released.

DESCRIPTION:

A task issues a Release Semaphore macro call when it has finished using the resource controlled by the semaphore indicated in the call. The semaphore must have been previously defined by a Define Semaphore macro call.

When the release function is executed, the counter whose initial value was set in the Define Semaphore macro call is incremented.

If tasks are waiting for the resource to become available, the first task queued on this semaphore is awakened.

NOTES

1. The system places in $R6 the semaphore id supplied by argument 1. If this argument is omitted, the system assumes that $R6 contains the correct id.

2. On return, $R1 and $R6 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0502 - Semaphore not defined

   $R6 - Semaphore id (as supplied).

Example:

See the example given for the Define Semaphore macro call.

RELEASE TERMINAL ($RLTML)

Function Code: 17/04

Equivalent Command: None

Issued by a task group to release a secondary user's terminal back to the Listener component after the terminal file has been closed and removed.

FORMAT:

```
[label]  $RLTML  [location of terminal LRN],
                 [location of status code]
```

ARGUMENTS:

location of terminal LRN

Any address form valid for a data register; provides the logical resource number (LRN) of the terminal to be released.

location of status code

Any address form valid for a data register; provides a one or two byte completion status code that is reported to Listener when the terminal is released. If the completion status is non-zero, Listener displays on the terminal the status code, prefixed with the Listener component code ("39"), and associated message library text.

DESCRIPTION:

This macro call is used to return a secondary user's terminal that was previously obtained by the calling task group through a Request Specific Terminal or Request Terminal macro call. Until this call is issued, the terminal is reserved for the task group.

NOTES

1. The system places in $R6 the LRN of the addressed terminal supplied by argument 1. If this argument is omitted, the system assumes that $R6 contains the terminal's LRN.

2. The system places in $R7 the status code supplied by argument 2. If this argument is omitted, the system assumes that $R7 contains the status code.

3. On return, $R1 contains one of the following status codes:

   0000 - Terminal successfully released

   3902 - Invalid LRN

   3921 - Terminal not assigned to task group

   3928 - Unable to release terminal; file not removed.

Example:

In this example, the Release Terminal macro call is used to release a terminal previously reserved through a Request Terminal ($RQTML) call. $RQTML returned the terminal's LRN in word 0 of the area that received the login parameters (see the Request Terminal macro call). Subsequently, the LRN was stored in the field LRN_STR. A status code of 0000 is to be used; it will not be displayed.

```
REL_TA  $RLTML  =LRN_STR, =0
```

REMOVE FILE ($RMFIL)

Function Code: 10/25

Equivalent Command: Remove (REMOVE)

Cancel the file reservation previously established by a Get File macro call. The user identifies the file to be removed by supplying either a logical file number or a pathname. This function is usually done outside program execution.

FORMAT:

```
[label]  $RMFIL  [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown.

logical file number

A 2-byte logical file number (LFN) used to refer to the file; must be a binary number from 0 through 255, or ASCII blanks (2020), which indicate that an LFN is not specified.

pathname pointer

A 4-byte address, which may be any address form valid for an address register; it points to a pathname (which must end with an ASCII space character) that identifies the directory in the file hierarchy in which the file to be removed is found (as well as the name of the file itself). Binary zeros in this entry indicate that a pathname is not specified.

DESCRIPTION:

This macro call removes the file reservation established for the specified file, provided it is not currently open (see the Open File macro call) in the task group in which you are executing. It does not dissociate the LFN from a pathname (see the Dissociate File macro call).

Also, if the file is a temporary file (see the Create File macro call), this macro call has the same effect as the Delete File macro call previously described.

The file to be removed can be specified only by an LFN or a pathname. When only an LFN is specified, the file must have been reserved previously with a Get File or Create File macro call, or with an equivalent command.

A Remove File macro call does not remove a file that has been reserved through the GET command; the REMOVE command must be used.

Since the Remove File macro call removes all information about the file from the system, subsequent Get File macro calls may require that multiple directory levels be searched to locate the file again. Thus, the Remove File macro call should be used carefully and only after all references to the file are complete.

NOTES

1. If the argument is coded, the system loads the address of the parameter structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0208 - LFN or file currently open in same task group

   0209 - Named file or directory not found

   020C - Volume not found

   0210 - LFN conflict

0222 - Pathname cannot be expanded; no working directory

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

0229 - File not known to task group.

In the following examples, the macro call specifies an argument structure built by a previous Get File macro call; this technique, as opposed to building a separate argument structure, results in using fewer bytes of memory while achieving the cancellation. The macro call is coded as shown in the two examples:

Example 1:

```
WRTFIL  DC      5         LFN = 5
        DC      2,0
        $RMFIL  !WRTFIL
```

Example 2:

```
WRTFIL  DC      Z'2020'   NO LFN
        DC      <FILE_A   PATHNAME POINTER
        RESV    2-$AF
FILE_A  DC      '^VOL03>SUB>FILE_A '
        $RMFIL  !WRTFIL
```

RENAME FILE/RENAME DIRECTORY ($RNFIL)

Function Code: 10/40

Equivalent Command: Rename (RN)

Change the name of a disk file or directory to the name specified by the macro call. The user identifies the disk file or directory to be renamed by supplying either a logical file number or a pathname. This function is usually done outside program execution.

FORMAT:

```
[label]  $RNFIL  [argument structure address]
```

ARGUMENT:

argument structure address

Any address form valid for an address register; provides the location of the argument structure defined below, which must contain the following entries in the order shown.

logical file number

A 2-byte logical file number (LFN) used to refer to the file; must be a binary number in the range 0 through 255, or ASCII blanks (X'2020'), which indicate that an LFN is not specified.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end with an ASCII space character) that identifies the file or directory whose name is to be changed. Binary zeros in this entry indicate that a pathname is not specified.

new name

A 1- to 12-byte name, specifying the new name of the file or directory; must be a simple name (i.e., must not contain "^", "<", ">", etc.).

DESCRIPTION:

This call changes the name of the specified file or directory. However, the volume major directory cannot be renamed (any attempt to do so will cause a status code of 0228 to be returned in $R1). To rename the volume major directory, use the Create Volume command (see the Commands manual).

The file can be renamed by specifying (1) an LFN only or (2) a pathname only. If only an LFN is specified, the file must have been reserved (through a Create File or Get File macro call, or equivalent command) with that LFN.

A restorable disk file (i.e., one created/modified with the -RESTORE attribute) and its related files can be renamed only if the system's journal file is open.

NOTES

1. If the argument is coded, the system loads the address of the parameter structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the parameter structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0209 - Named file or directory not found

   020C - Volume not found

   0212 - Attempted creation of existing file or directory

   0213 - Cannot provide requested file concurrency

0222 - Pathname cannot be expanded; no working directory

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

0228 - Invalid file type

022C - Access control list (ACL) violation

0260 - Journal file not open.

Example:

In this example, it is assumed that a file has been created in the directory SUB.INDEX.A by the name of FILEA. Its full pathname is ^VOL03>SUB.INDEX.A>FILEA. In addition, this file is reserved with LFN=2. User executes this code:

```
ANEWAA  $RNFIL  !NEWNM1
        .
        .
        .
NEWNM1  DC      2         LFN = 2
        RESV    2,0       NO PATHNAME POINTER
        DC      'OLDF_1 '
```

The result is that FILEA in the directory SUB.INDEX.A is renamed to OLDF_1.

REPORT MESSAGE

REPORT MESSASGE ($RPMSG)

Function Code: 0F/03

Equivalent Command: Display

Report a message contained in a message library, with optional substitution of parameters, to the user's terminal. Alternatively, return the message to the caller or display it on the user-out file.

FORMAT:

```
[label]  $RPMSG  [location of component code],
                 [location of  message id],
                 [location of indicators word],
                 [location of descriptor list],
                 [location of buffer to receive message]
```

ARGUMENTS:

location of component code

Any address form valid for a data register; provides the 4-digit hexadecimal code (00xx) of the software component that reports the message. The first pair of digits (00) must be zero; the second pair (xx) identifies the reporting component. Each pair of digits occupies one byte.

location of message id

Any address form valid for a data register pair; provides the 5-digit hexadecimal key value used to locate a message in a message library. (Two hexadecimal digits occupy a byte and 2 bytes form a word. A 5-digit value, therefore, occupies 1 1/2 words, requiring two data registers.)

Values for this argument can take one of the following two forms:

1. A 4-byte string, whose format is shown below:

Byte	Meaning
0	Must be zero.
1	Error/help identifier; one of the following hexadecimal values: 0 = error message 1,2 = help message (first element in a chain) 3-7 = any element following the first in a chained message.
2-3	Message number; a 4-digit hexadecimal number (xxyy)in range 0000-FFFF. The first pair of digits (xx) identifies the software component that owns the message; the second pair (yy) identifies the specific error or help number.

2. =$R7

   The 4-byte message id is already in $R6,$R7.

NOTE

The expanded code of $RPMSG includes an LDI instruction that loads the message id. Users who, when specifying argument 2, index an address register should note that the index register will be aligned to count double words. Users who, when specifiying argument 2, use immediate operand addressing, should note that their string constants must be exactly 32 bits.

location of indicators word

Any address form valid for a data register; provides 16 indicator-bits that specalize execution of this call. Bit meanings are shown below. Bit 0 is the most significant bit.

Bit	Meaning
0	0 = Display message. 1 = Return message to caller's buffer.
1	0 = Display message identifier (error/help id and message number) and code of component reporting message. 1 = Suppress message identifier and code of component reporting message.
2	0 = Descriptor list not provided. 1 = Descriptor list provided.
3	0 = Display message to REGION3 of menu screen or (if terminal in ECL mode) to error-out file (normal option). 1 = Display message to user-out file; suppress message chaining.
4	0 = Return error message to buffer if $R1 = non-zero value. 1 = Return only good message to buffer (i.e., return message only if $R1 = 0). This bit is significant only if bit 0 is set to 1 (return message to buffer).
5	0 = Provide slew character when returning message to buffer. 1 = Do not provide slew character when returning message to buffer. This bit is significant only if bit 0 is set to 1 (return message to buffer).
6	0 = Display message to REGION3 of menu screen (normal option). 1 = Display message to REGION2 of menu screen; suppress message chaining.
7-14	Reserved for future use; must be zero.
15	Used internally; must be zero.

NOTE

If value of indicators word bit 2 is zero, the next argument (location of descriptor list) is ignored.

location of descriptor list

Any address form valid for an address register; provides the location of the parameter descriptor list. This argument is applicable only if the message to be displayed or returned contains substitutable parameters and the caller wishes to substitute arguments for those parameters. If a descriptor list is provided and the message to be reported contains no substitutable parameters, the descriptor list is ignored.

The first item of a descriptor list is a word specifying the number of descriptors in the list; the remaining items are descriptors. Each descriptor corresponds to a substitutable parameter in the message. Information provided by the descriptor includes the address of the argument to be substituted for a parameter and the identifying number of that parameter. A parameter's identifying number and its attributes (e.g., character type, maximum length) are specified by a parameter designator embedded in the preformatted message library text. (Parameter designators are described in the System Messages manual.)

Each descriptor is 4 words long; its format is shown below.

Word	Contents
0	Parameter number/byte indicator.

Parameter number. Two hexadecimal digits in the range 1-99 (decimal); specifies the number of the parameter for which an argument is being supplied. The identifying number of each parameter is established by the parameter's designator.

Byte indicator. Two hexadecimal digits, the first of which must be zero. The second digit has these possible values:

0 = Parameter argument being supplied begins in left byte of word pointed to by this descriptor.

1 = Parameter argument being supplied begins in right byte of word pointed to by this descriptor.

Word	Contents
2	Parameter size. A hexadecimal value specifying the size in bytes of the parameter argument being supplied. The value specified here should not exceed the length (field size) specified by the parameter's designator. If the parameter size is greater than the field size, the supplied argument is displayed as a string of asterisks. A value of zero instructs the system to pick up the number of bytes necessary to fill the parameter field, starting at the location pointed to by this descriptor.
3-4	Pointer to parameter argument. A null pointer signifies that the caller does not choose to substitute an argument for this parameter. The corresponding parameter designator and any associated message text enclosed in brackets are not displayed.

The number of descriptors in a list need not agree with the number of substitutable parameters in the message to be reported. Descriptors can be listed in any order (i.e., the descriptor referring to parameter 1 need not be the first in the list and need not precede the descriptor for parameter 2). If this argument is specified, bit 2 of the indicators word (argument 3) must be set on (descriptor list provided).

Example of descriptor list:

In this example, the first descriptor is the descriptor for parameter 1. The parameter argument is 5 units long and begins in the left byte of location 21A76. Note that the list omits a descriptor for parameter 2; the caller can selectively supply arguments for the substitutable parameters in a message.

```
0002     NUMBER OF DESCRIPTORS IN LIST
0100     PARAMETER NUMBER AND BYTE INDEX
0005     LENGTH OF ARGUMENT
0002     2-WORD POINTER TO ARGUMENT
1A76
0300     START OF DESCRIPTOR FOR THIRD PARAMETER
  .
  .
  .
```

location of buffer to receive message

Any address form valid for an address register; provides the location of the buffer to receive the message. This argument is supplied only if the message is to be returned to the calling application rather than displayed at a terminal. If this argument is supplied, bit zero of the indicators word must be set on (return message; do not display it).

The word preceding the buffer must contain the buffer size, in bytes. When $RPMSG returns the message in the buffer, it also returns, in the word preceding the buffer, the length of the returned message.

The recommended buffer size is 241 bytes--the maximum allowable message length plus one slew byte. The maximum message length accommodates all the elements of a fully formatted message: the message id (described under argument 1) and message text (including substituted parameters).

DESCRIPTION:

$RPMSG allows the caller to report a message to a terminal running in menu or ECL mode. To report a message to a terminal running in forms mode, use the Report Message, Display Formatting and Control ($RPDFC) macro call.

If message chaining is enabled, $RPMSG supports chained messages. To the first message in a chain, specified by argument 2, the Message Reporter appends the "more help?" prompt. A positive response causes the Message Reporter to to display the next message in the chain; a negative response returns control to the caller.

To the last message in a chain, the Message Reporter appends the text "end of help". In menu mode, any response (except one that breaks or interrupts program execution) returns control to the caller. In ECL mode, the message reporter returns control immediately after displaying the "end of help" text, without waiting for the user's response.

The actions that return control to the caller, after $RPMSG is issued, can be summarized as follows:

1. Chaining is disabled. Message Reporter returns control after displaying message; no response by the user is necessary for return of control.

2. Chaining is enabled; message is not chained.

a. Terminal is in ECL mode. Message reporter returns control after displaying message, as in 1. above.

b. Terminal in menu mode. Message reporter returns control after user acknowledges message by hitting any key (except one that breaks or interrupts program execution).

3. Chaining is enabled; message is chained.

a. Terminal is in ECL mode. Message reporter returns control unconditionally after displaying "end of help message" or after a negative response to "more help?" prompt.

b. Terminal is in menu mode. Message Reporter returns control after any response to "end of help? message or after a negative response to "more help?" prompt.

A message displayed in REGION3 remains there for reference by the user after control is returned to the caller.

Argument 5 (location of buffer) allows the caller to receive a message and then display it by a subsequent command. If the receiving buffer is shorter than the messsage specified by argument 2 (location of message id), the message is truncated. If the buffer is shorter than the message id (5 bytes) no part of the message is returned to the buffer.

Argument 5 allows the caller to specialize a message before its display. The argument also allows the caller to take the following precaution against faulty execution of the call. Before requesting the return of a message, the caller can place a backup message in the buffer and set on bit 4 of the indicators word (return only good message to buffer). If $RPMSG then fails to locate or read the requested message, it leaves the buffer contents unchanged, only changing the buffer size to zero. The caller can then display the existing backup message in lieu of no message at all.

NOTES

1. The component code, supplied by argument 1, is placed in $R3. If argument 1 is omitted, $R3 is assumed to contain the component code.

2. The message id, supplied by argument 2, is placed in $R6,$R7. If =$R7 is specified for argument 2, $R6,$R7 are assumed to contain the message number. If argument 2 is omitted, the message is assumed to be an error message, $R1 is assumed to contain the message number, $R7 is loaded with the value in $R1, and $R6 is set to zero.

3. The indicators word, supplied by argument 3, is placed in $R4. If argument 4 is omitted, $R4 is assumed to contain the indicators word.

4. The address of the descriptor list, if supplied by argument 4, is placed in $B2.

5. The address of the buffer, supplied by argument 5, is placed in $B3; if argument 5 is omitted and bit 0 of the indicators word is set off (display message), the message is reported by the system.

6. No values are returned when the caller requests $RPMSG to display a message. When the caller requests $RPMSG to return a message, $R6,$R7, $R2, and $R1 contain the following information:

   $R6,$R7 - Link to next message in chain. Zero indicates end of chain.

   $R2 - Byte offset from beginning of buffer to beginning of message text returned to buffer.

   $R1 - Return status; one of the following:

   0 - Good message returned.

   1 - "ML ERROR" message returned to buffer.

   2 - Only the message identifier was returned; no text.

   3 - A truncated message was returned (i.e., buffer was too small).

   4 - No message was returned (i.e., indicator bit 4 set on or buffer smaller than 5 bytes); size set to zero.

Example 1:

In this example, $RPMSG displays a message at the user's terminal. The displayed message will include the code of the reporting component (X'0017') and the category/specific error code. The category/specific error code has already been returned to $R1 by a preceding macro call. Because argument 2 is omitted, the system assumes that $R1 already contains a category/specific error code (referred to above, under "location of message id", as the "message number"). The omission of argument 4 and the zero value of the indicators word both indicate that the message text will be displayed without substitution of parameters. Because the caller wishes the message displayed rather than returned, argument 5 (buffer location) is omitted.

```
          $RPMSG     COMP,,INDIC
COMP      DC         X'0017'
INDIC     DC         0
```

Example 2:

In this example. $RPMSG displays text to the user's terminal. The value of the indicators word indicates that the component code and message number will not appear in the displayed message. The omission of argument 4 and the value of the indicators word both indicate that the message text will be displayed without substitution of parameters. Because the caller wishes the message to be displayed rather than returned, argument 5 (buffer location) is omitted.

```
          $RPMSG     COMP, HELP, INDIC
COMP      DC         X'0017'
HELP      DC         Z'00010201'
INDIC     DC         Z'4000'
```

REPORT MESSAGE, DISPLAY FORMATTING AND CONTROL ($RPDFC)

Function Code: 0F/04

Equivalent Command: Display

Report a message contained in a message library, with optional substitution of parameters, to a terminal running in forms mode. The message is displayed on the terminal's "25th line".

FORMAT:

```
[label]  $RPDFC  [location of component code],
                 [location of  message id],
                 [location of indicators word],
                 [location of descriptor list],
                 [location of VTCRB]
```

ARGUMENTS:

location of component code

Any address form valid for a data register; provides the 4-digit hexadecimal code (00xx) of the software component that reports the message. The first pair of digits (00) must be zero; the second pair (xx) identifies the reporting component. Each pair of digits occupies one byte.

location of message id

Any address form valid for a data register pair; provides the 5-digit hexadecimal key value used to locate a message in a message library. (Two hexadecimal digits occupy a byte and 2 bytes form a word. A 5-digit value, therefore, occupies 1 1/2 words, requiring two data registers.)

Values for this argument can take one of the following two forms:

1. A 4-byte string, whose format is shown below:

Byte	Meaning
0	Must be zero
1	Error/help identifier; one of the following hexadecimal values: 0 = error message 1,2 = help message (first element in a chain) 3-7 = any element following the first in a chained message
2-3	Message number; a 4-digit hexadecimal number (xxyy)in range 0000-FFFF. The first pair of digits (xx) identifies the software component that owns the message; the second pair (yy) identifies the specific error or help number.

2. =$R7

   The 4-byte message id is already in $R6,$R7.

NOTE

The expanded code of $RPMSG includes an LDI instruction that loads the message id. Users who, when specifying argument 2, index an address register should note that the index register will be aligned to count double words. Users who, when specifying argument 2, use immediate operand addressing, should note that their string constants must be exactly 32 bits.

location of indicators word

Any address form valid for a data register; provides 16 indicator-bits that specalize execution of this call. Bit meanings are shown below. Bit 0 is the most significant bit.

Bit	Meaning
0	Must be zero.
1	0 = Display message identifier (error/help id and mesaage number) and code of component reporting message. 1 = Suppress message identifier and code of component reporting messaage.
2	0 = Descriptor list not provided. 1 = Descriptor list provided.
3-14	Reserved for future use; must be zero.
15	Used internally; must be zero.

NOTE

If value of indicators word is zero, the next argument (location of descriptor list) is ignored.

location of descriptor list

Any address form valid for an address register; provides the location of the parameter descriptor list. This argument is applicable only if the message to be displayed contains substitutable parameters and the caller wishes to substitute arguments for those parameters.

The first item of a descriptor list is a word specifying the number of descriptors in the list; the remaining items are descriptors. Each descriptor corresponds to a substitutable parameter in the message. Information provided by the descriptor includes the address of the argument to be substituted for a parameter and the identifying number of that parameter. A parameter's identifying number and its attributes (e.g., character type, length) are specified by a parameter designator embedded in the preformatted message library text.

Each descriptor is 4 words long; its format is explained in the description of the Report Message ($RPMSG) macro call.

location of VTCRB

Any address form valid for an address register; provides the location of the address of the VDAM terminal control request block (VTCRB).

DESCRIPTION:

$RPDFC reports an error or help message to the message/response line ("25th line") of a terminal running in forms mode.

If the message is longer than one line (80 characters) the Message Reporter displays up to 60 characters on line 25, attempting to partition the message on a blank. The text "message cont'd" is appended to the partial message. After the terminal user acknowledges the partial message, the Message Reporter displays the remaining portion(s) of the message. If the message is chained and chaining is enabled, the Message Reporter appends the "more help?" prompt to the last portion of the message. To the last portion of the final message in the chain, the Message Reporter appends the text "end of help".

The actions that return control to the caller after $RPDFC is issued can be summarized as follows:

1. <u>Chaining is disabled and/or message is unchained.</u> The message reporter returns control after the final portion of the message is displayed and is acknowledged by the user.

2. <u>Chaining is enabled; the message is chained.</u> The Message reporter returns control after the user responds to the "more help?" prompt or "end of help" text by hitting a key other than HELP.

NOTES

1. The component code, supplied by argument 1, is placed in $R3. If argument 1 is omitted, $R3 is assumed to contain the component code.

2. The message id, supplied by argument 2, is placed in $R6,$R7. If =$R7 is specified for argument 2, $R6,$R7 are assumed to contain the message number. If argument 2 is omitted, the message is assumed to be an error message, $R1 is assumed to contain the message number, $R7 is loaded with the value in $R1, and $R6 is set to zero.

3. The indicators word, supplied by argument 3, is placed in $R4. If argument 4 is omitted, $R4 is assumed to contain the indicators word.

4. The address of the descriptor list, supplied by argument 4, is placed in \$B2; if argument 4 is omitted, \$B2 is assumed to contain the address of the descriptor list.

5. The address of the VTCRB, supplied by argument 5, is placed in \$B3; if argument 5 is omitted, \$B3 is assumed to contain the VTCRB address.

REQUEST BATCH ($RQBAT)

Function Code: 0E/00

Equivalent Command: Enter Batch Request (EBR)

Add a request to the queue of files to be processed by the Command Processor executing in the batch task group. If batch requests are queued on disk, the request can be deferred to a specified date/time (see argument 5).

FORMAT:

```
[label]  $RQBAT  [location of address of argument list],
                 [location of address of fixed parameter block]
```

ARGUMENTS:

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the Parameter Block macro call, to be used to build the batch request block. The batch request block is built in the system area of memory and is used by the Command Processor to specialize commands read from the command-in file.

The argument list provides the pathname of the command-in file to be read by the Command Processor and, optionally, arguments to be substituted for parameters in that file. Items in the argument list (i.e., arguments supplied with the $PRBLK call) must be the following:

Item	Content
Argument 1	Ignored by system; null.
Argument 2	Pathname of command-in file read by Command Processor; must be supplied.
Argument 3 . . . argument n	Arguments to be substituted for parameters in command-in file; these arguments are optional.

NOTE

All non-null arguments must be enclosed by single or double quotation marks and must terminate with a blank.

location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block, which can be generated by the Parameter Block macro call. This parameter block has the following arguments:

Argument 1

A string specifying the user id to be associated with this batch request (for system use). The user id currently associated with the issuing task group will be used when the call is executed from a user task group.

Argument 2

A pathname string specifying the command-in and the initial user-in files for the batch request. A non-zero value is required.

Argument 3

A pathname string specifying the error-out and initial user-out files for this batch request. If this entry is zero, one of the following assumptions is made:

- If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.AO.
- If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.
- If in-path specifies an input-only device, the pathname for the output files is null.

Argument 4

A pathname string specifying the initial value of the working directory for this batch request.

Argument 5

The external date/time of the deferred request (disk-queued batch requests only).

Argument 6

A pathname string specifying the message library file for this request. If this argument is not specified, the message library pathname of the requestor is used.

DESCRIPTION:

This macro call causes a request to execute the commands contained in the file identified by the second item in the fixed parameter block (argument 2) to be queued against the batch task group. The batch task group has a first-in/first-out queue of command processor files.

If the batch task group is dormant when the Request Batch macro call is issued, execution begins immediately; otherwise, the request is queued.

The Command Processor is executed as the lead task of the batch task group. Since the Command Processor obtains its commands from the file named in the second entry of the fixed parameter block, the file must begin with a command.

Task group requests can be queued on disk, using the Message Facility, if a group request queue was created for the target group prior to the target group's creation. Group requests queued on disk using the Message Facility can be deferred until a specified date/time.

Batch requests cannot be waited upon.

Task group requests have message library definitions associated with them. Each task within the request group uses the supplied message library. If the message library pathname is not supplied, the requestor's message library is used.

NOTES

1. The system places in $B4 the address of the argument list to be used to build the request block, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. The system places in $B5 the address of the fixed parameter block, supplied by argument 2. If this argument is omitted, the system assumes that $B5 contains the correct address.

3. On return, $R1 contains one of the following status codes:

   0000 - No error
   0209 - Invalid pathname.

Example:

In this example, the Request Batch macro call causes a request to execute the command contained in the file ^V1124>UDD>TEST>JONES>ASM_TST to be queued against the batch task group. This file will also be used as the user-in file. Since argument 3 is null, the user-out and error-out files will default to ^V1124>UDD>TEST>JONES>ASM_TST.AO. The user id, initial working directory, and message library will be JONES.TEST.B., ^V1124>UDD>TEST>JONES>MSGLIB, respectively. The arguments -XREF and -PRINT will be passed to the Command Processor to specialize the control file ASM TST (&1 and &2 in the control file will be replaced by -XREF and -PRINT, respectively). The Parameter Block ($PRBLK) macro call used in this example is described earlier in this section.

```
        $RQBAT  !ARGS,  !INFO
           .
           .
           .
INFO    $PRBLK  'JONES.TEST.BΔ',
                '^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                ,'^V1124>UDD>TEST>JONESΔ',
                ,'^V1124>UDD>TEST>JONES>MSGLIBΔ'
ARGS    $PRBLK  ,'^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                '-XREFΔ'
                '-PRINTΔ'
```

REQUEST BLOCK DISPLACEMENTS ($RBD)

Generated Label Prefixes:

R_RRB
R_SEM
R_RS
R_CT1
R_CT2
R_ADR

See Appendix C for the format of the request block.

REQUEST CLOCK ($RQCL)

Function Code: 05/00

Equivalent Command: None

Request the Clock Manager to mark the specified clock request block (CRB) as complete when the interval specified in that CRB has elapsed.

FORMAT:

[label] $RQCL [location of CRB address]

ARGUMENT:

location of CRB address

Any address form valid for an address register; provides the address of the clock rquest block to be posted when its specified time interval has elapsed.

DESCRIPTION:

This macro call connects the specified CRB to the timer queue.

If the clock request block is not cyclic (see the Clock Request Block macro call), when the specified interval elapses, the CRB is dequeued from the timer queue. Another Request Clock macro call must be issued to requeue the CRB. Note that a noncyclic CRB can specify an absolute time value rather than an interval.

If the CRB is cyclic, when the specified interval elapses, the CRB is posted and a new request for the originally specified interval is automatically initiated. The automatic resetting continues until a Cancel Clock Request macro call is issued. A cyclic CRB cannot have a time interval of zero and cannot specify an absolute time value.

NOTES

1. The system places in $B4 the address of the CRB to be connected, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0401 - Invalid time value (zero value for cyclic CRB)

   0402 - Invalid receiving field length

   0403 - Invalid control interval unit specified

   $B4 - Address of CRB.

Example:

See the example given for the Wait on Request List ($WAITL) macro call.

REQUEST GROUP ($RQGRP)

Function Code: 0D/00

Equivalent Command: Enter Group Request (EGR)

Request the execution of the lead task of a specified task group. The request is placed in the first-in/first-out request queue maintained for that task. If group requests are queued on disk, the request can be deferred to a specified date/time (see Argument 5).

FORMAT:

```
[label]  $RQGRP  [location of group id],
                 [location of address of argument list],
                 [location of address of fixed parameter block]
```

ARGUMENTS:

location of group id

Any address form valid for a data register; provides the group id of the task group to be requested. This task group must have been previously defined by a Create Group macro call.

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the Parameter Block macro call.

If the lead task is the Command Processor, the argument list provides the pathname of the command-in file to be read by the Command Processor and, optionally, arguments to be substituted for parameters in that file. Items in the argument list (i.e., arguments supplied with the $PRBLK call) must be the following:

Item	Content
Argument 1	Ignored by system; null.
Argument 2	Pathname of command-in file read by Command Processor; must be supplied.
Argument 3 . . . argument n	Arguments to be substituted for parameters in command-in file; these arguments are optional.

NOTE

All non-null arguments must be enclosed by single or double quotation marks and must terminate with a blank.

If the lead task activated by $RQGRP is not the Command Processor, the argument list is optionally used to specialize execution of the lead task. The order in which the arguments are listed is the order expected by the lead task. Unless the argument is a pathname, it is not necessarily enclosed in quotation marks.

location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block, which can be generated by the Parameter Block macro call. This parameter block has the following arguments:

Argument 1:

A string specifying the user id to be associated with this request (for system use). If this entry is zero, the user id currently associated with the issuing task group is used at the time the call is executed from a user task group.

Argument 2

A pathname string specifying the command-in and initial user-in files for this request for the lead task of the referenced task group. If this entry is zero, no command-in and initial user-in files will be available to the group. However, the group can later obtain a user-in file by means of the New User Input macro call. A nonzero entry is required if the command processor is the lead task.

Argument 3

A pathname string specifying the error-out and initial user-out files for this request of the task group. If this entry is zero, one of the following assumptions is made when the call is executed:

- If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.AO.

- If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.

- If in-path specifies an input-only device, the pathname for the output files is null.

Argument 4

A pathname string specifying the initial value of the working directory for this request of the referenced task group.

Argument 5:

A string specifying the external date/time of the deferred request (disk-queued group requests only).

Argument 6:

A pathname string specifying the message library file for this request. If this argument is not specified, the message library pathname of the requestor is used.

DESCRIPTION:

This macro call initiates the execution of the lead task of a task group previously created by a Create Group macro call. If the task group is dormant at the time the Request Group macro call is issued, execution begins immediately. If the task group has been activated by a previous Request Group function and has not yet terminated, execution of this Request Group macro call begins when the group becomes dormant.

Task group requests can be queued on disk using the Message Facility if a group request queue was created for the target group prior to the target group's creation. Group requests queued on disk using the Message Facility can be deferred until a specified date/time.

Execution begins with the lead task specified in the Create Group macro call. The second argument of the Request Group macro call provides an argument list used to specialize a request block that, in turn, is used to request the lead task. (This request block is built in space taken from the memory pool of the requested group.)

It is not possible to wait on the execution of a Request Group macro call.

Task group requests have message library definitions associated with them. Each task within the requested group uses the supplied message library. If the message library pathname is not supplied, the requestor's message library is used.

NOTES

1. The system places in $R2 the group id supplied by argument 1. If argument 2 is omitted, the system assumes that $R2 contains the group id to be used.

2. The system places in $B4 the address of the argument list supplied by argument 2. If this argument is omitted, the system assumes that $B4 contains the address of the list.

3. The system places in $B5 the address of the fixed parameter block supplied by argument 3. If this argument is omitted, the system assumes that $B5 contains the address of the fixed parameter block to be used.

4. On return, $R1 contains one of the following status codes:

   0000 - No error
   0601 - Invalid memory size or memory pool
   0602 - Memory unavailable
   0806 - Group id not currently defined
   160A - Insufficient memory.

Example 1:

In this example, the Request Group macro call causes a request to execute the commands contained in the file ^V1124>UDD>TEST>JONES>ASM_TST to be queued against the Q2 task group. (It is assumed that task group Q2 has already been created with the Command Processor as its lead task. See the Create Group macro call for information on creating task groups.) The ASM_TST file will also be used as the user-in file. The file ^V1124>UDD>TEST>JONES>L>ASM_TST.AO will be used as both the user-out file and the error-out file. The user id, initial working directory, and message library will be JONES.TEST.M, ^V1124>UDD>TEST>JONES, and ^V1124>UDD>TEST>JONES>MSGLIB, respectively. The arguments -XREF and -PRINT will be passed to the Command Processor (group Q2's lead task) to specialize the control file ASM_TST (&1 and &2, in the control file, will be replaced by -XREF and -PRINT, respectively). Refer to Parameter Block macro described previously.

```
            $RQGRP  ='Q2',!ARGS,!INFO
               .
               .
               .
     INFO   $PRBLK  'JONES.TEST.B ',
                    '^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                    '^V1121>UDD>TEST>JONES>ASM_TEST.AOΔ',
                    '^V1124>UDD>TEST>JONESΔ',
                    ,'^V1124>UDD>TEST>JONES>MSGLIBΔ'
     ARGS   $PRBLK  ,'^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                    '-XREFΔ'
                    '-PRINTΔ'
```

Example 2:

In this example, the Request Group macro call activates task group Q5, whose lead task is the Compare utility. (It is assumed that the task group has already been created with Compare as its lead task.) No command-in or user-in file is initially required. The user id, initial working directory, and message library are SMITH.TEST.B, ^VOLA>TEST>SMITH, and ^VOLA>TEST>SMITH>MESSAGELIB, respectively. The files to be compared are FILEA and >UDD>BOOKS>FILEA. The first 20 miscompared records are to be printed on the user-out file LPT00.

Note that because the lead task of the requested group is not the Command Processor, the format of the argument list differs from that shown in Example 1.

```
        $RQGRP   = 'Q5',!ARGS, !INFO
            .
            .
            .
INFO    $PRBLK   'SMITH.TEST.BΔ',
                 ,'LPT00Δ',
                 'VOLA>TEST>SMITHΔ',
                 ,'^VOLA>TEST>SMITH>MESSAGELIBΔ'
ARGS    $PRBLK   'FILEAΔ',
                 '>UDD>BOOKS>FILEAΔ',
                 -PR 20
```

REQUEST I/O ($RQIO)

Function Code: 02/00

Equivalent Command: None

Request an I/O transfer in which the device involved in the transfer and the parameters defining the transfer are identified in the I/O request block (IORB) referred to in the call.

FORMAT:

[label] $RQIO [location of IORB address]

ARGUMENT:

location of IORB address

Any address form valid for an address register; provides the address of the IORB containing the device designation and all information about the nature of the I/O transfer. The IORB can be hand-coded or constructed through the Input/Output Request Block Offsets or Input/Output Request Block macro calls.

DESCRIPTION:

This macro call requests an I/O transfer using a defining IORB.

You should initially reserve the device named in the IORB. Device reservation can be accomplished by the Get File macro call using device-level access (i.e., the pathname is in the form SPD dev_name [volid]).

The IORB requires a logical resource number (LRN) to refer to the device. The LRN can be obtained by issuing a Get File Information macro call. The LRN returned by the Get FIle Information call will be the LRN assigned to the device at system building time.

NOTES

1. The system places in $B4 the address of the IORB supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the IORB to be used.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0801 - IORB in use (t-bit on)

   0802 - Invalid LRN

   0803 - Invalid wait or the R/S/D bit in the IORB is nonzero.

   If the IORB specifies that the issuing task is to wait for the completion of the request, one of the following codes could be returned:

   0104 - Invalid arguments

   0105 - Device not ready

   0106 - Device timeout

   0107 - Hardware error (check IORB status word)

   0108 - Device disabled

   0109 - File mark encountered

   010A - Controller unavailable

   010B - Device unavailable

   010C - Inconsistent request

   010D - Magnetic tape end-of-tape (EOT) marker (reflective strip) detected

   0817 - Memory access violation

   $B4 - Address of IORB.

Example:

In this example, the Request I/O macro call is used to request an I/O transfer involving a device whose logical resource number is 143. The device has been reserved by a Get File macro call; its LRN has been obtained by a Get File Information macro call. In addition to the LRN, the IORB provides the following information about the I/O transfer:

- The issuing task is to be suspended until the request is complete.
- The address of the buffer to be used in the I/O transfer is BUFAD.
- The buffer begins in the left byte of BUFAD.
- The buffer is 326 bytes long.

```
AF001   $RQIO  !IORB21
          .
          .
          .
IORB21  $IORB  143,WAIT,,BUFAD,L,326
```

REQUEST SEMAPHORE ($RQSM)

Function Code: 06/00

Equivalent Command: None

Request reservation of a resource controlled by the semaphore specified in the indicated semaphore request block (SRB). If it is available, reserve the resource. If the resource is not available, queue the SRB until the resource becomes available.

FORMAT:

```
[label]  $RQSM  [location of SRB address]
```

ARGUMENT:

location of SRB address

Any address form valid for an address register; provides the address of the SRB to be queued if the resource is not available. See the Semaphore Request Block macro call later in this section.

DESCRIPTION:

This macro call is an asynchronous request for a resource controlled by the semaphore identified in the SRB. The semaphore itself must have been defined by a Define Semaphore macro call. The SRB can be generated by a Semaphore Request Block macro call.

When the Request Semaphore macro call is executed, the counter, whose initial value was established by the Define Semaphore macro call, is decremented by 1.

If the resource is available, it is reserved. If the resource is not available, the SRB is queued until the resource becomes available.

If WAIT was specified in argument 2 of the Semaphore Request Block macro call, the issuing task is suspended until the resource becomes available. The resource is then reserved, the SRB is marked as terminated, and control is returned to the issuing task.

If argument 2 of the Semaphore Request Block macro call is not WAIT, control is immediately returned to the issuing task, which can then perform other processing. When the resource becomes available, it is reserved and the SRB is marked as terminated. The issuing task can then use the Test Completion Status, Wait, or Wait on Request List macro calls to check the completion status of the SRB. (Alternatively, the task can use the request-task or post-semaphore termination options.)

NOTES

1. The system places in $B4 the address of the SRB supplied in argument 1. If this argument is omitted, the system assumes that $B4 contains the SRB address.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0502 - Invalid SRB

   $B4 - Address of SRB.

Example:

In this example, the Request Semaphore and Wait ($WAIT) macro calls are used to replace the P-op on semaphore TH used in the example given for the Define Semaphore macro call. This technique allows the requesting task to start the process of reserving a resource before it is actually needed and to continue concurrent processing until the resource is required (at which time the requesting task will wait for the SRB). Processing then continues as in the Define Semaphore example.

```
*
*  START THE PROCESS OF CAPTURING A RESOURCE BY ISSUING
*  A REQUEST SEMAPHORE CALL TO RESERVE A RESOURCE
*
                     $RQSM   !SRB
*
*  NOW CONTINUE NORMAL PROCESSING
*
                        .
                        .
                        .
*
*  ROUTINE TO FINISH GETTING A RESOURCE
*
*  WAIT FOR THE REQUEST SEMAPHORE CALL TO FINISH
*
                     $WAIT   !SRB
*
*  NOW LOCK THE FREE RESOURCE LIST
*
                     $RSVSM  ='LK'
*
*  NOW TAKE A RESOURCE FROM THE FREE RESOURCE LIST
*
                        .
                        .
                        .
*
*  THEN UNLOCK THE FREE RESOURCE LIST
*
                     $RLSM   ='LK'
*
*  NOW THE RESOURCE IS RESERVED
*
                        .
                        .
                        .
SRB                  $SRB    TH, WAIT
```

REQUEST TASK ($RQTSK)

Function Code: 0C/00

Equivalent Command: Enter Task Request (ETR)

Request the execution of a previously created task within the same task group from which this request is issued.

FORMAT:

[label] $RQTSK [location of request block address]

ARGUMENT:

location of request block address

Any address form valid for an address register; provides the address of the task request block that identifies the requested task and specifies whether the issuing task is to wait for the completion of the request.

DESCRIPTION:

This macro call activates a task that was previously defined by a Create Task macro call. The Request Task macro call allows a running task to request the execution of another task. The issuing task must supply a task request block that identifies the requested task and the characteristics of the request.

A task request block is constructed through the Task Request Block macro call. The first argument of the Task Request Block macro call specifies the logical resource number (LRN) of the requested task. The second and third arguments specify whether or not the issuing task is to be suspended until the request is complete. The fourth argument specifies the start address of the task.

Using the LRN supplied in the request block, the Task Manager ascertains the task control block of the requested task. The Task Manager then places the request block in the request queue of the requested task. If the request queue was previously empty, the task is queued to its priority level. If the priority level was empty, it is activated. In addition, if the newly activated priority level is higher than that of the calling task, the Task Manager (operating at the priority level of the calling task) is interrupted and the requested task begins execution.

When the priority level of the calling task again becomes the highest active priority level, the Task Manager checks the task request block to ascertain if the calling task is to wait for the completion of this request (for the requested task) before continuing. If the calling task is to wait (and the requested task has not already signaled its completion relative to the request), the Task Manager associates the identity of the calling (and now waiting) task with the request block for the requested task. The Task Manager then removes the calling task from its priority level and activates the next task in the queue. If the calling task is not to wait for completion of this request for the requested task, the Task Manager returns control to the calling task.

The calling task can explicitly supply the address of the requested task's entry point in the request block it uses. If it does not, the requested task's entry point, derived when the task was created or last terminated, is used.

When a requested task is entered, the Task Manager provides the address of the request block that is being honored. This address is that of the first request block in the request queue for the priority level of the requested task.

If a calling task waits for the completion of its request for a requested task, the Task Manager returns the completion status of the request to the calling task when the latter regains control. (See also the Wait and Wait on Request List macro calls.)

NOTES

1. The system places in $B4 the address supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the task request block for the task. The address of the task request block must be a legal address in the space of both the requesting and requested task. If the requested task is to be able to interrogate its own request block, that task request block must be in a memory segment shared by both tasks.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0801 - Request block in use

   0802 - Invalid LRN used in request block

   0803 - Invalid wait (a task cannot wait on a request for itself)

   If wait specified:

   0000-FFFF - Completion status

   $B4 - Address of task request block.

Example:

In this example, the Request Task macro call is used to request the execution of the task created in the first example for the Create Task macro call (assuming that both macro calls are executed in the same task group). The task request block used is generated so that (1) the issuing task will not be suspended awaiting completion of the requested task, (2) the semaphore named TD will be V-oped at completion of the requested task, and (3) the requested task will be started at entry point ENTRY3 instead of the address specified when the task was created. The task request block is also to contain the argument -PRINT, and by default will contain no additional space for use by the requested task.

```
         $RQTSK    !TRB
            .
            .
            .
TRB      $TRB      10,NWAIT,SM=TD;
                   ENTRY3,,-PRINT
```

REQUEST SPECIFIC TERMINAL ($RQSPT)

Function Code: 17/02

Equivalent Command: None

Request that a secondary login be made to the issuing task group from the specified terminal.

FORMAT:

```
[label]  $RQSPT  [location of requested terminal's LRN],
                 [location of caller's IORB]
```

ARGUMENTS:

location of requested terminal's LRN

Any address form valid for a data register; specifies the LRN of the terminal from which a secondary login will be accepted by the issuing task group. The specified terminal must be monitored by Listener.

location of caller's IORB

Any address form valid for an address register; provides the address of an input/output request block (IORB) generated by the issuing task group. The IORB wait-bit specifies whether control returns to the caller immediately or after the request terminates. The IORB buffer address field points to a buffer that holds a prompt message to be displayed on the requested terminal. When a secondary login satisfies this request, $RQSPT returns login information to the buffer. The IORB range field indicates the size of the buffer, which determines both the length of the prompt message to be displayed and the amount of login information to be returned to the caller. In a user registration environment, $RQSPT retrieves the language key from the registration record of the secondary user and returns the key to the device specific field of the caller's IORB. The format of the information returned to caller's buffer is shown below.

Word	Contents
0	LRN of requested terminal.
1-6	Symbolic device name of requested terminal.
7-12	Person field of user id.
13-18	Account field of user id.
19-22	Unspecified; this field is not blank filled.
23-xx	Login line entered from requested terminal. Maximum length of login line is 32 words.

DESCRIPTION

Like the Request Terminal ($RQTML) macro call, this call is made by a user group to announce that it will accept a secondary login. However, while $RQTML is satisfied by any secondary login to the issuing group, this call is satisfied only by a secondary login to the issuing group from a specified terminal.

If, when $RQSPT is issued, the specified terminal is not monitored by Listener, or is associated with with a group other than the caller, or already has a specific_terminal_ request against it, the present request is posted back with an error status. Otherwise, $RQSPT displays at the specified terminal the prompt supplied by the caller.

If, as is usually the case, a read is pending on the requested terminal, Listener cancels the read, writes the prompt message, then issues another read to obtain a login line.

When writing the prompt message to the terminal, Listener turns off the device specific word bit that causes the first word of the message to be treated as a control character. The message displayed, therefore, is identical to the message supplied in the buffer.

If the next login at the terminal is a secondary login to the calling group, the request is satisfied. A primary login or a secondary login to another group causes the request to be posted back with an error status, and is processed normally.

NOTES

1. The requested terminal's LRN provided by argument 1 is placed in the right byte of $R6. If argument 1 is omitted, the right byte of $R6 is assumed to contain the LRN.

2. The address of the IORB provided by argument 2 is placed in $B4. If this argument is omitted, $B4 is assumed to conain the IORB address.

3. On return, all registers are preserved except $R1, which contains the following information:

   0000 - If no wait was specified: request was issued successfully. If wait was specified: a secondary login has occurred; the buffer pointed to by the IORB contains information about the login.

   393D - Terminal is not monitored by listener.

   393E - Terminal not available for specific request.

REQUEST TERMINAL ($RQTML)

Function Code: 17/03

Equivalent Command: None

Permit the issuing task group to accept a user who is logging into that task group through the Listener component.

FORMAT:

```
[label]   $RQTML   [location of request IORB]
```

ARGUMENT:

location of request IORB

Any address form valid for an address register; provides the address of the input/output request block (IORB) associated with this request.

DESCRIPTION:

This macro call enables the task group of the issuing task to be notified when a terminal user logs in as a secondary user of the task group.

If a secondary user logs in to the calling group after this call has been issued, the terminal from which the user logs in is passed to that task group. The task group can use the Release Terminal macro call to release the terminal. The task group can cancel the request by a Cancel Request macro call.

The buffer address field of the request IORB specifies an area that is to receive some or all of the login parameters in the format specified below. (The actual amount of data transferred is determined by the IORB buffer range field.)

Word(s)	Contents
0	Terminal LRN (in right byte)
1-6	Terminal symbolic peripheral device name (e.g., TTY0)
7-12	Person identification from login line
13-18	Account name from login line, if any
19-22	Not used
23-xx	Entire login line as entered from terminal

The setting of the IORB's W-bit determines whether control is returned immediately or is returned after a login has occurred.

The IORB's I/O bit must be set; the D-bit is reset. The S- and R-bits specify how the task group is to be notified when the request is satisfied. The requesting task group must issue a Get File macro call to the terminal file to reserve the file.

On return, the device specific information field of the request IORB contains the secondary user's language key, or X'2020' if no key was specified.

NOTES

1. The system places in $B4 the address of the terminal IORB supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the current address.

2. On return, $R1 contains one of the following return status codes:

   0000 - If no wait specified, request was issued successfully; if wait specified, successful login

   0817 - Memory access violation

   0824 - Request canceled

   082E - Parameter error (invalid control bits in IORB).

3. On return, $B4 contains the request block address.

4. This macro call modifies item I_CT2 of the IORB.

Example:

In this example, the Request Terminal macro call is used to ensure that the issuing task group is notified when a terminal user logs in as a secondary user of the task group. The information returned to the task group consists only of the terminal LRN, terminal symbolic peripheral device name, person identification, and account name. Note that control is returned immediately to the issuing task group; the group does not wait for a login to occur.

```
CHK_1           $RQTML      !IORB
*
*     DEFINE IORB
*
IORB            RESV        $AF,0           RSU
                TEXT        Z'00';          RETURN STATUS
                            B'0';           T BIT (IN USE)
                            B'1';           W BIT (DON'T WAIT)
                            B'0';           U BIT (USER)
                            Z'0';           MBZ
                            B'1';           MUST BE ONE
                TEXT        Z'03';          LRN
                            B'0';           MBZ
                            B'0';           B BIT (BYTE INDEX)
                            B'00';          MBZ
                            Z'1';           FUNCTION CODE
                DC          <SEC_USR        BUFFER ADDRESS
                DC          IN_LNG          RANGE
                DC          0
                DC          0               RESIDUAL RANGE
                DC          0               STATUS WORD
*
*     END IORB
*
SEC_USR         RESV        18,0
IN_LNG          EQU         2*($-SEC_USR)
*
```

RESERVE SEMAPHORE ($RSVSM)

Function Code: 06/02

Equivalent Command: None

Reserve a resource controlled by the specified semaphore, if the resource is available (i.e., do a P-op or P-test). If the resource is not available, perform one of the following actions, depending on the value of argument 2:

- Return immediately to the issuing task (do a P-test).
- Suspend the issuing task until the resource becomes available. Then, reserve the resource and return to the issuing task (these three actions are known collectively as a P-op).

FORMAT:

$$\text{[label]} \quad \text{\$RSVSM} \quad \text{[location of semaphore id]}, \left[\begin{Bmatrix}\text{DENY}\\\text{WAIT}\end{Bmatrix}\right]$$

ARGUMENTS:

location of semaphore id

Any address form valid for a data register; provides the two ASCII characters that identify the semaphore associated with the resource to be reserved.

DENY

Specifies that if the resource is not available for reservation, an immediate return to the issuing task is to be made (i.e., a P-test is to be done).

WAIT

Specifies that if the resource is not available for reservation, the issuing task is to be suspended until the resource becomes available; then the resource is to be reserved and a return to the issuing program is to be made (i.e., a P-op is to be done).

WAIT is assumed if the argument is omitted.

DESCRIPTION:

This macro call is a synchronous request for a resource controlled by the semaphore identified in argument 1. This semaphore must have been defined by a Define Semaphore macro call.

When a P-op is performed, the counter, whose initial value was established by the Define Semaphore macro call, is decremented by 1.

Since the reserve function does not queue a semaphore request block (see Request Semaphore macro call), the Reserve Semaphore macro call must be reissued when DENY is specified for argument 2.

NOTES

1. The system places in $R6 the sempahore id supplied by argument 1. If this argument is omitted, the system assumes that $R6 contains the id of the semaphore to be tested.

2. If DENY was specified for argument 2, $R2 is set to 0 (P-test to be done); if WAIT is specified for argument 2, or if the argument is omitted, $R2 is set to -1 (P-op to be done).

3. On return, $R1 and $R6 contain the following information:

   $R1 - Return status; one of the following:

      0000 - No error

      0501 - Unsuccessful reservation (only if DENY specified)

      0502 - Semaphore not defined

      0507 - Invalid return condition indicator

   $R6 - Semaphore id (as supplied).

Example:

For an example of the Reserve Semaphore macro call, see the example given for the Define Semaphore ($DFSM) macro call.

RESTART ($RS)

Function Code: 0D/10

Equivalent Command: Restart Initiation (RESTART)

Perform a restart of the most recent valid checkpoint on the currently assigned checkpoint file. If no checkpoint file is currently assigned, perform a restart on the most recent valid checkpoint on the checkpoint file designated by argument 1.

FORMAT:

```
$RS    [location of pathname of checkpoint file],
       [location of group id],
       {WTMBM}
       {NWMBM}
```

ARGUMENTS:

[location of pathname of checkpoint file]

Any address form valid for an address register; provides the pathname of the checkpoint file to be assigned if there is no currently assigned checkpoint file.

[location of group id]

Any address form valid for a data register; provides the group identification of the task group to be restarted. If this argument is omitted, the task group issuing the macro call is restarted. If a group id is specified, it must be the same as that used in the Create Group macro call that initialized that task group.

{WTMBM}
{NWMBM}

WTMBM causes the restart procedure to wait for specific memory blocks required to effect the restart. NWMBM causes the restart to fail if the required memory blocks are not available.

DESCRIPTION:

This macro call aborts the specified task group and then performs a restart to the most valid checkpoint on the currently assigned checkpoint file. If there is no currently assigned checkpoint file, the pathname of the file to be assigned is specified in argument 1.

NOTES

1. The system places in $B4 the address of the pathname supplied by argument 1 in $B4.

2. The system places in $R2 the location of the group id supplied by argument 2. If this argument is omitted, $R2 is set to zero to indicate the issuing task group is to be restarted.

3. $R4 and $R7 contain the following information: $R4 is set to one if the pathname of the checkpoint file was supplied in argument 1, or set to zero if the currently assigned checkpoint files were used.

   $R7 is set to one if argument 3 was set for a wait (WTMBM) for required memory block availability, or is set to zero if waiting for memory block availability was not specified.

Example:

This example illustrates the use of the Restart macro to restart a session. The Validate Checkpoint call ($VLCKP) is used to determine whether some previous session has terminated abnormally. If so, a valid checkpoint still exists on the checkpoint files, and a Restart is performed back to that checkpoint. The Restart waits for availability of any user memory required. If the previous session has terminated normally, the current session proceeds.

```
         .
         .
         .
         $VLCKP    !Path               Validate specific
                                       checkpoint file

         bnez      $r1,>skiprs

         $RS       !Path,, wtmem       Restart to previous
                                       checkpoint

skiprs   equ       0
         .
         .
         .
Path     text      '^myprog>ckptfile '
```

RETURN ($RETRN)

Function Code: None

Equivalent Command: None

Issue a standard return sequence for tasks or called subroutines.

FORMAT:

```
[label]   $RETRN   [location of completion status],
                   [location of return address]
```

ARGUMENTS:

location of completion status

Any address form valid for a data register; provides the user-selected status code to be returned when the subroutine or system service routine finishes processing. Any code can be selected.

location of return address

Any address form valid for an address register; provides the address in the calling task to which the subroutine or system service routine returns when it has finished processing.

DESCRIPTION:

This macro call allows a procedure (which can be called as a subroutine or invoked to service a task request) to have a common return interface to the calling task.

If the procedure was statically linked with its caller, the return address supplied in argument 2 is placed in $B5, and a JMP $B5 instruction is issued. The completion status is placed in $R1.

If the procedure was invoked as a subtask, the procedure's task is terminated and its request block is marked as complete. (See the Terminate Request macro call for further information about task termination.)

Note that $B5 is set to the address of a system-supplied termination routine when either of the following occurs:

- A task is initially activated to service a request
- A return request block macro call is issued.

NOTES

1. The system places in $R2 the status code specified by argument 1. If this argument is omitted, the system assumes that $R2 contains the intended status code.

2. The system places in $B5 the address supplied by argument 2 and executes a JMP $B5 instruction. If this argument is omitted, the system assumes that $B5 contains the return address.

Example:

In this example, the Return macro call is used by a semaphore to return to its caller with a completion status of zero. The example assumes that the procedure was entered at the entry point named BEGIN and that the contents of SAV_B5 are not altered within the procedure other than at its entry point. If the procedure was statically linked with its caller, the macro call causes a JMP $B5 return to the caller, with the completion status in $R1. If the procedure was invoked as a subtask, the macro call causes the procedure's task to be terminated and its request block marked as complete.

```
            EDEF       BEGIN
BEGIN       STB        $B5,SAV_B5
              .
              .
              .
            $RETRN     =0,SAV_B5
              .
              .
              .
SAV_B5      RESV       2
```

RETURN MEMORY/RETURN PARTIAL BLOCK OF MEMORY (SRMEM)

Function Code: 04/04 (Return Memory)
04/05 (Return Partial Block)

Equivalent Command: None

Return all or part of the previously allocated memory block to the memory pool of the task group of the issuing task. If argument 2 is omitted, return all of the memory block; if argument 2 is specified, return the number of words it indicates.

FORMAT:

```
[label]   $RMEM   [location of memory block address],
                  [location of number of words to be returned]
```

ARGUMENTS:

location of memory block address

Any address form valid for an address register; provides the location of the address of the leftmost word (excluding the block header) of the memory block to be returned (either partially or totally).

location of number of words to be returned

Any address form valid for a data register; provides the number of words to be returned (starting at the rightmost part of the block). If this parameter is omitted, the entire memory block is returned.

DESCRIPTION:

The Return Memory and Return Partial Block of Memory macro calls are the means by which a task returns a previously allocated memory block to the task group's memory area. If the entire block is to be returned, argument 2 is omitted. If a part of the block is to be returned, argument 2 specifies the number of words to be returned.

When a partial block of memory is returned, the return is done in 32-word increments of memory; the actual amount of memory returned is the specified amount rounded down to the next lower 32-word increment.

The memory block address referred to by argument 1 is the same address as that returned in $B4 when the task issued a Get Memory or Get Available Memory macro call and was allocated this block.

NOTES

1. The system places in $B4 the memory block address derived from argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the memory block to be returned.

2. The system places in $R6 and $R7 the number of words to be returned (partial return only) derived from argument 2. If argument 2 is =$R7, the system assumes that $R6 and $R7 contain the number of words to be returned. If argument 2 is omitted, the system returns the entire memory block.

3. On return, $R1, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0603 - Block returned is not within its own memory pool

   0604 - Size of memory to be returned is greater than size of memory block (partial return only)

   0818 - No task group with specified group id exists (system software error)

   081B - Rollout of online task group attempted (system software error)

081C - Rollin attempted when batch group not rolled out (system software error)

081E - Unrecoverable media error during rollin

081F - Group not suspended when rollin attempted (system software error)

$R6, $R7 - Partial return only; remaining size of block still allocated

$B4 - Partial return only; address of first (leftmost) word of allocated memory block (excluding header word).

Example:

In this example, the Return Memory/Return Partial Block of Memory macro call is used to return all of the memory obtained in the first example for the Get Memory/Get Available Memory macro calls. The Return Memory/Return Partial Block of Memory macro call is contained in the same procedure as the coding shown in that example.

```
    $RMEM    M_PTR
```

In this example, the Return Memory/Return Partial Block of Memory macro call is used to return 100 words of the memory obtained in the first example for the Get Memory/Get Available Memory Macro calls. Upon return from the system, $B4 contains the address of the first usable word of the memory area, and $R6 and $R7 specify the number of words still remaining in the memory area. The Return Memory/Return Partial Block of Memory macro call is assumed to be in the same procedure as the coding shown in the Get Memory example.

```
    $RMEM    M_PTR,=100
```

RETURN REQUEST BLOCK ADDRESS ($RBADD)

Function Code: 01/07

Equivalent Command: None

Return the address of the request block currently at the head (top) of the issuing task's request queue.

FORMAT:

```
[label]   $RBADD
```

ARGUMENTS:

None

DESCRIPTION:

This macro call returns the address of the first request block in the request queue for the task. The request block is not removed or altered.

The system places the address of the request block in $B4. The system places the address of the argument list (if any) associated with the request block in $B7 (see Appendix C).

Upon return to the issuing task, $B5 contains the address of the system-supplied termination routine.

NOTES

On return, $R1, $B4, $B5, and $B7 contain the following information:

$R1 - Return status; one of the following:

0000 - No error

0801 - Specified request block already in use

$B4 - Address of current request block (if $R1 is 0000)

$B5 - Address of system-supplied termination routine

$B7 - Address of request block argument list (if $R1 is 0000).

Example:

In this example, the Return Request Block Address macro call returns the address of the issuing task's request block in $B4. When the lead task of a user task group is started, a Request Block argument list and a Request Block parameter block are created. The Return Request Block Address call initially causes the address of the argument list to be placed in $B7. When the task is actually executed, the starting address of the parameter block is placed in $B7. The address of a system-supplied termination routine is returned in $B5.

```
    CHEK_L   $RBADD
```

REVERIFY PASSWORD

REVERIFY PASSWORD ($RVFPW)

Function Code: 24/01

Equivalent Command: None

Request password from a terminal (attached to the calling group) that has experienced physical disconnnection. This call ensures that the person who resumes use of the terminal is the same user who logged in before the disconnection.

FORMAT:

```
[label]  $RVFPW  [location of terminal id]
```

ARGUMENT:

location of terminal id

Any address form valid for a data register; provides in the right byte the identity of the terminal from which reverification is requested. The value of this argument may be one of the following:

lrn

Logical resource number (LRN) of a terminal used for primary or secondary login to the calling group. Must be a binary number in the range 0 through 255.

X'FF'

Signifies that the terminal is one used for primary login to the calling group.

DESCRIPTION:

This macro call should be used when a 010B (device unavailable) error is returned on a terminal I/O order and user registration is in effect.

Before issuing this call, the caller must place a (logical) disconnect request against the terminal from which reverification is to be requested.

The call causes to be displayed at the specified terminal a message requesting the user's password if all of the following conditions exist:

1. User registration is active.

2. The specified terminal is monitored by Listener.

3. The user logged in at the specified terminal submitted a valid password.

The password submitted in response to this call is compared with the password of the user logged in at the terminal (see the third condition listed above). If the two passwords do not match, the call returns a X'3938' (password does not verify) error. The terminal remains attached to the caller in a (logically) disconnected state whether or not the passwords match.

If any of the conditions listed above is not present, the call returns a zero (successful) status, without having initiated any dialogue with the user.

NOTES

1. The system places in $R6 the terminal id supplied by the argument. If this argument is omitted, the system assumes that $R6 contains the terminal id.

2. On return, $R1 and $R6 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   010C - Inconsistent request (e.g., request for a disconnect when connect has not been made)

   3938 - Password did not verify

   $R6 - LRN of terminal specified by either form of the argument.

REWRITE RECORD ($RWREC)

Function Code: 11/40 (current), 11/41 (key)

Equivalent Command: None

Change the contents of the specified logical record in the file. This macro call is valid for all file organizations except tape-resident sequential files and device files.

FORMAT:

$$\text{[label]} \quad \text{\$RWREC} \quad \text{[FIB address]} \quad \left[\begin{Bmatrix}\text{,CURRENT} \\ \text{,KEY}\end{Bmatrix}\right]$$

ARGUMENTS:

FIB address

Any address form valid for an address register; provides the location of the file information block (FIB).

CURRENT
CUR

This mode argument indicates that the last record read is to be rewritten by the record defined in the FIB. The previous data management call must have been a read next or read with key; otherwise, a "no current record pointer" error will result. CURRENT is the default value for this macro call. You must code the following FIB entries:

logical file number
user record pointer
output record length.

This mode is referred to as rewrite current record.

KEY

This mode argument indicates that the position in the file associated with the key value specified in the FIB is to be written over by the record identified by the FIB. You must code the following FIB entries:

logical file number

user record pointer

output record length

input key pointer (unless this is an indexed file that contains the key embedded in the logical record)

input key format

This mode is referred to as rewrite with key.

DESCRIPTION:

Before this macro call can be executed, the file must be opened (see the Open File macro call) with a program view word that allows access through data management (bit 0 is zero) and allows rewrite operations (bit 3 is one). The file must be reserved (see the Get File call) with write access concurrency control (type 3, 4, or 5). The Rewrite Record macro call has no effect on the read or write pointer. If the file is an indexed file, the embedded key must not be altered.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Data Management Access) macro call.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4. If this argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0203 - Invalid function

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0207 - LFN not open

020A - Address out of file

020B - Invalid extent description information

020E - Record not found

0217 - Access violation

0219 - No current record pointer

021A - Record length error

021D - Attempt to change the symbolic key value

021E - Key length or location error

022A - Record lock area overflow or not defined

022B - Requested record is locked or causes deadlock

022F - Unknown or invalid record type

0237 - Invalid control interval or record format

023A - Recovery file I/O error

0263 - Journal file I/O error.

Example:

In this example, it is assumed that the file is reserved with write access concurrency control and opened. The FIB identified in the first parameter is defined in "Assumptions for File System Examples" in Appendix A. The macro call is specified as follows:

```
    BACREC    $RWREC    !MYFIB,CURRENT
```

ROLL BACK (RECOVER) FILES ($ROLBK)

Function Code: 0C/14

Equivalent Command: None

Write out onto the media all "before" images recorded in the recovery file by the issuing group; that is, roll back (recover) all files updated since execution of the last Clean Point macro call. Reset the recovery file.

FORMAT:

```
[label]   $ROLBK
```

ARGUMENTS:

None

DESCRIPTION:

The macro call rolls back (or recovers) all files updated since the last execution of the Clean Point macro call, erasing all updates done by the issuing task group.

File recovery is the ability to save and retrieve parts of a file (its "before" images) before that file is updated by a Clean Point macro call. When a record on a recoverable file is to be altered, the system writes the record as it exists before the alteration ("before" image) to the recovery file.

A phase, or interval between Clean Point executions, is the time during which all I/O activity takes place. During this time, data is in an inconsistent or alterable condition. A phase change, when data is declared to be consistent, is accomplished by the Clean Point macro call. File recovery is done on a phase basis; i.e., a phase roll back (recovery) to the last Clean Point execution, with the Roll Back (Recover) Files macro call. The call also resets the recovery file.

File recovery is done on a task group basis. Therefore, when a file is accessed by more than one task group, and one of the groups performs a roll back, the file may be left in an inconsistent state. To prevent this, either the file should be reserved exclusively or, if reserved as sharable, should be reserved with record locking in effect. When a recoverable file is reserved as sharable without record locking in effect, the user should provide some other controls to prevent more than one task group from updating the same control interval.

NOTES

1. If no recovery file exists or the recovery file does not contain any before images, the Roll Back (Recover) Files macro call performs no function.

2. When record contention occurs (see the Get File macro call), resulting in a 022B return code, the user can respond with a Roll Back (Recover) Files macro call to roll back (recover) updates done since the last Clean Point execution, and start over again.

3. On return, $R1 contains one of the following:

   0000 - No error

   01xx - Physical I/O error

   023A - Recovery file I/O error

   2063 - Journal file I/O error.

SEMAPHORE REQUEST BLOCK ($SRB)

Function Code: None

Equivalent Command: None

Generate a 5-word semaphore request block.

FORMAT:

```
[label]   $SRB   [semaphore id],
                 [issuing task suspension option],
                         or
                 [termination action]
```

ARGUMENTS:

semaphore id

A 2-character (ASCII) identifier that must have been defined by the task issuing the semaphore request. If this argument is omitted, the semaphore id is set to an initial value of zero.

issuing task suspension option

One of the following values is specified to indicate whether the requesting task is to be suspended until the resource associated with the semaphore becomes available:

WAIT

Suspend the issuing task until the resource becomes available (see W-bit to zero).

NWAIT

Do not suspend the issuing task (set W-bit to one). If this argument is omitted, the value NWAIT is assumed. If WAIT is specified, argument 3 must be omitted.

termination action

One of the following values is specified to indicate the action to be taken when the resource becomes available to the issuing task:

SM=aa

Do not suspend the issuing task; release (V-op) the semaphore identified by aa (two ASCII characters), when requested task is completed.

RB=label

Do not suspend the issuing task; issue a request for the request block identified by label, when requested task is completed.

Note that the requesting task must be asynchronous, may not wait on the requested task later on, and can only point to a task request block (TRB). The requested task must already have been created (not spawned), be asynchronous, and have a valid LRN. When the requesting task terminates, the TRB pointed to by "label" must be inactive.

If this argument is omitted (or argument 2 is WAIT), the generated Semaphore Request Block (SRB) contains no termination option.

DESCRIPTION:

The SRB is used to request asynchronously the reservation of a resource controlled by the specified semaphore. The SRB contains a semaphore id that identifies the (previously defined) semaphore being requested.

Example:

In this example, the Semaphore Request Block macro call generates a semaphore request block with identifier AA. The W-bit is set to zero to indicate the requesting task is to be suspended until the resource becomes available. No suspension action is given.

```
GTRAA    $SRB    AA,WAIT
```

SEMAPHORE REQUEST BLOCK OFFSETS ($SRBD)

Counterpart: $SRB (see the Semaphore Request Block macro call)

Generated Label Prefixes

	S_RRB/S_SEM
SRB label	offset 0
	S_CT1
	S_CT2
	S_ADR

See Appendix C for the format of the semaphore request block.

SET DIAL

SET DIAL ($SDL)

Function Code: 1B/00

Equivalent Command: Set Autodial Telephone Number (SDL)

Insert the specified telephone number into the first entry in the Auto Call Unit (ACU) telephone number list for the specified line. This telephone number will be used first when the Auto Call Unit Facility attempts to establish a connection on the switched circuit line, which is identified either by channel number or by the name of a device on the line.

FORMAT 1:

```
[label]    $SDL  [location of address of telephone number],
                 [location of channel number],
                 [location of address of device name],
```

ARGUMENTS:

location of address of telephone number

Any address form valid for an address register; provides the address of the telephone number to be inserted in the ACU list. The telephone number must be stored as a character string containing at least one trailing space and no embedded spaces. The telephone number can contain from 5 through 16 ASCII characters chosen from the set 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, *.

location of channel number

Any address form valid for a data register. If the user chooses to identify the line by channel number, this argument provides the four hexadecimal digits that define the 10-bit channel number. The channel number must be stored left-justified and zero-filled. If, alternatively, the user chooses to identify the data line by the name of a device on the line, the value of this argument must be zero.

location of address of device name

Any address form valid for an address register; provides the name assigned to the device at configuration time by means of the optional DEVICE directive (described in MOD 400 System Building and Administration (CZ02-00)). If the device is not configured with the DEVICE directive (i.e., is not accessible through the File System), then the user must identify the line by channel number (argument 2).

The device name must be stored as a string of ASCII characters starting with an exclamation point and ending with a space character (e.g., '!TTY01 ').

If the line is identified by channel number (i.e., if argument 2 has a non-zero value), this argument is ignored.

DESCRIPTION:

During system building, the user can specify that the communications Auto Call Unit be applied to one or more communications lines. For each line supported by auto-dialing, the user supplies one or more telephone numbers. The system constructs a list of these numbers, leaving the first entry of the list empty.

The Set Dial macro call allows you to dynamically insert a telephone number into the empty entry in the list for a particular line. When the Auto Call Unit handler is invoked, this telephone number is dialed first in the attempt to establish a connection with the terminal(s) on the line. If no successful connection is established, the next telephone number in the list is dialed, and so on until a successful connection is made or every number in the list has been dialed. (Each telephone number is dialed three times at 40-second intervals.)

When using this macro call, the user supplies either a channel number or a device name to identify the line. If a device name is supplied, the value of argument 2 must be zero.

NOTES

1. The system places in $B4 the address of the telephone number supplied by argument 1. if argument 1 is omitted, the system assumes that $B4 contains the address of the telephone number.

2. The system places in $R6 the channel number or zero value supplied by argument 2. If argument 2 is omitted, the system assumes that $R6 contains zeros and that argument 3 supplies a device pathname.

3. The system places in $B2 the device pathname, if any, supplied by argument 3. If argument 3 is omitted, the system assumes that $B2 contains the device pathname.

4. On return, $R1 contains one of the following status codes:

   0000 - No error

   0201 - Invalid pathname

   0701 - Channel not configured

   0702 - Auto Call Unit (ACU) control unit not configured on this channel

   0703 - ACU in progress

   1704 - Invalid argument length

   170F - Invalid digit in telephone number.

Example 1:

In this example, a terminal assigned to channel number X'FF80' is to be connected by dialing the number 1-617-555-4444.

```
DIALAA    $SDL    !NUM_12, =CHAN
           .
           .
           .
NUM_12    TEST    '1617555444 '
CHAN      EQU     X'FF80'
```

Example 2:

In this example, the terminal whose pathname is TTY01 is to be connected by dialing the number 1-617-555-4444.

```
    DIALAA    $SDL    !NUM_12, CHAN, !PATH
                .
                .
                .
    NUM_12    TEXT    '16175554444 '
    CHAN      DC      0
    PATH      TEXT    '!TTY01 '
```

SET EXTERNAL SWITCHES ($SETSW)

Function Code: 0B/01

Equivalent Command: Modify External Switches (MSW)

Set the specified external switches in the task group's external switch word to on; return the inclusive logical OR of the previous settings.

FORMAT:

```
[label]   $SETSW   external switch name,
                   [external switch name],
                              .
                              .
                              .
                   [external switch name]
```

ARGUMENTS:

external switch name ... external switch name

A single hexadecimal digit (0 through F) specifying the external switch in the task group's external switch word. A maximum of 16 external switches (0 through F) can be specified. If no arguments are supplied, $R2 is assumed to contain a mask word specifying the switches to be set on. If ALL is specified, all external swiches are set on.

DESCRIPTION:

This call provides a mask by which switches can be set in the external switch word of the issuing task's task group. It also provides an indication of the previous settings of these switches.

$R2 is the mask word. Each bit in $R2 that is one causes the corresponding bit in the external switch word to be set on; each bit that is zero causes the corresponding bit to remain unchanged.

When the Set External Switches macro call is executed, $R2 contains the new settings of the external switch word. Bit 11 (bit-test indicator) of the I-register provides an indication of the previous setting of the switches in the switch word, as follows:

- If bit 11 is zero, no switch set on had previously been set on.
- If bit 11 is one, at least one switch of these set on had previously been set on.

NOTES

1. The bits corresponding to the external switches in the arguments are set on in $R2; if no arguments are supplied, $R2 is assumed to contain the mask to be used. If ALL is specified, all bits are set on in $R2.

2. On return, $R2 and the I-register contain the following information:

   $R2 - External switch word after modification

   I-register (Bit 11) - Inclusive OR of previous settings of switches set on:

   0 - No switch set on was on

   1 - At least one switch of those set was on.

Example:

In this example, the Set External Switches macro call is used to turn on external switches 2, 4, and B of the task group in which the issuing task is executing.

```
    SET_AA    $SETSW    2,4,B
```

SET GROUP ATTRIBUTES

SET GROUP ATTRIBUTES ($SGRPA)

Function Code: 0D/13

Equivalent Command: None

Set/Reset one of the following attributes for the issuing task group: message chaining on/off, ready prompt on/off, break key on/off, memory clear on/off.

FORMAT:

```
[label]  $SGRPA  [attribute code]
```

ARGUMENT:

attribute code

One of the following alphabetic strings or numeric codes; specifies which attribute is to be set/reset.

Alphabetic string	Numeric code	Significance
MHOFF	0	Message chaining off
MHON	1	Message chaining on
RDF	2	Ready prompt off
RDN	3	Ready prompt on
BRKN	4	Break key on
BRKF	5	Break key off
MCF	6	Memory clear off
MCN	7	Memory clear on

DESCRIPTION:

This macro call establishes the requested attribute in the issuing task's Group Control Block (GCB).

The message chaining attribute of a task group is used to govern the extent to which messages (e.g., errors) are reported by any task within the task group. When message chaining is on and the particular message being reported is designated as having chained elements, a "more help?" prompt is displayed after the first message element. A positive response to the prompt causes the next message element to be displayed. If message chaining is disabled, only the first message element is displayed. The default for all task groups is to have message chaining on. Equivalent commands are MHON and MHOFF. See the System Messages Manual for a detailed description of message chaining.

The ready attribute governs the display of the RDY: prompt that signals to the user the completion of the previous command. Equivalent commands are RDN and RDF.

The break key attribute governs the usage of the break key within the caller's task group. The break key is enabled when the task group is started up. It can be disabled and and then re-enabled any number of times during the user session by means of this macro call. The break off call takes effect immediately, whereas break on takes effect only when the next command is processed.

The memory clear attribute governs action taken by the Memory Manager when any task in the task group returns a block of memory. When memory clear is on, the contents of the memory block are cleared (set to FFFF) before the block is returned to the user pool. This is primarily a means to achieve data privacy among task groups at the expense of the additional overhead required to perform the clear operation.

NOTES

1. The numeric attribute code supplied (or derived from the supplied alphabetic string) is placed in $R2. If this argument is omitted, $R2 is assumed to contain the numeric attribute code.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   082E - Invalid attribute code supplied.

Example:

In this example, the Set Group Attributes macro call is used to disable message chaining for the issuing task group.

```
              •
              •
              •
MCOFF         equ       $
              $SGRPA    MHOFF
```

SET TERMINAL FILE CHARACTERISTICS ($STTY)

Function Code: 10/45

Equivalent Command: Set Terminal Characteristics (STTY)

Set the file characteristics of a terminal.

FORMAT:

[label] $STTY [location of parameter structure address]

ARGUMENT:

location of parameter structure address

Any address form valid for an address register; provides the location of the parameter structure defined below. The parameter structure must contain the following entries in the order shown.

device name

A left-justified 6-byte field that contains the device name of the terminal

line length

A 2-byte binary integer specifying the line length of the terminal. If this word is zero, the terminal's line length is not changed.

reserved

A 2-byte field that must be zero if you are using either of the device-specific words described below. For compatibility with other releases, this may contain a device-specific word.

file indicator

A 2-byte field that details the following specific terminal characterisitcs (0 means do not change the characteristics; 1 means change the characteristics):

Bit	Meaning
0	Input-only device type
1	Output-only device type
2	Bidirectional device type
3	Tab simulation required
4	Tab simulation not required
5	Asynchronous input
6	Asynchronous output
7	Synchronous input
8	Synchronous output
9	Use system buffer
10	Do not use system buffer
11	Field transfer
12	Block transfer
13	Restart on power fail
14	No restart on power fail
15	Reset device-specific word to value specified at system generation.

NOTE

Consistency checks are not made on the above fields.

device-specific word 1

A 2-byte field that is used in conjunction with the device-specific mask 1 (see below) to set or reset the terminal's device-specific word that is used at open (connect) and close (disconnect) time. The meaning of bit values in device specific word 1 depends on the type of line protocol handler (LPH) supporting the terminal. For details, see the <u>System Programmer's Guide, Vol. I</u>, which describes each LPH and its interpretation of device-specific-word bit settings.

device-specific word 2

A 2-byte field that is used in conjunction with device-specific mask 2 (see below) to set or reset the terminal's device-specific word that is used during read and write operations. The meaning of bit values in device specific word 2 depends on the type of line protocol handler (LPH) supporting the terminal. For details, see the <u>System Programmer's Guide, Vol. I</u>, which describes each LPH and its interpretation of device-specific-word bit settings.

Device-specific Mask 1

A 2-byte field that indicates which bits of device specific word 1 are to be used to set or reset the specified option.

Device-Specific Mask 2

A 2-byte field that indicates which bits of device-specific word 2 are to be used to set or resert the specified option.

DESCRIPTION:

This macro call allows the issuing task to dynamically alter terminal file characteristics. The original file characteristics, established at system generation, can be altered to reflect the needs of the issuing task.

NOTES

1. The system places in $B4 the address of the parameter structure supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0201 - Invalid pathname
   0203 - Invalid function code
   0205 - Invalid argument
   0209 - Named file not found.

Example:

In this example, the terminal whose symbolic peripheral device name is TTY4 is changed to an output-only device for asynchronous output. Neither the line length nor the device-specific word is changed. (It is assumed that the file is not open.)

```
TER_AA   DC      'TTY4  '
         RESV    2,0
         DC      B,0100001000000000'
         DC      (10) Z'0000'
          .
          .
          .
SETAA    $STTY   !TER_AA
```

SHRINK FILE ($SHFIL)

Function Code: 10/37

Equivalent Command: SHRINK_FILE
SHRINK

Release, for reallocation, the disk space that has been allocated to the specified file but contains no data.

FORMAT:

```
[label]   $SHFIL   [argument structure location]
```

ARGUMENT:

argument structure location

Any address form valid for an address register; provides the location of the argument structure defined below. The argument structure must contain the following entries in the order shown.

logical file number

A 2-byte logical file number (LFN) that refers to the file to be shrunk; must be a binary number from 0 to 255, or ASCII blanks (2020), which indicate that an LFN is not specified. If this entry contains blanks, the pathname pointer (below) must point to a pathname.

pathname pointer

A 4-byte address that may be any address form valid for an address register; points to a pathname (which must end in an ASCII space character) identifying the file to be shrunk. Binary zeros in this entry indicate that a pathname is not specified. A pathname must be specified if an LFN (see above) is not.

new size

A 4-byte field that currently must contain binary zeros. Zeros indicate that the file's new size will extend to the logical sector that contains the last data control interval.

DESCRIPTION:

This macro call releases disk space that was allocated to a file at the time of file creation, but was not subsequently loaded with data. The function is for situations in which users cannot accurately predict the size of the file they are creating. Such users should specify a high value for the (initial) size or growth_size arguments of the Create File function/command. Then, the file system loads the file into a single, continuous extent; the Shrink File function releases any unused space between the last physical sector containing file data (EOD) and the end of the extent (EOF). An alternative procedure would be to create a file without specifying initial size or growth size, letting the system dynamically allocate space as needed. This procedure is not recommended because it may fragment the file into multiple extents that cannot be efficiently accessed.

The Shrink File function/command is normally performed outside program execution. It can be performed at any time; the file need not be opened or reserved.

The function does not apply to directories or temporary files. The user must have modify access to the immediately superior directory, into which the function rewrites.

The disk file to be shrunk can be specified in the argument structure by either an LFN or pathname. If an LFN is specified, the file must have been previously reserved through that LFN by means of the Create File or Get File function/command.

A restorable disk file (i.e., one with the -RESTORE attribute) cannot be shrunk unless the system's journal file has been opened by the Open Journal command (described in the Commands manual).

If an indexed file is specified, both the data and index portions are shrunk. Before shrinking an indexed file, the user should consider that when an indexed file is closed, any unused space allocated for data is designated as a general overflow area. If this overflow area is desired, the user should shrink the file after closing it; if this overflow area is not desired, the user should shrink the file before closing it. The index portion is shrunk in both cases.

If an alternate index is specified, only that index is shrunk, not the associated data portion of the file.

The Shrink File function does not apply to files that are non-expandable (i.e., files whose specified initial size is the same as the specified maximum size).

NOTES

1. If the argument structure address is coded, the system loads that address into \$B4. If the argument is omitted, the system assumes that \$B4 contains the address of the argument structure.

2. On return, \$R1 contains one of the following status codes:

   01xx - Media error

   0201 - Invalid pathname

   0202 - Pathname not specified

   0205 - Invalid argument

   0206 - Unknown or invalid logical file number (LFN)

   0208 - LFN or file already open (in the same task group)

   0209 - Named file or some superior directory not found

   020C - Volume not found

   0210 - LFN conflict

   0213 - Cannot provide requested file concurrency

   0222 - Pathname cannot be expanded; no working directory

   0225 - Not enough system memory for buffers or structures

   0226 - Not enough user memory for buffers or structures

0228 - Invalid file type (i.e., a directory)

022C - Access control list (ACL) violation

0260 - Journal file not open.

SHRINK FILE PARAMETER STRUCTURE BLOCK OFFSETS ($SHPSB)

Associated Macro Call: $SHFIL

Structure:

Word	Fields
0	Logical File Number (LFN)
1 2	Pathname Pointer
3 4	New Size of File; must be zero
5 6 7 8 9 10 11 12 13 14 15	Reserved; Must be Zero

Generated Offset Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
S_LFN	0	Logical File Number (LFN)
S_PTHP	+1	Pathname pointer
S_NSZ	+3	New size of file; must be zero
S_SZ	16	Size of structure (not a field in the block)

SIGNAL TRAP

SIGNAL TRAP ($SGTRP)

Function Code: 0A/03

Transmit a software-generated trap condition to a specified task.

FORMAT:

```
[label]   $SGTRP   [location of LRN],
                   [trap condition]
```

ARGUMENTS:

location of LRN

Any address form valid for a data register; provides the logical resource number (LRN), a value from 0 through 255, of the task to be signaled. An LRN value of -1 specifies that the task is signaling itself.

trap condition

Can be either (1) location of a trap number or (2) a keyword described below.

1. location of trap number

   Any address form valid for an address register; provides the trap number for the trap condition to be signaled to the task, and can be only one of the following trap numbers:

   0 - Cleanup
   1 - Program interrupt
   48 - Quit

2. keyword

   One of the following, to indicate the trap condition to the issuing task:

   CLEANUP
   PI (for program interrupt)
   QUIT

DESCRIPTION:

This macro call transmits a cleanup, program interrupt, or quit trap condition to a specified task. If the appropriate trap is enabled for the task, the task processes as indicated in the user-written trap-handling routine. The keywords CLEANUP, PI, and QUIT specify trap numbers 0, 1, and 48, respectively (see Appendix A).

The system transforms trap 0 (cleanup) into trap 49 (cleanup due to external termination). Internal system-generated cleanup conditions are received as trap 0 by the task's generalized trap handler.

When trap 1 is signaled to a task without that trap enabled, the unclaimed signal causes the system to signal trap 0 to the task.

When trap 48 is signaled to a task without that trap enabled, the unclaimed signal causes the system to suspend the task.

NOTES

1. The system places in $R2 the LRN of the task to be signaled, supplied by argument 1. When the argument is omitted, the system assumes that $R2 contains the correct LRN. A value of -1 indicates that the task is signaling itself.

2. The system places in $R6 the trap number to be signaled, supplied by argument 2. When this argument is omitted, the system assumes that $R6 contains the correct trap number.

3. On return, $R1 contains one of the following status codes:

   0000 - No error
   0342 - Invalid trap number
   0802 - Invalid LRN.

Example:

The macro call is used to transmit a quit condition to the task (within the issuing task's task group) whose LRN is 40. If trap 48 is not enabled, the task will be suspended.

```
    ALRN    $SGTRP    =40,QUIT
```

SPAWN GROUP ($SPGRP)

Function Code: 0D/05

Equivalent Command: Spawn Group (SG)

Define a new task group within the system. Request the execution of the group's lead task. Delete the group from the system when the group request terminates. If the request for this group is to be queued on disk, task group activation can be deferred to a specified date/time.

FORMAT:

```
[label]  $SPGRP   [location of group id],
                  [location of address of argument list],
                  [location of address of fixed parameter block],
                  [location of memory pool id],
                  [location of base level],
                  [location of high logical resource number],
                  [location of high logical file number],
                  [location of root entry name address]
```

ARGUMENTS:

location of group id

Any address form valid for a data register; provides the group identification of the task group to be spawned. The group id must be a 2-character (ASCII) name that does not have the $ character as its first character.

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the Parameter Block macro call.

If the lead task is the Command Processor, the argument list provides the pathname of the command-in file to be read by the Command Processor and, optionally, arguments to be substituted for parameters in that file. Items in the argument list (i.e., arguments supplied with the $PRBLK call) must be the following:

Item	Content
Argument 1	Ignored by system; null.
Argument 2	Pathname of command-in file read by Command Processor; must be supplied.
Argument 3 . . . argument n	Arguments to be substituted for parameters in command-in file; these arguments are optional.

NOTE

All non-null arguments must be enclosed by single or double quotation marks and must terminate with a blank.

If the lead task activated by $SPGRP is not the Command Processor, the argument list is optionally used to specialize execution of the lead task. The order in which the arguments are listed is the order expected by the lead task. Unless the argument is a pathname, it is not necessarily enclosed in quotation marks. For an example of an argument list used to specialize the execution of a lead task that is not the Command Processor, see Example 2 of Request Group ($RQGRP).

location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block, which can be generated by the Parameter Block macro call. This parameter block has the following arguments:

Argument 1

A string specifying the user id to be associated with the spawned task group (for system use). If this entry is zero, the user id currently associated with the issuing task group is used when the call is executed from a user task group.

Argument 2

A pathname string specifying the command-in and initial user-in files for this request of the lead task of the spawned task group. If this entry is zero, no command-in or initial user-in files are available to the spawned group. However, the spawned group can later obtain a user-in file by means of the New User Input macro call. A nonzero entry is required if the command processor is the lead task.

Argument 3

A pathname string specifying the error-out and initial user-out files of the spawned task group. If this entry is zero, one of the following assumptions is made when the call is executed:

- If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.AO.

- If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.

- If in-path specifies an input-only device, the pathname for the output files is null.

Argument 4

A pathname string specifying the initial value of the working directory to be used by the spawned task group.

Argument 5

External date/time of deferred task. (Disk-queued group requests only).

Argument 6

A pathname string specifying the message library file for this request. If this argument is not specified, the message library pathanme of the requestor is used.

location of memory pool id

Any address form valid for a data register; provides the id of the memory pool used to service all memory requests emanating from the spawned task group. The memory pool id consists of two ASCII characters that name a pool defined at system generation. If this argument is omitted, the spawned task group uses the memory pool associated with the issuing task group.

location of base level

Any address form valid for a data register; provides the base priority level, relative to the system level, at which the lead task executes.

A base level of zero, if specified, is the next higher level above the last system priority level. The sum of the highest system physical level plus 1, and the base level of a group, and the relative level of a task within that group, must not exceed 62.

location of high logical resource number

Any address form valid for a data register; specifies the highest logical resource number (LRN) that will be used by any task in the spawned task group. The LRN can be a value from 0 through FF (hexadecimal). If this argument is omitted, or if the value specified is less than the highest LRN used by the system task group, the system task group's LRN is used.

location of high logical file number

Any address form valid for a data register; specifies the highest LFN to be used by any task in the spawned task group. The LFN can be a value from 0 through FF (hexadecimal). If this argument is omitted, the value 15 is assumed. (Refer to the Associate File macro call.)

location of root entry name address

Any address form valid for an address register; provides the address of the root entry name string that specifies the pathname of the bound unit to be executed as the lead task of the spawned group. The bound unit pathname can have an optional suffix in the form of ?entry, where entry is the symbolic start address within the root segment. If this suffix is not given, the default start address (established at assembly or link time) is used. For example, to specify the command processor as the lead task, use the pathname EC?ECL.

DESCRIPTION:

This call combines the Create Group, Enter Group Request, and Delete Group macro calls. Spawn group implicitly causes the execution of these calls in sequence (i.e., it (1) allocates and creates the data structures required to define and control the execution of the task group, (2) places a request against the group, thereby activating it, and (3) when execution terminates, removes all controlling data structures and returns memory used by the task group to the appropriate memory pool).

To queue a task group request on disk using the Message Facility, create a mailbox for the task group through the Create Group Request command (CGRQ) prior to spawning the group (see the Commands manual). Note that only one group request can be queued on disk using the Message Facility when using the Spawn Group macro call.

Spawned task groups cannot be requested, nor can they be waited upon.

Task group requests have message library definitions associated with them. Each task within the request group will use the supplied message library. If the message library pathname is not supplied, the requestor's message library is used.

The request block generated according to the second argument in the macro call is constructed in space taken from the memory pool of the spawned task group.

A Spawn Group macro call can be issued from a task group that was itself spawned.

NOTES

1. The system places in $R2 the group identifier specified by argument 1. If this argument is omitted, the system assumes that $R2 contains the group id to be used.

2. The system places in $B4 the address of the argument list supplied by argument 2. If this argument is omitted, the system assumes that $B4 contains the address of the argument list used to build the request block.

3. The system places in $B5 the address of the fixed parameter block supplied by argument 3. If this argument is omitted, the system assumes that $B5 contains the address of the block to be used.

4. The system places in $R4 the memory pool id specified by argument 4. If this argument is omitted, $R4 is set to zero to indicate that the memory pool of the issuing task group is to be used by the spawned task group.

5. The system places in $R5 the base priority level specified by argument 5. If this argument is omitted, the system assumes that $R5 contains the base priority level to be used.

6. The system places in $R6 the high LRN value specified by argument 6. If argument 6 is omitted, $R6 is set to zero to indicate that the value of the highest LRN created for the system task group will be used.

7. The system places in $R7 the high LFN value specified by argument 7. If this argument is omitted, $R7 is set to 15.

8. The system places in $B2 the address of the root entry name supplied by argument 8. If this argument is omitted, the system assumes that $B2 contains the address of the root entry name of the bound unit to be executed as the lead task of the spawned group.

9. On return, $R1 and $R2 contain the following information:

$R1 - Return status; one of the following:

0000 - No error

0601 - Invalid memory size or memory pool

0602 - Memory unavailable

0804 - Group id in use

0806 - Invalid group id

0807 - Invalid memory pool id

0808 - Invalid base level

0809 - Invalid high LRN

080A - Invalid high LFN

080C - Unresolved start address

1609 - Bound unit not found

160B - Invalid overlay nesting

$R2 - Group id of spawned task group.

Example:

In this example, the Spawn Group macro call is used to create a task group, execute a task in that group, and then delete the group. The task group is created with a group id of Q2, use of memory pool P2, and a relative base level of 5 (decimal). Both the high LRN and the high LFN are defaulted (only system logical resources will be available, and the highest logical file number available will be 15, decimal). The task group's lead task will be the Command Processor. The request part of the spawn is the same as the request given in the example for the Request Group macro call.

```
        $SPGRP   ='Q2',!ARGS,!INFO;
                 ='P2',=5,,,!ROOT
            .
            .
            .
   INFO  $PRBLK   'JONES.TEST.BΔ',
                  '^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                  '^V1121>UDD>TEST>JONES>ASM_TEST.AOΔ',
                  '^V1124>UDD>TEST>JONESΔ',
                  ,'^V1124>UDD>TEST>JONES>MSGLIBΔ'
   ARGS  $PRBLK   ,'^V1124>UDD>TEST>JONES>ASM_TSTΔ',
                  '-XREFΔ'
                  '-PRINTΔ'
    ROOT  TEXT    'EC?ZXECLΔ'
```

SPAWN TASK ($SPTSK)

Function Code: 0C/05 (different bound unit)
0C/06 (same bound unit)
0C/15 (deferred spawn task)

Equivalent Command: Spawn Task (ST)

Create, request execution of, and then cause a task to be deleted within the task group of the issuing task.

FORMAT:

```
[label]    $SPTSK    [location of task request block address],
                     [location of relative priority level],
                     [location of start address],
                     [location of root entry name address],
                     [loction of clock request block]
```

ARGUMENTS:

location of task request block address

Any address form valid for an address register; provides the location of the address of the request block for the spawned task. The request block indicates whether the issuing task is to wait for the execution of the spawned task; the request block may contain parameters to be passed to the spawned task.

location of relative priority level

Any address form valid for a data register; provides the location of the priority level, relative to the task group's priority level, at which the spawned task is to execute. If this argument is omitted, the priority level used is that of the issuing task.

location of start address

Any address form valid for an address register; provides the location of the task start address to be used when the spawned task is to execute the same bound unit as the issuing task (function code 0C/06).

location of root entry name address

Any address form valid for an address register; provides the location of the address of the pathname of the bound unit root segment to be loaded for execution by the newly created task. The bound unit pathname can have an optional suffix in the form ?entry, where entry is the symbolic start address within the root segment. If no suffix is given, the default start address (established at link time) is used (function code 0C/05).

location of clock request block

Any address form valid for an address register; provides the address of the clock request block (CRB), which specifies when the task will be spawned. Used to create a deferred spawn task (function code 0C/15).

DESCRIPTION:

This macro call combines the functions of the Create Task, Request Task, and Delete Task macro calls in that it constructs the requisite structures for the execution of the task; activates the task; and, when the task becomes inactive, deletes the task. When the spawned task is deleted, its associated data structures are removed. The memory they occupied is returned to the task group's memory pool.

A spawned task is not assigned a logical resource number (LRN); therefore, the spawned task is local to the spawning task (i.e., is visible only to the spawning task). A spawned task cannot be requested or referred to by any other task; nor can its memory space or code be shared. However, a spawned task can share the memory space and code of another task that was assigned an LRN by a previously issued Create Task macro call. This sharing is indicated by the presence of argument 3.

Either the location of the start address or the location of the root entry name address, but not both, can be specified.

Multiple task requests can be made to execute concurrently within a given task's bound unit; this is accomplished by issuing multiple Spawn Task macro calls.

NOTES

1. The system places in $B4 the address of the request block supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the request block.

2. The system places in $R6 the relative priority level supplied by argument 2. If this argument is omitted, $R6 is set to -1 to indicate that the priority level of the issuing task is to be used.

3. Arguments 3 and 4 are mutually exclusive; if both are supplied, argument 3 is used and a diagnostic is issued. The system places information derived from either argument in $B2. If these arguments are omitted, the system assumes that $B2 contains the start address within the bound unit.

4. If an address of a CRB was supplied by argument 5, it is placed in $B3, and an MCL of 0C/15 is issued; otherwise an MCL 0C/05 is issued.

5. On return, $B1 contains the address of the CRB in system space used to achieve the deferred spawn task.

6. On return, $R1 contains one of the following status codes:

   0000 - Task successfully spawned (if no wait condition was indicated in the request block)

   0000-FFFF - Posted completion status of spawned task (if wait condition specified)

   01xx - Media error

   0209 - File or directory not found

   0602 - Memory unavailable

   0801 - Request block in use (T-bit on)

   0817 - Memory access violation on request block

0827 - Invalid file type for bound unit

082D - Group available memory quota exceeded

0E02 - No memory available for nonswappable task

1604 - Unresolved symbolic start address

160A - Insufficient memory

1613 - Invalid pathname format

1614 - Access violation (root segment not user segment)

1615 - Invalid bound unit file (header incorrect or number of overlays plus the root is equal to zero).

Example:

In this example, the Spawn Task macro call creates a task, requests its execution, and then deletes the task. The task creation part of the spawn is the same as that given in the first example for the Create Task macro call, except that there is no LRN. The request part of the spawn is the same as that given in the example for the Request Task macro call, except that a synchronous request is made instead of an asynchronous request, and no semaphore is V-oped (see "Semaphore Functions" in Section 2, Vol. I). The delete part of the spawn is the same as given in the example for the Delete Task macro call.

```
        $SPTSK    !TRB,=2,,!ROOT
           .
           .
           .
TRB     $TRB      ,,,ENTRY3,,-PRINT
ROOT    TEXT      'PROG10 '
```

STATUS MEMORY POOL ($STMP)

Function Code: 04/06

Equivalent Command: None

Determine the amount of memory available in a specified memory pool.

FORMAT:

```
[label]    $STMP    [location of memory pool id]
```

ARGUMENT:

location of memory pool id

Any address form valid for a data register; provides the memory pool id of the memory pool to be examined. If this argument is omitted, the memory pool examined is that associated with the task group of the issuing task.

DESCRIPTION:

This macro call allows the issuing task to determine the amount of memory currently available in a specified memory pool. The amount of available memory is returned to the issuing task, both as the actual number of words now available in the pool and as the percentage of the pool's total memory now available. The total available memory may not be contiguous.

If the memory pool being examined has the preempt batch option, the statistics returned are for the specified memory pool combined with the batch task group's memory pool.

NOTES

1. The system places in $R2 the memory pool id of the memory pool to be examined, supplied by argument 1. If this argument is omitted, $R2 is set to -1 to indicate that the memory pool of the task group of the issuing task is to be examined.

2. On return, $R1, $R2, $R6, and $R7 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0606 - Invalid or undefined memory pool id

   $R2 - If $R1 is 0000, percentage of the memory pool's total memory that is currently available. The percentage is returned as an integer with the fractional value truncated.

   $R6, $R7 - If $R1 is 0000, the number of words of memory currently available in the memory pool.

Example:

In this example, the Status Memory Pool macro call is used to determine the amount of memory available in the memory pool of the issuing task's task group. The number of words of memory available in the pool is returned in $R6 and $R7. A double-word 2500 is subtracted from the double-word size, and the high-order word of the result is checked if the result is still positive.

```
POOLCT   $STMP
         SUB     $R7 = 2500
         BCT      +SA
         ADV     $R6 -1
    $A   BGEZ    $R6,SOMMEM
         .
         .
         .
SOMMEM   $GMEM   =2500
```

SUSPEND GROUP ($SUSPG)

Function Code: 0D/08

Equivalent Command: Suspend Group (SSPG)

Suspend the specified task group.

FORMAT:

```
[label]    $SUSPG    [location of group id]
```

ARGUMENT:

location of group id

Any address form valid for a data register; provides the group id of the task group to be suspended. This task group must have been previously defined by a Create Group macro call.

DESCRIPTION:

This macro call causes the system to suspend the specified task group. The task group is marked as suspended when:

- All tasks of the group have exited from critical areas of the Monitor.
- All active task control blocks have been removed from their level queue.
- All external requests (system driver, clock, memory, semaphore) have been satisfied.

A suspended task group can be activated through the Activate Group macro call.

When the suspended task group is aborted, or no other task group issues a Activate Group macro call to enable the suspended group, the operator must issue an Activate Batch or Activate Group command to allow the suspended group to continue.

NOTES

1. The system places in $R2 the group id of the task group to be suspended, supplied by argument 1. If this argument is omitted, the system assumes that $R2 contains the correct group id.

2. On return, $R1 and $R2 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   0806 - Specified group id not currently defined

   $R2 - Group id as supplied.

Example:

In this example, the Suspend Group macro call is used to suspend the task group whose group id is G1. Task group G1 will not be suspended until all its tasks have exited from critical areas of the Monitor and all external requests have been satisfied.

```
SUSGAA    $SUSPG    =G1
```

SUSPEND FOR INTERVAL ($SUSPN)

Function Code: 05/02

Equivalent Command: None

Remove the issuing task from the active queue for its priority level until the specified interval has elapsed.

FORMAT:

```
[label]   $SUSPN   [interval unit designator],
                   [location of interval value]
```

ARGUMENTS:

interval unit designator

One of the following codes must be specified to indicate the manner in which the interval is to be measured.

Code	Interval Measurement
MS	Milliseconds
T	Tenths of a second
S	Seconds
M	Minutes
C	Clock resolution units

location of interval value

Provides the interval for which the issuing task is to be suspended; can be one of the following:

=$R7

Interval value is in $R6 and $R7

=hexadecimal string

String specifies the interval value

fieldname

Fieldname represents the first word of a 2-word field containing the interval value

DESCRIPTION:

This call causes the issuing task to be suspended for the period of time specified in the call arguments.

The Suspend Until Time macro call also suspends the issuing task, but the suspension exists until a particular date/time is reached.

NOTES:

1. The system places in $R2 the interval unit designator supplied by argument 1. The contents of $R2 depend on the interval designator chosen, as follows:

Interval Unit Designator	Contents of $R2
MS	1
T	2
S	3
M	4
C	5

2. The system places in $R6 and $R7 the interval value supplied by argument 2. If this argument is omitted or is =$R7, the system assumes that $R6 and $R7 contain the correct interval value.

3. On return, $R1 contains one of the following return status codes:

   0000 - Specified time has elapsed
   0401 - Invalid time interval specified.

4. Periodic use of this call by central processor bound tasks will allow other tasks with the same or numerically higher hardware priority level to obtain CPU time more often.

Example:

In this example, the Suspend for Interval macro call suspends the issuing task for one unit of time measured in units of clock resulution.

```
    ASTPA    $SUSPN    C,=1
```

SUSPEND UNTIL TIME ($SUSPN)

Function Code: 05/03

Equivalent Command: None

Remove the issuing task from the active queue for its priority level until the date/time specified in the call.

FORMAT:

```
[label]    $SUSPN    [TIME],
                     [location of internal date/time value]
```

ARGUMENTS:

TIME

Optional keyword; explicitly notes that a date/time value will be used to govern the suspension of the issuing task.

location of internal date/time value

Any address form valid for a data register; provides the address of a 3-word internal date/time value that, when reached, causes the task to be activated. The value is a binary count of milliseconds since January 1, 1901.

DESCRIPTION:

This macro call causes the issuing task to be suspended until the date/time value indicated by argument 2 is reached.

The Suspend For Interval macro call also suspends the issuing task, but for a particular interval of time.

NOTES

1. If argument 1 is omitted, date/time format is assumed.

2. The system places in $R2, $R6, and $R7 the internal date/time value supplied by argument 2. If this argument is omitted or is either =$R2 or =$R7, the system assumes that these registers contain the correct internal date/time value.

3. On return, $R1 contains one of the following status codes:

   0000 - Specified date/time has been reached

   0401 - Invalid internal date/time, or interval value.

Example:

The Get Date/Time ($GDTM) macro call is used to get the current date/time (in internal format), leaving it in registers R2, R6, and R7. The External Date/Time, Convert To ($EXDTM) macro call is then used to convert this internal format to an external format, replacing the date portion (first 10 characters) of the field labeled TODAY. The External Time, Convert To ($EXTIM) macro call is then used to convert the internal format date/time to an external format, storing the hour of the date in the field labeled HOUR. The Internal Date/Time, Convert To ($INDTM) macro call converts the contents of the field TODAY back to the internal format contained in $R2, $R6, and $R7. The field HOUR is then compared to the constant 08. If HOUR is greater than or equal to 08, one day (86,400,000 milliseconds) is added to $R2, $R6, and $R7. Thus, $R2, $R6, and $R7 now contain the internal format date/time value for the next time, either today or tomorrow, that 0800 hours occurs. The Suspend Until Time ($SUSPN) macro call then suspends the issuing task until the next time the clock reads 0800 hours. The addition of one day to $R2, $R6, and $R7 is programmed, assuming a central processor that has the add integer double (AID) instruction. (See the example given for the Internal Date/Time, Convert To macro call for the same addition performed without the use of the AID instruction.)

```
*
*    GET THE CURRENT DATE/TIME VALUE.
*
              $GDTM
*
*    CONVERT IT TO AN EXTERNAL FORMAT DATE.
*
              $EXTDT    ,!TODAY,=10
*
*    CONVERT IT TO AN EXTERNAL FORMAT HOUR OF DAY.
*
              $EXTIM    ,!HOUR,=2
*
*    NOW CONVERT THE EXTERNAL FORMAT DATE/TIME
*    BACK TO THE INTERNAL FORMAT.
*
              $INDTM    !TODAY,,=15
*
*    IF IT'S BEFORE 0800 HOURS THE INTERNAL FORMAT
*    DATE/TIME IS CORRECT ELSE IT'S ONE DAY TOO SMALL.
*
              LDR       $R1,HOUR
              CMR       $R1,='08'
              BL        >SUSPND
              AID       A_DAY
              CAD       =$R2
SUSPND        $SUSPN    TIME
                •
                •
                •
TODAY         TEXT      'YYYY/MM/DD 0800'
HOUR          TEXT      'HH'
A_DAY         DC        86400000B(31,0)
```

SWAP FILE ($SWFIL)

Function Code: 10/5A

Equivalent Command: None

Close the tape or disk file on the current volume (writing end-of-volume labels as required); force a swap to the next volume in the set; open the file on that volume.

FORMAT:

[label] $SWFIL [FIB address]

ARGUMENT:

FIB address

Any address form valid for an address register; provides the location of the 16-word file information block used in data and storage management calls.

DESCRIPTION:

This function is meaningful only for labeled tape files and sequential disk files on a serial, multivolume set. It returns an "end of file" error (021F) if applied to other types of files.

This call enables the user to finish a magnetic tape file as though an end-of-tape signal (output mode) or an end-of-volume trailer (input mode) had been encountered. If a continuation reel is online, it is selected; otherwise a mount request occurs.

The FIB used for data management and storage management calls can also be used for the Swap File call. Swap File clears the Out Record Address field (in which the system places the relative record number of the last record transferred by the last data management call). After subsequent read record operations, this field specifies a record number that is relative to the beginning of the current file section.

The file must be opened for either data management or storage management. If the file is opened in output mode for data management, end-of-volume trailer records (EOV1/EOV2) are written, followed by two tape marks at the current tape position.

If the tape file is opened in input mode for data management access, the tape is rewound and unloaded, and a normal reel swap is required. The reel with the next subsequent file section is expected.

If the tape file is opened for storage management access, the tape is rewound and cycled down.

Since there is no way of knowing that a file section is the last one in a set until the trailer records are read, it is the user's responsibility to identify the last file section and issue a Close File call rather than a Swap File call.

The user is responsible for writing any trailer records and tape marks for output files reserved for device (volume) level access.

NOTES

1. On return, $R1 contains the following status codes:

   01XX - Media error

   0205 - Invalid argument

   0206 - Unknown or invalid logical file number (LFN)

   0207 - LFN or file not open

   021F - end of file.

SYSTEM ATTRIBUTE INFORMATION, GET ($SYSAT)

Function Code: 14/11

Equivalent Command: None

Provides the user with attribute information about the software/hardware execution environment.

FORMAT:

[label] $SYSAT [location of marketing identifier string]

ARGUMENT:

location of marketing identifier string

Any address form valid for an address register; provides the address of a 5-word field that is to receive the marketing identifier string.

DESCRIPTION:

This macro call provides the user with the operating system identify and software/hardware attribute information.

NOTES

1. The system places in $B4 the address of the receiving field for the marketing identifier, supplied by argument 1. If argument 1 contains =$B4, the system assumes that $B4 contains the address of the receiving field. If argument 1 is omitted, $R2 is set to zero. If any argument is present in argument 1, $R2 is set to -1.

2. On return, $R1, $R2, $R6, and $R7 contain the following:

   $R1 - 0

   $R2 - Provides operating system identity 4. This value identifies the operating system as MOD 400.

$R6 - Provides hardware information as follows:

- 3 for Model 3x
- 4 for Model 4x and Model 5x

$R7 - Indicates the presence/absence of either a SIP or CIP context. Bit positions 12 and 13 of $R7 have the following significance:

00 - No SIP context present; instructions not executable

X1 - SIP simulator specified on system generation card

1X - SIP hardware present

Bit positions 14 and 15 of $R7 have the following significance:

00 - No CIP context present

X1 - CIP simulator specified on system generation card

1X - CIP hardware present.

SYSTEM IDENTIFICATION ($SYSID)

Function Code: 14/04

Equivalent Command: None

Return the identification of the system under which this task is running to a receiving field. The format of the receiving field is one word containing the number of characters in the system id, followed by 15 words containing the system id itself.

FORMAT:

```
[label]   $SYSID   [location of system id field address]
```

ARGUMENT:

location of system id field address

Any address form valid for an address register; provides the address of a 30-character, aligned, varying receiving field into which the system will place the system identification.

DESCRIPTION:

This macro call returns the system id to a field in the issuing task. The system id is in the form:

GCOS6/MOD400-rrrr-mm/dd/hh/mm

where rrrr is the system software release number and mm/dd/hh/mm are the date and time that the Monitor was linked.

NOTES

1. The system places in $B4 the address of the receiving system id field supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the field.

2. On return, $R1 contains the following status code:

   0000 - No error
   0817 - Memory access violation.

Example:

In this example, the System Identification macro call is used to return the identification of the currently executing system to a field whose address is SYIDFL.

```
    ASYSID     $SYSID     !SYIDFL
                 .
                 .
                 .
    SYIDFL     RESV       15,A'   '
```

TASK GROUP INPUT ($TGIN)

Function Code: 14/0C

Equivalent Command: None

Returns the pathname of the initial command-in file of the calling task group.

FORMAT:

```
[label]   $TGIN   [location of task group input address]
```

ARGUMENT:

location of task group input address

Any address form valid for an address register; provides the address of a 58-character, aligned, nonvarying field into which the system will place the pathname.

DESCRIPTION:

This macro call returns the pathname of the initial command-in file of the calling task group into a 58-character, aligned, nonvarying field whose address is provided by the argument.

NOTES

1. When the argument is entered, the system loads the task group input address into $B4. When the argument is omitted, the system assumes that $B4 contains the address of the receiving home directory field.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   080F - System file undefined
   0817 - Memory access violation

   $B4 - Address of the receiving task group.

TASK REQUEST BLOCK ($TRB)

Function Code: None

Equivalent Command: None

Generate a task request block (TRB) whose length is variable.

FORMAT:

```
[label]   $TRB   [logical resource number],
                 { ,                              }
                 { WAIT,                          } ,
                 { NWAIT, [termination action]    }
                 [task start address],
                 [size of request block argument],
                 [user argument 1],
                 [user argument 2],
                          .
                          .
                          .
                 [user argument n]
```

ARGUMENTS:

logical resource number

A value from 0 through 252 specifying the LRN for this task. If this argument is omitted, the task request block does not have an LRN.

{ WAIT / NWAIT } ,

One of the following values is specified to indicate whether the requesting task is to be suspended until the completion of the request:

WAIT

Suspend the issuing task until the request is complete (set W-bit to zero).

NWAIT

Do not suspend the issuing task (set W-bit to one).

If this argument is omitted, the value NWAIT is assumed.

If WAIT is specified, argument 3 (termination action) must be omitted.

termination action

One of the following values is specified to indicate the action to be taken upon completion of the request.

SM=aa

Release (V-op) the semaphore identified by aa (two ASCII characters), when requested task is completed.

RB=label

Issue a request for the request block identified by label, when requested task is completed.

The requesting task must be asynchronous, cannot wait on the requested task later on, and can only point to a TRB. The requested task must have already been created (not spawned), be asynchronous, and have a valid logical resource number (LRN). When the requesting task terminates, the TRB pointed to by "label" must be inactive.

If this argument is omitted (or argument 2 is WAIT), the generated task request block contains no termination option.

task start address

Any address form valid for an address register; provides the start address to be used when the requested task is turned on to service the request. If this argument is omitted, the implicit task start address is to be used (bit 15 of the T_CT1 word is set to one; see Appendix C).

size of request block argument

Value specifying the number of words in the added portion of the task request block. If this argument is omitted, the generated request block will be large enough to contain only the user arguments specified in the macro call. If no user arguments are specified, the request block will be generated to contain only the standard fixed format request block fields (arguments 1 through 4). If this argument is specified in addition to user arguments, an area is reserved following those arguments.

user argument 1 ... user argument n

Begins the optional, variable-sized area containing user arguments to be passed to the requested task in response to a Spawn Task or Request Task macro call or command. This variable portion of the task request block is built in the following standard format.

entry 1 - One-word count of number of argument pointers

entry 2 - Address of first argument length field

entry 3 - Address of second argument length field

.
.
.

entry r - Address of nth argument length field

entry z - Length (in bytes) of first argument (one word)

entry y - First argument value (of specified size)

entry x - Length (in bytes) of second argument (one word)

entry w - Second argument value (of specified size)

.
.
.

entry p - Length (in bytes) of nth argument (one word)

entry o - nth argument value (of specified size)

DESCRIPTION:

The task request block is used to communicate between tasks. It serves to pass arguments between the requested and requesting tasks within a task group. When a previously created task is requested, the task request block contains the LRN that identifies the requested task. When a task is spawned, the TRB does not require an LRN.

The task request block may contain the start address to be used when the requested task is turned on to service the request.

The task request block may contain a variable-size portion that contains optional information to be passed to the requested task, and has a fixed size protion that contains standard control information.

When a task is activated, its $B4 register points to offset zero of the request block, and its $B7 register points to a parameter list (if one is expected by the task). The proper $B7 address is established by the Task Request Block macro call, when it has a parameter list pointer, or by placing that pointer at the Task Request Block Offsets macro call's T_PRM offset.

Any task-specific arguments are permitted (as if the TRB had been constructed by the command processor).

Example:

In this example, the Task Request Block macro call is used to create a task request block that has a 10-word argument (in addition to space added) to accommodate the parameters passed to the task in control arguments when the task is requested. The generated request block is 18 words long, has an LRN of 30, and when its task terminates, releases semaphore AA.

```
ATRBA     $TRB     30,,SM=AA,,5,XR643MX77B
```

TASK REQUEST BLOCK OFFSETS ($TRBD)

Generated Label Prefixes:

	T_RRB/T_SEM
TRB label	offset 0
	T_CT1
	T_CT2
	T_ADR
	T_PRM

See Appendix C for the format of the task request block.

DESCRIPTION:

See the Task Request Block macro call.

TERMINATE REQUEST ($TRMRQ)

Function Code: 01/04, 01/03

Equivalent Command: None

Terminate the current request being processed by the issuing task. End the current request for the execution of the task and mark its associated request block as terminated.

FORMAT:

```
[label]   $TRMRQ   [location of completion status],
                   [location of new task start address]
```

ARGUMENTS:

location of completion status

Any address form valid for a data register; provides the user-selected status code that is to be returned when the current request and its associated request block are terminated. Completion status codes 0801, 0802, and 0803 should not be used; they are indistinguishable from error codes with the same values.

location of new task start address

Any address form valid for an address register; provides the new task start address for the terminating task. This address is subsequently requested by a request block that does not explicitly specify a start address.

DESCRIPTION:

This macro call is used to end a request for the execution of a task. The Terminate Request function marks a current request block as terminated and removes it from the appropriate request queue.

If there are no other request blocks on the request queue affected by the Terminate Request function, the Task Manager places the task in a dormant state. If there are one or more request blocks in the affected queue, the Task Manager immediately uses the next request block to begin execution of the task at the indicated start address. If the task is requested for deletion and there is no other request for it, the task is deleted; if this is a spawned task, it is deleted.

The Task Manager will do one of the following:

1. Activate a task, if that task is awaiting completion of the current request block being terminated.

2. Release (V-op) the semaphore indicated by the current request block.

3. Schedule the task request block indicated by the current request block being terminated.

If the terminating task will subsequently be requested by a request block that does not explicitly specify a task start address, the terminating task can specify the new task address through argument 2.

NOTES

1. The system places in $R2 the completion status code supplied through argument 1. If this argument is omitted, the system assumes that $R2 contains the completion status code.

2. If argument 2 contains the location of the new task start address, the system places that address in $B4 and issues and MCL 01/04. If argument 2 specifies =$B4, the system assumes that $B4 contains the new start address and issues an MCL 01/04. If argument 2 is omitted, the system does not modify $B4 and issues an MCL 01/03 (no new task start address).

3. On return, $B4, $B5, and $B7 contain the following information:

   $B4 - Address of request block for new request

   $B5 - Address of system supplied termination routine

   $B7 - Address of the request block parameter list.

Example:

In this example, the Terminate Request macro call labeled TRM_NM terminates the issuing task with a completion status of zero without changing the task's start address. The Terminate Request macro call labeled TRM_AB terminates the issuing task with a completion status of one and changes the task's start address to RETRY.

```
TRM_NM     $TRMRQ     =0
                .
                .
                .
TRM_AB     $TRMRQ     =1,!RETRY
```

TEST COMPLETION STATUS ($TEST)

Function Code: 01/02

Equivalent Command: None

Return the completion status of any type of specified request block (e.g., task, clock, I/O, or semaphore).

FORMAT:

```
[label]   $TEST   [location of request block address]
```

ARGUMENT:

location of request block address

Any address form valid for an address register; provides the address of the request block whose completion status is to be tested.

DESCRIPTION:

This macro call permits a running task to ascertain whether a specified request block has been marked as terminated by another task. When the call is executed, control is returned to the issuing task; $R1 contains a return status that shows whether the request block has been terminated and $B4 contains the address of the tested request block.

The Test Completion Status macro call does not cause a wait for the request block to be terminated; that function is performed by the Wait macro call.

A given request block can be tested by any number of tasks.

NOTES

1. The system places in $B4 the request block address supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the request block to be tested.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   yyzz - Where yy can be 00, or 00 through EE for user status, or as defined for other yy values in the System Messages manual.

   $B4 - Address of tested request block.

Example:

In this example, the Test Completion Status macro call is used to determine the status of a task that was requested using a request block labeled TRB. If the requested task has not run to completion yet, a status of 0801 (hexadecimal) will be returned in $R1 and the T-bit in the request block will be on. If the requested task has run to completion, or has so indicated by posting the request block through a Terminate Request macro call, the posted completion status is returned in $R1 and the T-bit in the request block is off.

```
$TEST    !TRB
```

TEST FILE ($TIFIL (INPUT), $TOFIL (OUTPUT))

TEST FILE ($TIFIL (input), $TOFIL (output))

Function Codes: 10/62 ($TIFIL), 10/63 ($TOFIL)

Equivalent Command: None

Test the status of any outstanding I/O activity. These macro calls are used in conjunction with I/O operations where the device to/from which the data transferred is a terminal. The user identifies the file by supplying its logical file number (LFN) in the file information block (FIB).

FORMAT:

```
[label]    {$TIFIL}    [FIB address]
           {$TOFIL}
```

ARGUMENT:

FIB address

Any address form valid for an address register; provides the location of the FIB. The FIB must contain a valid LFN.

DESCRIPTION:

The FIB logical file number (LFN) must be set up prior to a Test File call. The file must be open.

Test File is used in conjunction with Read Record and Write Record functions. Test File does not, in itself, force I/O completion.

Test file is meaningful only for terminals configured as buffered and allowing asynchronous I/O. Terminals not so configured can be dynamically changed by means of the STTY ECL command.

$TIFIL tests the status of a file system input buffer to see whether or not data has been read in and is ready to be transferred to the user record area (by means of Read Record macro call). A 0204 (file busy) return code indicates that I/O is still in progress for the file. A 0000 (normal) return code means that input has been received and is ready to be transferred to the user record area for processing.

$TOFIL tests the status of a file system output buffer, to see whether or not data in that buffer has been written to the terminal. A 0204 (file busy) return code indicates that output to the terminal is still in progress. A 0000 (normal) return code means that no output is currently in progress and that therefore the buffer is ready to receive more data from the user record area (by means of a Write Record macro call).

When a terminal file is opened (by means of the Open File macro call), a connect is issued asynchronously. That is, a wait-until-complete (i.e., until the line is physically connnected) is not done. Test File monitor calls issued after an Open File return a busy code until the connect is physically complete. If the standard time for having a connect raised (i.e., five minutes) expires, any function (e.g., Test File, Read Record, Write Record) returns a 0110 (connect timeout) code. The file is not closed; thus, any attempt to re-raise the connect must be preceded by a Close File function.

After the connect is satisfied, the first Test File issued to an input-only or bidirectional LFN will cause an anticipatory read to be issued. Once a Read Record function has successfully moved data from the file system buffer to the the user record area, a read order will be requeued.

When a terminal file is closed, the file system waits for the completion of any outstanding write orders, dequeues any anticipatory read orders, and optionally issues a disconnect.

The description of the Wait File macro call explains how that call can be used in conjunction with Test File.

NOTES

1. If the argument is coded, the system loads the address of the FIB into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0110 - Timeout occurred on connect
   0204 - File busy
   0205 - Invalid argument
   0206 - Unknown or invalid LFN
   0207 - LFN not open
   0217 - Access violation.

Example:

In this example, a terminal file, FILE_T, associated with LFN 0006, has been reserved (see the Get File macro call) and opened (see the Open File macro call). The following macro calls function as shown in the flowchart below. The FIB for FILE_T is defined as:

```
FILE_T    DC    Z'0006'
           .
           .
           .
(Remainder is the FIB)
```

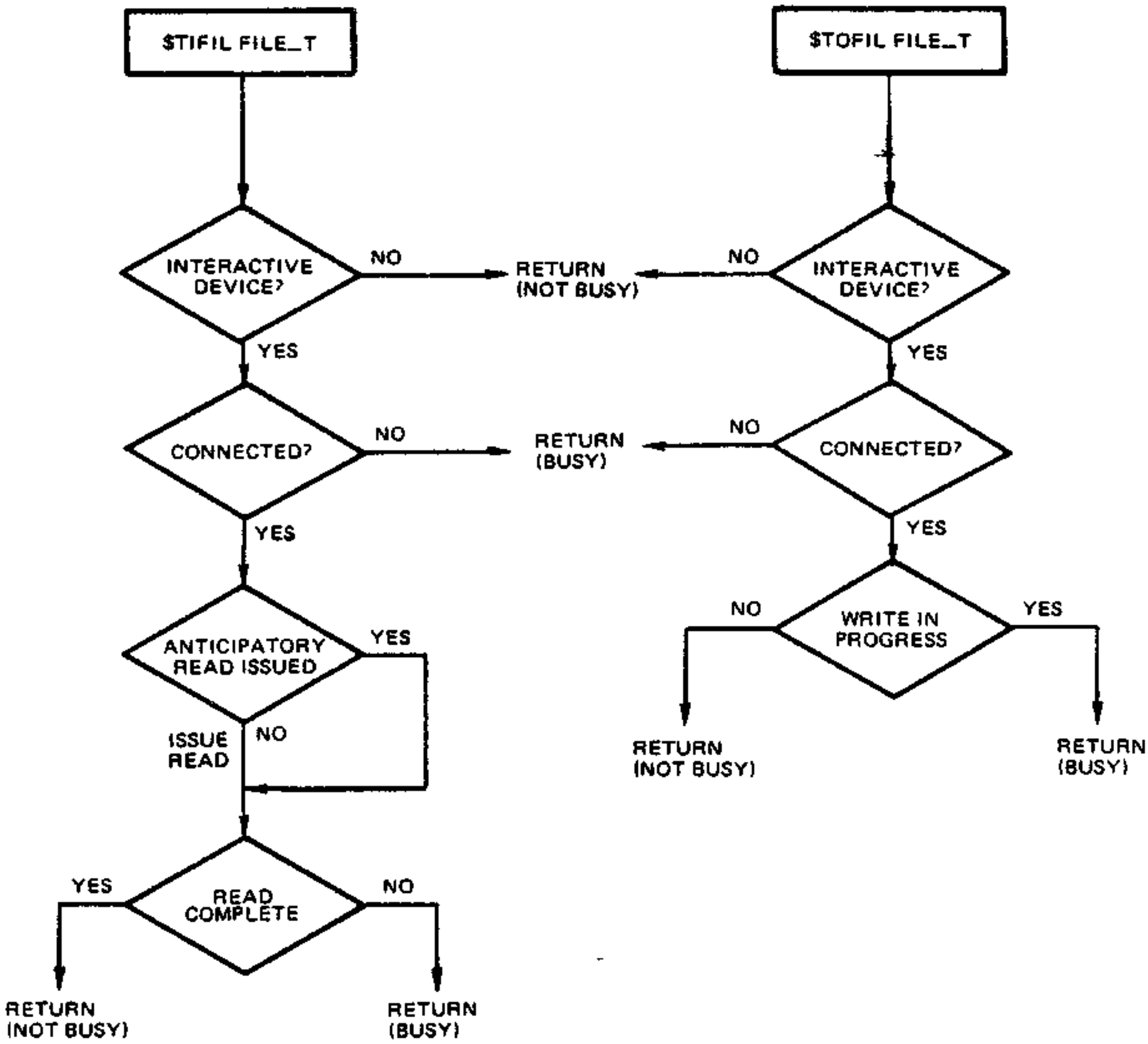


Figure 2-5. Flowchart for Test File ($TIFIL and $TOFIL) Macro Calls

TRANSFER AND RETURN USER ($XRETU)

Function Code 17/07

Equivalent Command: None

Pass a secondary user's terminal from the calling task group back to the Listener component along with the user's next login line arguments. Request that when the next login terminates, the user be returned as a secondary user to the calling group.

FORMAT:

```
[label]  $XRETU  [location of terminal LRN],
                 [location of buffer address],
                 [location of buffer length]
```

ARGUMENTS:

location of terminal LRN

Any address form valid for a data register; provides, in the right byte, the LRN of the terminal to be transfered.

location of buffer address

Any address form valid for an address register; provides the address of a buffer that contains login line arguments.

The text supplied in the buffer must consist only of control arguments for the Login command; the text must not include the Login command itself (L) or the user id, which is not a control argument.

location of buffer length

Any address form valid for a data register; provides the length in bytes of the buffer containing the login line arguments.

DESCRIPTION:

This macro call enables a task group to release a terminal user who has logged in to it as a secondary user, to specify to Listener the user's next login line arguments, and to generate a secondary login from the terminal to the calling group when that next login terminates.

Thus, by means of this call, the user can be transferred from one subsystem to another and back without having to sign off or enter a login line. There is no limit to the number of times a user can be transferred.

The call terminates the user's secondary login as if a Release Terminal macro call had been made. Listener then uses the login line arguments in the specified buffer to define the next login line from the terminal, while retaining the user_id of the current user. If user registration is in effect, the language key value most recently supplied is also retained.

When the next login terminates, Listener acts as if the user then makes a secondary login back to the group that issued the Transfer and Return. For this login to be successful, the group must have a Request Terminal or Request Specific Terminal call outstanding. If no request is found, the user is logged out.

Listener reports at the terminal any error in the supplied login line arguments, and also stores any error code in the device status field of the Request Terminal IORB when it returns the user to the calling group.

NOTES

1. The system places in $R6 the terminal id supplied by the first argument. If this argument is omitted, the system assumes that $R6 contains the terminal id.

2. The system places in $B2 the address of the buffer supplied by the second argument. If this argument is omitted, the system assumes that $B2 contains the address of the buffer.

3. The system places in $R2 the buffer length supplied by the third argument. If this argument is omitted, the system assumes that $R2 contains the buffer length.

4. On return, $R1 contains one of the following status codes:

   0000- Terminal satisfactorily transferred
   3921- Terminal not assigned to task group
   3928- Unable to release terminal; file not removed.

Example:

By means of Transfer and Return User, the calling task releases a terminal to print 10 lines of the file CLM_USER. When the print operation completes, the terminal will be returned to Listener (rather than to the caller), because the caller does not have an outstanding request for a secondary terminal.

The caller previously reserved the terminal as a secondary user by means of the Request Terminal macro call ($RQTML). $RQMTL returned the terminal's LRN in word 0 of the area that received the terminal's login parameters (see the description of Request Terminal). Subsequently, the LRN was stored in the field TRMLRN.

The login line supplied with Transfer and Return User specifies PRINT (PR) as the lead task of the group to be spawned by the login procedure. The -ARG control arguments specialize the execution of PRINT.

```
            $XRETU    =TRMLRN, !ARGBUF, =BUFSIZ
              .
              .
              .
ARGBUF      DC        a'-PO PR  -ARG >SID>CLM_USER  -FROM 10
                      -LIMIT 10'
BUFSIZ      DC        ($-ARGBUF)*2
```

TRANSFER USER ($XFERU)

Function Code: 17/06

Equivalent Command: None

Pass a primary or secondary user's terminal from the calling task group to the Listener component, along with the user's next login line arguments.

FORMAT:

```
[label]  $XFERU  [location of terminal id],
                 [location of buffer address],
                 [location of buffer length]
```

ARGUMENTS

location of terminal id

Any address form valid for a data register; provides in the right byte the identity of the terminal to be transferred. The value of this argument may be one of the following:

lrn

Logical resource number (LRN) of a terminal used for primary or secondary login to the calling group. Must be a binary number in the range 0 through 255.

X'FF'

Signifies that the terminal is one used for primary login to the calling group.

location of buffer address

Any address form valid for an address register; provides the length in bytes of the buffer containing the login line arguments.

The text supplied in the buffer must consist only of control arguments for the Login command; the text must not include the Login command itself (L) or the user id, which is not a control argument.

location of buffer length

Any address form valid for a data register; provides the length in bytes of the buffer containing the login line arguments.

DESCRIPTION

This macro call enables a task group to release a terminal user who has logged into it as a primary or secondary user, and to specify to Listener the user's next login line arguments. Thus, by means of this call, the user can be transferred from one subsystem to another without having to sign off and enter another login line. There is no limit to the number of times a user can be transferred.

The call terminates the user's primary or secondary login as if an Abort Group or Release Terminal macro call had been issued. Listener then uses the login line arguments in the specified buffer to define the next login from the terminal, while retaining the user id of the current user. If user registration is in effect, the language key value most recently supplied is also retained through subsequent transfers.

If the supplied buffer contains the single login line argument BYE, the user's login session terminates. Listener logs out the user and physically disconnects the terminal.

Listener reports at the terminal any error in the supplied login line arguments, and logs the user out.

NOTES

1. The system places in $R6 the terminal id supplied by the first argument. If this argument is omitted, the system assumes that $R6 contains the terminal id.

2. The system places in $B2 the address of the buffer supplied by the second argument. If this argument is omitted, the system assumes that $B2 contains the address of the buffer.

3. The system places in $R2 the buffer length supplied by the third argument. If this argument is omitted, the system assumes that $R2 contains the buffer length.

4. On return, $R1 contains one of the following status codes:

   0000 - Terminal satisfactorily transferred
   3921 - Terminal not assigned to task group
   3928 - Unable to release terminal; file not removed.

Example:

See example for Transfer and Return User ($XRETU)

TRAP HANDLER CONNECT ($TRPHD)

Function Code: 0A/00

Equivalent Command: None

Connect a user-written trap handling routine entry point to the issuing task.

FORMAT:

```
[label]   $TRPHD  [location of trap handling routine address]
```

ARGUMENT:

location of trap handling routine address

Any address form valid for an address register; provides the address of the user-written trap handling routine. This address (entry point) is entered at each occurrence of a user trap that has been enabled for that task.

DESCRIPTION:

This macro call identifies a user-written routine that provides an alternative to the system default trap handler's response to user trap conditions. If user trap conditions are handled by the system default trap handler, the task in which the condition occurs is aborted.

Since trap conditions are handled in a task context, each task must identify the trap handler and enable the particular trap numbers to be serviced on behalf of the task (see Enable user Trap macro call). When an enabled user trap condition occurs, control is transferred to the user-written trap handling routine rather than the system default routine. (See Appendix A, Volume I for more information about trap handling.)

NOTES

1. The system places in $B4 the address of the user-written trap handling routine supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the correct address.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   0341 - Invalid trap handling routine address.

3. This macro call is required to enable a software simulated trap in a task that the user interrupts with the break function, and for which a PI or UW break response is entered.

Example:

In this example, the Trap Handler Connect macro call connects the routine labeled TRAPS as the issuing task's trap handler. The Enable User Trap macro call ($ENTRP) enables the program interrupt and unwind traps for the task. All program interrupt and unwind traps for the issuing task will be directed to the routine labeled TRAPS. The Disable User Trap macro call ($DSTRP) disables all user traps for the issuing task.

The remaining code illustrates the basic techniques used to write a user traphandler. The example is not meant to be typical.

```
*
*        NAME THE PROGRAM TRAPS.
*
PI_T       EQU      1
UW_T       EQU      49
*
*        NAME THE PERTINENT TSA FIELDS
*
TSA_W      EQU      $B3.4
*
*        CONNECT "TRAPS" AS THE TRAP HANDLER.
*
           $TRPHD   !TRAPS
*
*        ENABLE PROGRAM INTERRUPT AND UNWIND.
*
           $ENTRP   =PI_T
           $ENTRP   =UW_T
              .
              .
              .
*
*        READ A NEW DIRECTIVE FROM USER INPUT.
*
GETLIN     $USIN    !LINE,=80
              .
              .
              .
*
*        DISABLE ALL TRAPS.
*
           $DSTRP   =-1
              .
              .
              .
FINISH     $TRMRQ   =$R2
*
*        TRAP HANDLER FOR THIS TASK:
*          SEND PROGRAM INTERRUPT TO "GETLIN"
*          SEND UNWIND TO "FINISH".
*
TRAPS      CMN      +$B3         INCREMENT B3 BY POINTER SIZE
           STB      $B4,TSA_W    SAVE B4
           LAB      $B4,GETLIN
           CMV      $R3,PI_T
           BE       >+$A
           LAB      $B4,FINISH
$A         STB      $B3,$B3
           LDB      $B4,TSA_W    RESTORE B4
           RTT
```

UNLOCK DUMPFILE ($RLDMP)

Function Code: 20/04

Equivalent Command: Unlock Dumpfile (RLDMP)

Unlock the dump file configured for use by the Software Reboot Facility (SRF).

FORMAT:

```
[label] $RLDMP
```

ARGUMENT: None

DESCRIPTION:

The dumpfile unlocked is that used by the Software Reboot Facility (SRF) to take a memory dump before reinitializing the system. When activated, the SRF attempts to take a dump if instructed to do so by an argument of the REBOOT CLM directive. The dumpfile is locked after a successful dump has been taken. Once locked, the dumpfile is unavailable for a later dump until unlocked by the Unlock Dumpfile macro call or command. To ensure a successful dump, an application should issue $RLDMP before activating the SRF.

$RLDMP is a privileged call; it can be issued only by a task running in a privileged memory pool.

NOTE

1. On return $R1 contains one of the following status codes:

   0000 - Dumpfile successfully unlocked

   083A - Function invalid for unprivileged task group

   0865 - Dumpfile not configured

   0869 - Error reported by file system when attempting to reference dumpfile

   086C - WARNING: Dumpfile too small for current system.

Example:

In this example, $RLDMP is issued to ensure that the dumpfile is available when the subsequent Reboot ($RBOOT) macro call is executed.

```
$RLDMP
.
.
.
$RBOOT
```

USER IDENTIFICATION ($USRID)

Function Code: 14/00

Equivalent Command: None

Returns the user id of the calling task group to a 29-character blank-filled receiving field.

FORMAT:

```
[label]   $USRID   [location of user id field address]
```

ARGUMENT:

location of user id field address

Any address form valid for an address register; provides the address of a 29-character, aligned, nonvarying blank-filled field, into which the system will place the user id associated with the issuing task group.

DESCRIPTION:

This macro call returns the task group's user id to a field in the issuing task. The user id will consist of person.account.mode. The unused portion of the field is blank-filled.

NOTES

1. The system places in $B4 the address of the receiving user id field, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the receiving field for the user id.

2. On return, $R1 contains the following status code:

   0000 - No error
   0817 - Memory access violation.

Example:

In the following example, a 16-word field is set up in the issuing task, and the User Identification macro call is issued to place the user id of the task group in that field.

```
    ID01    $USRID    !USIDFL
               .
               .
               .
    USIDFL  RESV      16,A'  '
```

USER INPUT ($USIN)

Function Code: 08/00

Equivalent Command: None

Read the next record from the current input file for the issuing task.

FORMAT:

```
[label]    $USIN    [location of record area address],
                    [location of record size],
                    [byte offset of beginning of record area]
```

ARGUMENTS:

location of record area address

Any address form valid for an address register; provides the address of a record area in the issuing task into which the next record read from the current user-in file will be placed.

location of record size

Any address form valid for a data register; provides the size (in bytes) of the input record area whose address is given in argument 1.

byte offset of beginning of record area

Any address form valid for a data register; provides the byte offset of the beginning of the record area (from the address provided in argument 1). If argument 3 is L, the record area begins at the left byte of the address specified in argument 1. If argument 3 is R, the record area begins at the right byte of this address. Any other value is taken to be the location of the byte offset of the beginning of the record area from the address specified in argument 1. If argument 3 is omitted, the record area is assumed to begin at the left byte of the address specified in argument 1.

DESCRIPTION:

This macro call allows a task to read the next record from the current user-in file. Unless it has been changed by a New User-Input macro call, the user-in file is that file identified in the Request Group or Enter Batch Request macro call.

NOTES

1. The system places in $B4 the address of the record area supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the record area address.

2. The system places in $R6 the record size supplied by argument 2. If argument 2 is omitted, the system assumes that $R6 contains the record size.

3. If argument 3 is L, $R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, $R7 is set to one to designate that the record area begins in the right byte of the specified address. Any other argument 3 value is assumed to designate the location of the byte offset from the address specified by argument 1 and is placed in $R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address, and $R7 is set to zero.

4. On return, $R1, $R6, $R7, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error
   0817 - Memory access violation

   All data management read-next-record error codes may also be returned to $R1. See the <u>System Messages</u> manual.

   $R6 - Residual range (number of bytes not filled in input record area)

   $R7 - File status/type (see the Command In macro call)

   $B4 - Address of input record area.

Example:

In this example, the issuing task is to read the next record of the current user-in file into a 128-byte record area whose address is in RECAD. The record area begins at the left byte of the indicated address.

```
INAA     $USIN    !RECAD,=128
         .
         .
         .
RECAD    RESV     64,0
```

USER OUTPUT ($USOUT)

Function Code: 08/01

Equivalent Command: None

Write the next output record to the current user-out file for the task group of the issuing task.

FORMAT:

```
[label]    $USOUT   [location of record area address],
                    [location of record size],
                    [byte offset of beginning of record area]
```

ARGUMENTS:

location of record area address

Any address form valid for an address register; provides the address of the output record area containing the record to be written to the user-out file. The first byte of the record must be a slew byte (print file format control byte; see "Printer Driver" in Section 6, Vol. I). The record text begins in the second byte.

location of record size

Any address form valid for a data register; provides the size (in bytes) of the record area whose address is given in argument 1. The size value must include the slew byte.

byte offset of beginning of record area

Any address form valid for a data register; provides the byte offset of the beginning of the record area (from the address provided in argument 1). If argument 3 is L, the record area begins in the left byte of the address specified in argument 1; if argument 3 is R, the record area begins in the right byte of this address. Any other value is taken to be the location of the byte offset of the beginning of the record area from the address specified in argument 1. If argument 3 is omitted, the record size is assumed to begin at the left byte of the address specified in argument 1, the $R7 is set to zero.

DESCRIPTION:

This macro call allows a task to write the next record to the current user-out file. Unless it has been changed by a New User Output macro call, the user-out file is as identified by the Request Group or Enter Batch Request macro call.

NOTES

1. The system places in $B4 the address of the record to be written, supplied by argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the output record.

2. The system places in $R6 the output record size, supplied by argument 2. If this argument is omitted, the system assumes that $R6 contains the size of the output record.

3. If argument 3 is L, $R7 is set to zero to designate that the record area beings in the left byte of the specified address. If argument 3 is R, $R7 is set to one to designate that the record area beings in the right byte of the specified address. Any other value is assumed to be the location of the byte offset to be used, and is placed in $R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address, and $R7 is set to zero.

4. On return, $R1, $R6, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000 - No error

   All data management write-next-record error codes may also be returned. See the System Messages manual.

   $R6 - Residual range (number of bytes not transferred from record area).

   $B4 - Address of record area containing output record.

Example:

In this example, the issuing task is to write the next record to the current user-out file for its task group. The record length is 137 bytes (including the slew byte). The output record begins at the right byte of the word labeled REC_AR.

```
            $USOUT      !REC_AR,=137,R
            .
            .
            .
REC_AR      TEXT        ' A', (136)' '
```

VALIDATE CHECKPOINT ($VLCKP)

Function Code: 0D/12

Equivalent Command: Validate Checkpoint (VALIDCKPT)

Determine the availability of a valid, restartable checkpoint.

FORMAT:

[label] $VLCKP [location of pathname of checkpoint files]

ARGUMENT:

location of pathname of checkpoint files

Any address form valid for an address register; provides the pathname of the checkpoint files to be validated. If this argument is omitted, the currently assigned checkpoint files are checked to determine if a valid, restartable checkpoint is accessible.

DESCRIPTION:

This macro call determines the availability of a valid, restartable checkpoint. If argument 1 is specified, the pair of checkpoint files specified by the pathname supplied are checked to determine whether a valid, restartable checkpoint is accessible. If argument 1 is omitted, the currently assigned checkpoint files are checked.

The macro calls associated with the Checkpoint/Restart Facility are: Validate Checkpoint, Checkpoint, Restart, Defer Checkpoint, Checkpoint File.

NOTES

1. The system places in $B4 the address of the pathname supplied by argument 1. $R2 is set to one if a pathname was specified, or is set to zero if no pathname was specified.

2. On return, $R1 contains one of the following return codes:

   0000 - No error
   084C - No valid checkpoint exists
   0209 - File or directory not found
   0213 - Exclusive access not available
   0849 - No checkpoint/restart file assigned.

Example:

This example illustrates the use of the Validate Checkpoint macro in a check of the current checkpoint session. If a checkpoint has been previously taken, a restart is performed that goes back to that checkpoint. If not, the current session continues. This sequence can be performed when an error is detected by the program and a restart is desired.

```
          .
          .
          .
        $VLCKP                      Check current files

        bnez      $rl,>skiprs

        $RS                         Restart to previous
                                    checkpoint

skiprs  equ       0
          .
          .
          .
```

WAIT ($WAIT)

Function Code: 01/00

Equivalent Command: None

Wait for the completion of the operation that uses the specified request block (task, I/O, semaphore, clock, or overlay).

FORMAT:

[label] $WAIT [location of request block address]

ARGUMENT:

location of request block address

Any address form valid for an address register; provides the address of the request block whose termination is to be awaited by the issuing task.

DESCRIPTION:

This macro call permits a running task to indicate, as a separate action, that it wishes to wait for the completion of a particular request for the execution of another task. (The capability of the synchronous Wait function is available through the Request Task function.)

When a Wait macro call is issued, the issuing task must supply the address of the request block to be waited upon. If the Task Manager discovers that this request block is marked as terminated, it immediately returns control to the calling task, supplying the completion status code of the terminated request. If the requst block is not marked as terminated, the Task Manager stores the identity of the calling (and now waiting) task in the request block and then suspends the calling task. Another task can run at this task's level. Later, when the Task Manager is notified of the completion of the request being waited upon, it activates the waiting task and reports the completion status code of the terminated request.

NOTES

1. The system places in $B4 the request block address derived from argument 1. If this argument is omitted, the system assumes that $B4 contains the address of the request block.

2. On return, $R1 and $B4 contain the following information:

   $R1 - Return status; one of the following:

   yyzz - Where yy can be 00 or 80 through EE for user status, or as defined for other yy values in the <u>System Messages</u> manual.

   0000-FFFF - Posted completion status

   0802 - Invalid LRN

   0803 - Invalid wait (request block already waited on, waiting on self, or request block not pending for this task)

   $B4 - Address of request block being waited upon.

Example:

In this example the Wait macro call is used to block the issuing task until a task that was requested using the request block labeled TRB posts its completion to that request block. See the Terminate Request ($TRMRQ) macro call for information about task termination. When the issuing task is returned to the ready state, the task's posted completion status will be in $R1.

```
    WAIT_1     $WAIT     !TRB
```

WAIT ANY ($WAITA)

Function Code: 01/01

Equivalent Command: None

Check the completion status of all marked request blocks generated by the calling task.

FORMAT:

```
[label]  $WAITA
```

ARGUMENT:

None

DESCRIPTION:

This macro call permits a task to indicate that it wishes to wait for request blocks (of any type) to be marked as terminated. All requests waited on must have been issued by the calling task.

Unlike the Wait on Request List ($WAITL) macro call, this call does not require the caller to supply labels locating the request blocks to be waited on. Instead, the caller must set the P-bit of each request block to be waited on. (Request block formats are shown in Appendix C.) If, after the call is issued, no request block with its P-bit set is marked as terminated, the Task Manager suspends the calling task. Upon termination of a marked request block, the Task Manager activates the waiting task, supplies in $B4 the address of the terminated request block, and reports the completion code of the terminated request.

NOTE

1. Upon return, $R1 and $B4 contain the following information:

   $R1 - 0000-FFFF: Posted completion status of first completed request block

   0803 - Invalid wait (no request block generated with P-bit set is outstanding)

   $B4 - Address of terminated request block.

WAIT BLOCK ($WTBLK)

Function Code: 12/20

Equivalent Command: None

Wait for the completion of the I/O operation associated with the specified buffer. This macro call is used only when the asynchronous bit is set in the program view entry in the file information block (FIB) for the preceding Read Block or Write Block macro call.

FORMAT:

```
[label]    $WTBLK    [FIB address]
```

ARGUMENT:

FIB address

Any address form valid for an address register; provides the location of the FIB. The following FIB entry is required:

logical file number

DESCRIPTION:

This macro call immediately follows a Read Block or Write Block macro call. The buffer identified by the user buffer pointer entry in the FIB must not be accessed between the Read block or Write Block macro call and the Wait block macro call, as shown below:

```
$RDBLK (block 0)
$WTBLK (block 0)
$RDBLK (block 1)
(process block 0)
$WTBLK (block 1)
$RDBLK (block 2)
(process block 1)
        .
        .
        .
```

Furthermore, only one asynchronous operation per file can be outstanding at any given time.

The FIB can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Storage Management Access) macro call.

NOTES

1. If the argument is coded, the system loads the address of the FIB into \$B4. If the argument is omitted, the system assumes that \$B4 contains the address of the FIB.

2. On return, \$R1 contains one of the following status codes:

   0000 - No error
   01xx - Physical I/O error
   0203 - Invalid function
   0205 - Invalid argument
   0206 - Unknown or invalid LFN
   0207 - LFN not open
   020A - Address out of file
   020B - Invalid extent description information
   0217 - Access violation
   021F - End of file.

Example:

In this example, it is assumed that the Read Block macro call was coded as described above, except that the program view entry was specified as Z'E001'. Therefore, the Wait Block macro call is coded as follows:

```
    WAITAA    $WTBLK    !BLKFIB
```

WAIT FILE ($WIFIL (input), $WOFIL (output))

Function Codes: 10/64 ($WIFIL), 10/65 ($WOFIL)

Equivalent Command: None

Wait for the completion of an asynchronous I/O activity. The Wait File (input) and Wait File (output) macro calls are used in conjunction with I/O operations in which the device to or from which data is transferred is a terminal. The user specifies a list of logical file numbers (LFNs) identifying the files to be checked by the Wait function. If the $WIFIL macro call is used, the function waits until at least one anticipatory read to one of the specified files is complete. If the $WOFIL call is used, the function waits until at least one write to one of the specified files is complete. The first LFN for which an anticipatory read ($WIFIL) or an asynchronous write ($WOFIL) is complete is placed in the field that the user specifies.

FORMAT:

```
                 {$WIFIL}
     [label]     {$WOFIL}    [parameter structure address]
```

ARGUMENT:

parameter structure address

Any address form valid for an address register; provides the location of the argument structure defined below, which must contain the following entries in the order shown.

out-LFN

A 2-byte field into which file management places the LFN that was the first LFN in the list for which I/O was complete.

list length

A 2-byte field containing a binary number that specifies the number of LFNs in the list. If this field is zero (meaning no list of LFNs is specified), file management assumes a list of LFNs consisting of all LFNs in the task group that are currently associated with opened, interactive devices.

LFN entries

A series of 2-byte fields, each containing the 2-byte LFN used to refer to the file. The LFN is a binary number in the range 0 through 255. Each referenced file must have been reserved (see the Get File macro call) and opened (see the Open File macro call) through this LFN. The LFNs in the list are considered to be in order of priority; the first LFN specified for which I/O has completed is returned in the out-LFN field.

DESCRIPTION:

A Wait File (input) function ($WIFIL) is meaningful only for interactive device files that allow asynchronous input; this function gives up control of the central processor until at least one anticipatory read to any of the specified files is complete. A Wait File (output) function ($WOFIL) is meaningful only for interactive device files that allow asynchronous output; this function gives up control of the central processor until output to one of the specified files is complete.

When a Wait File (input) function is executed, the out-LFN field is set to identify the first LFN in the list for which an anticipatory read is complete. Since more than one read may be completed at the same time, a Test File (input) macro call can be used after the $WIFIL call to ascertain those LFNs for which input is complete. Note that the first $WIFIL call issued after the file has been opened waits for the connect termination (initiated at the time of the open) in addition to waiting for the completion of the first read.

When a Wait File (output) function is executed, the out-LFN field is set to identify the first LFN in the list for which an asynchronous write operation is complete. This function returns the status of the write operation. If the write terminated normally, the file can be considered available for output.

The LFNs in the list are ordered by priority; thus, the first LFN for which I/O has completed will be returned in the out-LFN field. The user can ignore the output LFN and establish his own priority by using the Test File (input) and Test File (output) functions (see the Test File macro call). For example, the user could:

1. Issue a $WIFIL for LFNs 1, 2, and 3.

2. Issue a Test File ($TIFIL) for LFN 2; read and process if not busy.

3. Do the same for LFN 1 and LFN 3.

4. Return to 1.

NOTES

1. If the argument is coded, the system loads the address of the argument structure into $B4. If the argument is omitted, the system assumes that $B4 contains the address of the argument structure.

2. On return, $R1 contains one of the following status codes:

   0000 - No error
   01xx - Media error
   0205 - Invalid argument (duplicate LFN)
   0206 - Unknown or invalid LFN
   0207 - LFN not open
   0217 - Access violation.

Example:

In this example, a Wait File (input) function is executed to wait for the completion of a write operation on any of three interactive files whose LFNs are 1, 2, and 3. The completion of a write operation on the file associated with LFN 3 is checked first; if the write is complete, LFN 3 is placed in the out-LFN field. If the write is not complete, LFN 1's file is checked; if not complete, the file associated with LFN 2 is checked. If none of the write operations are completed, the task is placed in the wait state.

```
IWTLST    DC       0
          DC       3
          DC       1
          DC       2
ONWTBB    $WIFIL   !IWTLST
```

WAIT LIST, GENERATE

WAIT LIST, GENERATE ($WLIST)

Function Code: None

Equivalent Command: None

Generate a wait list consisting of a count field followed by the specified number of request block pointers.

FORMAT:

```
[label]    $WLIST    [request block label 1],
                     [request block label 2],
                               .
                               .
                               .
                     [request block label n]
```

ARGUMENTS:

request block label 1 ... request block label n

Label of the request block to be placed in the wait list.

If a label having a value of zero is specified before the last label is supplied, an address of zero is generated for the wait list entry that corresponds to that argument position. See Appendix C for the format of the wait list.

DESCRIPTION:

A wait list consists of a count of the number of request blocks to be waited on, followed by the specified number of request block pointers.

When any request block referenced in the wait list provided in the Wait on Request List macro call has been posted as complete, the issuing task is awakened.

A wait list can refer to any mixture of request blocks.

If any pointer in the wait list is zero, it is ignored by the Wait on Request List macro call.

The count field format is 01nn (where nn is the number of request block pointers specified in the macro call).

Example:

In this example, a Wait List, Generate macro call is used to generate a list of three request block addresses (following the count field of 0103).

```
    ALSTA    $WLIST    TSKB01,TSKB02,TSKB03
```

WAIT ON REQUEST LIST ($WAITL)

Function Code: 01/01

Equivalent Command: None

Check the completion status of request blocks. The request blocks specified in the list can be a mixture of types (task, clock, I/O, semaphore, or overlay).

FORMAT:

```
[label]    $WAITL    [request block label 1],
                     [request block label 2],
                              .
                              .
                              .
                     [request block label n]
```

ARGUMENTS:

request block label 1 ... request block label n

Label of the request block to be placed in the wait list.

If a label having a value of zero is specified before the last label is supplied, an address of zero is generated for the wait list entry that corresponds to that argument position. See Appendix C for the format of the wait list.

DESCRIPTION:

This macro call permits a running task to indicate that it wishes to wait for any one of up to 255 request blocks (of any type) to be marked as terminated.

The Task Manager scans the wait list and checks the status of the specified request blocks. If it finds any request block marked as terminated, the task manager returns immediately to the calling task. If it finds that no request block in the list is marked as terminated, the Task Manager suspends the calling task until at least one of the blocks is marked as terminated. When the Task Manager is notified of the termination of a request block specified in the list, it activates the waiting task and reports the completion code of the terminated request.

NOTES

1. If arguments are specified, a wait list is generated. The address of the wait list is placed in $B2; if the arguments are omitted, $B2 is assumed to contain the address of the wait list.

2. Upon return to the issuing task, $R1, $B2, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   yyzz - Where yy can be 00 or 00 through EE for user status, or as defined for other yy values in the System Messages manual.

   0000-FFFF - Posted completion status of first completed request block detection.

   0802 - Invalid LRN.

   0803 - Invalid wait (request block already waited on or not pending for this task; or all pointers on this wait list were null).

   $B2 - Address of wait list

   $B4 - Address of request block that caused return (i.e., first completed request block found); if null, all pointers in the wait list were null.

Example:

In this example, the Request Clock macro call ($RQCL) is issued to start a 5-second timer using the clock request block labeled TIMER. Then the Wait on Request List macro call is used to block the issuing task until either the task that was requested using a request block labeled TRB posts its completion or the Clock Manager posts completion of the 5-second interval on the clock request block labeled TIMER. If the task goes to completion first, the Cancel Clock Request macro call ($CNCRQ) will cancel the request on TIMER, thus freeing it for later reuse. To simplify the example, the return status will not be checked for errors that might occur.

```
         $RQCL      !TIMER
         $WAITL     TRB,TIMER
         CMB        $B4,=TIMER
         BE         T_OUT
*
*   THE SUBTASK COMPLETED FIRST
*
         $CNCRQ     !TIMER
            .
            .
            .
*
*   THE CLOCK TIMED OUT FIRST
*
T_OUT    EQU        S
            .
            .
            .
TIMER    $CRB       R, NWAIT,,MS=5000
```

WAIT ON MULTIPLE REQUESTS ($WAITM)

Function Code: 01/01

Equivalent Command: None

Check the completion status of a list of specified request blocks. The request blocks specified in the list can be a mixture of types (task, clock, I/O, semaphore, or overlay).

FORMAT:

```
[label]  $WAITM  [number of requests to be waited on],
                    [request block label 1],
                    [request block label 2],
                                .
                                .
                                .
                    [request block label n]
```

ARGUMENTS:

number of requests to be waited on

Any address valid for an address register. Specifies the number of requests to be waited on. If the total number of requests completed exceeds the value specified, control is returned to the issuing task.

request block label 1 ... request block label n

Label(s) of the request block(s) to be placed in the wait list.

If a label having a value of zero is specified before the last label is supplied, an address of zero is generated for the wait list entry that corresponds to that argument position. See Appendix C for the format of the wait list.

DESCRIPTION:

This macro call checks the completion status of each of a list of specified request blocks. If the total number of completed requests equals or exceeds the number specified in argument 1, control is returned to the issuing task. Otherwise, the issuing task is suspended until the requisite number has completed.

1. If arguments are specified, a wait list is generated. The address of the wait list is placed in $B2; if the arguments are omitted, $B2 is assumed to contain the address of the wait list.

2. Upon return to the issuing task, $R1, $B2, and $B4 contain the following information:

   $R1 - Return status; one of the following:

   0000-FFFF - Posted completion status of first completed request block detection.

   0802 - Invalid LRN.

   0803 - Invalid wait (request block already waited on; or not pending for this task; or all pointers on this wait list were null).

   $B2 - Address of wait list

   $B4 - Address of request block that caused return (i.e., first completed request block found); if null, all pointers in the wait list were null.

WAIT LIST, GENERATE MULTIPLE ($WLSTM)

Function Code: None

Equivalent Command: None

Generate a wait list in which one or more of the requests waited on must complete before the issuing task is reactivated.

FORMAT:

```
[label]  $WLSTM  [number of requests required to reactivate],
                 [request block label 1],
                 [request block label 2],
                           .
                           .
                           .
                 [request block label n]
```

ARGUMENTS:

number of requests required to reactivate

Specifies the number of completed request blocks necessary for issuing task reactivation.

request block label 1 ... request block label n

Label of the request block to be placed in the wait list.

DESCRIPTION:

The wait list consists of a count of the number of request blocks in the list and a quota of those blocks required to reach completion before issuing task is to be activated, followed by the specified number of request block pointers. When the requisite number of request blocks in the list have completed, the issuing task is activated.

WRITE BLOCK ($WRBLK)

Function Code: 12/10 (normal), 12/11 (tape mark)

Equivalent Command: None

Write (i.e., transfer) a block from a buffer in main memory to a file. The user must supply a buffer and specify both the size of the block and its relative location in the file.

FORMAT:

```
[label]    $WRBLK    [FIB address]    [{,NORMAL}]
                                      [{,TM    }]
```

ARGUMENTS:

FIB address

Any address form valid for an address register; provides the location of the file information block (FIB). The following FIB entries are required.

logical file number

Program view (should include buffer alignment and whether the next write operation is synchronous or asynchronous).

user buffer pointer

transfer size

Block size (must be a multiple of the physical sector size).

block number

{NORMAL}
{NOR}

For disk-resident files, this mode argument indicates that the contents of the buffer are to be written in a control interval whose block number is specified in the block number entry in the FIB.

For nondisk-resident files, this mode argument indicates that the block is to be transferred from the buffer to the next sequential position on the file.

NORMAL is the default value for this macro call.

TM

(For tape-resident files only.) This mode argument indicates that a tape mark is to be written at the next sequential position on the tape.

DESCRIPTION:

Before this macro call can be executed, the Logical File Number (LFN) must have been opened (see Open File macro call) with a FIB program view word that allows access through storage management (bit 0 is one) and allows write operations (bit 2 is one). To write a file sequentially, it is necessary only to issue successive Write Block macro calls in NORMAL mode, which causes the block number entry to be incremented by one after each transfer. The system extends the file space up to the limit specified in the Create File macro call when required to do so as a result of a Write Block macro call. In addition, the system updates the logical end-of-file as the file is extended.

The following end-of-file/end-of-tape considerations must be noted:

- Tape-resident files. If logical end-of-tape (i.e., EOT reflector) is detected during a Write Block macro call, all data is written and status code 0231 is returned. If physical end-of-tape is reached before all data is written, a code of 0231 is also returned.
- Disk-resident files. If there is insufficient space to contain the data defined by the transfer size entry in the FIB (i.e., the file has reached its maximum size), a status code of 0223 is returned. If the file has not reached its maximum size but no more sectors are available for allocation to it, a code of 0215 is returned. No data is written.
- All files. Partial blocks are not written.

Only one asynchronous I/O operation per LFN can be outstanding at any given time.

The FIB can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Storage Management Access) macro call.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4; if the argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0203 - Invalid function

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0207 - LFN not open

   020A - Address out of file

   020B - Invalid extent description information

   0215 - Not enough contiguous logical sectors available

   0217 - Access violation

   0223 - File space limit reached or file not expandable.

   0224 - Directory space limit reached or not expandable.

   0231 - Unexpected tape EOT encountered.

Example:

This example assumes that the FIB was defined as follows:

```
BLKFIB   DC     Z'0005'        LFN = 5
         DC     Z'E000'        PROGRAM VIEW = ALLOW READ/WRITE;
                               SYNCHRONOUS PROCESSING
         DC     <BLKBUF        BUFFER POINTER
         RESV   2-$AF
         DC     256            TRANSFER SIZE = 256
         DC     256            BLOCK SIZE = 256
         DC     Z'00000000'    BLOCK NUMBER
```

Where the above FIB is defined (assuming the appropriate Open File and Get File macro calls are specified), the following macro call can be executed to write the contents of the buffer into the first block (i.e., block 0) in the file:

```
$WRBLK    !BLKFIB,NORMAL
```

WRITE RECORD ($WRREC)

Function Code:
11/20 (next), 11/21 (key), 11/22 (position equal), 11/23 (position greater than), 11/24 (position greater than or equal), 11/25 (position forward), 11/26 (position backward)

Equivalent Command: None

Transfer logical records to a file from the user's record area, or merely position the write pointer to a desired record. Whether to transfer or position is specified by the second (i.e., mode) parameter.

FORMAT:

```
                                      ┌  ┌ ,NEXT    ┐ ┐
                                      │  │ ,KEY     │ │
                                      │  │ ,POSEQ   │ │
[label]   $WRREC   [FIB address]      │  { ,POSGR   } │
                                      │  │ ,POSGREQ │ │
                                      │  │ ,POSFWD  │ │
                                      └  └ ,POSBWD  ┘ ┘
```

ARGUMENTS:

FIB address

Any address form valid for an address register; provides the location of the file information block (FIB).

{NEXT | NXT}

(For all files.) This mode argument indicates that the record is to be written into the position in the file identified by the write pointer. The write pointer is set to the next logical record in the file after the write is complete. The system ensures that the position pointed to is unused or contains a deleted record. Records are written in the file as described in the Data File Organizations and Formats manual. This is the default for this macro call. The user must code the following FIB entries:

logical file number (LFN)
program view (record area alignment)
user record pointer
output record length

After the record is transferred to the file, the system updates the following FIB entry:

output record address (serial sequence number if device file; BSN if tape file; relative key for relative files and simple key for other disk files).

This mode is referred to as write next.

KEY

(For disk files accessed by key, only.) This mode argument indicates that the record is to be written into a position in the file, based upon the key value. The write pointer is set to the next logical record in the file after the write is complete. Records are written as described in the Data File Organizations and Formats manual. The user must code the following FIB entries:

logical file number
program view (record and key area alignment)
user record pointer
output record length
input key pointer
input key format
input key length

After the record is transferred to the file, the system updates the following FIB entry:

output record address

This mode is referred to as write with key.

$\begin{Bmatrix} \text{POSEQ} \\ \text{PEQ} \end{Bmatrix}$

(For disk files accessed by key, only.) This mode argument positions the write pointer to the first position in the file whose key value is equal to the one specified in the FIB. It is normally followed by Write Next macro calls to load the file starting from that position. The user must code the following FIB entries:

logical file number
program view
input key pointer
input key format
input key length

This mode is referred to as read position equal.

$\begin{Bmatrix} \text{POSGR} \\ \text{PGR} \end{Bmatrix}$

(For disk files accessed by key, only.) This mode argument positions the write pointer to the first position in the file whose key value is greater than the one specified in the FIB. It is normally followed by Write Next macro calls to load the file starting from that position. The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as write position greater than.

$\begin{Bmatrix} \text{POSGREQ} \\ \text{PGE} \end{Bmatrix}$

(For disk files accessed by key, only.) This mode argument positions the write pointer to the first position in the file whose key value is greater than or equal to the one specified in the FIB. It is normally followed by Write Next macro calls to load the file starting from that position. The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as write position greater than or equal.

$\begin{Bmatrix} \text{POSFWD} \\ \text{PFD} \end{Bmatrix}$

(For tape-resident, disk sequential, and relative files, only.) This mode argument moves the write pointer forward the number of record positions specified by the key value identified in the FIB (but not beyond the end of file). The same FIB entries as for POSEQ, above, must be coded. This mode is referred to as write position forward.

$\begin{Bmatrix} \text{POSBWD} \\ \text{PBD} \end{Bmatrix}$

(For tape-resident, disk sequential, and relative files, only.) This mode argument is the same as for POSFWD, above, except that the pointer is moved backward the number of record positions specified by the key value in the FIB (but not before the first record). This mode is referred to as write position backward.

DESCRIPTION:

Before this macro call can be executed, the LFN must be opened (see the Open File macro call) with a program view word that allows access through data management (bit 0 is zero) and allows write operations (bit 2 is one). The file must be reserved (see the Get File macro call) with write access concurrency control (type 3, 4, or 5). The write pointer is a logical pointer to where the next record is to be written; it is maintained separately from the read pointer. There is one write pointer per LFN per user. At open file time, the write pointer is set to the first record (if RENEW is specified) or logical end of file (if PRESERVE is specified). The write pointer is modified by each write record operation.

The file information block can be generated by a File Information Block macro call. Displacement tags for the FIB can be defined by the File Information Block Offsets (Data Management Access) macro call.

The following illustrates the effect of write actions according to file organizations.

File Organization	Effects of Write Action
Sequential	Write next: If the file is being created (i.e., opened in RENEW mode), the records start at the beginning of the file. If the file is not being created, the records are appended to the end of the existing file. The position modes POSEQ, POSGR, POSGREQ, POSFWD, and POSBWD may be specified to do a "partial file renewal" or a "file shrink". These modes use a simple key to address (set write concurrency) an active record. The resulting new end-of-data must lie within the file limits that existed before the write operation. Write next and write with key produce identical results when dealing with random files. A write with key verifies that the key length and key pointer references are in the proper position in the user record area. These checks are not done in the write next operation.

File Organization	Effects of Write Action
Relative	Write next, issued immediately after an open file, appends a record to the end of an existing file. In RENEW mode, this action can be used to create the file sequentially. Write next issued after a write next, write with key, or with any position mode, inserts a record in the next available (unused or deleted) space. A write next searches for the next available spaces in which to place the record. Write with key uses a relative or simple key that must address a deleted record or an unused space. All position modes use a relative or simple key to address (set write currency to) an active record, deleted record, or unused space.
Indexed	Write next and write with key (using a key format that indicates a primary key) produce identical results. A write with key operation verifies that the key lengths and key format information in the FIB are correct and that the key pointer refers to the proper position in the user record area. The write next operation does not perform these checks. If the file is being initially loaded, it should be opened in RENEW mode, with the data to be written sequenced in ascending order by primary key. Data management will verify that the supplied keys are in order, and will generate a key-out-of-sequence error if they are not. When inserts are to be made, the existing file should be opened in PRESERVE mode.
Fixed Relative	Fixed relative with nondeletable records: Write next, issued immediately after an open file, appends a record to the end of an existing file. In RENEW mode, this action can be used to create the file sequentially. When issued after a write next, write with key, or any position mode, it inserts a record in the next logical record position.

File Organization	Effects of Write Action
Fixed Relative (cont)	Write with key inserts a record in the space addressed by the relative key. All position modes set write currency to the specified record. Fixed relative with deletable records: Write next, issued immediately after an open file, appends a record to the end of an existing file. In RENEW mode, this action can be used to create the file sequentially. Issued after a write next, write with key, or any position mode, write next inserts a new record in the next available (unused or deleted) space. This write next operation searches for the next available space in which to place the record. Write with key uses a relative key that must address a deleted record or an unused space. All position modes can address (set write concurrency to) an active record, deleted record, or an unused record.
Device Files	Write next allows sequential writing, provided the device can be written to and has been so defined.
Tape Files	If the file was opened in RENEW mode, records start at the beginning of the file. If the file was opened in PRESERVE mode with only write permission granted, records are appended to the end of the existing file. In PRESERVE mode with both read and write permission, write next inserts a record following the last record just read or written. If write is the first function after the open, data is written at the beginning of the file. A tape file can be positioned n records forward or backward.

NOTES

1. If the first argument is coded, the system loads the address of the FIB into $B4. If this argument is omitted, the system assumes that $B4 contains the address of the FIB.

2. On return, $R1 contains one of the following status codes:

   0000 - No error

   01xx - Physical I/O error

   0203 - Invalid function

   0205 - Invalid argument

   0206 - Unknown or invalid LFN

   0207 - LFN not open

   020A - Address out of file

   020B - Invalid extent description information

   0217 - Access violation

   0219 - No current record pointer

   021A - Record length error

   021B - Duplicate key

   021C - Key out of sequence

   021E - Key length or location error

   0223 - Disk space limit reached

   0224 - Directory space limit reached

   0227 - Index limit exceeded while loading an indexed file

   022A - Record lock area overflow

   022B - Record deadlock occurred

022F - Unknown or invalid record type

0231 - Unexpected end-of-tape encountered

0237 - Invalid record or control interval format

023A - Recovery file I/O error

0263 - Journal file I/O error.

Example:

In this example, the FIB (i.e., MYFIB) described under "Assumptions for File System Examples" in Appendix A is identified by the first argument. Assuming that the file has been reserved with write-access concurrency control, and that it has been opened as defined in the Open File example, the macro call is specified as follows:

```
$WRREC    !MYFIB,NEXT
```

After the record is written in the file, the system updates the following entry, which you can interrogate with the FIB offset tag:

F_ORA (output record address)

Appendix A
ASSUMPTIONS FOR FILE SYSTEM EXAMPLES

The examples shown in file system macro call descriptions are based on the following assumptions.

1. All the following displacement definition macros were specified:

   $CRPSB
   $GTPSB
   $GIPSB
   $GIFAB
   $GIKDB
   $TFIB
   $MDPSB
   $STPSB

2. The following argument structures were defined:

   a. Argument structure for Create File Parameter Structure Block Offsets ($CRPSB)

```
FILE_A  DC    Z'0005'  LFN 5
        DC     IDX01   PATHNAME PTR.
        RESV 2-$AF
        DC    Z'4900'  FILE ORG = I (INDEXED)
        DC    80       LOG. RCD. SZ. = 80
        DC    512      C.I. SZ. = 512
        DC    5        INIT. ALLOC. SZ. = 5
        DC    5        ALLOC. INCR. SZ. = 5
        DC    10       MAX. ALLOC. SZ. = 10
        DC    320      FREE SPACE = 320
        DC    2        LOCAL OVERFLOW ALLOCATION INCREMENT=2
        DC    1        NO. OF KEY DESCR. = 1
        DC     KEY     KEY DESCRIPTOR PTR.
        RESV 2-$AF
        RESV 4,0       RESERVED
```

b. The pathname addressed by the previous structure (FILE_A)

```
IDX01  DC    '^VOL03>SUBINDEX.A>FILE_A '
```

c. File information block ($FIB)

```
MYFIB  DC    Z'0005'      LFN 5
       DC    Z'2000'      PROG. VIEW = ALLOW WRITE
       DC     INBUF       USER RECORD PTR.
       RESV  2-$AF
       DC    256          MAX. INPUT RCD. SZ. = 256
       DC    256          OUTPUT RCD. SZ. = 256
       DC    Z'0000FFFF'  RESERVED
       DC    Z'0000'      RESERVED
       DC     MYKEY       INPUT KEY PTR.
       RESV  2-$AF
       DC    Z'010A'      INPUT KEY=PRIMARY; KEY LENGTH=10
       RESV  2,0          RECORD ADDRESS
       RESV  2,0          RESERVED
```

When necessary, other structures are defined in the file system macro call examples.

Appendix B
SUMMARY OF REGISTER CONTENTS FOR SYSTEM SERVICE MACRO CALLS

Table B-1 lists the register contents before and after execution of the system service macro calls. Since data structure macro calls do not affect registers, these are not listed. For a discussion of the registers that are altered or preserved, refer to "Register Conventions and Contents" in Section 1.

The table is arranged in function code sequence.

Table B-1. Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Request and Return Functions														
$WAIT	01/00					Address RB	Status					Address RB		
$WAITA	01/01						Status					Address RB		
$WAITL	01/01				Wait list address		Status				Wait list address	Address RB		
$WAITM	01/01				Wait list address		Status				Wait list address	Address RB		
$TEST	01/02					Address RB	Status					Address RB		
$RPTRN		Code					Status						Terminate routine address	
$TRMRQ	01/03	Code					Status					Address RB	Terminate routine address	RB parameter list address
$TRMRQ	01/04	Code				Start address	Status					Address RB	Terminate routine address	RB parameter list address
$RBADD	01/07						Status					Address RB	Terminate routine address	RB parameter list address
	Contents Before Execution						Contents Returned							
	R1	R2	R5	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Request and Return Functions														
$LQPST	01/0B	Completion code					Status							
$DPRTL	01/0C		Defer priority			New task start	Status					Address next RB	Address termination sequence	Address RB argument list

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
	Request and Return Functions (cont.)													
$DFHRD	01/0D		Defer priority			New task start address	Status					Address next RB		Address RB argument list
$PPNTL	01/0E		Defer priority				Status							
$RCLHD	01/0F	Completion code	Recall priority				Status					Address recalled RB		
	Physical I/O Functions													
$RQIO	02/00					Address IORB	Status					Address IORB		
$ELST	02/05				Device name address	User log table address	Status							
$ELEX	02/07				Device name address	User log table address	Status							
$ELGT	02/08				Device name address	User log table address	Status							
$ELEND	02/09				Device name address	User log table address	Status							
	Memory Allocation Functions													
$GMEM	04/02	Return condition	Size	Size			Status		Size	Size		Address memory		
$GMEM	04/03		Size	Size			Status		Size	Size		Address memory		
$RMEM	04/04					Address memory	Status							
$RMEM	04/05		Size	Size		Address memory	Status		Size	Size		Address memory		
$STMP	04/06	Pool id					Status	Memory (percent)	Memory (words)					

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
		Clock Functions												
$RQCL	05/00					Address CRB	Status					Address CRB		
$CNCRQ	05/01					Address CRB	Status					Address CRB		
$SUSPN	05/02	Code	Interval value				Status							
$SUSPN	05/03	Internal	date/time				Status							
		Date/Time Functions												
$EXTDT	05/04	Internal	date/time			Receiving field address	Status	Internal	date/time			Receiving field address		
		R5=Receiving field size												
$EXTIM	05/05	Internal	date/time			Receiving field address	Status	Internal	date/time			Receiving field address		
		R5=Receiving field size												
$GDTM	05/06						Status	Internal	date/time					
$INDTM	05/07	Internal	date/time			External date/time	Status	Internal	date/time			External date/time address		
		R5=Size of B4 field												
		Semaphore Functions												
$RQSM	06/00					Address SRB	Status					Address SRB		
$CNSRQ	06/01					Address SRB	Status					Address SRB		
$RSVSM	06/02	Code	Identifier				Status		Identifier					
$RLSM	06/03		Identifier				Status		Identifier					
$DFSM	06/04	Value	Identifier				Status		Identifier					
$DLSM	06/05		Identifier				Status		Identifier					
		Overlay Handling Functions												
$OVEXC	07/00	Overlay id	Offset	BU id		Base address	Status							
$OVLD	07/01	Overlay id		BU id		Base address	Status	Overlay id	Offset			Base address		

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
	Overlay handling functions (cont.)													
$OVST	07/03	Overlay id		BU id			Status	Overlay status	Offset	Size		Base address		
$OVRSV	07/05	Overlay id	Offset	BU id		OAT address	Status	Overlay id				OAT address		
$OVRLS	07/06	$B5=Return point address		BU id			Code 0006							
$OVRCL	07/07	Overlay id	Offset	BU id		RB address	Status	Overlay id	Offset			RB address		
$CROAT	07/0A	Size of overlay area entry	Number of entries in overlay area				Status	Actual size of overlay area	Actual size of entries in overlay area			Overlay area table address		
$OVUN	07/0C	Overlay id B5=Return point address		BU id		Base address	Status							
$DLOAT	07/0D					OAT address	Status							
	Standard System File I/O Functions													
$USIN	08/00		Record size	Offset		Address record area	Status		Range	File Type		Address record area		
$USOUT	08/01		Record size	Offset		Address record area	Status		Range			Address record area		
$CIN	08/02		Record size	Offset		Address record area	Status		Range	File Type		Address record area		
$EROUT	08/03		Record size	Offset		Address record area	Status		Range			Address record area		
$NUIN	08/04	0,1, or 2				Address pathname	Status		Record length	File Type		Address pathname		
$UNOUT	08/05	0 or 1				Address pathname	Status		Record length	File Type		Address pathname		
$NCIN	08/06	0 or 1			Address argument list	Address pathname	Status		Record length	File Type	Address argument list	Address pathname		

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Standard System File I/O Functions (cont.)														
$NMLF	08/08					Address ML path-name	Status					Address ML path-name		
Operator Interface Functions														
$OPMSG	09/00					Address IORB	Status					Address IORB		
$OPRSP	09/01					Address IORB list	Status					Address input IORB		
$CMSUP	09/02						0002							
$CMSUP	09/03						0003							
Trap Handling Functions														
$TRPHD	0A/00					Address handler	Status							
$ENTRP	0A/01	Trap number					Status	Trap number						
$DSTRP	0A/02	Trap number					Status	Trap number						
$SCTRP	0A/03	Task LRN	Trap number				Status							
External Switch Functions														
$RDSW	0B/00	Mask						Value switch word						
$SETSW	0B/01	Mask						Value switch word						
$CLRSW	0B/02	Mask						Value switch word						
Task Control Functions														
$RQTSK	0C/00					Address TRB	Status					Address TRB		
$CANRQ	0C/01					RB address	Status		Posted RB code			RB address		

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
	Task Control Functions (cont.)													
$CRTSK	0C/02	LRN	Level		Address start		Status	LRN						
$CRTSK	0C/03	LRN	Level		Address root name		Status	LRN						
$DLTSK	0C/04	LRN					Status	LRN						
$SPTSK	0C/05		Level		Address root name	Address TRB	Status							
$SPTSK	0C/06		Level		Address start	Address TRB	Status							
$BUFXR	0C/07				Address root name		Status							
$CMDLN	0C/08		Size			Address command line	Status					Address command line		
$BUAT	0C/09		Access rights, code	Access rights, data	Address root entry name		Status		Index id	B6=Address data section		Address entry point		
$BULD	0C/0A		Access rights, code	Access rights, data	Address root entry name		Status		Index id	B6=Address data section		Address entry point		
$BUDT	0C/0B		Index id (If B2=0)		Address BU pathname (or 0)		Status	0	Index id					
$CRSEG	0C/0C	Access rights	Segment	size	Address segment word		Status		Segment	size	Segment address			
$DLSEG	0C/0D				Address segment word		Status							
$KILLT	0C/11	LRN					Status							
$CLPNT	0C/13					FIB address	Status							
$ROLBK	0C/14						Status							
$DFCKP	0C/19						Status							

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	[illegible]	B4	B5	B7
Task Group Control Functions														
$RQGRP	0D/00	Group id	B5-Address fixed parameter block		Address argument list		Status							
$CNGRP	0D/03	Group id	LRN	LFN	Address root name		Status	Group id						
		R4=Memory pool id R5=Priority level												
$DLGRP	0D/04	Group id					Status	Group id						
$SPGRP	0D/05	Group id	LRN	LFN	Address root name	Address argument list	Status	Group id						
		B5=Address fixed parameter block R4-Memory pool id R5-Priority level												
$ABGRQ	0D/07	Group id	Abort code				Status	Group id						
$SUSPG	0D/08	Group id					Status	Group id						
$ACTVG	0D/09	Group id					Status	Group id						
$ABGRP	0D/0A	Group id	Abort code				Status	Group id						
$NPROC	0D/0B						Status							
$CKPT	0D/0F						If R2=0, status	If R1=0, status						
$RS	0D/10	Group id				Address checkpoint file				0 or 1	R4=0 or 1			
$CKPFL	^D/11					Address checkpoint file	Status							
$VLCKP	0D/12					Address checkpoint file	Status							

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Task Group Control Functions (cont.)														
$SGRPA	0D/13	Attribute code					Status							
Batch Functions														
$RQBAT	0E/00	R5=Address fixed parameter block					Address argument list	Status						
Error Handling Functions														
$RPMSG	0F/03	R3=Component error code B5=Buffer address R4=Indicators word	Message id		Descriptor list address		Status	Buffer offset	Next Message Link or 0					
$RPDFC	0F/04	R3=Component error code B3=VTCRB address R4=Indicators word	Message id		Descriptor list address		Status		Next Message Link or 0					
File Management Functions														
$ASFIL	10/10					Address argument structure	Status							
$DSFIL	10/15					Address argument structure	Status							
$GTFIL	10/20					Address argument structure	Status							
$RMFIL	10/25					Address argument structure	Status							
$CRFIL	10/30					Address argument structure	Status							

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
File Management Functions (cont.)														
$RLFIL	10/35					Address argument structure	Status							
$SHFIL	10/37					Address argument structure	Status							
$GRFIL	10/38					Address argument structure	Status							
$GNFIL	10/3C					Address argument structure	Status							
$RNFIL	10/40					Address argument structure	Status							
$MDFIL	10/41					Address argument structure	Status							
$GTACT	10/42					Address argument structure	Status							
$STTY	10/45					Address argument structure	Status							
$OPFIL	10/50, 10/51					Address FIB	Status							
$CLFIL	10/55, 10/56, 10/57					Address FIB	Status							
$SWFIL	10/5A					Address FIB	Status					Address FIB		
$GIFIL	10/60					Address argument structure	Status							
$TIFIL	10/62					Address FIB	Status							
$TOFIL	10/63					Address FIB	Status							

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
File Management Functions (cont.)														
$WIFIL	10/64					Address argument structure	Status							
$WOFIL	10/65					Address argument structure	Status							
$GACIL	10/7C					Address argument structure	Status							
$CRDIR	10/A0					Address argument structure	Status							
$DLDIR	10/A5					Address argument structure	Status							
$CWDIR	10/B0					Address argument structure	Status							
$GWDIR	10/C0					Address receiving field	Status							
$XPATH	10/D0					Address argument structure	Status							
$DLFIL	10/35					Address argument structure	Status							
$GIDEV	10/66					Address argument structure	Status							
Data Management Functions														
$RDREC	11/10 through 11/16					Address FIB	Status							
$WRREC	11/20 through 11/26					Address FIB	Status							
$DLREC	11/30, 11/31					Address FIB	Status							

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Content Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Data Management Functions (cont.)														
$WRREC	11/40, 11/41					Address FIB	Status							
Storage Management Functions														
$RDBLK	12/00 through 12/04					Address FIB	Status							
$WRBLK	12/10, 12/11					Address FIB	Status							
$WTBLK	12/20					Address FIB	Status							
Identification and Information Functions														
$USRID	14/00					Address receiving field	Status							
$TERID	14/01					Address receiving field	Status					Address receiving field		
$ACTID	14/02					Address receiving field	Status							
$MODID	14/03					Address receiving field	Status					Address receiving field		
$SYSID	14/04					Address receiving field	Status							
$SATID	14/06					Address receiving field	Status					Address receiving field		
$ENTID	14/07		Entry point name length	BU id		Address entry point name	Status				Address entry point			
$GRPID	14 08					Address user id	Status		Group id					
$HDIR	14/0B					Address receiving field	Status					Address receiving field		

Table B-1 (cont.) Macro Calls, Function Codes, and Register Contents

	Contents Before Execution						Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
		Communications Function												
$SDL	18/00		Channel number or 0		Address device pathname	Address telephone number	Status							
		Software Reboot Functions												
$RLDMP	20/04						Status							
$RBPRM	20/05	Reboot volume id	Dump condition			CLM path-name address	Status							
$RBOOT	20/06	Halt condition	Dump condition				Status							
		User Registration Functions												
[illegible]	24/01		Terminal id				Status		Terminal LRN					
$PRF[illegible]	24/10	Record type	User id		Buffer address		Status	Record type	LRN		Buffer address			
$PRFIF	24/12	Buffer size	User id		Buffer address		Status		Language key		Buffer address			
$PRFCR	24/20	Record type	User id				Status	Record type	LRN					
$PRFUL	24/30	Record type	User id		Buffer address		Status	Record type	LRN		Buffer address			
$PRFUD	24/40	Record type	User id				Status	Record type	LRN					
$PRFAU	24/22	Record type	User id	Buffer length	Buffer address		Status	Record type	LRN					

Appendix C
DATA STRUCTURE FORMATS

This appendix describes the following data structures:

- Clock request block (CRB)
- File information block (FIB)
- Input/output request block (IORB)
- Task request block (TRB)
- Parameter block
- Wait list
- Semaphore request block (SRB)
- Message group request blocks (MGCRB, MGIRB, MGRRB).

Any of the structures can be hand coded or generated by macro calls. All structures but the parameter block and wait list can be defined by macro call templates.

The first four items of the request blocks have an identical format (but slightly different contents, depending on the block type) as shown in Figure C-1. Later diagrams show the format of each block type; tables show the contents of the block entries.

The offset symbol $AF signifies that number of words required to specify a memory address. In this system, $AF is equivalent to two words.

The first field (-$AF or -1) of a request block need be present only when the request block pointer/semaphore name is needed.

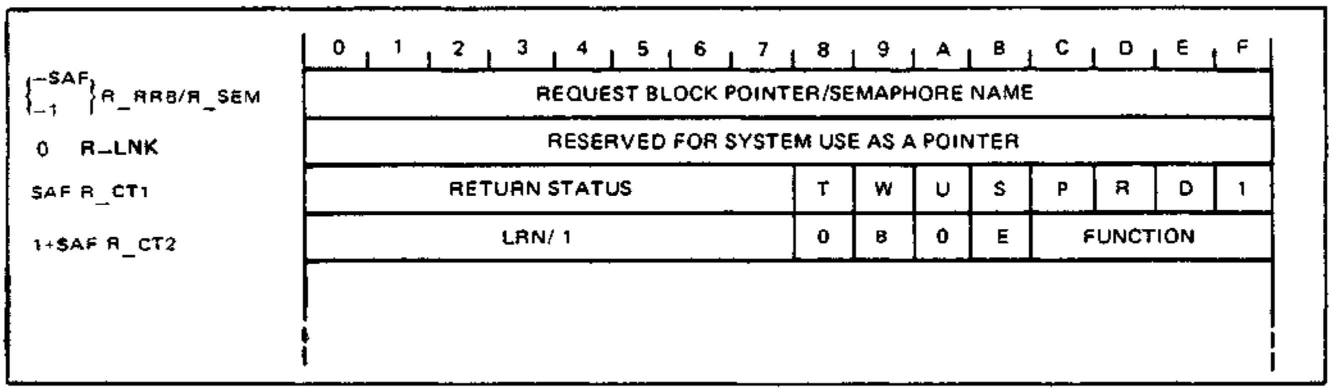


Figure C-1. First Four Items of Request Blocks

CLOCK REQUEST BLOCK FORMAT

Figure C-2 shows the format of the clock request block; Table C-1 shows its contents.

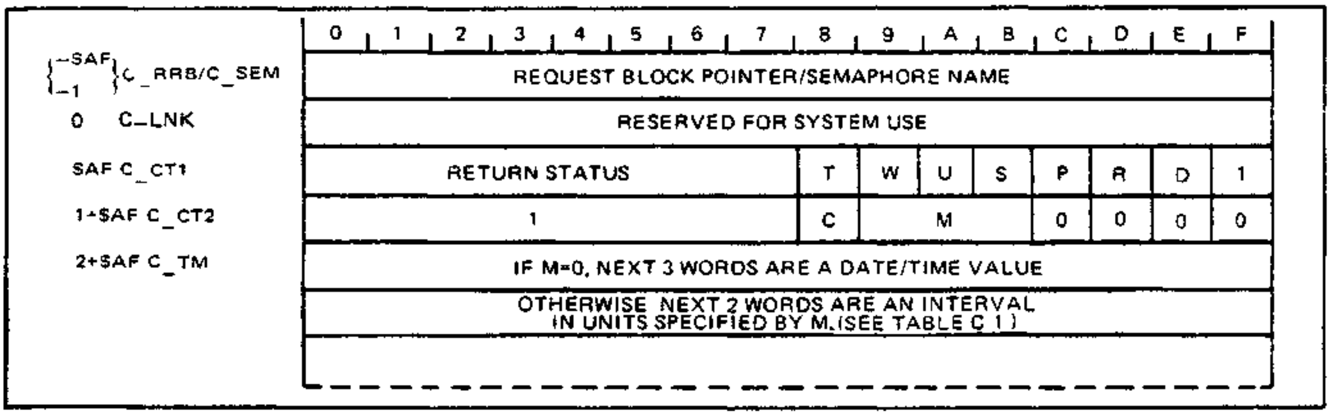


Figure C-2. Format of Clock Request Block

Table C-1. Contents of Clock Request Block

Item	Label	Bit(s)	Contents
-$AF	C_RRB/	0-31	Depending on the condition of the S- or R-bits of C_CTl, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on).
	C_SEM	0-15	
0	C_LNK	0-15	Reserved for system use.
$AF	C_CTl	0-7	Return status.
		8(T)	This bit is set on while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
		9(W)	Wait bit. Set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it. In a user-built CRB, must be 0 initially.
		B(S)	Release semaphore indicator. 0 = No release; 1 = Release, on completion of this request, semaphore item named in C_SEM.
		C(P)	Must be set by user if CRB is to be referenced by a Wait Any ($WAITA) macro call. If set, CRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return clock RB indicator. 0 = No dispatch; 1 = Dispatch task request block named in C_RRB after completion of this request.
		E(D)	Delete clock RB indicator, used usually with the B(S) and D(R) bits. 0 = No delete; 1 = Delete and, when task terminates, return memory to pool where CRB is first entry of its memory block.

Table C-1 (cont). Contents of Clock Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	C_CT1	F	Implicit task start address. Must always be 1 for CRB.
1+$AF	C_CT2	0-7	Value is -1.
		8(C)	When set, indicates this block is associated with a cyclic clock function.
		9-B(M)	When set, last two words contain an interval in units specified by M. Each interval value is as follows: 001 - in milliseconds; 010 - in tenths of a second; 011 - in seconds; 100 - in minutes; 101 - in units of clock resolution. When reset (off), the last <u>three</u> words contain a date/time interval.
2+$AF	C_TM		Contents depend on M bit of C_CT2.

FILE INFORMATION BLOCK (FIB) FORMAT AND CONTENTS

Tables C-2 and C-3 show the format, and Tables C-4 and C-5 show the contents, of the file information block (FIB) for data management (record level) access, and for storage management (block level) access, respectively.

Table C-2. Format of FIB for Data Management

Word	Label(s)	0 1 2 3 4 5 6 7 8 9 A B C D E F
0	F_LFN	Logical file number (LFN)
1	F_PROV	Program view
2	F_URP	User record area pointer
3		
4	F_IRL	Input record length
5	F_ORL	Output record length

Table C-2 (cont). Format of FIB for Data Management

Word	Label(s)	0 1 2 3 4 5 6 7	8 9 A B C D E F
6	F_IRS/F_ORS	Input record status	Output record status
7	F_IRT	Input record type	
8	F_ORT	Output record type	
9	F_IKP	Input key pointer	
10			
11	F_IKF/F_IKL	Input key format	Input key length
12	F_ORA	Output record address	
13			
14	F_RFU2	Reserved	
15			

Table C-3. Format of FIB for Storage Management

Word	Label(s)	0 1 2 3 4 5 6 7 8 9 A B C D E F
0	F_LFN	Logical file number (LFN)
1	F_PROV	Program view
2	F_UBP	User buffer pointer
3		
4	F_BFSZ	Buffer size
5	F_BKSZ	Block size
6	F_BKN1	Block number
7	F_BKN2	

Table C-3 (cont). Format of FIB for Storage Management

Word	Label(s)	0 1 2 3 4 5 6 7 8 9 A B C D E F
8	F_RFU3	Reserved
9		
10		
11		
12		
13		
14		
15		

Table C-4. Contents of FIB for Data Management

Word	Label	Bit(s)	Contents
0	F_LFN	0-15	Logical file number (LFN)
1	F_PROV	0	Access level. Set off for data management.
		1-4	Process rules. Bit 1 for $RDREC, bit 2 for $WRREC, bit 3 for $RWREC, bit 4 for $DLREC.
		5-9	Key type. Bit 5 for primary keys, bit 8 for relative keys, bit 9 for simple keys (bits 6 and 7 must be 00).
		10	Record class. Set on for fixed-length records only; off for fixed- and variable-length records.
		11	Record visibility. Set on if deleted records are to be visible; off if invisible.
		12	Key storage alignment. Set on if storage area begins at odd-byte boundary; off if even-byte boundary.

Table C-4 (cont). Contents of FIB for Data Management

Word	Label	Bit(s)	Contents
1 (cont)	F_PROV (cont)	13	Record storage area. Set on if record storage area begins on odd-byte boundary; off if even-byte boundary.
		14	Transcription mode. Set on if data transferred in binary transcription mode; off if ASCII mode.
		15	Must be 0.
2,3	F_URP	0-31	Start address of user record area.
4	F_IRL	0-15	Input record length (in bytes).
5	F_ORL	0-15	Output record length (in bytes).
6	F_IRS	0-3	0000 - Unknown terminal control information; 0001 - Records contain no terminal control information; 0010 - Records contain standard GCOS 6 printer control characters.
		4-7	Must be zero.
	F_ORS	8	Read operations. Set on if the key of the record just read duplicates the key of the record previously read. Write/rewrite operations. Set on if the key of the record just written is a duplicate.
		9	Read operations. Set on if the key of the record just read duplicates a record that is yet to be read.
		10-15	Must be zero.
7	F_IRT	0-15	Must be set to X'FFFF' (all bits set on).
8	F_ORT	0-15	Must be set to X'0000' (all bits set off).
9,10	F_IKP	0-31	Start address of user key area.
11	F_IKF	0-7	Input key format. 0 for none specified; 1 for primary key; 2 for simple key.

Table C-4 (cont). Contents of FIB for Data Management

Word	Label	Bit(s)	Contents
	F_IKL	8-15	Input key length (in bytes).
12,13	F_ORA	0-31	Output record address.
14,15	F_RFU2	0-31	Reserved for later use; must be X'00000000'.

Table C-5. Contents of FIB for Storage Management

Word	Label	Bit(s)	Contents
0	F_LFN	0-15	Logical file number (LFN).
1	F_PROV	0	Access level. Set on for storage management.
		1-2	Process rules. Bit 1 for $RDBLK; bit 2 for $WRBLK.
		4-12	Must be X'00000000'.
		13	Buffer alignment. Set on when buffer begins on odd-byte boundary; off when even-byte boundary.
		14	Transcription mode. Set on when data transferred in binary transcription mode; off when transfer is in ASCII mode.
		15	Synchronous/asynchronous indicator. Set on when $RDBLK and $WRBLK calls executed asynchronously; off when synchronously.
2,3	F_UBP	0-31	Start address of user buffer area.
4	F_BFSZ	0-15	Buffer transfer size (in bytes).
5	F_BKSZ	0-15	Block size (in bytes).
6,7	F_BKNO	0-31	Block number.
8-15	F_RFU3	All	Reserved for later use; must be all zeros.

INPUT/OUTPUT REQUEST BLOCK (IORB) FORMAT

Figure C-3 shows the format of a nonextended input/output request block (IORB) (see Vol. I, Section 7 for descriptions of IORB extensions). Table C-5 defines the specific fields for a non- extended IORB. Table C-6 summarizes the IORB fields for operator interface functions.

Offset	Label	0–7	8	9	A	B	C	D	E	F
-$AF / -1	I_RRB/I_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME								
0	I_LNK	RESERVED FOR SYSTEM USE AS A POINTER								
$AF	I_CT1	RETURN STATUS	T	W	U	S	P	R	D	1
1+$AF	I_CT2	LRN	IBM	B	P	E	FUNCTION			
2+$AF	I_ADR	BUFFER ADDRESS								
2+2*$AF	I_RNG	RANGE								
3+2*$AF	I_DVS	DEVICE SPECIFIC WORD								
4+2*$AF	I_RSR	RESIDUAL RANGE								
5+2*$AF	I_ST	STATUS WORD/HIGH-ORDER BITS OF WORD7 FOR STORAGE MODULE								
6+2*$AF	I_EXT	TOTAL EXTENSION LENGTH	PIO EXTENSION LENGTH							

Figure C-3. Format of I/O Request Block

Table C-6. Contents of I/O Request Block

Item	Label	Bit(s)	Contents
-$AF -1	I_RRB/ I_SEM	0-31 0-15	Depending on the S- or R-bits of I_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user; used by system at termination of request.
0		0-31	Reserved for system use. 2-word pointer to indirect request block.
$AF	I_CT1	0-7	Return status
		8(T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.

Table C-6 (cont). Contents of I/O Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	I_CT1 (cont)	9(W)	Wait bit. Set by user if the requesting task is not to be suspended pending completion of the request that uses this IORB.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
			0 = No release; 1 = Release, on completion, semaphore item named in I_SEM.
		B(S)	Release semaphore indicator.
		C(P)	Must be set by user if IORB is to be referenced by a Wait Any ($WAITA) macro call. If set, IORB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return IORB indicator.
			0 = No dispatch; 1 = Dispatch task request block named in I_RRB after completion of this request. If 1, system executes $RQTSK, using I_RRB, when the task terminates.
		E(D)	Delete IORB indicator. Used usually with the B(S) and D(R) bits.
			0 = No delete; 1 = Delete and when task terminates, return memory to pool where IORB is first entry of its memory block.
		F	Implicit task start address. Must always be 1 for IORB.
1+$AF	I_CT2	0-7	Logical resource number (LRN). Identifies device to be used.
		8(IBM)	IBM-type request. Changes interpretation of I_DVS to task word, and of I_RSR and I_ST to configuration words A and B, respectively.

Table C-6 (cont). Contents of I/O Request Block

Item	Label	Bit(s)	Contents
1+$AF (cont)	I_CT2 (cont)	9(B)	Byte index. 0 = buffer begins in leftmost byte of word; 1 = buffer begins in rightmost byte.
		A(P)	Private space; reserved for system use.
		B(E)	Extended IORB indicator. 0 = Standard (nonextended) IORB; 1 = IORB extended to at least 6+2*$AF items. Set by user. (See I_EXT below.)
		C-F	Function code. Driver or LPH function, see Vol. I, Table 6-1.
2+$AF	I_ADR	0-31	Buffer address. 2-word pointer.
2+2*$AF	I_RNG	0-15	Range. Number of bytes to be transferred. Used as input field for cartridge disk or mass storage unit.
3+2*$AF	I_DVS	0-15	Device-specific information.
4+2*$AF	I_RSR	0-15	Residual range. Indicates the number of bytes <u>not</u> transferred. Filled in by the system on completion of the order. Used by the cartridge disk and mass storage unit drivers as a data offset value.
5+2*$AF	I_ST	0-15	Modified device status. Shows mapping of hardware status into software status format. See Vol. I, Table 9-4. Set by user as input field high-order bits of sector number of mass storage unit. Set by system after I/O completion.
6+2*$AF	I_EXT	0-7	Left byte. Number of words, in binary, in the IORB extension, not including this I_EXT word.
		8-15	Right byte. Number of words, in binary, in physical I/O part of IORB extension, not including this I_EXT word. This count must be less than or equal to the total extension length specified in the left byte (0-7). This word is present only when the B(E) bit in I_CT2 is 1. (See Vol. I, Section 7 for a description of IORB extensions.)

Table C-7. Summary of IORB Fields for Operator Interface

Item	Label	Bit(s)	Contents
$AF	I_CT1	9(W)	For a $OPMSG call, the setting of the W-bit in the output IORB controls return to the caller. For a $OPRSP call, the setting of the W-bit in the input IORB controls return to the caller; the setting of the W-bit in the output IORB has no significance. For either call, return to the caller is immediate if the significant W-bit is on. If the significant W-bit is off, return to the caller occurs after the order is completed.
1+$AF	I_CT2	0-7	LRN = 0.
		9(B)	Must be off if the input/output buffer begins at the left byte of the word whose address is contained in word 3 (I_ADR) of this IORB. Must be on if the input/output buffer begins at the right byte.
2+$AF	I_ADR	0-15	The word address of the message buffer (which contains an output message or is to receive an input message).
2+2*$AF	I_RNG	0-15	The buffer size in bytes. This is the length of an output message or the maximum length allowed for an input message.

SEMAPHORE REQUEST BLOCK FORMAT

Figure C-4 shows the format of the semaphore request block; Table C-8 shows its content.

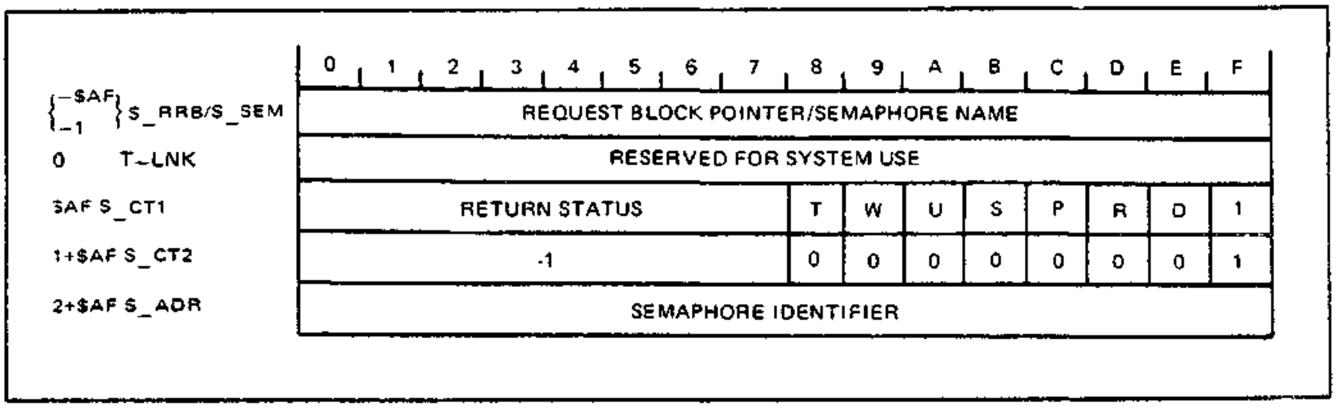


Figure C-4. Format of Semaphore Request Block

Table C-8. Contents of Semaphore Request Block

Item	Label	Bit(s)	Contents
-$AF -1	S_RRB S_SEM	0-31	Depending on the S- or R-bits of S_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user; used by system when request terminates.
0	S_LNK	0-15	Reserved for system use.
$AF	S_CT1	0-7	Return status
		8(T)	This bit is set (on) while the request using the block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
		9(W)	Wait bit. Set if the requesting task is <u>not</u> to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. 0 = No release; 1 = Release, on completion, semaphore item named in S_SEM.

Table C-8 (cont). Contents of Semaphore Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	S_CT1	C(P)	Must be set by user if SRB is to be referenced by a Wait Any ($WAITA) macro call. If set, SRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return semaphore RB indicator. 0 = No dispatch; 1 = Dispatch task request block named in S_RRB after completion of this request.
		E(D)	Delete SRB indicator. Used usually with the B(S) and D(R) bits. 0 = No delete; 1 = Delete and, when task terminates, return memory to pool where SRB is first entry of its memory block.
		F	Implicit task start address. Must always be 1 for SRB.
1+$AF	S_CT2	0-7	Value is -1.
		8-14	Must be zero.
		15	Must be one.
2+$AF	S_ADR	0-15	Semaphore identifier - two ASCII characters.

TASK REQUEST BLOCK FORMAT

Figure C-5 shows the format of the task request block; Table C-9 shows its contents.

	0 1 2 3 4 5 6 7	8	9	A	B	C	D	E	F
{-$AF / -1} T_RRB/T_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME								
0 T_LNK	RESERVED FOR SYSTEM USE AS A POINTER								
1 T_CT1	RETURN STATUS	T	W	U	S	P	R	D	I
1+$AF T_CT2	LRN	0	0	0	0	0	0	0	0
2+$AF T_ADR	START ADDRESS IF I 0								
2+2*$AF T_PRM	BEGINNING OF ARGUMENT LIST								

Figure C-5. Format of Task Request Block

Table C-9. Contents of Task Request Block

Item	Label	Bit(s)	Contents
-$AF -1	T_RRB/ T_SEM	0-31 0-15	Depending on the condition of the S- or R-bits of T_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user, used by system when request terminates.
0	T_LNK	0-31	Reserved for system use.
$AF	T_CT1	0-7	Return status.
		8(T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; the user should not change it.
		9(W)	Wait bit. Set by user if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.

Table C-9 (cont). Contents of Task Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	T_CT1 (cont)	B(S)	Release semaphore indicator. 0 = No release; 1 = Release, on completion, semaphore item named in T_SEM.
		C(P)	Must be set by user if TRB is to be referenced by a Wait Any ($WAITA) macro call. If set, TRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return task RB indicator. 0 = No dispatch; 1 = Dispatch task request block named in T_RRB after completion of this request.
		E(D)	Delete TRB indicator. Used usually with the B(S) and D(R) bits. 0 = No delete; 1 = Delete and when task terminates, return memory to pool where TRB is first entry of its memory block.
		F	Implicit task start address. Must always be 1 for TRB.
1+$AF	T_CT2	0-7	Logical resource number (LRN).
		8-15	Must be zero.
2+$AF	T_ADR	0-15	Start address if the I-bit of T_CT1 is reset (zero).
2+2*$AF	T_PRM		Beginning of argument list.

PARAMETER BLOCK FORMAT

Figure C-6 shows the format of the parameter block.

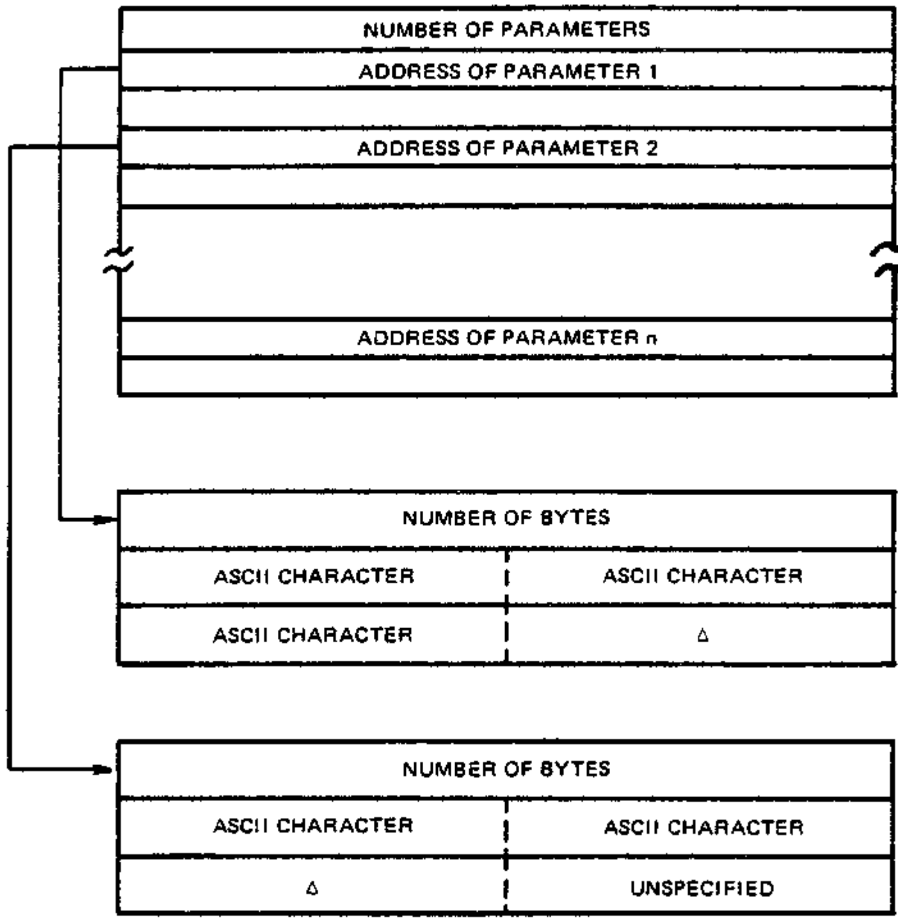


Figure C-6. Format of Parameter Block

NOTE

The parameter value strings need not be contiguous with the address portion of the parameter block; if the block is system-generated, each parameter will have a trailing blank that is not included in the byte count.

WAIT LIST FORMAT

Figure C-7 shows the format of the wait list.

NUMBER/ITEMS TO WAIT FOR	TOTAL ITEMS IN LIST
ADDRESS OF FIRST REQUEST BLOCK	
ADDRESS OF EIGHTH REQUEST BLOCK	

Figure C-7. Format of Wait List

MESSAGE GROUP REQUEST BLOCKS

Tables C-10, C-11, and C-12, respectively, show the content of the following message group request blocks:

- Message group control request block (MGCRB)
- Message group initialization request block (MGIRB)
- Message group recovery request block (MGRRB).

Templates for these request blocks are generated by the $MGCRT, $MGIRT, and $MGRRT macro calls, respectively.

The request blocks can be generated by the $MGCRB, $MGIRB, and $MGRRB macro calls, respectively.

Message group request blocks are used by the message facility for sending requests between task groups or tasks.

Table C-10. Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
0	MC_OS	0-31	Pointer; reserved for system use.
$AF	MC_MAJ		Major status.
		0-7	Reserved for system use.
		8(T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
$AF	MC_MAJ	9(W)	Wait bit. Set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it. Display processing uses this bit during a write.
		B(S)	Release semaphore indicator. Values: 0 = No release; 1 = Release (on close-out) of semaphore, which must be in MC_OS -1.
		C(P)	Must be set by user if MGCRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGCRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return request block indicator. Values: 0 = No dispatch; 1 = Dispatch request block whose address must be contained in MC_OS -$AF, after closeout of this request.
		E(D)	Delete request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where MGCRB is the first entry of its memory block.
		F	I/O bit. Must be set.

Table C-10 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
1+$AF	MC_OPT		General options:
		0-7	Reserved for system use.
		8	Must be 0.
		9	Byte index. 0 = Buffer begins in leftmost byte of the word; 1 = Buffer begins in rightmost byte.
		A	Must be 0.
		B	Must be 1 (extended MGCRB).
		C-F	Must be 0.
2+$AF	MC_BUF	0-31	Buffer pointer.
2+2*$AF	MC_BSZ	0-F	Buffer range (in bytes).
3+2*$AF	MC_DVS		Record-type code.
	MC_REC	0-F	On send, insert record-type code; on receive, return assigned record-type code.
4+2*$AF	MC_RSR	0-F	Residual range (in bytes).
5+2*$AF	MC_MRU	0-7	End message recovery unit (MRU). Reserved for system use.
	MC_WTI	8-F	Wait test indicator. 00 = Return null value to application. 01 = Wait.
6+2*$AF	MC_EXT		Extension mechanism.
		0-7	Binary value of 13+2*$AF, i.e., number of words in MGCRB following the extension word.
		8-F	Must be hexadecimal '7'.
7+2*$AF	Next 7 words		Reserved for system physical I/O use.

Table C-10 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
14+2*$AF	MC_FNC	0-7	Function. Reserved for system use.
	MC_REV	8-F	Revision. Must be hexadecimal '2'.
15+2*$AF	MC_MGI		Message group id.
		0-F	Returned in the $MINIT and $MACPT macro calls.
16+2*$AF	MC_LVL		Enclosure level.
	MC_LVR	0-7	Enclosure level requested.
	MC_LVD	8-F	Enclosure level detected according to the following ASCII values: 0 = Not end of record; 1 = End of record; 2 = End of quarantine unit; 5 = End of message.
17+2*$AF	MC_PCI	0-F	Must be 0.
18+2*$AF	MC_VDP	0-31	Must be zero.
18+3*$AF	MC_TGI	0-F	Reserved for system use.
19+3*$AF	MC_TSK	0-31	Pointer. Reserved for system use.
19+4*$AF	MC_NPI	0-F	Must be 0.
22+3*$AF	MC_LEN	0-F	Length of text received.

Table C-11. Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
0	MI_OS	0-31	Pointer. Reserved for system use.
$AF	MI_MAJ		Major status.
		0-7	Reserved for system use.
		8(T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
		9(W)	Wait bit. Set if the requesting task is not suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. Values: 0 = No release; 1 = Release, on termination of this request, semaphore whose name must be in MI_OS -1.
		C(P)	Must be set by user if MGIRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGIRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return request block indicator. Values: 0 = No dispatch. 1 = Dispatch,, after termination of this request, request block whose address must be contained in MI_OS -$AF.

Table C-11 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
		E(D)	Delete I/O request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where this MGIRB is the first entry of its memory block.
		F	I/O bit. Must be set.
1+$AF	MI_OPT		General options.
		0-7	Reserved for system use.
		8-A	Must be 0.
		B	Must be 1 (extended MGIRB).
		C-F	Must be 0.
2+$AF	MI_BUF	0-31	Must be zero.
2+2*$AF	MI_BSZ	0-F	Buffer range (in bytes). Must be 0.
3+2*$AF	MI_MPD	0-F	Message path description identifier. Must be hexadecimal '01'.
4+2*$AF	MI_RSR	0-F	Residual range (in bytes).
5+2*$AF	MI_MDE	0-7	Must be 0.
	MI_IOP	8-F	Must be 0.
6+2*$AF	MI_EXT		Extension mechanism.
		0-7	Binary value of 31+2*$AF, i.e., number of words in MGIRB following the extension word.
		8-F	Must be hexadecimal '7'.
7+2*$AF	MI_DV2 (three words)	0-F 0-F 0-F	Maturity date/time in standard internal date/time format (see $INDTM).
14+2*$AF	MI_FNC	0-7	Function. Reserved for system use.
	MI_REV	8-F	Revision. Must be hexadecimal '2'.

Table C-11 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
15+2*$AF	MI_MGI		Message group id.
		0-F	Returned in the $MINIT and $MACPT macro calls.
16+2*$AF	MI_PCM (Two words)	0-F 0-F	Must be 0. Must be 0.
18+2*$AF	MI_ADT		Address type.
		0-7	Address type (initiator); must be hexadecimal '1'.
		8-F	Address type (acceptor); must be hexadecimal '1'.
19+2*$AF	MI_NWI	0-F	Must be 0.
20+2*$AF	MI_NDI	0-F	Must be 0.
21+2*$AF	MI_MBI		Initiator mailbox name.
	(Six words)	0-F 0-F 0-F 0-F 0-F 0-F	Must be from 1 to 12 ASCII characters, blank-filled, left justified as specified when the mailbox was created, indicating that only messages with this identifier will be accepted; or six words of zeros, indicating that messages with any identifier will be accepted.
27+2*$AF	MI_NWA	0-F	Must be 0.
28+2*$AF	MI_NDA	0-F	Must be 0.
29+2*$AF	MI_MBA		Acceptor mailbox name.
	(Six words)	0-F 0-F 0-F 0-F 0-F 0-F	Must be from 1 to 12 ASCII characters specifying the acceptor mailbox id, blank-filled, left-justified.

Table C-11 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
36+2*$AF	MI_CNT	0-F	Count of number of active messages in the mailbox. Returned with $MCMG macro call.
37+2*$AF	MI_TGI	0-F	Reserved for system use.
38+2*$AF	MI_TSK	0-31	Pointer. Reserved for system use.
38+3*$AF	MI_SIP	0-31	Security information pointer. Points to the security information block (SIB) that points to the logical submittor block containing the user id (SI_PER), the account id (SI_ACC), and the mode (SI_MOD).

Table C-12. Message Group Recovery Request Block (MGRRB)

Item	Label	Bit(s)	Contents
0	MR_OS	0-31	Pointer. Reserved for system use.
$AF	MR_MAJ		Major status.
		0-7	Reserved for system use.
		8(T)	This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.
		9(W)	Wait bit. Set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. Values: 0 = No release; 1 = Release, on closeout, of semaphore which must be in MC_OS -1.
		C(P)	Must be set by user if MGRRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGRRB can be referenced only by $WAIT or $WAITA issued by the requesting task.
		D(R)	Return request block indicator. Values: 0 = No dispatch; 1 = Dispatch request block, whose address must be in MC_OS -$AF, after closeout of this request.

Table C-12 (cont). Message Group Recovery Request Block (MGRRB)

Item	Label	Bit(s)	Contents
$AF (cont)	MR_MAJ (cont)	E(D)	Delete I/O request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where MGRRB is the first entry of its memory block.
		F	I/O bit. Must be set.
1+$AF	MR_OPT		General options.
		0-7	Reserved for system use.
		8-A	Must be 0.
		B	Must be 1 (extended MGRRB).
		C-F	Must be 0.
2+$AF	MR_BUF	0-31	Pointer. Must be 0.
2+2*$AF	MR_BSZ	0-F	Buffer range. Must be 0.
3+2*$AF	MR_ITP	0-F	Must be 0.
4+2*$AF	MR_RES		Residual range.
		0-F	Reserved for system use.
5+2*$AF	MR_RSN	0-7	Reason-for-terminate code. 0 = Normal message group termination; 22-26 = User-defined abnormal termination of message group.
		8-F	Reserved for system use.
14+2*$AF	MR_FNC	0-7	Function. Reserved for system use.
	MR_REV	8-F	Revision. Must be hexadecimal '02'.
15+2*$AF	MR_MGI	0-F	Message group id. Returned in the $MINIT and $MACPT macro calls.

Table C-12 (cont). Message Group Recovery Request Block (MGRRB)

Item	Label	Bit(s)	Contents
17+2*$AF	MR_CNC	0-F	Reserved for system use.
16+2*$AF	MR_FMT	0-31	Pointer. Must be 0.
18+3*$AF	MR_MRU (Two words)	0-F 0-F	Reserved for system use. Reserved for system use.
19+3*$AF	MR_AMU (Two words)	0-F 0-F	Reserved for system use. Reserved for system use.

INDEX

MACRO CALL ACRONYMS (CONT)

MACRO CALL ACRONYMS (CONT)

Honeywell

Honeywell Information Systems