SERIES 60 (LEVEL 6)

GCOS 6 SYSTEM SERVICE MACRO CALLS ADDENDUM A

SUBJECT

Description of and User Procedures for System Service Macro Calls, Device Drivers, and Data Structures

SPECIAL INSTRUCTIONS

Insert the attached pages into the manual according to the Collating Instructions on the back of this cover. Change bars in the margins indicate new or changed information and asterisks denote deletions. Note that the Error Logging macro calls (pages 5-147 through 5-165) have been deleted from this manual. The Release Directory and Release File macro calls (pages 5-366 through 5-371) have been deleted and renamed Delete Directory and Delete File, and reinserted in alphabetical sequence.

Note:

Insert this cover behind the manual cover to indicate that the manual has been updated with this Addendum.

SOFTWARE SUPPORTED

This manual supports the Series 60 (Level 6) GCOS 6 MOD 600 (Release 0110 Executive). See the manual directory of the latest *MOD 600 System Concepts* (Order No. CB50) manual for information as to later releases supported by this manual.

PREREQUISITE INFORMATION

MOD 600 System Concepts manual and *GCOS 6 Assembly Language Reference* manual.

ORDER NUMBER

CB08-02A

July 1979

24257
3779
Printed in U.S.A.

Honeywell

COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

Remove	Insert
Title, Preface	Title, Preface
iii, iv	iii, iv
vii through x	vii through x
1-7 through 1-10	1-7 through 1-10
1-13 through 1-18	1-13 through 1-18
1-21, 1-22	1-21, 1-22
2-5 through 2-8	2-5 through 2-8
2-11, 2-12	2-11, 2-12
3-17, 3-18	3-17, 3-18
4-1, 4-2	4-1, 4-2
5-11, 5-12	5-11, 5-12
5-19, 5-22	5-19, 5-22
5-33, 5-34	5-33, 5-34
5-43, 5-44	5-43, 5-44
5-55 through 5-58	5-55 through 5-58
5-67 through 5-74	5-67 through 5-74
5-77, 5-78	5-77, 5-78
5-83 through 5-86	5-83 through 5-86
5-91, 5-92	5-91, 5-92
5-103 through 5-108	5-103 through 5-108
5-109 through 5-112	5-109 through 5-112
5-115 through 5-118	5-115 through 5-118
5-121, 5-122	5-121 through 5-122.6
5-127, 5-128	5-127, 5-128
5-131, 5-132	5-131, 5-132
5-143 through 5-166	5-143 through 5-147, blank
	5-166, blank
5-169, 5-170	5-169, 5-170
5-187, 5-188	5-187, 5-188
5-213, 5-214	5-213, 5-214
5-221, 5-222	5-221, 5-222
5-225, 5-226	5-225, 5-226
5-229, 5-230	5-229, 5-230
5-241, 5-242	5-241, 5-242
5-251 through 5-254	5-251 through 5-254
5-257, 5-258	5-257, 5-258
5-267, 5-268	5-267, 5-268
5-275 through 5-278	5-275 through 5-278
5-283 through 5-288	5-283 through 5-286
5-297, 5-298	5-297, 5-298
5-307, 5-308	5-307 through 5-308
5-317, 5-318	5-317, 5-318.1
5-327, 5-328	5-327, 5-328
5-337, 5-338	5-337, 5-338
5-347, 5-348	5-347, 5-348
5-359 through 5-372	5-359 through 5-372
	blank, 5-372
5-377 through 5-380	5-377 through 5-380
5-393, 5-394	5-393, 5-394
5-423 through 5-430	5-423 through 5-430
5-445, 5-446	5-445, 5-446
5-449, 5-450	5-449, 5-450
5-459, 5-460	5-459, 5-460

File No.: 1S13

7/79

COLLATING INSTRUCTIONS (cont)

Remove	Insert
5-465 through 5-468	5-465 through 5-468
5-471, 5-472	5-471, 5-472
5-487 through 5-494	5-487 through 5-494
5-509 through 5-512	5-509 through 5-512
5-521, 5-522	5-521, 5-522
5-525, blank	5-525, blank
6-1 through 6-14	6-1 through 6-14
6-33 through 6-36	6-33 through 6-36
6-39 through 6-42	6-39 through 6-42
A-3, A-4	A-3, A-4
A-7 through A-14	A-7 through A-14
A-17 through A-24	A-17 through A-24
B-5, B-6	B-5, B-6
B-9 through B-14	B-9 through B-14

SERIES 60 (LEVEL 6)
GCOS 6 SYSTEM SERVICE MACRO CALLS

SUBJECT

Description of and User Procedures for System Service Macro Calls, Device Drivers, and Data Structures

SPECIAL INSTRUCTIONS

This revision supersedes Revision 1 of the manual dated July 1978. Appendix B was deleted and succeeding appendixes renumbered. Except for revised Section 6, new Section 8, and revised Appendixes A and C, change bars indicate new or changed information, asterisks denote deletions. Major additions are macro calls for MOD 600 software.

SOFTWARE SUPPORTED

This manual supports the Series 60 (Level 6) GCOS 6 MOD 400 (Release 0120) and MOD 600 (Release 0100) Operating Systems. See the manual directory of the latest *MOD 400 System Concepts* (Order No. CB20) or *MOD 600 System Concepts* (Order No. CB50) manual for information as to later releases supported by this manual.

PREREQUISITE INFORMATION

MOD 400 System Concepts manual or *MOD 600 System Concepts* manual and *GCOS 6 Assembly Language Reference* manual.

ORDER NUMBER

CB08, Rev. 2

December 1978

Honeywell

PREFACE

This manual is for assembly language programmers who use the GCOS system service macro routines and macro calls in writing application programs. The manual describes the macro calls for monitor services, for using the file system, and for generating data structures.

The manual also discusses Honeywell peripheral device drivers. *

Section 1 concerns macro call syntax, register conventions, and addressing conventions.

Sections 2, 3, and 4 briefly summarize and list macro calls for monitor services, for the file system, and for defining data structures, respectively.

Section 5 describes in detail the use, structure, function, and error return codes for each macro routine and macro call, some with examples. These descriptions are arranged alphabetically by function description name, according to the function description shown in column 2 of Table 1-1.

Section 6 describes the GCOS 6 Honeywell device drivers for data transfer in system and applications programs with Level 6 peripheral devices.

Sections 7 and 8 discuss trap handling for hardware and software traps under MOD 400 and MOD 600, respectively.

Appendix A describes various block data structures that are related to certain macro routines. Appendix B summarizes register contents before and after execution of the system service macro calls. Appendix C shows the ASCII and EBCDIC character sets.

MANUAL DIRECTORY

The following publications constitute the GCOS 6 manual set. See the "Manual Directory" of the appropriate System Concepts manual for the current revision number and addenda (if any) of the relevant operating-system-specific publications.

Base Publication	Manual Title
CB01	GCOS 6 Program Preparation
CB02	GCOS 6 Commands
CB03	GCOS 6 Communications Processing
CB04	GCOS 6 Sort/Merge
CB05	GCOS 6 Data File Organizations and Formats
CB06	GCOS 6 System Messages
CB07	GCOS 6 Assembly Language Reference
CB08	GCOS 6 System Service Macro Calls
CB09	GCOS 6 RPG Reference
CB10	GCOS 6 Intermediate COBOL Reference
CB12	GCOS 6 Entry-Level COBOL Reference
CB13	GCOS 6 FORTRAN Reference
CB14	GCOS 6 Advanced COBOL Reference
CB15	GCOS 6 Advanced COBOL Reference Guide
CB16	GCOS 6 I-D-S/II Reference Card
CB20	GCOS 6 MOD 400 System Concepts
CB21	GCOS 6 MOD 400 Program Execution and Checkout
CB22	GCOS 6 MOD 400 Programmer's Guide
CB23	GCOS 6 MOD 400 System Building
CB24	GCOS 6 MOD 400 Operator's Guide
CB27	GCOS 6 MOD 400 Programmer's Pocket Guide
CB28	GCOS 6 MOD 400 Master Index
CB30	Remote Batch Facility User's Guide
CB31	Data Entry Facility User's Guide
CB32	Data Entry Facility Operator's Quick Reference Guide
CB33	Level 6/Level 6 File Transmission Facility User's Guide
CB34	Level 6/Level 62 File Transmission Facility User's Guide
CB35	Level 6/Level 64 (Native) File Transmission Facility User's Guide
CB36	Level 6/Level 66 File Transmission Facility User's Guide Guide

Base Publication	Manual Title
CB37	Level 6/Series 200/2000 File Transmission Facility User's Guide
CB38	Level 6/BSC 2780/3780 File Transmission Facility User's Guide
CB39	Level 6/Level 64 (Emulator) File Transmission Facility User's Guide
CB40	2780/3780 Workstation Facility User's Guide
CB41	HASP Workstation Facility User's Guide
CB42	Level 66 Host Resident Facility User's Guide
CB43	Terminal Concentration Facility User's Guide
CB44	Interactive Function User's Guide
CB48	3270 Interactive Facility User's Guide
CB50	GCOS 6 MOD 600 System Concepts
CB51	GCOS 6 MOD 600 Program Execution and Checkout
CB52	GCOS 6 MOD 600 Programmer's Guide
CB53	GCOS 6 MOD 600 System Building
CB54	GCOS 6 MOD 600 Administrator's Guide
CB55	GCOS 6 MOD 600 Transaction Driven System
CB56	I-D-S/II Data Base Administrator's Guide
CB57	I-D-S/II Data Base User's Guide
CB58	GCOS 6 MOD 600 Operator's Guide
CD46	Display formatting and Control
CF11	RBF/64 User's Guide
CG65	GCOS 6 MOD 600 Operator's Pocket Guide
CG66	GCOS 6 MOD 600 Programmer's Pocket Guide
CG71	GCOS 6 MOD 600 System Building Memory Calculator
CG72	GCOS 6 MOD 600 Software and Documentation Directory

In addition, the following publications provide supplementary information:

Order Number	Manual Title
AT97	Level 6 Communications Handbook
CC71	Level 6 Minicomputer Systems Handbook
FQ41	Writable Control Store User's Guide

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

CONTENTS (cont)

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	0D0A	Task group control
$ABGRQ	Abort group request	0D07	Task group control
$ACTCL[b]	Accounting files, close	1E05	Accounting
$ACTD1[b]	Accounting raw record, delete	1E03	Accounting
$ACTD2[b]	Accounting update record, delete	1E13	Accounting
$ACTID	Account identification	1402	Identification and information
$ACTOP[b]	Accounting files, open	1E04	Accounting
$ACTR1[b]	Accounting raw record, read	1E01	Accounting
$ACTR2[b]	Accounting update record, read	1E11	Accounting
$ACTVG	Activate group	0D09	Task group control
$ACTVT[b]	Activate task	0C10	Task control
$ACTW1[b]	Accounting raw record, write	1E02	Accounting
$ACTW2[b]	Accounting update record, write	1E12	Accounting
$ACUPD[b]	Accounting field, update	1E06	Accounting
$ASFIL	Associate file	1010	File management

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$BUAT[b]	Bound unit, attach	0C09	Task control
\$BUDT[b]	Bound unit, detach	0C0B	Task control
\$BUID	Bound unit identification	1406	Identification and information
\$BULD[b]	Bound unit, load	0C0A	Task control
\$BYE[b]	Terminate user session	0D0B-0D0D	Task group control
\$CANRQ	Cancel request	0C01	Task control
\$CIN	Command in (read command-in file)	0802	Standard system file I/O
\$CLFIL	Close file	1055-1057	File management
\$CLPNT	Clean point	0C13	Task control
\$CLRSW	Clear external switches	0B02	External switch
\$CMDLN	Command line, process	0C08	Task control
\$CMSUP	Console message suppression	0902/0903	Operator interface
\$CNCRQ	Cancel clock request	0501	Clock
\$CNSRQ[a]	Cancel semaphore request	0601	Semaphore handling
\$CRB	Clock request block	-	Data structure generation
\$CRBD	Clock request block offsets	-	Data structure generation
\$CRDIR	Create directory	10A0	File management

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$CRFIL	Create file	1030	File management
$CRGRP	Create group	0D03	Task group control
$CRKDB	Create file, key descriptor block offsets	-	Data structure generation
$CRMSG[b]	Create maximum segment	0C0E	Task control
$CROAT	Create overlay area table	070A	Overlay handling
$CRPSB	Create file parameter structure block - offsets	-	Data structure generation
$CRSEG[b]	Create segment	0C0C	Task control
$CRTSK	Create task	0C02/0C03	Task control
$CSGRP[b,c]	Create system group	0D02	Task group control
$CUSID[b,c]	Change user identification	0D0E	Task group control
$CWDIR	Change working directory	10B0	File management
$DFSM	Define semaphore	0604	Semaphore handling
$DLDIR	Delete directory	10A5	File management
$DLFIL	Delete file	1035	File management
$DLGRP	Delete group	0D04	Task group control
$DLREC	Delete record	1130/1131	Data management

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Code (4)
$DLOAT	Delete overlay area table	070D	Overlay handling
$DLSEG[b]	Delete segment	0C0D	Task control
$DLSM	Delete semaphore	0607	Semaphore handling
$DLTSK	Delete task	0C04	Task control
$DSDV	Disable device	0202	Physical I/O
$DSFIL	Dissociate file	1015	File management
$DSTRP	Disable user trap	0A02	Trap handling
* $ENDV	Enable device	0204	Physical I/O
$ENTID	Entry point identification	1407	Identification and information
$ENTRP	Enable user trap	0A01	Trap handling
$EROUT	Error output file - write to	0803	Standard system file I/O
$EXTDT	External date/time - convert to	0504	Date/time
$EXTIM	External time - convert to	0505	Date/time
$FIB	File information block - create or change	-	Data structure generation

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$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
$MINIT	Message group, initiate	1502	Intergroup message facility
$MODID	Mode identification	1403	Identification and information
$MRECV	Message group, receive	1503	Intergroup message facility
$MSEND	Message group, send	1505	Intergroup message facility
$MTMG	Message group, terminate	1504	Intergroup message facility
$NCIN	New command-in	0806	Standard system file I/O
$NPROC	New process	0D0B	Task group control

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$NUIN	New user input file -	0804	Standard system file I/O
$NUOUT	New user output file - redefine	0805	Standard system file I/O
$OPFIL	Open file	1050/1051	File management
$OPMSG	Operator information message - display only	0900	Operator interface
$OPRSP	Operator response message - display/respond	0901	Operator interface
$OVEXC	Overlay, execute	0700	Overlay handling
$OVLD	Overlay, load	0701	Overlay handling
$OVRCL	Overlay release, wait, and recall	0707	Overlay handling
$OVRLS	Overlay area, release	0706	Overlay handling
$OVRSV	Overlay area reserve, and execute overlay	0705	Overlay handling
$OVST	Overlay status	0703	Overlay handling
$OVUN	Overlay, unload	070C	Overlay handling
$PERID	Person identification	1401	Identification and information
$PRBLK	Parameter block	-	Data structure generation
$RBADD	Return request block address	0107	Request and return

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$RDBLK	Read block	1200-1204	Storage management
$RDREC	Read record	1110-1116	Data management
$RDSW	Read external switches	0B00	External switch
$RDVAT	Reset device attention	0203	Physical I/O
$RETRN	Return sequence - establish	-	Request and return
$RLSM	Release semaphore	0603	Semaphore handling
$RLTML	Release terminal	1704	Terminal Function
$RMEM	Return memory; return	0404/0405	Memory allocation
$RMFIL	Remove file	1025	File management
$RNFIL	Rename file/directory	1040	File management
$ROLBK	Roll back	0C14	Task control
$RPTER	Report error condition	0F00/0F01	Error handling
$RQBAT	Request batch execution	0E00	Batch
$RQCL	Request clock	0500	Clock
$RQGRP	Request group	0D00	Task group control

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$RQIO	Request I/O transfer	0200	Physical I/O
$RQSM[a]	Request semaphore	0600	Semaphore handling
$RQTML	Request terminal	1703	Terminal function
$RQTSK	Request task	0C00	Task control
$RSVSM	Reserve semaphore	0602	Semaphore handling
$RWREC	Rewrite record	1140/1141	Data management
$SDL	Set dial	1B00	Communications
$SETSW	Set external switches	0B01	External switch
$SGTRP[b]	Signal trap	0A03	Trap handling
$SPGRP[a]	Spawn group	0D05	Task group control
$SPTSK	Spawn task	0C05/0C06	Task control
$SRB	Semaphore request block	-	Data structure generation
$SRBD	Semaphore request block offsets	-	Data structure generation
$STMP[a]	Status memory pool	0406	Memory allocation
$STTY	Set terminal file characterstics	1045	File management
$SUSPG	Suspend group	0D08	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	0502/0503	Clock
$SUSPT[b]	Suspend task	0C0F	Task control
$SWFIL	Swap file	1054	File management
$SYSAT	System attribute information	1411	Identification and information
$SYSID	System identification	1404	Identification and information
$TCPUR[b]	Task CPU time remaining	050B	Date/time
$TEST	Test completion status	0102	Request and return
$TFIB	File information block - offsets	-	Data structure generation
$TGIN	Task group input	140C	Identification and information
$TIFIL	Test file for input	1062	File management
$TINFO[b]	Task information	1409	Identification and information
$TOFIL	Test file for output	1063	File management
$TRB	Task request block	-	Data structure generation
$TRBD	Task request block offsets	-	Data structure generation

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
$TRMRQ	Terminate request	0103/0104	Request and return
$TRPHD	Trap handler connect	0A00	Trap handling
$TRPHD[b]	Trap handler, query	0A04	Trap handling
$UNSBU	Unload sharable bound unit	070E	Overlay handling
$USIN	User input file - read	0800	Standard system file I/O
$USMSG[b]	User message	1700	Intergroup task message
$USOUT	User output file - write	0801	Standard system file I/O
$USRID	User identification	1400	Identification and information
$USRSP[b]	User response message	1701	Intergroup task message
$WAIT	Wait for operation complete	0100	Request and return
$WAITL	Wait on request list	0101	Request and return
$WIFIL	Wait for file input	1064	File management
$WLIST	Wait list structure	-	Data structure generation
$WOFIL	Wait for file output	1065	File management
$WRBLK	Write block	1210/1211	Storage management
$WRREC	Write record	1120/1126	Data management

TCBs representing task code are assigned to execute on physical priority level of the central processor. One or more TCBs may be assigned to use a level, and will be queued awaiting availability if there is a request for the task. When the TCB heading the level queue terminates with an empty request queue or is temporarily suspended by the system, the next TCB on that level moves to the head of the level queue. The system may suspend a task while processing a system service call, e.g., fetching a system overlay; the task may also explicitly suspend by performing a wait or suspend operation. When a suspended task reactivates, its TCB is placed at the end of the appropriate level queue.

The following sequence of events illustrates an example of request queue manipulation as one task (e.g., task A, identified as logical resource number 1 at priority level 7) requests the execution of another task (e.g., task B, identified as logical resource number 2 at priority level 10, a lower priority level) and later waits for completion of the requested task.

1. Task A requests task B (specifying logical resource number 2 in the request block). The task manager places this request block at the end of the request queue for Task B which executes at priority level 10. See Diagram 1.

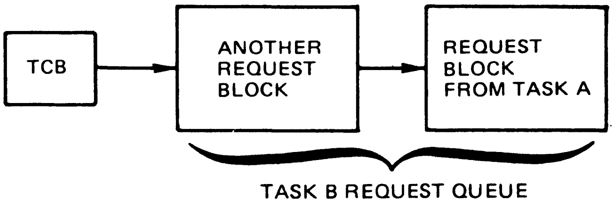


Diagram 1 - Request Block From Task A is Queued in Request Queue for Task B

2. Task A issues a wait call, indicating that it wishes to be suspended until its request for Task B is completed. Task A is now suspended.

3. Task B runs and terminates relative to the first request block in the request queue for the task. As Task B terminates, the first request block is removed from the request queue for the task. See Diagram 2. The TCB for Task B on priority level 10 remains active because another request block (the one generated by Task A) exists in its request queue.

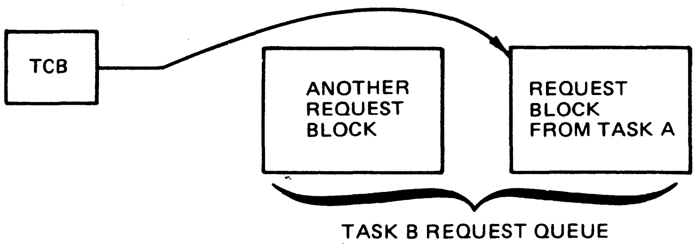


Diagram 2 - First Request Block is Dequeued as Task B Terminates Relative to It

4. Task B runs and terminates relative to the request block generated by Task A. Task A, which was waiting for this event, is now reactivated. The request block generated by Task A is removed from the request queue for priority level 10. Task A will resume execution when priority level 7 becomes the highest active level, and the Task A TCB again reaches the beginning of the level 7 TCB queue.

Function	Macro Call
Home directory pathname	$HDIR
Bound unit identification	$BUID
System identification	$SYSID
Task group account identification	$ACTID
Task group input file name	$TGIN
Task group mode identification	$MODID
Task group person identification	$PERID
Task group user identification	$USRID
Entry point identification	$ENTID
Group information	$GINFO
Group identification	$GRPID
Group status information	$GRPST
Installation identification	$INSID
Task information	$TINFO
System attribute information, get	$SYSAT

MEMORY ALLOCATION FUNCTIONS

The macro calls for memory allocation functions allow you to dynamically obtain memory from the task group's memory pool, to return this memory when it is no longer needed, and ascertain the amount of memory available in a specified pool.

The macro call that allocates a memory block has two forms: one form allows you to obtain a memory block of the specified size only; the other allows you to obtain the largest existing contiguous memory block if a block of the specified size cannot be found. The macro call that returns a memory block also has two forms: one form allows you to return an entire memory block; the other allows you to return a specified part of the block.

The macro routines/calls are:

Get memory; get available memory	$GMEM
Return memory; return partial block of memory	$RMEM
Status memory pool	$STMP

MESSAGE FACILITY FUNCTIONS

The message facility allows two task groups, using assembly language code, to have online communication between them by sending a message (one or more records) through message queues called mailboxes. A message group is a set of records that constitute a message sent through a mailbox.

The message facility macro calls are issued by the task groups to perform message group and message functions. (The MOD 400 System Concepts manual describes the message facility.)

The intergroup message facility macro calls have the following functions:

- o Open the send function of the message facility (accept)
- o Ascertain number of messages in the mailbox
- o Open the receive function of the message facility (initiate)
- o Terminate the message group
- o Receive the data
- o Send the message data

The message facility macro calls are:

Message group, accept	$MACPT
Message group, count	$MCMG
Message group, initiate	$MINIT
Message group, terminate	$MTMG
Message group, receive	$MRECV
Message group, send	$MSEND

INTERGROUP TASK MESSAGE FUNCTIONS (MOD 600 ONLY)

The macro calls for intergroup task message functions permit communications between tasks in different task groups by:

- o Sending a message between tasks in separate task groups
- o Sending a message from a task in one group to a task in another, and receiving a response

The macro calls use an intergroup request block, discussed in Section 4 and detailed in Appendix A.

The macros are:

User message	$USMSG
User response message	$USRSP

OPERATOR INTERFACE FUNCTIONS

The macro calls for operator interface functions enable tasks to communicate with the operator terminal by:

- o Displaying an information message on the operator terminal

- o Sending a message to the operator terminal and receiving a response

- o Activating or deactivating console suppression, i.e., suspend or restore issuance of messages to the operator terminal for the issuing task group

The macro routines/calls are:

Console message suppression	$CMSUP
Operator information message	$OPMSG
Operator response message	$OPRSP

THE $OPMSG and $OPRSP macro calls require input/output request blocks (IORB's), which can be generated by the $IORB macro call (see Sections 4 and 5 and Appendix A).

OVERLAY HANDLING FUNCTIONS

Overlays may be loaded at a fixed displacement from the base of the root-segment at link time, or if "floatable," into a block of memory allocated explicitly by the user or implicitly by the system.

The user may create a set of overlay areas and have the system load floatable overlays into them, managing the availability of free areas, and locating available copies of requested overlays. The user may unload from memory, at one time, all sharable bound units with a user count of zero.

The macro routines/calls are:

Overlay, release, wait, and recall	$OVRCL
Overlay area, release	$OVRLS
Overlay area reserve, and execute overlay	$OVRSV
Create overlay table	$CROAT
Delete overlay area table	$DLOAT
Overlay, execute	$OVEXC
Overlay, load	$OVLD
Overlay status	$OVST
Overlay, unload	$OVUN

*

PHYSICAL I/O FUNCTIONS

The macro calls described in this subsection allow you to interact with device drivers. If direct access to devices is not a requirement, use the File System macro calls.

The physical I/O macro calls allow you to:

- Request input and output
- Disable a device when an attention interrupt occurs
- Set the resource control table (RCT) of a device to the enable status
- Turn off the attention status indicator in the RCT of the specified device

See Section 6 for a complete description of Level 6 physical I/O functions, including details on device drivers and resource control tables.

The macro routines/calls for physical I/O are:

Disable device on attention	$DSDV
Enable device	$ENDV
Reset device attention	$RDVAT
Request I/O transfer	$RQIO

* REQUEST AND RETURN FUNCTIONS

The macro calls for request and return functions enable you to control requests for tasks and to provide a standard return sequence for called subroutines. Specifically, the macro routines are used to:

- Terminate the current execution of a task
- Wait for the completion of another task

4. Task B issues a $RLSM when it finishes with the resource; the counter is incremented to 0, Task C now gets the resource. After the $RLSM for Task C, the value is 1 again.

Use of resources by more than one user at a time can be arranged by adjusting the initial value of the semaphore, e.g., an initial value of 2 allows two users, a value of 4 allows four users, and so on, depending on the nature of the resource and its intended use.

If it is undesirable for a task to be suspended while a resource is in use, the $RQSM macro call can be used instead of $RSVSM to reserve a resource. $RQSM is an asynchronous reservation request ($RSVSM is a synchronous request) which causes a request block to be queued for the resource, so that the issuing task can do other processing before the needed resource is available.

The macro routines/calls for semaphore handling are:

Cancel semaphore request	$CNSRQ
Define semaphore	$DFSM
Release semaphore	$RLSM
Request semaphore	$RQSM
Reserve semaphore	$RSVSM
Delete semaphore	$DLSM

STANDARD SYSTEM FILE I/O FUNCTIONS

The macro calls for standard system file I/O functions make the standard system files (command-in, user-in, user-out, and error-out) available to a task group. Other macro calls shown below allow the task to redefine the user-in and user-out files. Specifically, the macro routines enable you to:

- Read the next record from the command-in file
- Write the next record to the error-out file
- Read the next record from the user-in file
- Write the next record to the user-out file
- Redefine the user-in file
- Redefine the user-out file

The macro routines/calls are:

Command in (read command-in file)	$CIN
Error output file	$EROUT
New command in	$NCIN
New user input file	$NUIN
New user output file	$NUOUT
User input file	$USIN
User output file	$USOUT

TASK CONTROL FUNCTIONS

The macro calls for task control allow you to:

- Cancel a previously issued request
- Create, request, spawn, suspend, activate, delete, and abort a task
- Attach, load, and detach a bound unit to/from a task
- Create and delete a segment for a task's bound unit
- Process command lines
- Roll back (recover) updated records in all files updated since the last execution of clean point.
- Define "clean" point at which updated records are valid and may be recovered; unlock records for all files in the task group

Some macro calls involve the use of request blocks. Sections 4 and 5 discuss and describe macro calls that generate request blocks; Appendix A shows the format of the request blocks.

Macro routines/calls for task control are:

Cancel request	$CANRQ
Clean point	$CLPNT
Command line, process	$CMDLN
Create task	$CRTSK
Delete task	$DLTSK
Request task	$RQTSK
Spawn task	$SPTSK
Bound unit, attach	$BUAT
Bound unit, load	$BULD
Bount unit, detach	$BUDT
Unload sharable bound unit	$UNSBU
Create segment	$CRSEG
Create maximum segment	$CRMSG
Delete segment	$DLSEG
Suspend task	$SUSPT
Activate task	$ACTVT
Kill task	$KILLT
Roll back	$ROLBK

TASK GROUP CONTROL FUNCTIONS

A task group is a named set of one or more tasks, memory space, files, peripheral devices, and priority levels. Any number of task groups may be defined. The macro calls for task control allow you to:

- Create a file
- Delete a file
- Get a file (reserve a file for processing)
- Open a file
- Close a file
- Remove a file from processing
- Rename a file
- Associate a logical file number with a pathname
- Dissociate a logical file number from a pathname
- Create a directory
- Delete a directory
- Rename a directory
- Change the working directory
- Get the name of the current working directory
- Expand pathname (develop a full pathname from a relative pathname)
- Get information about a file
- Test the status of an I/O activity (terminal)
- Wait for the completion of an asynchronous I/O activity (terminal)
- Set the file characteristics of a terminal

Some of the macro calls use file information blocks (FIBs); some can use FIB offsets or parameter structure offsets. The macro calls available to generate FIBs and offsets are summarized in Section 4 and described in detail in Section 5.

The macro routines/calls for file management are:

Associate file	$ASFIL
Change working directory	$CWDIR
Close file	$CLFIL
Create directory	$CRDIR
Create file	$CRFIL

Delete file	$DLFIL
Delete directory	$DLDIR
Dissociate file	$DSFIL
Expand pathname	$XPATH
Get file	$GTFIL
Get file information	$GIFIL
Get working directory	$GWDIR
Open file	$OPFIL
Release directory	$RLDIR
Release file	$RLFIL
Remove file	$RMFIL
Rename file/directory	$RNFIL
Set terminal file characteristics	$STTY
Test file for input	$TIFIL
Test file output	$TOFIL
Swap tape file	$SWFIL
Wait for file input	$WIFIL
Wait for file output	$WOFIL

Section 5 describes these macros in detail.

Many of the macro calls can be logically paired, as follows:

- Open file - Close file
- Create file - Delete file
- Associate file - Dissociate file
- Get file - Remove file
- Create directory - Delete directory

Although the following functions are available through macro calls, they are typically performed outside of program execution via execution control commands.

- Associate file
- Dissociate file
- Get file
- Remove file
- Create file
- Delete file
- Rename file
- Create directory
- Delete directory
- Change working directory
- Get working directory
- Set terminal file characteristics.

Figure 3-1 shows the life cycle of a file. Create file ($CRFIL) and get file ($GTFIL) are actually on the same level. The same is true for delete file ($DLFIL) and remove file ($RMFIL). (Associate file and dissociate file provide a way of supplying a pathname as input to create file and get file.)

SECTION 4

DATA STRUCTURE GENERATION

This section summarizes the macro routines that generate and/or define the system data structures. There are two kinds of data structures, those that apply to the monitor service functions, and those that apply to the file system functions.

The macro calls for data structure generation for both monitor services and for the file system functions, are described in detail in Section 5, in the alphabetic order of their function descriptions (see column 2 of Table 1-1).

NOTE: Macro calls that are usable with only one operating system, (e.g., MOD 400 or MOD 600) are so identified in Table 1-1 and in Section 5.

MONITOR SERVICES DATA STRUCTURES

Monitor service data structures are the following:

- Request blocks
- Parameter block and wait list
- Request block offsets

The macro routines for generating the monitor services data structures, summarized in this subsection and described in Section 5, cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for detailed information about SAF/LAF independent code.

Request Blocks

Request blocks are data structures used by an application to coordinate the processing of events. The request blocks provide a standard system interface that specifies the conditions for execution to proceed. For example, one element in a request block can be set to indicate that a task issuing a request for another task has the option to wait until the second task finishes processing before the issuing task continues its own processing.

Request blocks provide the means of specifying the following options:

- o Wait for requested task completion
- o Explicit start address of requested task
- o Termination action for requested task
- o Deletion of request block upon termination

The wait option allows synchronization of a requesting and a requested task; for example, the issuing task could name a semaphore to be released or it could specify an address of a request block to be scheduled.

The selection of an explicit start address allows a requesting task to control the entry point of the requested task.

Possible termination options of the requested task include release of a semaphore or request of another request block on task termination. These options allow flexible synchronization among tasks of an application and permit the issuing task to terminate before the requested task completes. For example, a slave task that runs asynchronously with the remainder of the application can repetitively reserve a semaphore and be activated only by release of that semaphore as requested at termination of other tasks. The option of scheduling another task request at task termination allows, for example, a dispatching task to be notified of completion of certain tasks without explicitly waiting for their completion.

The request block deletion option causes the system to return the request block to the appropriate pool upon task termination without further application intervention.

Often used in conjunction with the semaphore and/or schedule request options, this is a way for memory to be properly returned even though the issuing task has itself terminated. For example, the system uses this feature on asynchronous task requests such as Spawn Task, with the NWAIT argument.

These options are controlled by the following specific bits in the request blocks, and apply to all types of requests (unless otherwise indicated).

- o W-bit, or wait

- o I-bit, or implicit start address (not optional for IORBs or clock request blocks (CRBs), always set)

- o S-bit, or semaphore

- o R-bit, or return request

- o D-bit, or delete

ACCOUNTING FILES, OPEN (MOD 600 ONLY)

Macro Call Name: $ACTOP

Function Code: 1E/04

Equivalent Command: None

Open the system's accounting files, and make them available to this user.

FORMAT:

```
[label]  $ACTOP
```

ARGUMENT DESCRIPTION:

None

FUNCTION DESCRIPTION:

This call opens the system accounting files to access by an authorized user. (This call must precede all other accounting macro calls, except $ACUPD).

The system automatically activates the accounting function and creates the accounting files when the operator startup EC file includes the MESSAGE OF THE DAY command. Unless this command is removed from the startup EC file, the accounting function will be in the system; accounting data will be accumulated and placed in the raw accounting file.

NOTE: A user who does not utilize the accounting function, when it is present in the system, should, with the CLEAR_ACCT command, or with his own program, periodically delete the automatically generated and accumulated accounting records from the raw accounting file.

Accounting uses these system accounting files:

- Raw accounting file (ACCT.SA.RAW)
- Update accounting file (ACCT.SA.UPDT)
- Hold accounting file (ACCT.SA.HOLD)

The accounting function creates, for every user who logs in, an accounting record for that login period only. When that user logs off, the record is written to the raw accounting file. If the same user again logs in, the system creates another accounting record. (See the accounting record, read from raw accounting file ($ACTR1) macro call for a description of the raw accounting record.)

Accounting information in an accounting record comprises the following:

1. Date and time user logs in
2. Date and time user logs off
3. Central processor time used (in milliseconds)
4. Number of lines printed
5. Number of cards read
6. Number of cards punched
7. Number of physical I/O order issued (inclusive of 4, 5, and 6)
8. Number of times user program was rolled out
9. Number of times overlay area table (OAT) was loaded
10. Number of times OAT was already in memory
11. Number of pages printed
12. Group's base software level
13. Name of terminal
14. Name of last bound unit loaded

Accounting information is numeric binary. The user is responsible for retrieving the data from the raw accounting file, for changing that data into units/dollars to be charged or reported, and for writing that data out to the accounting update file.

A possible user accounting program sequence might be the following:

1. Issue $ACTR1 call to read the next record from the raw accounting file. ($ACTOP must be first call issued.)
2. Convert data from raw accounting file, into the update record.

```
STORE    STH    $R6,$B3.-$R3   STORE 1 BYTE, STARTING FROM RIGHT OF
                               A 3-BYTE GROUPING; BRANCH TO GET NEXT
         BDEC   $R4,>NXTCHR    UNTIL $R4 = -1

         ADV    $R3,3

         BDEC   $R5,>NXTWRD    BRANCH TO GET NEXT WORD UNTIL $R5 =
                               -1

         JMP    $B5            RETURN TO CALLER

ST_BLK   LDV    $R6,'△'

         B      >STORE
```

ACCOUNTING RAW RECORD, WRITE (MOD 600)

ACCOUNTING RAW RECORD, WRITE (MOD 600 ONLY)

Macro Call Name: $ACTW1

Function Code: 1E/02

Equivalent Command: Write Accounting Record (WRITE_ACCT_R)

Write a raw record to the raw accounting file.

FORMAT:

```
[label]  $ACTW1  [location of the record to be written out],
                 [location of the record length (in bytes)]
                 [location of record type number]
```

ARGUMENT DESCRIPTION:

location of record to be written out

Any address form valid for a data register; provides the address of the raw record to be written to the raw accounting file.

location of length of record to be written out

Any address form valid for a data register; provides the address of the length of the record to be written out. Length is in bytes (decimal). Default value is zero.

location of record type number

Location of the record type number (2 to 65,535) of the record to be written, which is loaded into $R7 prior to the macro call. If this argument is not specified, it is assumed to have been loaded into $R7.

FUNCTION DESCRIPTION:

This call should be used only when the user has implemented his own accounting system, e.g., to provide only selected accounting information such as measuring disk use. For a user-designed system that uses the accounting record, the following rules apply:

1. The user must reserve 16 bytes for record header infor-amation. User character data may not exceed 162 bytes.

2. Record length may not exceed 178 (decimal) bytes.

3. When record length is less than 178 bytes, the user must provide the length in $R6, or provide a pointer to the location where the length value is stored.

This call writes a user-designed accounting record, which must conform to the above rules, into the raw accounting file.

NOTES: 1. This call must be preceded by successful execution of the $ACTOP macro call.

2. The address of the record to be written out, supplied by argument 1, is placed in $B4. When this argument is omitted, the system assumes that $B4 contains this address.

3. The length of the record to be written out, supplied by argument 2, is placed in $R6. When this argument is omitted, the system sets $R6 to zero (the default record length).

4. On return, $R1 contains one of:

0000 - No error

0824 - Request canceled; accounting files were not opened.

ACCOUNTING UPDATE RECORD, DELETE (MOD 600)

ACCOUNTING UPDATE RECORD, DELETE (MOD 600 ONLY)

Macro Call Name: $ACTD2

Function Code: 1E/13

Equivalent Command: None

Delete the current update record from the accounting update file.

FORMAT:

```
[label]  $ACTD2
```

ARGUMENT DESCRIPTION:

None

FUNCTION DESCRIPTION:

This call deletes the current update record (the last update record read) from the accounting update file, and is effective only when a successful $ACTR2 macro call was the last operation against that file.

The record to be deleted is the last update record read into the accounting update file. A deleted record can no longer be read by any other user, but any update record in the user's receiving area (see the $ACTR2 macro call) is not affected by the $ACTD2 macro call and remains available to the user until a subsequent record is read into the receiving area.

This macro call should be used only when an update record causes an error condition to be reported by the system administrator command UPDT_ACCT, which normally deletes update records after it has successfully processed them. When the update record causes an error, the system leaves that record in the update file for user correction.

ASSOCIATE FILE

Macro Call Name: $ASFIL

Function Code: 10/10

Equivalent Command: Associate Path (ASSOC)

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

FORMAT:

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

ARGUMENT DESCRIPTION:

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 in the range 0 through 255.

pathname pointer

A 4-byte address, which 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.

FUNCTION 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 or not the pathname identifies an existing file or directory (i.e., the pathname entry may identify an incomplete pathname, such as VOL1 SUBA). 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. It should be noted 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 address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain the address of the argument structure.

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

   0000 - No error

   0201 - Illegal pathname

   0202 - Pathname not specified

   0205 - Illegal argument

   0206 - Unknown or illegal LFN

   0210 - LFN already associated

   0222 - Pathname cannot be expanded, no working directory

   0226 - Not enough user memory for buffers or structures

BOUND UNIT IDENTIFICATION

Macro Call Name: $BUID

Function Code: 14/06

Equivalent Command: (MOD 600 only) USER BUID

Returns 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 DESCRIPTION:

location of bound unit id field address

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

FUNCTION 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 address of the receiving bound unit id field supplied by argument 1 is placed in $B4; if this argument is omitted, $B4 is assumed to contain 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 and the $BUID 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
```

CHANGE USER IDENTIFICATION (MOD 600 ONLY)

Macro Call Name: $CUSID

Function Code: 0D/0E

Equivalent Command: None

Change the user identification of the issuing task group to the specified id. *

NOTE: This macro routine is recommended for use only by specialized software system designers.

FORMAT:

[label] $CUSID [location of the identification field]

ARGUMENT DESCRIPTION:

location of the identification field

Any address form valid for a data register. Provides the address of the changed user-id field, which consists of three elements as follows:

```
12-character person_id
12-character account_id
 3-character mode
```

Each element must contain exactly 12, 12, and 3 characters, respectively, filled with trailing blanks if necessary.

FUNCTION DESCRIPTION:

This call changes the calling privileged task group's previous user_id as specified in argument 1. For the call to be executed, the task must be executing in a privileged system group (initiated by a create systems group ($CSGRP) macro call or CREATE DAEMON GROUP (CDG) command). *

NOTES:

1. The address of the new user_id field is placed in $B4. When the argument Ts omitted, the system assumes that $B4 contains the new user_id.

2. On return, $R1 contains the following:

   0000 - No error

   0602 - Memory unavailable

   082E - Argument error; unable to pack identity field

   083A - Use of privileged executive function attempted.

CHANGE WORKING DIRECTORY

Macro Call Name: $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 DESCRIPTION:

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.

FUNCTION 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 address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

   0202 - Pathname not specified

   0205 - Illegal argument

   0209 - Named 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 - Illegal file type (not a directory)

Example:

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

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 satisified.

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.

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 will be posted; has one of the following values:

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

MS indicates milliseconds; TS tenths of seconds; SC seconds; MN minutes; and CT units of clock resolution.

n is an integer value from 1 through 65535; m is an integer value from 1 through 32767.

If this argument is omitted, the CRB is initialized with an interval value of zero milliseconds (MS=0).

FUNCTION DESCRIPTION:

The clock request block (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.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

Example:

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

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

CLOCK REQUEST BLOCK OFFSETS

Macro Call Name: $CRBD

Generated Label Prefixes:

```
                    C_RRB/C_SEM
          CRB label offset 0
                    C_CT1
                    C_CT2
                    C_TM
```

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

DESCRIPTION:

See the clock request block macro call.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

CLOSE FILE

Macro Call Name: $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. You identify the file to be closed by supplying its logical file number.

FORMAT:

```
                                      ┌ ( ,NORMAL ) ┐
[label]   $CLFIL   [fib address]      │ { ,LEAVE  } │
                                      └ ( ,UNLOAD ) ┘
```

ARGUMENT DESCRIPTION:

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.

NORMAL
NOR

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 will be disconnected according to the specifications made at system building time.

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

NORMAL is the default value for this macro call.

$\left\{\begin{array}{l}\text{LEAVE}\\\text{LEV}\end{array}\right\}$

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

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

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

$\left\{\begin{array}{l}\text{UNLOAD}\\\text{UNL}\end{array}\right\}$

For tape-resident files the action is the same as for NORMAL mode, except that after the rewinding, 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).

FUNCTION 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 unloaded, and the logical end-of-file (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 1 (write permitted) and bit 1 set to 0 (read not permitted).

The following information applies only to magnetic tape. The actions performed on closing a tape file are determined by the way the write permit bit (bit 2) in the FIB program view word was set when the file was opened. Note that when a tape volume is opened for storage management access, and both volume and file names are not specified, then no trailer labels nor tape marks are written; that is the user's responsibility.

1. Write permission granted:

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

      If no write operations were performed, the trailer lable group will not be 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 will be positioned at the end of the current trailer label group, unless the tape is being processed at the volume level.

2. No write permission granted:

   a. If the end-of-file tape mark was 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 end-of-file tape mark was not detected, the trailer label group is not processed. When the LEAVE option is specified, the tape will be mispositioned. Opening the next file may result in an "invalid tape file header" condition.

The file information block can be generated by a $FIB macro call. Displacement tags for the FIB can be defined by the $TFIB macro call.

NOTES: 1. If the first argument is coded, the address of the FIB is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal argument

0206 - Unknown or illegal LFN

0207 - LFN not open

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

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

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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).

COMMAND LINE PROCESS

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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.

FUNCTION 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 of 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 address of the command line, supplied by argument 1, is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the command line to be processed.

2. The size of the command line, supplied by argument 2, is placed in $R6; if this argument is omitted, $R6 is assumed to contain the size.

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

$R1 - Return status; one of the following:

0000 -	No error
0000-00FF -	Completion status returned by spawned task
0601 -	Insufficient memory
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
FFFF -	Honeywell component error previously reported

$B4 - Address of supplied command line

Example:

In this example, the $CMDLN macro call causes a command line to be processed which will execute 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 unit will be stored in the file

- An initial allocation of eight physical sectors (allowing 32 entries) for diskette, eight 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 four physical sectors (allowing 16 entries each) for diskette, eight 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 address of the parameter structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

0202 - Pathname not specified

0205 - Illegal argument

0209 - Named subdirectory 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 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.A '
```

The macro call can be specified as follows:

```
          $CRDIR   !SUBDIR
```

CREATE FILE (MOD 600 ONLY)

Macro Call Name: $CRFIL

Function Code: 10/30

Equivalent Command: Create File (CR)

Creates a new disk file by placing a description of the file in the file system hierarchy and, optionally, allocating space for it. The user identifies this file by either a logical file number (LFN) a pathname, or both. At the completion of create file execution, the file is reserved exclusively for the task group. If both an LFN and pathname are supplied then, in addition to creating and reserving the file, it is assigned to the LFN. Subsequent macro calls (open file, read record, etc.) can then be directed to the file via this LFN. $CRFIL can be used to create any of the disk files which are described in the Data File Organizations and Formats manual, including:

o Fixed-Relative
o Relative
o Sequential
o Indexed
o Random (calc)

In addition $CRFIL can be used to create a temporary disk file which will exist only during this task group's execution. This function is normally done outside program execution.

FORMAT:

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

ARGUMENT DESCRIPTION:

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 (2020) which indicates that an LFN is not specified, or -1 (FFFF), which indicates that the system should assign an LFN from the pool of available LFNs.

pathname pointer

A 4-byte address of the pathname, which 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 (1) the name of the file to be created, and (2) the directory in the file system hierarchy in which to add the name and attributes of the file. 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 organization

A 1-byte field specifying the file organization, as follows:

2 - Fixed-relative without deletable records

5 - Fixed-relative with deletable records

R - Relative

S - Sequential

I - Indexed

C - Calc (random)

A - I-D-S/II data base area

reserved

This 1-byte field must contain zeros.

0202 - Pathname not specified

0205 - Illegal argument

0206 - Unknown or illegal LFN

0208 - LFN or file already open

0209 - Same named subdirectory 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 Section 3, 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 was defined as follows:

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

With the preceding definitions having been made, the following macro call will create FILE_A:

```
          DOMYAA    $CRFIL   !FILE_A
```

```
*
*        CREATE A WORK SEGMENT OF UP TO 10K WORDS
*
         $CRMSG    =B'000C110000000000' =10240
*
*        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 WORK SEGMENT
*
         $DLSEG    SEG_A
          .
          .
          .
SEG_A    DC        <$
SEG_S    DC        0B(31,0)
```

CREATE OVERLAY AREA TABLE

CREATE OVERLAY AREA TABLE

Macro Call Name: $CROAT

Function Code: 07/0A

Equivalent Command: None

Create an overlay table to be used with overlay loading functions that require a pointer to an overlay area table (OAT). The overlay area described by this OAT is created in real memory space. (See the appropriate System Concepts manual for details on overlays and overlay area tables.)

FORMAT:

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

ARGUMENT DESCRIPTION:

location of OAT address

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

location of size of overlay area entry

Any address form valid for a data register; 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 value for this argument depends on the number of overlays of this size used by the bound unit and the frequency of their release.

FUNCTION DESCRIPTION:

This macro call creates an overlay area table (OAT) to be used by subsequent loader functions that require (or imply) the existence of an OAT in the call.

The real memory space for the overlay area described by this call is obtained from the same memory pool used by the current bound unit of the issuing task. If the current bound unit is not sharable, memory will be obtained from the pool associated with the group of the issuing task. If the current bound unit is sharable, memory will be obtained from the system pool.

Once allocated, the overlay area table becomes a supporting resource of the current bound unit. That is, an OAT queue header field will be added to the definition of the bound unit descriptor, and as OATs are created, they will be placed in this queue. The OAT queue is maintained so that OATs are ordered by ascending area size.

Before an OAT is allocated, any existing OATs are searched for an OAT with area size equal to that specified in argument 2. If one is found equal, the number of areas in this OAT is returned to the caller (i.e., location specified in argument 1 or to register $R6). On return, the caller receives the address of the newly created OAT or an existing OAT.

The overlay area reserve and execute overlay ($OVRSV) and overlay area, release ($OVRLS) macro calls require that overlay areas be present. If no OAT that controls entries of the specified size can be found, the system creates an overlay area with the number of entries specified by argument 2, and then creates the controlling OAT.

When the system returns the address of the OAT, it also returns the actual size of the overlay area and the actual number of areas allocated or already present.

NOTES:

1. The address of the OAT is returned in $B4 and is stored as specified in argument 1. If argument 1 is omitted, the address is stored only in $B4.

2. The size of the entry supplied by argument 2 is placed in $R2; if this argument is omitted, $R2 is assumed to contain the correct size.

3. The number of entries supplied by argument 3 is placed in $R6; if this argument is omitted, $R6 is assumed to contain the correct number.

4. 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

   082D - Group's available memory quota exceeded

   0E02 - No memory available for nonswappable 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 MOD 600, rounded up to nearest 256 words)

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

   $B4 - Address of OAT (if $R1 is 0000)

5. On a return with error, the contents of $R2, $R6, and $B4 are unspecified.

Example:

In this example, an overlay area of three 512-word entries is created. (It is assumed that no existing overlay area table controls 512-word entries.) The address of the controlling OAT will be placed in OATAD.

```
OATAD  RESV  2,0
       $CROAT =OATAD,=512,=3
```

CREATE SEGMENT (MOD 600 ONLY)

Macro Call Name: $CRSEG

Function Code: 0C/0C

Equivalent Command: None

Create a segment in the task issuing this call; assign the segment to the initial bound unit.

FORMAT:

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

ARGUMENT DESCRIPTION:

location of segment access rights

Any address form valid for a data register; provides the access rights (read, write, execute) for this segment, as defined by the hardware segment descriptor. Bits 0 through 5 of the register are used to specify 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

Thus, for example, to read, write, and execute this segment from ring 3, bits 0 through 5 would be 000000.

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. A small segment must be less than or equal to 4K words (K=1024); a large segment must be less than or equal to 64K words. The actual size of the created segment will be the size specified, rounded up to the next 256-word increment.

location of segmented address

Any address form valid for a data register; provides the address of any word in the segment. When null is specified, the system selects a segment number, consistent with the size as specified and with the availability of segment numbers to users.

FUNCTION DESCRIPTION:

The call permits the requesting task to dynamically create a segment of the size specified, and assign the segment to the initial bound unit associated with that task. The user can create private segments at link time, or dynamically with this macro call. (The create maximum segment macro call ($CRMSG) allows the task to dynamically create a segment having the specified size or maximum available size.)

Argument 3 allows the user to specify the segment number or have the system select the first available segment number.

Conflicts in address assignments are the user's responsibility. The user can overwrite invalid segment descriptors, but cannot destroy protected overlay-area tables or addresses (e.g., group system area segment, group work area segment, and system segments). The user with write/execute access rights to the segment in the user ring can destroy that address.

When the macro call is executed, $B2 contains a pointer to the start of the created segment.

NOTES: 1. The MOD 600 System Concepts manual describes ring and segment access, segment size, and segment numbers, in detail.

2. The segment's access rights value supplied by argument 1 is placed in $R2. When the argument is omitted, the system assumes that $R2 contains this value. 7/79

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

4. The address of any word in the segment, supplied by argument 3, is placed in $B2. 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.

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

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

   0000 - No error

   0602 - Insufficient system memory

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

   - Sharable bound unit root
   - System segment
   - Creating a nonuser segment

   0828 - No virtual address available to create segment

   082D - Group's available memory quota exceeded

   082E - Argument error:

   - Size exceeds 64K

   - Size inconsistent with the specified segment number

   0E02 - No memory available for nonswappable task

$R6, $R7 - Actual size of created segment

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

Example:

With the macro call, the requesting task creates a 2K-word segment, and assigns it to the initial bound unit. Ring 3 will have read and write access rights, but execute access is restricted to ring 0. The segment number of the created segment will be 2.

```
          $CRSEG    =B'0000110000000000';
                    =2048;
                    +$A
            .
            .
            .
$A        DC        Z'00020000'
```

CREATE SYSTEM GROUP (MOD 600)

CREATE SYSTEM GROUP (MOD 600 ONLY)

Macro Call Name: $CSGRP

Function Code: 0D/02

Equivalent Command: Create Daemon Group (CDG)

Define a new task group with special privileges for performing system functions. Allocate and initialize the data structures necessary to control the new task group within the specified memory pool. Create the lead task as described in the create task ($CRTSK) macro call.

NOTE: This macro routine is recommended for use only by specialized software system designers.

FORMAT:

```
label  $CSGRP  [location of group identifier],                  *
               [location of base level],
               [location of high logical resource number],
               [location of high logical file number],
               [location of root entry name address]
```

ARGUMENT DESCRIPTION:

location of group identifier

Any address form valid for a data register; provides the group identification of the new task group. The group identifier must be a two-character (ASCII) name that may include the dollar sign ($) as its first character.

*

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 will execute.

* location of high logical resource number

Any address form valid for a data register; provides the highest logical resource number (LRN) that will be 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 will be used.

location of high logical file number

Any address form valid for a data register; provides the highest logical file number (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 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. 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?ECL specifies the command processor as the lead task.

CREATE TASK

Macro Call Name: $CRTSK (MOD 600 only)

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]
```

ARGUMENT DESCRIPTION:

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.)

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.)

FUNCTION DESCRIPTION:

This 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.

Either the location of the start address or the location of the root entry name address, but not both, can be specified.

If the new task is to execute the same bound unit as the issuing task, then 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 LRN supplied by argument 1 is placed in $R2; if this argument is omitted, $R2 is assumed to contain the LRN for the created task.

2. The relative priority level supplied by argument 2 is placed in $R6; if this argument is omitted, $R6 is set to the relative priority level of the task issuing this create 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, $B2 is assumed to contain 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 - Bound unit not found

   0602 - Memory unavailable

   0809 - LRN too large

   0813 - Referenced LRN already in use or invalid

   0827 - Bound unit file not fixed-relative

   0830 - LRN not available

   082D - Group's available memory quota exceeded

   0E02 - No memory available for nonswappable task

   1604 - Unresolved symbolic start address

   160A - Insufficient memory

   1611- Zero length root segment

   1613 - Invalid bound unit pathname

   1615 - Illegal bound unit file

$R2 - LRN of created task

Examples:

In this example, the $CRTSK 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, entering the bound unit at entry point PROG10.

```
                $CRTSK   =10,=2,,!ROOT
                      .
                      .
                      .
      ROOT      TEXT     'PROG10Δ'
```

In this example, the $CRTSK macro call makes a task known as logical resource number 12 (decimal) of the issuing group. The task will execute at the same priority level as the issuing task. The task will execute the same bound unit as the issuing task and will be started at the address represented by the label SSA.

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

LK is a semaphore which has an initial value of 1 and which 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 TH as described above, the task will unlink (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 semaphore LK, thus ensuring the integrity of this data base. After it stops using this data base, the task does a V-op on LK.

When the task finishes using the resource, it will return 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

Macro Call: $DLDIR

Function Code: 10/A5

Equivalent Command: Delete Directory (DD)

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

FORMAT:

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

ARGUMENT DESCRIPTION:

argument 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 entry.

pathname pointer

A 4-byte address, which 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.

FUNCTION DESCRIPTION:

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

NOTES:

1. If the argument is coded, the address of the parameter structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

   0202 - This function requires a pathname to be specified

   0205 - Illegal argument

   0209 - Named directory not found

   020C - Volume not found

   0213 - Cannot provide requested concurrency

   0220 - Attempted deletion of nonempty directory

   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 - Illegal file type (not a directory)

   022C - Access control list (ACL) violation

Example:

In this example, the $DLDIR 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.AΔ'
                $RLDIR !SUBDIR
```

DELETE FILE

DELETE FILE

Macro Call Name: $DLFIL

Function Code: 10/35

Equivalent Command: Delete File (DL); RL usable also in MOD 600

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. You identify the file to be deleted 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 DESCRIPTION:

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 in the range 0 through 255; or blank (which indicates that an LFN is not specified).

pathname pointer

A 4-byte address, which 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.

FUNCTION 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 case of the former, a return status code of 0208 is loaded in $R1; in the latter case, the file is deleted after the other task group is finished using it.

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.

NOTES:

1. If the argument is coded, the address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

   0202 - The LFN and pathname both were not specified

   0205 - Illegal argument

   0206 - Unknown or illegal 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

0228 - Illegal file type (a directory)

022C - Access control list (ACL) violation

Example:

In this example, the macro call deletes the file created in the create file macro call example. To do this, it references the same argument structure as the $CRFIL 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
```

KEY

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

logical file number
input key pointer
input key format

FUNCTION 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 via data management (bit 0 is 0) and allows delete operations (bit 4 is 1). 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 $FIB macro call. Displacement tags for the FIB can be defined by the $FIBDM macro call.

NOTES: 1. If the argument is coded, the address of the FIB is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal function
0205 - Illegal argument
0206 - Unknown or illegal LFN
0207 - LFN not open
020A - Address out of file
020E - Record not found
0217 - Access violation
0219 - No current record pointer
021E - Key length or location error

022A - Record lock area overflow

022B - Requested record is locked or causes deadlock

022F - Unknown or illegal record type

0237 - Invalid record or control interval format

Example:

The macro call in this example identifies the FIB that is described under "Assumptions for File System Examples" in Section 3. The $TFIB macro call reserved the FIB tags. The $DLREC macro call indicates that the current record is to be deleted; it is assumed that the file is open and that a $RDREC NEXT (read next record) macro call immediately precedes the $DLREC 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. In addition, offset tags can be used to access the LFN in later instructions in your program with the macro call $TFIB.

*

DELETE SEMAPHORE

Macro Call Name: $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 identifier]

ARGUMENT DESCRIPTION:

location of semaphore identifier

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

FUNCTION DESCRIPTION:

This call deletes a counting semaphore that was previously defined for the issuing task group with a define semaphore ($DFSM) macro call.

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

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

NOTES: 1. The semaphore identifier supplied by the argument is placed in $R6. When the argument is omitted, the system assumes that $R6 contains the identifier 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

0506 - Semaphore is currently active and cannot be deleted.

$R6 - Semaphore identifier (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'0504'
          BE        TH_BSY
DLSBB     $DLSM     ='LK'
          CMR       $R1,=Z'0504'
          BE        LK_BSY
             .
             .
             .
```

ENABLE USER TRAP

Macro Call Name: $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 DESCRIPTION:

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.

FUNCTION DESCRIPTION:

This 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 connect trap handler 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 RTT (return from trap) instruction is executed to return from the task's trap handler. (See Section 7 for more information about trap handling.)

NOTES: 1. The trap number of the trap to be enabled, supplied by argument 1, is placed in $R2; if this argument is omitted, $R2 is assumed to contain the binary number of the trap to be enabled.

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

$R1 - Return status; one of the following:

0000 - No error

0341 - Trap handler entry not connected

0342 - Illegal 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 "Trap Handler, Connect."

ENTRY POINT IDENTIFICATION (MOD 600 ONLY)

Macro Call Name: $ENTID

Function Code: 14/07

Equivalent Command: None

MOD 400 returns the address or value corresponding to a specified symbolic name which has been specified in an EDEF statement within the bound unit currently being executed by the issuing task, or within a bound unit made permanently resident by a CLM LDBU statement. For MOD 600, the currently executing bound unit is searched followed by a search of all other attached bound units.

FORMAT:

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

ARGUMENT DESCRIPTION:

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. The name must have been declared at link time in an EDEF statement.

location of id type

Any address form valid for a data register; provides the id type.

FUNCTION 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.

NOTES:

1. The address of the symbolic name field supplied by argument 1 is placed in $B4. When this argument is omitted, 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 i in argument 2, $R6 is loaded with -1. If argu-e ment 2 is omitted, the system assumes that $R6 c contains the id type.

3. 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,=6
                    .
                    .
                    .
         ENTNAM  TEXT      'ENTRY1'
```

Pages 147 through 165 have been deleted

ERROR OUT

ERROR OUT

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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). 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.

EXPAND PATHNAME

EXPAND PATHNAME

Macro Call Name: $XPATH

Function Code: 10/D0

Equivalent Command: None

Develop a full pathname from a relative pathname.

FORMAT:

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

ARGUMENT DESCRIPTION:

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.

input pathname pointer

A 4-byte address, which 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, which 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 on which to expand the relative path, as follows:

0000 - Working directory
0001 - System library-1
0002 - System library-2

FUNCTION 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 address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain the address of the argument structure.

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

0000 - Successful completion

0201 - Illegal pathname

0202 - Pathname not specified

0205 - Illegal 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, and you want to develop a fully expanded absolute pathname from the relative pathname<<ADF. In the macro call, you must identify the relative pathname (<<ADF) and the basis (working directory) on which the absolute pathname is to be developed, 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
```

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 Offsets Tags:

Tag	Corresponding Offsets (in Words)	Entry Name
F_LFN	0	Logical file number (LFN)
F_PROV	+1	Progam view
F_URP	+2	User record pointer[1]
F_IRL	+4	Input record length[1]
F_ORL	+5	Output record length[1]
F_IKP	+9	Input key pointer[1]
F_IKF	+11	Input key format (first byte)[1]
F_IKL	+11	Input key length (second byte)[1]

[1] Specific to $RDREC, $WRREC, $RWREC, and $DLREC macro calls.

[2] Specific to $RDBLK, and $WRBLK macro calls.

Tag	Corresponding Offsets (in Words)	Entry Name
F_ORA	+12	Output record address[1]
F_ORA1	F_ORA(+12)	(left half of F_ORA)[1]
F_ORA2	F_ORA+1(+13)	(right half of F_ORA)[1]
F_UBP	+2	User buffer pointer[2]
F_BFSZ	+4	Buffer size[2]
F_BKSZ	+5	Block size[2]
F_BKNO	+6	Block number[2]
F_BKN1	F_BNKO(+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

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 via primary key
MF_RKA	Z'0080'	Access via relative key
MF_SKA	Z'0040'	Access via 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)

[1] Specific to $RDREC, $WRREC, $RWREC, and $DLREC macro calls.

[2] Specific to $RDBLK, and $WRBLK macro calls.

0209 - Named file or directory not found

020C - Volume not found

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 - Recovery/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 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 Section 3.
                           (The pathname is defined in
                           the example for the create
                           file macro call.)
          DC     <IDX01
          RESV   2-$AF
          DC     Z'8501'   READ/WRITE SHARE; RECORD
                           LOCKING:  ISSUE MOUNT REQUEST
          RESV   2,0       IGNORED
          DC     Z'0200'   BUFFERS=2
          RESV   4,0       IGNORED
```

It is assumed that the following macro calls were issued before the $GTFIL macro call was issued:

```
$CRDIR    !SUBDIR    (See create directory macro example)

$CRFIL    !FILE_A    (See "Assumptions for File System
                     Examples")
```

The $GTFIL macro call altering FILE_A concurrency from exclusive to share can be specified as follows:

```
                     $GTFIL    !WRTFIL
```

Example 2:

In this example, the $GTFIL macro call is used to append characters to an incomplete pathname defined as follows:

```
DIRPTH    DC    '^VOL03>SUBINDEX.A△'    (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.A△'
          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△'
```

Peripheral Device	Device Type Code	Marketing Identifier
Line Printer	2000	PRU9104/9106
	2001	same as above but with Option PRF9102
	2002	PRU9103/9105
	2003	same as above but with Option PRF9102
Magnetic Tape	2045	MTU9104 (9-track NRZI, 45-ips)
	2046	MTU9105 (9-track NRZI, 75-ips)
	2045	MTU9109 (9-track, 800 bpi, NRZI, 45-ips)
	2049	MTU9109 (9-track, 1600 bpi PE, 45-ips)
	2046	MTU9110 (9-track, 800 bpi NRZI, 75-ips)
	204A	MTU9110 (9-track, 1600 bpi PE, 75-ips)
	2067	MTU9112 (7-track, 556-bpi)
	2071	MTU9112 (7-track, 800-bpi)
	206A	MTU9113 (7-track, 556-bpi)
	2071	MTU9113 (7-track, 800-bpi)
	2049	MTU9114 (9-track, 1600-bpi, 45-ips)
	204A	MTU9115 (9-track, 1600-bpi, 75-ips)

*logical resource number

A 2-byte entry into which the system places the logical resource number (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:

-1 - IBM diskette
0 - Device file
2 - Fixed-relative without deletable records
5 - Fixed-relative with deletable records
D - Directory
R - Relative
S - Sequential
I - Indexed
C - Random (calc) disk file
T - Tape-resident file

*data format

A 1-byte entry into which the system places a code identifying the format of the data, as follows:

F - Fixed-length record
D - Variable-length record (decimal count size)
V - Variable-length records (binary count size)
U - Undefined

file attribute pointer

A 4-byte address of a 32-byte field 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.

reserved

A 4-byte entry, containing zeros.

key 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.

reserved

A 4-byte entry, containing zeros.

The system places file attribute information in the 32-byte field pointed to by the file attribute entry described above. For disk-resident files, the structure contains the following:

*block size

A 2-byte entry specifying the maximum size (in bytes) of a physical record (i.e., the unit of transfer to a device file).

*reserved

A 28-byte entry containing zeros.

The following key descriptor information is placed in the 18-byte field pointed to by the key descriptors entry described above. This structure applies only to indexed files, and contains the following:

*reserved

A 4-byte entry that contains zeros.

*number of key components

A 1-byte entry that contains 1.

*reserved

A 9-byte entry that contains zeros.

*key component data type

A 1-byte entry that indicates the data type of the key component. The entry is hexadecimal 43 (i.e., C) for character, or hexadecimal 44 (i.e., D) for decimal.

*key component size

A 1-byte entry that specifies the size of the key component in bytes; that is, it specifies the size of the primary key stored in each logical record in the indexed file.

*key component location

A 2-byte entry that specifies the offset (in bytes) from the beginning of the record to the beginning of the key field; the first byte in the logical record is position 1.

FUNCTION 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 $GIPSB, $GIKDB, and $GIFAB macro calls.

NOTES: 1. If the argument is coded, the address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

0202 - Pathname not specified

0205 - Illegal argument

0206 - Unknown or illegal 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

0226 - Not enough user memory for buffers or structures

0228 - Illegal file type

022C - Access control list (ACL) violation

022E - Record lock concurrency conflict

0238 - Invalid file description

Structure for Disk Files:

Word	Fields
0	Logical Record Size
1	Control Interval Size
2	Current Allocation Size
3	Allocation Increment
4	Maximum Allocation Size
5	Free Space per Control Interval
6	Local Overflow Increment
7	Number of Key Descriptors
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 Offsets Tags:

For tape-resident and device files:

Tag	Corresponding Offsets (in words)	Entry Name
T_LRSZ	0	Logical record size
T_BKSZ	+1	Block size
T_RFU	+2	Reserved
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_LRSZ	0	Logical record size
K_CISZ	+1	Control interval/physical sector size
K_CASZ	+2	Current allocation size
K_AISZ	+3	Allocation increment size
K_MASZ	+4	Maximum allocation size
K_FREE	+5	Amount of free space per C.I.
K_LOV	+6	Local overflow allocation increment
K_NKS	+7	Number of keys
K_SZ	16	Size of structure (in words); not a field in the block

NOTE: If the tape file sequence number is 00 and

1. The file is opened at a storage management (block) level and,

2. No volume name or file name has been given in the pathname,

the tape is reserved for volume (device) level access.

GET WORKING DIRECTORY

Macro Call Name: $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 DESCRIPTION:

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.

FUNCTION 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 address of the argument structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain the address of the argument structure.

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

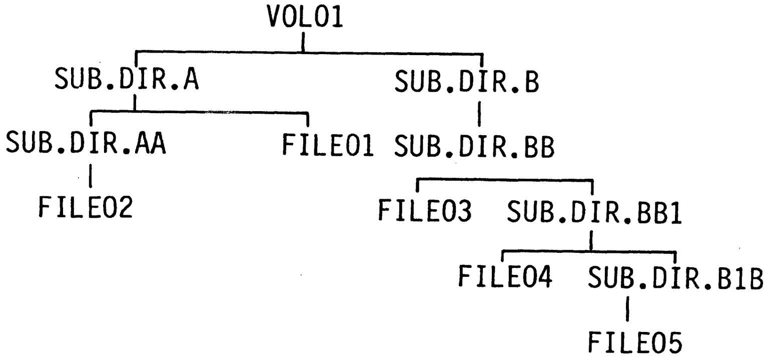
0000 - No error

0205 - Illegal 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 $GWDIR 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 STATUS (MOD 600 ONLY)

Macro Call Name: $GRPST

Function Code: 14/0E

Equivalent Command: None

Return the current status information about the specified group to a 47-word receiving field.

FORMAT:

```
[label] $GRPST [location of group id whose status is requested],
               [location of group-status field address]
```

ARGUMENT DESCRIPTION:

location of group id

Any address form valid for a data register; provides the group id of the task group whose status is to be returned.

location of group-status field address

Any address form valid for a data register; provides the address of a 47-word nonvarying receiving field where the system will place current status information about this task group.

FUNCTION DESCRIPTION:

This call returns to the issuing task, in $B4, the address of the 47-word receiving area that will contain the status information, in the following order, formatted as follows:

Number of Words	Contents
3	Terminal id (in ASCII) associated with this group as primary logon; blank if applicable
6	User id (in ASCII) associated with this task group
16	Group id (in ASCII)
1	Reserved for system use
1	Amount of CP time used
3	Private memory total
1	Shared memory total
1	Status of pre-emption and load control
1	I/O count
2	Segment swap count
2	Overlay miss count
2	Overlay hit count
2	Last bound unit loaded by create task
6	Size of group status block

NOTES:

1. The location of the group id supplied in argument 1 is placed in $R2. When this argument is omitted, the system assumes that $R2 contains the location of the group id.

2. The address of the 47-word receiving field supplied in argument 2 is placed in $B4. When this argument is omitted, the system assumes that $B4 contains the receiving field's address.

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

   $R1 - Return status codes, one of:

   0000 - No error

   0817 - Memory access violation

   0818 - No task group with specified group id

$B4 - Address of the group-status receiving field

Example:

This macro call obtains group status information about the task group whose id is JR, and places the information in the 47-word field labeled STATUS:

```
 JRSTAT     $GRPST     ='JR',!STATUS
                             .
                             .
                             .
 STATUS     RESV       47,0
```

HOME DIRECTORY

HOME DIRECTORY

Macro Call Name: $HDIR

Function Code: 14/0B

Equivalent Command: Home Directory (HOME_DIR)

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 DESCRIPTION:

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 will place the pathname of the default working directory of the calling task group.

FUNCTION DESCRIPTION:

This 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 address of the receiving home directory field, supplied by argument 1, is placed in $B4; if this argument is omitted, $B4 is assumed to contain the correct address.

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

0000 - No error
0817 - Memory access violation

NWAIT - Do not suspend the issuing task (set the w-bit to 1)

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.

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 IOR extension to be generated, resulting in an extended IORB (see Appendix A). When the argument is omitted, the system generates a standard length IORB.

EXT - Generate an extended IORB

FUNCTION DESCRIPTION:

The input/output request block (IORB) is used as the standard means of requesting a physical I/O service. The IORB contains an LRN which 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.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

Example:

In this example, the $IORB macro call generates a standard IORB having an LRN of 0, 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
```

0000 - No error
0802 - Invalid LRN

Example:

The issuing task issues a $KILLT macro call to abort another task (whose LRN is 34) in the same task group.

```
ABT34    $KILLT    =34
```

MESSAGE GROUP, ACCEPT

Macro Call Name: $MACPT

Function Code: 15/01

Equivalent Command: None

Establish a message connection, through a mailbox, between an initiator's task group and the acceptor (calling) task group issuing this $MACPT macro call.

FORMAT:

```
[label]  $MACPT  [location of MGIRB address]
```

ARGUMENT DESCRIPTION:

location of MGIRB address

Any address form valid for a data register; provides the address of the message group initialization request block (MGIRB), which must have been previously generated.

FUNCTION DESCRIPTION:

The acceptor task group issues this macro call in order to accept a connection request initiated (with a $MINIT macro call) by the initiator task group. The $MACPT macro call (1) indicates that the acceptor task group wishes to receive a message from a named mailbox (message queue), and (2) opens the receive function of the message facility. (See the appropriate System Concepts manual for a discussion about the message facility.)

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.

Table 5-3 (cont). Argument Values for $MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword with option				
Any	ALIGN=		Buffer byte alignment:	MC_OPT
		R	Buffer begins in rightmost byte of address specified by BUF= argument.	
		L (default value)	Buffer begins in leftmost byte of address specified by BUF= argument.	
Any	WTI=		Wait test indicator (for $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.	
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.	

[a] When WAIT is specified, argument 3 must be omitted.

[b] When this argument is omitted, or argument 3 is WAIT, the generated MGCRB contains no termination option. In that case, the user must issue a $WAIT, $WAITL, or $TEST macro call.

FUNCTION 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 ($MSEND) and message group receive ($MRECV) 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.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

MESSAGE GROUP CONTROL REQUEST BLOCK OFFSETS

MESSAGE GROUP CONTROL REQUEST BLOCK OFFSETS

Macro Call Name: $MGCRT

Generated Label Prefixes:

MC_OS
MC_MAJ
MC_OPT
MC_BUF
MC_BSZ
MC_DVS/MC_REC
MC_RSR
MC_MRU/MC_WTI
MC_EXT
MC_FNC/MC_REV
MC_MGI
MC_LVL
MC_PCI
MC_VDP
MC_TGI
MC_TSK
MC_NPI

Appendix A describes the contents of the message group control request block (MGCRB).

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for information about SAF/LAF independent code.

MESSAGE GROUP, COUNT

MESSAGE GROUP, COUNT

Macro Call Name: $MCMG

Function Code: 15/07

Equivalent Command: None

Provide a count of the number of completed message groups, not yet "accepted" by previous $MACPT macro calls, that are available for processing by subsequent $MACPT macro calls.

FORMAT:

```
[label]  $MCMG  [location of MGIRB address]
```

ARGUMENT DESCRIPTION:

location of MGIRB address

Any address form valid for a data register; provides the address of the message group initialization request block (MGIRB), which must have been previously created.

FUNCTION 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 $MACPT macro calls, and available to subsequent $MACPT macro calls. The mailbox is described in the MGIRB for this macro call (see Table 5-4 below).

NOTES: 1. Referenced mailboxes must have been created before this macro call is issued. (See the create mailbox (CMBX) command in the Commands manual.) References to mailbox fields when no mailbox has been created results in an error return.

Table 5-5 (cont). Argument Values for $MGIRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGIRB
Keyword only (cont)				
Any	RESV	None	Generates the MGIRB.	
Keyword with expression				
3			Issuing task termination[b] 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.	
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

[a] When WAIT is specified, argument 3 must be omitted.

[b] 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, $WAITL or $TEST macro call.

FUNCTION 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 ($MACPT), message group initiate ($MINIT), and message group count ($MCMG) 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.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

MESSAGE GROUP INITIALIZATION REQUEST BLOCK OFFSETS

MESSAGE GROUP INITIALIZATION REQUEST BLOCK OFFSETS

Macro Call Name: $MGIRT

Generated Label Prefixes:

```
     MI_OS
     MI_MAJ
     MI_OPT
     MI_BUF
     MI_BSZ
     MI_MPD
     MI_RSR
     MI_MDE/MI_IOP
     MI_EXT
     MI_FNC/MI_REV
     MI_MGI
     MI_PCM/MI_ADT
     MI_NWI
     MI_NDI
     MI_MBI
     MI_NWA
     MI_NDA
     MI_MBA
     MI_QSZ
     MI_CNT
     MI_TGI
     MI_TSK
     MI_SIP
```

Appendix A describes the contents of the message group initialization request block (MGIRB).

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for information about SAF/LAF independent code.

MESSAGE GROUP, INITIATE

Macro Call Name: $MINIT

Function Code: 15/02

Equivalent Command: None

Initiate a message connection, through a previously created mailbox, between the initiating task group (initiator) and the accepting task group (acceptor).

FORMAT:

```
[label]  $MINIT  [location of MGIRB address]
```

ARGUMENT DESCRIPTION:

location of MGIRB address

Any address form valid for a data register; provides the address of the message group initialization request block (MGIRB), which must have been previously generated.

FUNCTION DESCRIPTION:

A task group that is to send a message (initiator task group) to another task group must issue the $MINIT macro call to open the send function of the message facility. (See the MOD 400 System Concepts manual for a discussion about the message facility.) The macro routine informs the system that a message connection is requested in order to send a message, and provides the name of the initiator's mailbox.

NOTES: 1. Mailboxes must have been created before the macro call is issued. (See the create mailbox (CMBX) command in the Commands manual.)

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. Before the $MINIT macro call is executed, the user must generate the MGIRB (see Table A-8) with the argument values shown in Table 5-6.

Table 5-6. MGIRB Argument Values for $MINIT 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.	MI_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-path-description identifier Identifies one of a set of message path descriptions (MPDs) associated with the initiator task group. The MPD defines the connection between the mailboxes.	MI_MPD	Must be 0001, indicating local mail facility, and which assumes a default MPD that is for one-way message groups.
initiator identification Describes the following characteristics of the initiator at a mailbox where the message group originates.	As shown below.	As shown below.
address type Specifies that this is an initiator's address.	MI_ADT (bits 0-7)	Must be hexadecimal 01.
Reserved for future use.	MI_NWI	Must be 0.
Reserved for future use.	MI_NDI	Must be 0.

Table 5-6 (cont). MGIRB Argument Values for $MINIT Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
initiator identification (cont) initiator mailbox name	MI_MBI	Must be from 1 to 12 ASCII characters, blank-filled, left-justified.
destination identification acceptor mailbox name	MI_MBA	Must be from 1 to 12 ASCII characters, blank filled, left justified, and as specified when the mailbox was created.

4. The $MINIT macro 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 $MINIT macro call.

5. On successful macro execution, the system returns the message group identifier (MI_MGI field) of the "initiated" message group. A valid identifier is returned for all requests.

6. On return, $R1 contains the following return status codes:

   0000 - No error

   0C23 - Invalid message-path-description identifier

   0C25 - Acceptor mailbox may not be accessed by initiator

   0C26 - Acceptor mailbox not known

   0C34 through 0C44 - User-coded reason for abnormal message group

   0C62 - Normal message group termination

7. On return, register $B4 will point to the application's MGIRB, which is updated according to the specifications in the macro call.

MESSAGE GROUP RECOVERY REQUEST BLOCK OFFSETS

MESSAGE GROUP RECOVERY REQUEST BLOCK OFFSETS

Macro Call Name: $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 A describes the contents of the message group recovery request block (MGRRB).

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the *Assembly Language Reference* manual for information about SAF/LAF independent code.

MESSAGE GROUP, SEND

Macro Call Name: $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 DESCRIPTION:

location of MGCRB address

Any address form valid for a data register; provides the address of the message group control request block (MGCRB), which must have been previously generated.

FUNCTION DESCRIPTION:

The task group that issued a $MINIT macro call to initiate a message connection, issues one or more $MSEND macro calls to send message data via that connection. A task group sends a message through a named mailbox, from which the receiving task group obtains the message. The $MSEND macro call uses the same message group identifier, returned in the $MINIT 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 $MSEND call sends one record, which is the basic unit of data exchange. Each $MSEND transmission points to an MGCRB that describes the buffer of message data.

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 $MODID macro call is issued to place the mode identification of the task group in that field.

```
MODFL     RESV     2
          LAB      $B4,MODFL
              .
              .
              .
          $MODID
```

NEW COMMAND IN

NEW COMMAND IN

Macro Call Name: $NCIN

Function Code: 08/06

Equivalent Command: None

Reset the command-in file for the issuing task to that specified by the supplied pathname, or to its original definition.

```
  [label]  $NCIN  [location of pathname address]
                  [location of argument list address]
```

ARGUMENT DESCRIPTION:

location of pathname address

Any address form valid for a register; provides the address of the pathname of the new command-in file.

location of argument list address

Any address form valid for an address register; provides the address of the argument list containing the arguments for use in command line argument substitution.

FUNCTION DESCRIPTION:

This call allows a task to reset the command-in file to that specified by the supplied pathname.

NOTES: 1. The address of the pathname of the new command-in file supplied by argument 1 is placed in $B4; if this argument is omitted, $R2 is set to the value of 1 designating that the command-in file is to be reset to that initially defined for the issuing task.

2. The address of the argument list supplied by argument 2 is placed in $B2 and $R2 is set to zero; if this argument is omitted, $B7 contains the address of the argument list.

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 PROCESS

Macro Call Name: $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 DESCRIPTION:

There are no arguments for this macro call.

FUNCTION DESCRIPTION:

This 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 $NPROC macro call is used to terminate and restart the task group request.

```
                    AGAIN   $NPROC
```

   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 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 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 $FIB macro call. Displacement tags for the FIB can be defined through the $TFIB macro call.

NOTES: 1. If the first argument is coded, the address of the FIB is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal argument

0206 - Unknown or illegal 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

0225 - Not enough system memory for buffers or structures

0226 - Not enough user memory for buffers or structures

022C - Access control list (ACL) violation

022E - Recovery/record lock/concurrency conflict

0232 - Invalid tape file header or tape file trailer label

0237 - Invalid record or control interval format

0238 - Invalid file description information

Example:

This $OPFIL example opens a new file, in which records are to be written via 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.

NOTES:

1. The return point address supplied by argument 1 is placed in $B5; if this argument is omitted, $B5 is assumed to contain 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. In MOD 600, register $R1 will contain the value 0006 which is the subfunction code of the macro call.

Example:

In this example, the calling overlay uses the $OVRLS 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

OVERLAY AREA RESERVE, AND EXECUTE OVERLAY

Macro Call Name: $OVRSV

Function Code: 07/05

Equivalent Command: None

Reserve an overlay area within 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 defined through a create overlay area table macro call.)

FORMAT:

```
[label]  $OVRSV  [location of overlay id],
                 [location of entry point offset],
                 [location of OAT address]
```

ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the 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 an address 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.

1601 - Invalid overlay id

160A - Insufficient memory

1611 - Zero length overlay

1612 - Overlay not a user segment

1614 - Access violation:

- o Root of sharable bound unit
- o No access

$R2 - Overlay id (on a successful return)

$R6 - Overlay default start address offset (on a successful return)

$B4 - Overlay base address (MOD 400 only)

Overlay load address (MOD 600 only)

Example:

In this example, the $OVLD macro call causes the overlay named DPOSIT (of the bound unit being executed) to be loaded but not executed. Upon return from the system, in MOD 400 only, $B4 will contain the overlay base address or a null pointer value for floatable overlays. For nonfloatable overlays, $B4 is not applicable. $R6 will contain the offset from its base address to its default start address. The overlay base address and the offset to the default start address will be saved in OVLY_A and OVLY_E, respectively. Thus, the overlay can be entered later at its 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 $OVUN (overlay unload) macro call.

```
*
*    LOAD THE DPOSIT OVERLAY
*
        XVAL     DPOSIT
        $OVLD    =DPOSIT
*
        BNEZ     $R1, BAD_LD         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
*
        $OVUN    =DPOSIT, !OVLY_A
        .
        .
        .
OVLY_A  DC       <$
OVLY_E  DC       00
```

OVERLAY, UNLOAD

Macro Call Name: $OVUN

Function Code: 07/0C

Equivalent Command: None

Unload the specified overlay of the bound unit that contains the procedure being executed by the issuing task.

FORMAT:

```
[label]  $OVUN [location of overlay id],
               [location of overlay base address] (MOD 400 only),
               [location of return point address]
```

ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an 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 (MOD 400 only)

Any address form valid for an address register; provides the base address of the overlay to be unloaded.

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.

FUNCTION DESCRIPTION:

This call causes the named overlay to be unloaded. The overlay must not share a segment with any other overlay of the bound unit. You must have the proper access rights to the overlay.

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. In MOD 400, 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.

4a. In MOD 400, the overlay being updated must be floatable, and the memory it occupies must have been obtained by a get memory call, either directly by the user or indirectly by either the overlay load or overlay execute macro call. If that memory was obtained directly by the user, then the address of the first word of the memory block must have been specified as the base address of the overlay when it was loaded.

4b. In MOD 600, the overlay being unloaded must not share a segment with any other overlay of the bound unit. The overlay must start at location 0 of the segment and must occupy some part of the last 256 words of the segment. A check is made for flagrant misuses, however, not all possible errors are detected.

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

0000 - No error

0602 - Insufficient system memory

0603 - Illegal block memory address

0616 - Overlay not in a segment (refer to Note 4b)

0817 - Memory access violation

- System segment
- No access rights
- Root of sharable bound unit

When the $RDSW 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:

o If bit 11 is 0, none of the switches read was on.
o If bit 11 is 1, 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 $RDSW 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

Macro Call Name: $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

Retrieves one logical record from a file to your record area or merely positions 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   ⎭  │
                                         └               ┘
```

ARGUMENT DESCRIPTION:

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 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 1). 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

(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 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 1. 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

After the record is transferred to main memory, the system updates the following FIB entries:

output record length

output record address

(Simple or relative key.)

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 calc key is pointed to by the FIB input key pointer field.

POSEQ
PEQ

(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 to be active. The record can be read via a read next 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

This mode is referred to as read position equal.

POSGR
PGR

(For disk files accessed by key, only.) This mode argument positions the read and 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 via a read next 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 via a read next 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 specified by the key value identified in the FIB (but not beyond the end-of-file). It is not necessary for the record pointed to to be active. The record can be read via a read next 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 backwards 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.

FUNCTION DESCRIPTION:

Before this macro call can be executed, the LFN must have been opened (see the open file macro call) with a program view word that allows access via data management (bit 0 is 0) and allows read operations (bit 1 is 1). 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 file information block can be generated by a $FIB macro call. Displacement tags for the FIB can be defined by the $FIBDM 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.[1] 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.

NOTES: 1. If the first argument is coded, the address of the FIB is loaded into $B4; if the argument is omitted, $B4 is assumed to contain the address of the FIB.

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

[1] A read, with any position mode, positions the read pointer to the desired record, so that a subsequent READ-NEXT will retrieve that record.

0000	-	No error
01xx	-	Physical I/O error
0203	-	Illegal function
0205	-	Illegal argument
0206	-	Unknown or illegal LFN
0207	-	LFN not open
020A	-	Address out of file
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 or not defined
022B	-	Requested record is locked or causes deadlock
022F	-	Unknown or illegal record type
0233	-	Tape file sequence number error
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 Section 3. 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 you can interrogate using the FIB offset tags:

F_ORL (Output record length)
F_ORA (Output record address)

Pages 5-366 through 5-371 have been deleted

RELEASE SEMAPHORE

Macro Call Name: $RLSM

Function Code: 06/03

Equivalent Command: None

Release a resource controlled by the specified semaphore and activate the first waiting task enqueued 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 identifier]
```

ARGUMENT DESCRIPTION:

location of semaphore identifier

Any address form valid for a data register; provides the two ASCII characters that identify the semaphore controlling the resource to be released.

FUNCTION 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 semaphore identifier supplied by argument 1 is placed in $R6; if this argument is omitted, $R1 is assumed to contain the correct identifier.

FUNCTION DESCRIPTION:

This macro call removes the file reservation established for the specified file, provided it is not currently open (see "Open File") in the task group in which you are executing. It does not dissociate the LFN from a pathname (see "Dissociate File").

Also, if the file is a temporary file (see "Create File"), 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 either 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 was reserved through the command GET; the command REMOVE 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 again locate the file. 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 address of the parameter structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

0202 - Pathname not specified

0205 - Illegal argument

0206 - Unknown or illegal LFN

0208 - LFN or file currently open in same task group

0209 - Named file or directory not found

0210 - LFN conflict

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

0229 - File not known to task group

Example:

In the following example, 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 two examples:

```
Example 1:  WRTFIL    DC       5           LNF = 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_AΔ'
                      $RMFIL   !WRTFIL
```

RENAME FILE/RENAME DIRECTORY

RENAME FILE/RENAME DIRECTORY

Macro Call Name: $RNFIL

Function Code: 10/40

Equivalent Command: Rename (RENAME)

Change the name of a disk file or directory to the name specified by the macro call. You identify the disk file or directory to be renamed by supplying either a logical file number (LFN) or a pathname. This function is usually done outside program execution.

FORMAT:

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

ARGUMENT DESCRIPTION:

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 in the range 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; 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.).

FUNCTION 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 <u>Commands</u> 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.

NOTES:

1. If the argument is coded, the address of the parameter structure is loaded into $B4; if the argument is omitted, $B4 is assumed to contain 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 - Illegal pathname

   0202 - Pathname not specified

   0205 - Illegal argument

   0206 - Unknown or illegal 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

Example:

In this example, the $RQGRP 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 and the initial working directory will be JONES.TEST.M and ^V1124>UDD>TEST>JONES, 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). (See this section for a description of the $PRBLK macro used in this example.)

```
         $RQGRP    ='Q2',!ARGS,!INFO
            .
            .
            .
 INFO    $PRBLK    ,^1124>UDD>TEST>JONES>ASM_TST;
                   ^V1124>UDD>TEST>JONES>L>ASM_TST_AO;
                   ^V1124>UDD>TEST>JONES
 ARGS    $PRBLK    -XREF,-PRINT
```

REQUEST I/O

REQUEST I/O

Macro Call Name: $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 DESCRIPTION:

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 $IORBD or $IORB macro calls.

FUNCTION DESCRIPTION:

This 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 ($GTFIL) 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 ($GIFIL) macro call. The LRN returned by the $GIFIL call will be the LRN assigned to the device at system building time.

SEMAPHORE REQUEST BLOCK

SEMAPHORE REQUEST BLOCK

Macro Call Name: $SRB

Function Code: None

Equivalent Command: None

Generate a semaphore request block whose length is four words in SAF mode and five words in LAF mode.

FORMAT:

```
[label]  $SRB  [semaphore identifier],
               [issuing task suspension option],

                          or

               [termination action]
```

ARGUMENT DESCRIPTION:

semaphore identifier

A 2-character (ASCII) identifier that must have been defined by the task issuing the semaphore request. If this argument is omitted, the semaphore identifier 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 (set w-bit to 0)

NWAIT

Do not suspend the issuing task (set w-bit to 1)

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.

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

FUNCTION DESCRIPTION:

The semaphore request block (SRB) is used to request asynchronously the reservation of a resource controlled by the specified semaphore. The SRB contains a semaphore id which identifies the (previously defined) semaphore being requested.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

Example:

In this example, the $SRB 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

SEMAPHORE REQUEST BLOCK OFFSETS

Macro Call Name: $SRBD

Counterpart: $SRB (see "Semaphore Request Block")

Generated Label Prefixes:

	S_RRB/S_SEM
SRB label	offset 0
	S_CT1
	S_CT2
	S_ADR

See Appendix A for the format of the semaphore request block.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the *Assembly Language Reference* manual for more information about SAF/LAF independent code.

SET DIAL

Macro Call Name: $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 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.

FORMAT 1:

```
[label]  $SDL  [location of channel number],
               [location of address of telephone number],
               CHANNEL
```

FORMAT 2:

```
[label]  $SDL  [location of address of device pathname],
               [location of address of telephone number],
               [PATHNAME]
```

ARGUMENT DESCRIPTION:

location of channel number

Any address form valid for a data register; provides the four hexadecimal digits that define the 10-bit channel number of the data line. The channel number must be stored left-justified with low-order zero filling. (Applicable to format 1 only.)

location of address of telephone number

Any address form valid for an address register; provides the address of the telephone number to be associated with the data line. The telephone number must be stored as an aligned, nonvarying, 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, -, *. (Applicable to formats 1 and 2.)

$$\left\{\begin{array}{l}\text{CHANNEL}\\ \text{CHAN}\end{array}\right\}$$

Incicates that format 1 of the macro call is being used (channel number of line is provided).

location of address of device pathname

Any address form valid for an address register. For example, the device pathname could be >SPD>TTY1; see "Get File." The pathname must be stored as an aligned, nonvarying, character string containing at least one trailing space and no embedded spaces. (Applicable to format 2 only.)

$$\left\{\begin{array}{l}\text{PATHNAME}\\ \text{PATH}\end{array}\right\}$$

Indicates that format 2 of the macro call is being used (pathname of line is provided).

FUNCTION DESCRIPTION:

During system building, you can specify that the communications Auto Call Unit be applied to one or more communications lines. For each line that is to employ autodialing, you construct a list of telephone numbers. The first entry in this list is left empty by the system. The other entries are filled in according to your specifications.

The $SDL macro call allows you to dynamically insert a telephone number into the first entry in the list for a particular line. When the Auto Call Unit handler is invoked, this telephone number will be dialed first in the attempt to establish a connection with the terminal(s) on the line. If no successful connection is established, the next entry (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.)

NOTES:

1. For format 1, the channel number supplied by argument 1 is placed in $R6; if this argument is omitted, $R6 is assumed to contain the channel number.

2. The format 2, $R6 is cleared to zero and the address of the device pathname supplied by argument 1 is placed in $B2. If argument 1 is omitted, $B2 is assumed to contain the address of the device pathname.

3. For formats 1 and 2, the address of the telephone number supplied by argument 2 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the telephone number.

4. For format 1, CHANNEL (or CHAN) must be coded.

5. For format 2, all three arguments can be omitted. If this is done, $R6 is assumed to contain zeros, $B2 is assumed to contain the address of the device pathname, and $B4 is assumed to contain the address of the telephone number.

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

   0000 - No error

   0201 - Illegal pathname

   0701 - Channel not configured

   0702 - Auto Call Unit control unit not configured on this channel

   0703 - ACU in progress

   1704 - Illegal argument length

   170F - Invalid digit in telephone number

Example:

In this example, the terminal whose pathname is >SPD>TTY1 is to be automatically dialed using the number 1-617-555-4444.

```
DIALAA    $SDL    !PTNM,!NUM_12,PATH
            •
            •
            •
PTNM      TEXT    '>SPD>TTY1Δ'
NUM_12    TEXT    '16175554444Δ'
```

SET EXTERNAL SWITCHES

SET EXTERNAL SWITCHES

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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 switches are set on.

FUNCTION 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.

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.)

FUNCTION DESCRIPTION:

This 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 and 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 the issuing of multiple spawn task macro calls.

NOTES: 1. The address of the request block supplied by argument 1 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the request block.

2. The relative priority level supplied by argument 2 is placed in $R6; 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. Information derived from either argument is placed in $B2; if these arguments are omitted, $B2 is assumed to contain the start address within the bound unit.

4. 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 - Bound unit not found

   0602 - Insufficient memory

   0801 - Request block in use (T-bit on)

   0817 - Memory access violation on request block

   0827 - Bound unit is not a fixed relative file

   082D - Group available memory quota exceeded

   0E02 - No memory available for nonswappable task

   1604 - Unresolved symbolic start address

   160A - Insufficient memory

   1613 - Invalid bound unit pathname

   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).

–. On return, $R1, $R2, $R6, and $R7 contain the following information:

$R1 - Return status; one of the following:

0000 - No error
0606 - Illegal 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 $STMP 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     $R7 = 2500
        BCT      +$A
        ADV     $R6 -1
    $A  BGEZ    $R6,SOMMEM
           .
           .
           .
SOMMEM  $GMEM   =2500
```

SUSPEND GROUP

SUSPEND GROUP

Macro Call Name: $SUSPG

Function Code: 0D/08

Equivalent Command: Suspend Group (SSPG)

Suspend the specified task group.

FORMAT:

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

ARGUMENT DESCRIPTION:

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.

FUNCTION DESCRIPTION:

In MOD 400, this call causes the system to suspend the specified task group. The task group is marked as suspended when:

o All tasks of the group have exited from critical areas of the Monitor.

o All active task control blocks have been removed from their level queue.

o All external requests (system driver, clock, memory, semaphore) have been satisfied.

In MOD 600, this call suspends only the group request queue; a request that is active is allowed to go to completion.

A suspended task group can be activated through the $ACTVG macro call.

```
*
*      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 ITS BEFORE 0800 HOURS THE INTERNAL FORMAT
*      DATE/TIME IS CORRECT ELSE ITS 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

SWAP FILE

Macro Call Name: $SWFIL

Function Code: 10/5A

Equivalent Command:

Causes a simulated end-of-tape signal (output mode) or end-of-volume trailer (input mode). A continuation reel is then selected. If it is not online, a mount request occurs.

FORMAT:

```
                [label]  $SWFIL  [fib]
```

ARGUMENT DESCRIPTION:

fib

Any address form valid for an address register; provides the location of the 16 word file information block used in data and stroage management calls.

FUNCTION DESCRIPTION:

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.

NOTES: 1. The structure used for data management calls and storage management calls can also be used for the swap-file call.

2. This function is only meaningful for labeled tape files; it is ignored for other files.

3. The file must be opened for either data management or storage management. If the file is opened in output mode for data management, the following occurs:

   - End-of-volume trailer records (EOV1/EOV2) are written followed by two tape marks at the current tape position.

   If the tape files are opened in input mode for data management access, the following occurs:

   - The tape is rewound and unloaded and a normal reel swap is required. The reel with the next subsequent file section is expected.

   - 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.

   - Use of the swap-file call renders the FIB out-record-address returned on subsequent read operations meaningless. This field is set to the current relative record number for tape files.

   If the tape files are opened for storage management access, the following occurs:

   - The tape is rewound and cycled down.

   - The user is responsible for writing any trailer records and tape marks for output files reserved for device (volume) level access.

4. On return, $R1 contains the following status codes:

   01XX - Media error

   0205 - Illegal argument

   0206 - Unknown or illegal logical file number (LFN)

   0207 - LFN or file not open

SYSTEM ATTRIBUTE INFORMATION, GET (MOD 600 ONLY)

Macro Call Name: $SYSAT

Function Code: 14/11

Equivalent Command:

Provides the user with system attribute information about the software/hardware execution environment.

FORMAT:

```
[label]  $SYSAT  [location of marketing identifier string]
```

FUNCTION DESCRIPTION:

This call provides the user with the operating system identity and software/hardware attribute information.

NOTES: 1. The address of the receiving field for the marketing identifier supplied by argument 1 is placed in $B4. $B4 is assumed to contain the address of the receiving field if argument 1 contains: =$B4. 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 as follows:

- o 2 for MOD 200
- o 4 for MOD 400
- o 6 for MOD 600

$R6 - Provides hardware information as follows:

- o 3 for Model 3x
- o 4 for Model 4x and Model 5x

$R7 - Indicates the presence/absence of either a SIP or CIP context. If $R7 (12, '13) contains:

00 - No SIP context present; instructions not executable

X1 - SIP simulator present

1X - SIP hardware present

If $R7 (13, 14) contains:

00 - No CIP context present

X1 - CIP simulator present

1X - CIP hardware present

SYSTEM IDENTIFICATION

Macro Call Name: $SYSID

Function Code: 14/04

Equivalent Command: (MOD 600 only) USER SYSID

Returns 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 DESCRIPTION:

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..

FUNCTION DESCRIPTION:

This 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 address of the receiving system id field supplied by argument 1 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the address of the field.

$R1 - Return status; one of the following:

0000 - No error
0807 - No command input defined
0817 - Memory access violation

$B4 - Address of the receiving task group

TASK INFORMATION (MOD 600)

TASK INFORMATION (MOD 600 ONLY)

Macro Call Name: $TINFO

Function Code: 14/09

Equivalent Command: None

Returns a specific item of control information about the issuing task, depending on the argument value.

FORMAT:

```
[label] $TINFO [information code]
```

ARGUMENT DESCRIPTION:

information code

One of the following alphabetic character strings, or alternative numeric codes, specifying the item of information to be returned:

Alphabetic String	Numeric Code	Resulting Item of Information
CIN	0	Address of the task's command-in file control block (FCB)
UIN	1	Address of the task's user-in FCB
DIB	2	Address of task's command-in stream device interface module (DIM) interface block
CLL	3	Maximum length of command input line
LRN	4	Task's LRN
DB	5	Task's Debug indicator
LDT	6	Task's lead task indicator
PRV	7	Task's privileged task indicator

Alphabetic String	Numeric Code	Resulting Item of Information
STAD	8	Task's start address
ISTA	9	Task's initial start address

FUNCTION DESCRIPTION:

The call returns the requested item of control information about the issuing task; one execution of the macro call returns one information item. The information, returned according to the information code specified in the argument, is placed in $B4 or $R6.

Information resulting from strings/codes CIN/0, UIN/1, and DIB/2 is placed in $B4. Information from the remaining strings/codes is placed in $R6. For codes 5, 6, and 7 (indicators), $R6 contains zero when these indicators are not set, and a nonzero value when the indicators are set.

NOTES: 1. The string/code supplied by the argument is placed in $R2. When the argument is omitted, the system assumes that $R2 contains the appropriate string/code.

2. On return, $R1, $R6, and $B4 contain the following:

$R1 - 0000 - No error

$R6 - Returned value for code 3 through 7

$B4 - Returned address of the structure specified for codes 0 through 2.

Returned with the start or initial start address for codes 8 and 9, respectively.

Example:

The issuing task requests its LRN be returned.

```
GT_LRN    $TINFO    4
```

TASK REQUEST BLOCK

TASK REQUEST BLOCK

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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:

FUNCTION DESCRIPTION:

The task request block is used to communicate between tasks. It serves as the means by which arguments are passed between the requested and requesting tasks within a task group. When a previously created task is requested, the task request block contains the LRN (logical resource number) 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 portion that contains standard control information.

When a task is activated, its $B4 register points to offset 0 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 $TRB macro call when it has a parameter list pointer, or by placing that pointer at the $TRBD macro call's T_PRM offset.

Any task specific arguments are permitted (as if the TRB had been constructed by the command processor).

NOTES: 1. This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF.

2. In MOD 600, it is the user's responsibility to create task request blocks in an address space visible to both the requesting task and the requested task. Task request blocks created via the $GETMEM macro call are visible to all tasks in a task group.

Example:

In this example, the $TRB 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 will be 18 words long, have an LRN of 30, and, when its task terminates, will release semaphore AA.

```
        ATRBA    $TRB    30,,SM=AA,,5,XR643MX77B
```

TASK REQUEST BLOCK OFFSETS

TASK REQUEST BLOCK OFFSETS

Macro Call Name: $TRBD

Generated Label Prefixes:

	T_RRB/T_SEM
TRB label	offset 0
	T_CT1
	T_CT2
	T_ADR
	T_PRM

See Appendix A for the format of the task request block.

Description:

See the task request block macro call.

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

given condition is enabled (see enable user trap ($ENTRP) macro call). When there is no established trap handler for the specified trap condition, the system returns a zero (null) pointer.

NOTES: 1. The address for the trap handler for the specified trap condition is stored in the pointer provided by argument 1. When this argument is omitted, or is =$B4, the system does not store the pointer value; that value is available only in $B4.

2. The trap number of the designated trap condition, or the value -1, designated any connected trap handler, derived from argument 2 (which is mandatory) is placed in $R2.

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

$R1 - Return status code, one of:

0000 - No error
0342 - Invalid trap number

$B4 - Address of trap handler, or zero.

Example:

The macro call returns a pointer to the trap handler, previously established by the task for trap number 12. The pointer is returned only in $B4.

```
QRYAA    $TRPHD    ,=12
```

UNLOAD SHARABLE BOUND UNIT (MOD 600)

UNLOAD SHARABLE BOUND UNIT (MOD 600 ONLY)

Macro Call Name: $UNSBU

Function Code: 07/0E

Equivalent Command: Unload Sharable Bound Unit (UNLOAD_SH_BU)

Unload all sharable bound units that have a user count of zero.

NOTE: This macro routine is recommended for use only by specialized software system designers.

FORMAT:

```
                    [label]  $UNSBU
```

ARGUMENT DESCRIPTION:

None

FUNCTION DESCRIPTION:

This call unloads from memory all shareable bound units that have a user count of 0. The operating system will unload such bound units only if memory is needed. This call causes all shareable bound units, with no user, to be flushed out of memory.

NOTE: On return, $R1 contains one of:

0000 - No error; normal sharable bound units unloaded
083A - Use of privileged executive function attempted

USER IDENTIFICATION

Macro Call Name: $USRID

Function Code: 14/00

Equivalent Command: (MOD 600 only) USER ID

Returns the user identification of the calling task group to a 32-character, blank filled receiving field.

FORMAT:

```
[label]  $USRID  [location of user id field address]
```

ARGUMENT DESCRIPTION:

location of user id field address

Any address form valid for an address register; provides the address of a 32-character, aligned, nonvarying blank filled field, into which the system will place the user identification associated with the issuing task group.

FUNCTION DESCRIPTION:

This call returns the task group's user id to a field in the issuing task. The user identification will consist of person.account.mode. The unused portion of the field is blank filled. See the Operator's Guide for further details.

NOTES: 1. The address of the receiving user id field, supplied by argument 1, is placed in $B4; if this argument is omitted, $B4 is assumed to contain 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 $USRID macro call is issued to place the user identification of the task group in that field.

```
ID01      $USRID   !USIDFL
             .
             .
             .
USIDFL    RESV     16,A'ΔΔ'
```

USER INPUT

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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.

FUNCTION DESCRIPTION:

This call allows a task to read the next record from the current user-in file. Unless it has been changed by a new user-in ($NUIN) macro call, the user-in file is that file identified in the request group ($RQGRP) or enter batch request ($RQBAT) macro call.

NOTES:

1. The address of the record area supplied by argument 1 is placed in $B4; if this argument is omitted, $B4 is assumed to contain the record area address.

2. The record size supplied by argument 2 is placed in $R6; if argument 2 is omitted, $R6 is assumed to contain 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 begirns 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 in $R1. See the System Messages manual.

$R6 - Residual range (number of bytes not filled in input record area)

$R7 - File status/type (see "Command In")

$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 MESSAGE (MOD 600 ONLY)

Macro Call Name: $USMSG

Function Code: 17/00

Equivalent Command: Send Message (SEND_MSG or SM)

Send a message to another user task in another task group, which is identified by the group id specified in the issuing task's intergroup request block (IGRB).

FORMAT:

```
[label]  $USMSG  [location of intergroup request block address]
```

ARGUMENT DESCRIPTION:

location of intergroup request block (IGRB) address

Any address form valid for a data register; provides the address of the output IGRB that describes: (1) the group id of the task to which the message is to be sent, and (2), the location and range of the message.

FUNCTION DESCRIPTION:

The call allows a task to send a message to a task in another task group. The message must have read access, and its location and size specified in the IGRB. (Appendix A describes the intergroup request block (IGRB).)

The task group identified by the group id in the IGRB must have already been defined to the system. The IGRB must have write access.

The destination task group will receive the message only after it has issued a user response message ($USRSP) macro call.

WAIT LIST, GENERATE

Macro Call Name: $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 ]
```

ARGUMENT DESCRIPTION:

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 0 is specified before the last label is supplied, an address of 0 is generated for the wait list entry that corresponds to that argument position. See Appendix A for the format of the wait list.

FUNCTION 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 a 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).

NOTE: This macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

Example:

In this example, a $WLIST 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

Macro Call Name: $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]
```

ARGUMENT DESCRIPTION:

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 0 is specified before the last label is supplied, an address of 0 is generated for the wait list entry that corresponds to that argument position. See Appendix A for the format of the wait list.

FUNCTION DESCRIPTION:

This 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 supplied by argument 1 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 - Illegal 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.

3. If arguments are present, this macro call cannot be used in programs written in SAF/LAF independent code (SLIC). See the Assembly Language Reference manual for more information about SAF/LAF independent code.

POSFWD
PFD

(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.

POSBWD
PBD

(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.

FUNCTION DESCRIPTION:

Before this macro call can be executed, the LFN must have been opened (see the open file macro call) with a program view word that allows access via data management (bit 0 is 0) and allows write operations (bit 2 is 1). 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 specified) or logical end-of-file (if PRESERVE specified). The write pointer is modified by each write record operation.

The file information block can be generated by a $FIB macro call. Displacement tags for the FIB can be defined by the $FIBDM 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.
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.

0223 - File space limit reached or file not expandable

0224 - Directory space limit reached or not expandable

0227 - Index limit exceeded while loading an indexed file

022A - Record lock area overflow or not defined

022B - Requested record is locked or causes deadlock

0237 - Invalid record or control interval format

Example:

In this example, the FIB (i.e., MYFIB) described under "Assumptions for File System Examples" in Section 3 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)

SECTION 6

INPUT/OUTPUT DEVICE DRIVERS

This section describes the internal system software, known as device drivers, and some related data structures, that provide data transfer facilities for system and application programs with peripheral devices. Macro calls pertaining to standard system file input/output and to physical input/output are summarized in Section 2 and described in detail in Section 5.

INPUT/OUTPUT DRIVERS

Input/output peripheral drivers and the analogous communications device drivers (called line protocol handlers) perform all data transfers between a peripheral device and the system or application program that uses it. Drivers are provided for all Honeywell-supplied peripheral devices and the teleprinter, VIP, and BSC2780/3780 protocols.

The remainder of the section describes the peripheral device drivers. Line protocol handlers are described in the Communications Processing manual.

Applications programs can request the drivers directly or can use them indirectly by calling the file manager.

You select a driver and the priority level at which it executes at system building.

The input/output drivers are reentrant programs capable of supporting the concurrent operation of several devices of the same type. The driver runs at the priority level assigned to the particular device at system building. The drivers provide fully simultaneous operation of the central processor with multiple input/output operations. Device interrupts signal the termination of data transfers.

DEVICE DRIVER DATA STRUCTURES

Two data structures control the interactions among an application program, its device drivers, and the devices the program uses. The structures are the input/output request block (IORB) and the resource control table (RCT).

The IORB, which is partly described in this section and more fully in Appendix A, is the interface between the application task and its device driver, and is under user control.

The resource control table (RCT) is the interface between the driver and its device(s), and is not normally accessible to users of Honeywell-supplied drivers described in this section. The RCT is used by those who write their own device drivers; it is described in the Mod 400 and Mod 600 System Building manuals.

DEVICE DRIVER CONVENTIONS

The following conventions apply to all input/output device drivers.

- The I/O request block (IORB) is the standard control structure used by a driver (see "Data Structures," later in this section for definition).

- The $RQIO macro call is used to request a driver.

- The B4-register contains the address of the IORB supplied by the caller; the IORB contains the LRN of the device to be used.

- The I/O-specific words of the IORB (I_CT2 through I_DVS) are not modified by the driver.

- If a device becomes inoperable, it can be disabled with an operator command and another device can be substituted.

- Drivers are reentrant and interrupt driven; one driver supports many devices of the same type.

- Synchronous and asynchronous I/O are supported.

- The hardware status is always mapped into the software status word in the task's IORB (I_ST) before the driver relinquishes control.

Driver Functions and Function Codes

All drivers perform similar functions on behalf of the devices and application tasks they service. These functions are carried out by the driver's request processing and interrupt processing code.

The application task can request specific functions by providing a function code in the IORB it supplies when it requests I/O service. These specific function codes are summarized in Table 6-1 and discussed under the specific function heading in the following pages.

The application task uses the last four bits of the IORB entry I_CT2 to enter the function code for the functions summarized in Table 6-1.

Table 6-1. Input/Output Function Code

	Device					
IORB Function Code	ASR/KSR Keyboard Printer	Card Reader	Card Reader/ Punch	Printer	Disk	Magnetic Tape
0	Wait online	Wait online	Wait online	Wait online	Wait online	Wait online
1	Write	NA[a]	Write (punch)	Write	Write	Write
2	Read	Read	Read	NA	Read	Read
3	NA	NA	Write file mark (punch)	NA	NA	Write file mark
4	NA	NA	NA	NA	NA	Position block[b]
5	NA	NA	NA	NA	Format write	NA
6	NA	NA	NA	NA	Format read	Position file[c]
9	Break Notification	NA	NA	NA	NA	NA
A	Connect	Connect	Connect	Connect	Connect	Connect
B	Disconnect	Disconnect	Disconnect	Disconnect	Disconnect	Disconnect
E	NA	NA	NA	NA	Read disabled device	Read disabled device

[a] Not applicable.

[b] Positive range of one is forward space to start of next block.
Negative range of one is backspace to beginning of previous block.

[c] Positive range of one is forward space to next tape mark.
Zero range is backspace to previous tape mark.
Negative range of FFFF is rewind to BOT.
Negative range of FFFF is rewind to BOT and unload.

WAIT ONLINE FUNCTION (fc=0)

The "wait online" function, one element of a control mechanism used to synchronize task operation with device availability, allows a caller to wait until a device becomes ready for use, or until a specific time interval has passed.

All noncommunications devices (except KSR-like devices) generate interrupts when their availability changes. For example, when a printer runs out of paper, an interrupt is generated and the device is not ready for use; when the paper is installed and the device is again ready, another interrupt is generated.

When a driver receives a service request from a task using the "wait online" function code in the IORB that it supplies (0000 in the last four bits of I_CT2), and the device is not ready, the driver sets a timer for 5 minutes and suspends. When the driver is reactivated, either by a ready interrupt from the device or by a time-out, it deactivates the timer, checks the device-ready bit in the hardware status word and places a 0 or 6 value in the return status field of the IORB depending on the condition of that bit. See Table 6-2 and the return status codes for the $RQIO (Request I/O) macro call; the rightmost hexadecimal character is placed in the return status field.

The wait online function should not be issued to a device that is currently ready for use unless you expect it to become not ready before it becomes ready again (e.g., the operator has been instructed to change a volume mounted on a disk device currently in use).

WRITE FUNCTION (fc=1)

The write function is available for all devices except the card reader. This function allows the writing of data to a particular device. When a driver receives a write request, it transfers the indicated data from a user buffer to the device according to the specifications supplied in the task's IORB. *

READ FUNCTION (fc=2)

The read function is available for all devices except local and remote printers. This function allows reading data from a particular device. When a driver receives a read request, it transfers the data from the specified device to a user buffer according to the specifications supplied in the requesting task's IORB. *

Table 6-2. Return Status Codes (last digit)

Code Number (Hexadecimal)	Meaning
0	No error, operation complete
1	Request block already busy (T=1)
2	Invalid LRN
3	Illegal wait
4	Invalid parameters
5	Device not ready
6	Device time-out on other than connect
7	Hardware error, check IORB status word
8	Device disabled
9	File mark encountered
A	Controller unavailable
B	Device unavailable
C	Inconsistent request
10	Device time-out on connect

When these codes are found in I_CTl (IORB), or in $R1 on a resume after wait, look at I_ST (IORB) to identify the specific error. The status B is returned with every read or write IORB that has been aborted by a disconnect request with queue abort.

This status will be returned on an I/O request after an interrupt. The disks and tapes are disabled until the system's automatic volume recognition routine calls the enable device function.

This status indicates illogical peripheral driver requests: read or write before connect; duplicate connect or disconnect requests; write after disconnect.

READ DISABLED DEVICE FUNCTION (fc=E)

This function, available only to disk or magnetic tape devices, allows the driver to bypass the device-disabled test during validity checking.

This function is used by the system's automatic volume recognition (AVR) module, which recognizes the volume label of the volume on the disabled device, then enables the device so that attempts to read data from it can continue.

WRITE TAPE MARK FUNCTION (fc=3)

The write tape mark function, which is available to magnetic tape devices, allows you to put a mark block on a referenced magnetic tape.

POSITION BLOCK FUNCTION (fc=4)

The position block function, which is available to magnetic tape devices, allows you to position a referenced magnetic tape forward or backward one block.

FORMAT READ (fc=5)

The format read function, available only to disk and magnetic tape devices, allows you to read all identifier and data fields on a track. The read begins at the first sector following the index mark and proceeds in the order in which the identifiers are recorded.

FORMAT WRITE (fc=6)

The format write function, available only to disk device, allows you to format a disk device. The disk is partitioned into forty two equal length sectors starting at the index mark.

POSITION TAPE MARK FUNCTION (fc=6)

The position tape mark function, which is available to magnetic tape devices, allows you to:

- o Position a referenced magnetic tape forward to beyond the next tape mark.
- o Position a referenced magnetic tape backward to ahead of the current tape mark.
- o Rewind to BOT.
- o Rewind to BOT and unload.

BREAK NOTIFICATION FUNCTION (fc=9)

This function, available for any terminal device, is a request to notify the issuing task when a break occurs on a specific device. When a break does occur, the driver posts the break notification request and declares the device to be in break mode for the issuing task.

In break mode, all I/O requests issued from the "broken" task are rejected, i.e., posted without any data transfers being started. Execution of a subsequent break notification request will cause the driver to return to normal mode.

Communications Function Codes

The following function codes are for communications, and for interactive and noninteractive (such as card reader or printer) devices.

CONNECT FUNCTION (fc=A)

This function provides the logical and physical connection between an application program and a communications device. The function may be used for noncommunications devices for program compatibility; i.e., no matter how these devices are connected to the computer, all interactive KSR and KSR-like devices, and noninteractive devices such as card reader and printer, can be controlled by the same application program.

See the Communications Processing manual for descriptions of the connect function, and disconnect function (described below), as they pertain to communications devices.

DISCONNECT FUNCTION (fc=B)

This function code provides the logical (normal and abnormal) and physical disconnect between an application program and an interactive device. The function is processed as a no-op for noninteractive devices for program compatibility, i.e., a card reader or printer may be controlled by the same application program.

The disconnect function as a logical function indicates that use of the indicated device is terminated. Termination may be either normal or an abort of all queued read or write requests issued only by this user program.

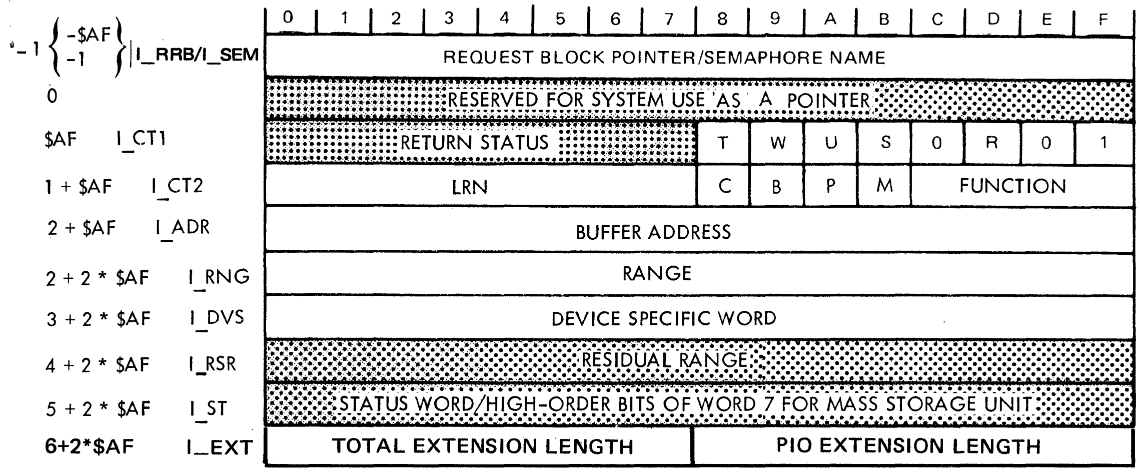
INPUT/OUTPUT REQUEST BLOCK

The input/output request block (IORB) contains all information that a task requesting an I/O service can specify to define the operation to be performed. In addition, it contains information returned by the driver to the requesting task concerning the outcome of its I/O request.

Figure 6-1 shows the format of a nonextended IORB. Unshaded fields must be initialized by the task requesting the I/O operation. The shaded fields are set by the driver in order to return information about the I/O request back to the caller, or are controlled by the Monitor.

Table 6-3 defines the specific IORB entries in a nonextended IORB. (See the Communications Processing manual for descriptions of IORB extensions.) Table 6-4 defines the software status word (I_ST) in the IORB. Device-specific IORB information is provided in the separate device driver descriptions later in this section.

NOTE: The labels (I_CT2, I_ADR, etc.) used in referring to the IORB entries are employed only for ease of presentation. The labels cannot be used for programming purposes.



		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
-1 { -$AF -1	I_RRB/I_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME															
0		RESERVED FOR SYSTEM USE AS A POINTER															
$AF	I_CT1	RETURN STATUS								T	W	U	S	0	R	0	1
1 + $AF	I_CT2	LRN								C	B	P	M	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 * $AF	I_ST	STATUS WORD/HIGH-ORDER BITS OF WORD 7 FOR MASS STORAGE UNIT															
6+2*$AF	I_EXT	TOTAL EXTENSION LENGTH								PIO EXTENSION LENGTH							

Figure 6-1. Format of I/O Request Block

Table 6-3. Contents of I/O Request Block

Item	Label	Bits	Contents
-$AF -1	I_RRB/ I_SEM	0 through 15 for SAF; 0 through 31 for LAF	Depending on the S- or R-bits of I_CT1, this word contains a task request block pointer (R-bit on), or a semaphore name (S-bit on); set by user, used by system at termination of request.
0	I_LNK	0 through 15; 0 through 31	Reserved for system use. 1- or 2-word pointer to indirect request block.
$AF	I_CT1	0 through 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. System controls this bit; user should not change it. In MOD 600, can be tested only with $TEST macro call.

Table 6-3 (cont). Contents of I/O Request Block

Item	Label	Bits	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. If W=0, then the D, R, and S bits may not be set.
		A (U)	User bit. User may or may not use this bit; system does not change it.
		B (S)	Release semaphore indicator. 0 = No semaphore in I_SEM, 1 = Release, on completion, semaphore item named in I_SEM.
		C	Must be zero.
		D (R)	Return IORB indicator. 0 = No request pointer in I_RRB. 1 = Dispatch task request block named in I_RRB after request timeout. 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 through 7	Logical resource number (LRN); identifies device to be used.
		8 (IBM)	IBM-type request. Changes interpretation of I_DVS to task word, and I_RSR and I_ST to configuration words A and B respectively.

Table 6-3 (cont). Contents of I/O Request Block

Item	Label	Bits	Contents
1+$AF (cont)	I_CT2 (cont)	9 (B)	Byte index; 0 = buffer begins in leftmost byte of word, 1=buffer begins in rightmost byte.
		A (P)	Reserved for system use.
		B (E)	Extended IORB indicator. 0 = Standard (nonextended) IORB. 1 = IORB extended to at least 6+2*$AF items. Set by user. (See I_EXT below.)
		C through F.	Function code. Driver function; See Table 6-1.
2+$AF	I_ADR	0 through 15	Buffer address, SAF mode. (See note.)
		0 through 31	Buffer address, LAF mode. 1- or 2-word pointer. (See note.)
* 2+2*$AF	I_RNG	0 through 15	Range - number of bytes to be transferred.
3+2+*$AF	I_DVS	0 through 15	Device-specific information.
4+2*$AF	I_RSR	0 through 15	Residual range. Indicates the number of bytes not transferred. Filled in by the system on completion of the order. Used by cartridge disk and mass storage unit driver as a data offset value on input.
5+2*$AF	I_ST	0 through 15	Modified device status; shows mapping of hardware status into software status format. See Table 6-4. Set by user as input field high order bits of sector number mass storage unit. Set by system after I/O completion.
* 6+2*$AF	I_EXT	0 through 7	Left byte: Number of words in the IORB extension, not including this I_EXT word.

Table 6-3 (cont). Contents of I/O Request Block

<table>
<tr><th>Item</th><th>Label</th><th>Bits</th><th>Contents</th></tr>
<tr><td>6+2*$AF
(cont)</td><td>I_EXT
(cont)</td><td>8 through 15</td><td>Right byte: Number of words 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 the GCOS 6 Communications Processing manual for description of IORB extensions.)</td></tr>
<tr><td colspan="4">NOTE: For break notification requests, the KSR driver gets the identification of the issuing task in this field. When break occurs, the contents of this field are transferred to the RCT.</td></tr>
</table>

Table 6-4. IORB Software Status Word[1] (I_ST)

Bit Position	ASR/KSR	Card Reader	Card Reader/Punch	Printer	Diskette	Cartridge Disk	Cartridge Module Disk and Disk Storage Unit	Magnetic Tape
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	Rewinding
2	Over/underrun	Over/underrun	Data service rate error	0	Over/underrun	Over/underrun	Over/underrun	Retryable error
3	Even parity error	Mark sense mode	Invalid ASCII code	End of form	Deleted field	Write protect error	Write/protect error	Write protect error
4	0	40-column	Punch echo or read registration	0	Read error	Read error	Read error	0
5	No stop bit	51-column mode	Light/dark check	0	Device fault	Illegal seek	Illegal seek	0
6	Long record	External clock track	Card jam	0	Missed data synchronization	Missed data synchronization	Missed data synchronization	BOT
7	Checksum error	Read check	0	0	Unsuccessful search	Unsuccessful search	Unsuccessful search	EOT
8	CC2 termination	ASCII code error	0	0	Two sided	Missed clock pulse	Missed clock pulse	Long record
9	CC3 termination	0	0	0	0	Missed sector pulse	Successful retry	Nonretryable error
10	0	0	0	0	Seek error	Seek error		0
11	0	0	0	0	0	0	0	Operation check
12	0	0	0	0	0	0	0	High density
13	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0
15	Fatal error	Fatal error	Fatal error	Fatal error	Fatal error	Fatal error	Fatal error	Fatal error

NOTES:

1. Nonexistent resource, bus parity, and uncorrected memory errors are combined into bit 15 of I_ST, but each occurrence is noted separately in the RCT.

2. The online drivers will flag, in the RCT, corrected memory errors and driver or hardware corrected errors.

[1] Equivalent to a modified status word 1.

CALLER INTERFACE WITH DEVICE DRIVER

To request execution of an I/O operation, the caller must issue a $RQIO macro call with $B4 pointing to the IORB that is to be serviced. If the IORB specifies synchronous I/O (W-bit reset), the issuing task will be suspended until the I/O operation is completed.

If IORB specifies asynchronous I/O, the instruction at the return point will be executed as soon as the system queues the IORB on the driver's level. The application may issue a $WAIT macro call when appropriate for the asynchronous request.

Thus, upon return from the driver at the completion of the I/O operation, the caller must check the R1 register first to see if the request was successful. Any interface error (illegal user argument) will be defined here. Hardware errors are defined in IORB entry I_ST (see Table 6-4).

Residual range denotes how much of the requested data transfer was actually performed. If I_RSR equals zero all data was transferred (see "Device Drivers" for details on device-specific basis). For an asynchronous request, register R1 should be checked on return; R1, I_ST and I_RSR should be checked after return from a $WAIT macro call.

Those fields not shaded in Figure 6-1 must be initialized by the task requesting the I/O operation. The remaining fields are set by the driver in order to return information about the I/O request back to the caller or are controlled by the Monitor. Table 6-3 describes the purpose of each field.

Other information needed to perform the I/O request is found in the IORB. The caller-supplied standard function code in I_CT2 is mapped by each driver into one or more device functions required to perform the actual request.

The LRN supplied by the caller in the IORB serves as a device identifier.

DEVICE DRIVERS

The remainder of this section discusses the device drivers in the following order:

- Card reader/Card reader-punch driver
- Printer driver
- Disk driver
- ASR/KSR driver
- Magnetic tape driver

Card Reader/Card Reader-Punch Driver

The card reader and card reader-punch devices are serviced by a single driver. The driver uses six function codes; i.e., read, write, write file mark (reader/punch only), connect, disconnect, and wait online. In addition, its IORB word I_DVS can be coded to define the character code of the input; namely, ASCII or verbatim. These values are specified in the IORB as defined in Table 6-6.

The translation/mapping of these codes from punched card format, into memory on reading, is described below.

In addition to the standard driver functionality discussed earlier, this driver also:

- Detects and discards unsolicited interrupts.

- Detects an end-of-file condition and sets the appropriate return status (ASCII GS character in column 1 of any card=EOF).

- Detects "device not ready" condition and sets appropriate error condition.

ASCII MODE

In this mode, punched cards are processed as shown in Figure 6-2. Each card column consisting of a 12-bit ASCII card code is converted into an 8-bit ASCII byte and stored in the main memory.

The ASCII card code table as specified in American National Standard X3.26 is given in Table 6-5. Note that no multiple punches in rows 1 through 7 are allowed and thus the 12-bit card code allows a maximum of 256 unique codes to be defined.

Translation is done by the card reader attachment which also provides a software-visible IORB status indicator that is set whenever an illegal ASCII card code is detected. This error condition is signaled by a 0107 in R1 register if any card column read had a hole pattern which was not one of the legal hole patterns given in Table 6-5. The illegal card code causes an ASCII-EO (all 1's) code to be loaded in the main memory.

Table 6-18. Cartridge Module Disk Status Code Mapping

Hardware Status	IORB I_ST	Meaning If Bit Set
0	-	
1	-	
2	2	Over-/underrun
3	3	Device fault
4	4	Read error
5	5	Illegal seek
6	6	Missed data synchronization
7	7	Unsuccessful search
8	8	Missed clock pulse
9	9	Successful recovery
10	10	Reserved
11	-	
12	-	
13	-	
14	-	
15	15	Fatal error

ASR/KSR Drivers

*

The keyboard/printer functions of an ASR are supported; the paper tape reader/punch functions are not. Thus, the K-bit within I_DVS word (Table 6-19) must be zero.

To examine the first character of a message sent in single character mode (from a local KSR terminal) before the rest of the message is transmitted, proceed as follows:

1. Issue a single character asynchronous read with no echo to the terminal.

2. When the read is completed, examine the character; then if the rest of the message is wanted, write the character to the terminal (with no carriage return or line feed).

3. Issue a read for the rest of the message (with echo).

Note that the operator terminal (keyboard/printer), when used, must be configured at LRN=0. For information about dialog with the operator's terminal, see the Operator's Guide.

Character codes, function codes, and device control availabl for the keyboard/printer are described below.

KEYBOARD INPUT

- Keyboard input is accepted until end-of-range, or carriage return, whichever occurs first. The carriage return character is not indicated as part of the input data.

- Keyboard control (line feed, carriage return, etc.) is definable in the IORB.

- Editing characters can control input:

  @ Deletes the previous character entered.

  CTL X Deletes all the previous characters entered on the same input line.

  (\) Character immediately following is treated as input.

NOTE: Since CAN is a nonprinting character, the *DEL* are displayed on a separate line when CAN is struck. Further input may begin after completion of the DEL output.

Causes character immediately following (@, CAN, CR, and \), to be treated as data input and not as editing characters; the back slash itself is not placed in memory.

PRINTER OUTPUT

- Printer output is accepted until end-of-range.

- Time-out period for keyboard/printer operation is 5 minutes.

ASR/KSR DEVICE-SPECIFIC IORB FIELDS

Table 6-19 shows the values of device-specific IORB fields for ASR/KSR devices.

Table 6-19. ASR/KSR IORB Fields

IORB Item	Field	Definition Keyboard/ Printer		Use

I_CT2	Function code	1=write
		2=read
		3=break notification
			A=Connect
			B=Disconnect
				Used by driver to complete the description of the requested I/O function.

Table 6-19 (cont). ASR/KSR IORB Fields

IORB Item	Field	Definition Keyboard/ Printer
I_DVS	Device-Specific	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 B 0 0 S F T 0 Q D K E L C M A H Bit Value B - (When function code is 9; i.e., break control request) 0 = Request break notification. 1 = Abort previous break notification requests. S - 0 = Function code 2 implies normal terminal read. 1 = Function code 2 implies single-character mode read. F - 0 = Assumes line printer format control. 1 = Assumes terminal format control. T - 0 = Use control characters in control functions. 1 = Treat all characters as data; bypass control character checks on input. Q - 0 = Stop output immediately on detecting "attention" when the detected character has No Stop bit status (e.g., a "break" key). 1 = Post "attention" and allow completion of output transfer. D - 0 = Read attention character with input (if present). 1 = Discard attention character on input. K - 0 = Transfer to keyboard/printer. (Must be 0.) E - 0 = Do not echo keyboard input. 1 = Echo keyboard input.

Table 6-19 (cont). ASR/KSR IORB Fields

IORB Item	Field	Definition Keyboard/ Printer	Use
I_DVS (cont)		Bit Value L - 0 = No line feed at end of transfer. 1 = Issue line feed after transfer. C - 0 = Issue carriage return after transfer. 1 = No carriage return after transfer. M - 0 = Transfer mode is 7-bit, with parity. 1 = Transfer mode is 8-bit direct transcription mode. A - In single-character mode: 0 = Do not abort previously buffered single-character mode characters in queue. 1 = Abort previously buffered single-mode characters in queue. A - On disconnect: 0 = Abort I/O requests on disconnect. 1 = Do no abort I/O requests on disconnect. H - 0 = Disconnect with phone hang up. 1 = Disconnect without phone hang up. NOTE: The MDC-connected ASR/KSR driver does not check this bit.	
I_ST	Software status word	Shown below	Mapped by driver from the hardware status in order to tell requesting task the hardware status of the I/O operation.

Table 6-21. Characteristics of Supported Tape Drives

Tape Drive Type	Speed (ips)		Density (bpi)				Parity		Mode	
	45	75	1600	800	556	200	Odd	Even	Packed	6-Bit
9-track NRZI	X	X	-	X	-	-	X	-	-	-
9-track PE	X	X	X	X	-	-	X	-	-	-
7-track NRZI[a]	X	X	-	X	X	X	X	X	X	X

[a] The application program must provide for tape positioning, creation and interpretation of labels, tape marks, control information, and data contents.

The driver provides the following callable functions:

- o Wait online
- o Write
- o Read (forward)
- o Position block (forward and backward)
- o Position forward or backward by tape mark, rewind to beginning of tape (BOT), rewind to BOT and unload.

The driver operates in the following modes:

- o Odd parity (9-track tape only)
- o Odd parity 6-bit (7-track tape)
- o Even parity 6-bit (7-track tape)
- o Packed, always odd parity (7-track tape)
- o Minimum data block, MDB (American National Standard specifies 18 or more characters per block in write, 8 or more in read)
- o MDB-inhibited (If fewer than the specified number of characters must be read or written, this mode is required.)

If MDB mode is specified for a write and the range is less than 18 characters, a parameter error is reported. If MDB mode is specified for a read and the range is less than 12 characters, the user will receive the first portion (requested range) of the first valid block and an unequal length check. If a "short record" is detected, a corrected media error is reported in status word, I_ST. If a record of less than 18 characters is written or less than 12 characters is read, the inhibit block size check bit (bit 12 of the device specific word, I_DVS) must be set.

Beginning of tape (BOT), end of tape (EOT), and end of file (EOF) conditions are reported for appropriate user action. If an error occurs in a case when the operation can be retried, the driver backspaces and reissues the order up to 32 times before reporting a hardware error. If an error occurs and no retry is possible, the driver rewinds and forward spaces to the problem block and reissues the order once before reporting a hardware error. The driver does not check the tape volume identifier.

The EOT return status is not returned for read operations; only the EOT status word bit is set. It is assumed that appropriate application software conventions will prevent reads that would force the tape off the end of the reel.

The resident magnetic tape driver is interrupt driven and must execute with a resident Monitor and with the central processor in the privileged state. It can support, on an adapter, one data transfer simultaneously with one or more rewind/rewind-unload orders.

MAGNETIC TAPE DEVICE-SPECIFIC IORB FIELDS

The IORB fields defined in Table 6-22 are specific to magnetic tape devices. All other IORB fields are defined in previous subsections.

Table 6-22. Magnetic Tape IORB Fields

Item	Field	Definition
I_CT2	Function code	0 = Wait online 1 = Write 2 = Read 3 = Write filemark 4 = Position by block (see range) 5 = Format read 6 = Position file (see range)

Table 6-22 (cont). Magnetic Tape IORB Fields

Item	Field	Definition
I_DVS	Device specific	0 12 13-15 0 0 0 0 0 0 0 0 0 0 0 0 I mode I: 0=Normal American National Standard block sizes 1=Inhibit sensing for American National Standard block size mode: 0 = 9-track tape; or 7-track in odd parity 6-bit mode 1 = 7-track tape in even parity 6-bit mode 2 = 7-track tape in packed mode
I_RNG	Range	Write: 1 through 7FFF Read: 0 means verify; 1 through 7FFF$_{16}$ is valid Position by block: Negative is backspace; 0 is illegal Positive is forward space Position by file: -2 = Rewind and unload -1 = Rewind to BOT 0 = Backspace to previous tapemark 1 = Forward space to tapemark
I_RSR	Residual range	Nonzero when physical block exceeds range.

A read with a range of zero verifies the selected sector with no data transfer to memory.

MAGNETIC TAPE HARDWARE/SOFTWARE STATUS CODE MAPPING

The hardware/software status code mapping for magnetic tape devices is shown in Table 6-23.

Table 6-23. Magnetic Tape Hardware/Software Status Code Mapping

RCT R_STTS	IORB I_ST	Meaning If Bit Set
0	-	Device ready
1	-	Attention
-	1	Rewinding
2	2	Error - Operation can be retried
3	-	MBZ
-	3	Write protected
4	-	Corrected media error
5	-	Tape mark
6	6	BOT
7	7	EOT
8	8	Unequal record length
9	9	Error - Operation cannot be retried
10	10	MBZ
11	11	Operation check
12	-	Corrected memory error
-	12	High density
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Memory error - correction impossible/fatal error

Table A-1. Contents of Clock Request Block

Item	Label	Bit(s)	Contents
-$AF	C_RRB/ C_SEM	0-15	Depending on the condition or the S or R bits of C_CT1, this word contains a task request block pointer (R-bit on), or a semaphore name (S-bit on).
0	C_LNK	0-15	Reserved for system use.
$AF	C_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. In MOD 600, can be tested only with $TEST macro call.
		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 MOD 600, user must not use this bit; in a user-built CRB, must be 0 initially.
		B(S)	Release semaphore indicator. 0 = No release, 1 = Release, on timeout, semaphore item named in C_SEM.
		C	Must be zero.
		D(R)	Return clock RB indicator. 0 = NO dispatch; 1 = Dispatch task request block named in C_RRB after request timeout. If 1, system executes $RQTSK, using C_RRB when the task terminates.
		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 A-1 (cont). Contents of Clock Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	C_CT1 (cont)	F	Implicit task start address. Must always be 1 for CRB.
1+$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 three 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

Figures A-3 and A-4 show the format, and Tables A-2 and A-3 show the contents, of the file information block (FIB) for data management (record level) access, and for storage management (block level) access, respectively.

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															
6	F_RFU1	Reserved															
7	F_IRT	Reserved															
8	F_ORT	Reserved															

Figure A-3. Format of FIB for Data Management

Table A-2 (cont). Contents of FIB for Data Management

Word	Label	Bits	Contents
11	F_IKF	0-7	Input key format: 0 for none specified 1 for primary key 2 for simple key
	F_IKL	8-15	Input key length.
12,13	F_ORA	0-31	Output record address.
14,15	F_RFU2	0-31	Reserved for later use, must be 00000000.

Table A-3. Contents of FIB for Storage Management

Word	Label	Bits	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 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.
5	F_BKSZ	0-15	Block size.
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 A-5 shows the format of a nonextended input/output request block (IORB) (See the GCOS 6 Communications Processing manual for descriptions of IORB extensions.) Table A-4 defines the specific fields in a nonextended IORB. Table A-5 summarizes the IORB fields for operator interface functions.

		0	1	2	3	4	5	6	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															
$AF	I_CT1	RETURN STATUS								T	W	U	S	0	R	D	1
1+$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 A-5. Format of I/O Request Block

Table A-4. Contents of I/O Request Block

Item	Label	Bit(s)	Contents
-$AF -1	I_RRB/ I_SEM	0-15 for SAF; 0-31 for LAF	Depending on the S or R bits of I_CT1, this word contains a task request block pointer (R-bit on), or a semaphore name (S-bit on). Set by user; used by system at termination of request.
0		0-15; 0-31	Reserved for system use. 1- or 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. In MOD 600, can be tested only with $TEST macro call.

Table A-4 (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.
		B(S)	Release semaphore indicator. 0 = No release, 1 = Release, on completion, semaphore item named in I_SEM.
		C	Must be zero.
		D(R)	Return IORB indicator. 0 = No dispatch. 1 = Dispatch task request block named in I_RRB after request timeout. 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.
		9(B)	Byte index: 0=buffer begins in leftmost byte of word, 1=buffer begins in rightmost byte.

Table A-4 (cont). Contents of I/O Request Block

Item	Label	Bit(s)	Contents
1+$AF (cont)	I_CT2 (cont)	A(P)	Private space; reserved for system use.
		B(E)	Extended IORB indicator. 0 = Standard (nonextended) IORB. 1 = IORB extended to at least 6+2*$AF items. Set by user. (See I_EXT below.)
		C-F	Function code. Driver or LPH function, see Table 6-1.
2+$AF	I_ADR	0-15 0-31	Buffer address, SAF Buffer address, LAF 1- or 2-word pointer.
2+2*$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 not 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*$AF	I_ST	0-15	Modified device status; shows mapping of hardware status into software status format. See Table 6-4. Set by user as input field high-order bits of sector number of mass storage unit. Set by system after I/O completion.
6+2*$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 toal extension length specified in the left byte (0-7).

Table A-4 (cont). Contents of I/O Request Block

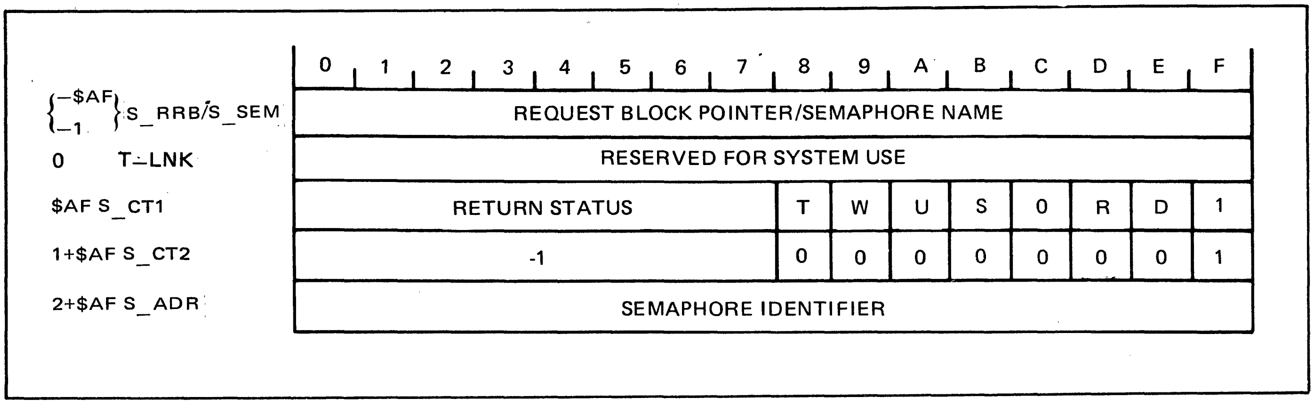
Item	Label	Bit(s)	Contents
6+2*$AF (cont)	I_EXT (cont)	8-15 (cont)	This word is present only when the B(E) bit in I_CT2 is 1. (See the GCOS 6 Communications Processing manual for description of IORB extensions.)

Table A-5. 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 A-6 shows the format of the semaphore request block; Table A-6 shows it contents.



	0 1 2 3 4 5 6 7	8	9	A	B	C	D	E	F
{−$AF −1} S_RRB/S_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME								
0 T–LNK	RESERVED FOR SYSTEM USE								
$AF S_CT1	RETURN STATUS	T	W	U	S	0	R	D	1
1+$AF S_CT2	-1	0	0	0	0	0	0	0	1
2+$AF S_ADR	SEMAPHORE IDENTIFIER								

Figure A-6. Format of Semaphore Request Block

Table A-6. Contents of Semaphore Request Block

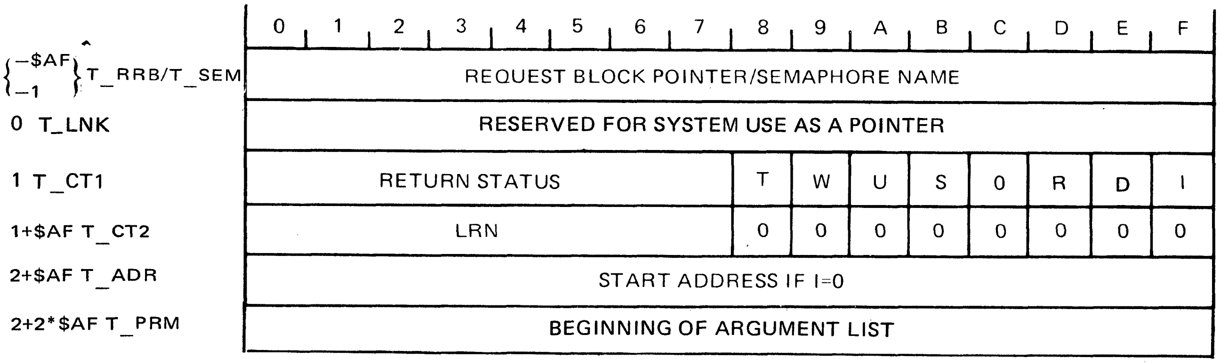
Item	Label	Bit(s)	Contents
-$AF -1	S_RRB S_SEM	0-15	Depending on the S or R bits of S_CT1, this word contains a task request block pointer (R-bit on), or a semaphore name (S-bit on). Set by user; used by system when request terminates.
0	S_LNK	0-15	Reserved for system use.
$AF	S_CT1	0-7	Return status
		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. In MOD 600, can be tested only with $TEST macro call.
		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. 0 = No release, 1 = Release, on completion, semaphore item named in S_SEM.

Table A-6 (cont). Contents of Semaphore Request Block

Item	Label	Bit(s)	Contents
$AF (cont)	S_CT1 (cont)	C	Must be zero.
		D(R)	Return semaphore RB indicator. 0 = no dispatch. 1 = Dispatch task request block named in S_RRB after request time-out. If 1, system executes $RQTSK, using S_RRB when the task terminates.
		E(D)	Delete SRB indicator, used usually with the B(S) and D(R) bits. 0 = No delete. 1 = Delete and when task terminates, return memory to pool where SRB is first entry of its memory block.
		F	Implicit task start address. Must always be 1 for SRB.
1+$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 A-7 shows the format of the task request block; Table A-7 shows its contents.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
{−$AF / −1} T_RRB/T_SEM	REQUEST BLOCK POINTER/SEMAPHORE NAME															
0 T_LNK	RESERVED FOR SYSTEM USE AS A POINTER															
1 T_CT1	RETURN STATUS								T	W	U	S	0	R	D	I
1+$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 A-7. Format of Task Request Block

Table A-7. Contents of Task Request Block

Item	Label	Bit(s)	Contents
-$AF -1	T_RRB/ T_SEM	0-15	Depending on the condition of the S or R bits of T_CT1, this word contains a task request block pointer (R-bit on), or a semaphore name (S-bit on). Set by user, used by system when request terminates.
0	T_LNK	0-15	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. In MOD 600, can be tested only with $TEST macro call.
		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.
		B(S)	Release semaphore indicator. 0 = No release, 1 = Release, on completion, semaphore item named in T_SEM.
		C	Must be zero.
		D(R)	Return task RB indicator. 0 = No dispath. 1 = Dispatch task request block named in T_RRB after request timeout. If 1, system executes $RQTSK, using T_RRB when the task terminates.
		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.

Table A-8 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
$AF	MC_MAJ		Major status.
		0-7	Left byte: 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. In MOD 600, can be tested only with $TEST macro call.
		9(W)	Wait bit - set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. Values: 0=No release, 1=Release, on closeout, of semaphore which must be in MC_OS -1.
		C	Must be zero.
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block whose address must be contained in MC_OS -$AF, after closeout of this request. System executes $RQTSK using the address of the request block contained in MC_OS -$AF upon request termination.
		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 A-8 (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_BIF	Address	Pointer
		0-15 (SAF)	Buffer pointer.
		0-31 (LAF)	Buffer pointer.
2+2*$AF	MC_BSZ	0-F	Buffer range.
3+2*$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*$AF	MC_RSR	0-F	Residual range.
5+2*$AF	MC_MRU	0-7	Left byte: end message recovery unit (MRU). Reserved for system use.
	MC_WTI	8-F	Right byte: wait test indicator.
			00 = Return null value to application.
			01 = Wait

Table A-8 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
6+2*$AF	MC_EXT		Extension mechanism.
		0-7	Left byte: binary value of 13+2*$AF, i.e., number of words in MGCRB following the extension word.
			Right byte: must be hexadecimal 7.
7+2*$AF	Next seven words.		Reserved for system physical I/O use.
14+2*$AF	MC_FNC	0-7	Left byte: function. Reserved for system use.
	MC_REV	8-F	Right byte: revision. Must be hexadecimal 1.
15+2*$AF	MC_MGI		Message group i.d.
		0-F	Returned in the $MINIT and $MACPT macro calls.
16+2*$AF	MC_LVL		Enclosure level.
		0-7	Left byte: enclosure level requested.
		8-F	Right byte: enclosure level detected according to 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	Address	Name-list pointer.
		0-15 (SAF) 0-31 (LAF)	Must be 0. Must be 0.
18+3*$AF	MC_TGI	0-F	Reserved for system use.

Table A-8 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
19+3*$AF	MC_TSK	Address	Pointer.
		0-15 (SAF) 0-31 (LAF)	Reserved for system use. Reserved for system use.
19+4*$AF	MC_NPI	0-F	Must be 0.

Table A-9. Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
0	MI_OS	Address	Pointer: reserved for system use.
		0-15 (SAF) 0-31 (LAF)	
$AF	MI_MAJ		Major status.
		0-7	Left byte: 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. In MOD 600, can be tested only with $TEST macro call.
		9(W)	Wait bit - set if the requesting task is not 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	Must be zero.

Table A-9 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
$AF (cont)	MI_MAJ (cont)	D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block whose address must be contained in MC_OS -$AF, after closeout of this request. System executes $RQTSK using the address of the request block contained in MC_OS -$AF upon request termination.
		E(D)	Delete I/O request block. Values: 0=No delete, 1=Delete, and return memory to the pool where MGIRB is the first entry of its memory block.
		F	I/O bit. Must be set.
1+$AF	MI_OPT		General options.
		0-7 8-A B C-F	Reserved for system use. Must be 0. Must be 1 (extended MGIRB). Must be 0.
2+$AF	MI_BUF	Address 0-15 (SAF) 0-31 (LAF)	Pointer. Must be 0. Must be 0.
2+2*$AF	MI_BSZ	 0-F	Buffer range. Must be 0.
3+2*$AF	MI_MPD	0-F	Message path description identifier. Must be hexadecimal 1.
4+2*$AF	MI_RSR	 0-F	Residual range. Reserved for system use.
5+2*$AF	MI_MDE MI_IOP	0-7 8-F	Left byte: must be 0. Right byte: must be 0.
6+2*$AF	MI_EXT		Extension mechanism.
		0-7	Left byte: binary value of 31+2*$AF, i.e., number of words in MGIRB following the extension word.
		8-F	Right byte: must be hexadecimal 7.

Table A-9 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
7+2*$AF	Next seven words. Reserved for system physical I/O use.		
14+2*$AF	MI_FNC	0-7	Function. Left byte: reserved for system use.
	MI_REV	8-F	Revision. Right byte: must be hexadecimal 1.
15+2*$AF	MI_MGI	0-F	Message group i.d. 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*$AF	MI_ADT	0-7 8-F	Address type. Left byte: address type (initiator); must be hexadecimal 1. Right byte: address type (acceptor); must be hexadecimal 1.
19+2*$AF	MI_NWI	0-F	Must be 0.
20+2*$AF	MI_NDI	0-F	Must be 0.
21+2*$AF	MI_MBI (Six words)	0-F 0-F 0-F 0-F 0-F 0-F	Initiator mailbox name. Must be from 1 to 12 ASCII characters, blank-filled, left justified.

Table A-9 (cont). Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
27+2*$AF	MI_NWA	0-F	Must be 0.
28+2*$AF	MI_NDA	0-F	Must be 0.
29+2*$AF	MI_MBA		Acceptor mailbox name.
	(Six words)	0-F	Must be from 1 to 12 ASCII characters, blank-filled, left justified.
		0-F	
		0-F	
		0-F	
		0-F	
		0-F	
35+2*$AF	MI_QSZ	0-F	Initiator - maximum size of quarantine unit.
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*$AF	MI_TSK	Address	Pointer. Reserved for system use.
38+3*$AF	MI_SIP	Address	Reserved for system use.

Table A-10 (cont). Message Group Recovery Request Block (MGRRB)

Item	Label	Bit(s)	Contents
$AF	MR_MAJ		Major status.
		0-7	Left byte: 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. In MOD 600, can be tested only with $TEST macro call.
		9(W)	Wait bit - set if the requesting task is not to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. Values: 0=No release, 1=Release, on closeout, of semaphore which must be in MC_OS -1.
		C	Must be zero.
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block whose address must be contained in MC_OS -$AF, after closeout of this request. System executes $RQTSK using the address of the request block contained in MC_OS -$AF upon request termination.
		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.

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					Status	Overlay status	Offset	Size		Base address		
$OVRSV	07/05	Overlay id	Offset			Overlay area table address	Status	Overlay id				Overlay area table address		
$OVRLS	07/06	$B5 = Return point address					Code 0006							
$OVRCL	07/07	Overlay id	Offset			Request block address	Status	Overlay id	Offset			Request block 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			Base address	Status							
$DLOAT	07/0D					Overlay area table address	Status							
$UNSBU	07/0E						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	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	Status		Range			Address record area		

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.)														
$NUIN	08/04	0,1, or 2				Address pathname	Status		Record length	File Type		Address pathname		
$NUOUT	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		
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						
$SGTRP	0A/03	Task LRN	Trap number				Status							
$TRPHD	0A/04	Trap number					Status					Address trap handler		
External Switch Functions														
$RDSW	0B/00	Mask						Value switch word						
$SETSW	0B/01	Mask						Value switch word						

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.)														
$ABGRQ	0D/07	Group id	Abort				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							
$BYE	0D/0B	0					Status							
$BYE	0D/0C	2					Status							
$BYE	0D/0D	1					Status							
$CUSID	0D/0E					New user-id	Status							
Batch Functions														
$RQBAT	0E/00	B5 = Address fixed parameter block				Address argument list	Status							
Error Handling Function														
$RPTER	0F/00	R2 = Component error code B3 = Expansion text address	Size	Code			Status			Code				
File Management Functions														
$ASFIL	10/10					Address argument structure	Status							

	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.)														
$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							
$RLFIL	10/35					Address argument structure	Status							
$RNFIL	10/40					Address argument structure	Status							
$STTY	10/45					Address argument structure	Status							
$OPFIL	10/50, 10/51					Address FIB	Status							
$SWFIL	10/5A					Address FIB	Status					Address FIB		
$CLFIL	10/55, 10/56, 10/57					Address FIB	Status							
$GIFIL	10/60					Address argument structure	Status							
$TIFIL	10/62					Address FIB	Status							
$TOFIL	10/63					Address FIB	Status							

	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							
$CRDIR	10/A0					Address argument structure	Status							
$DLDIR						Address receiving field	Status							
$DLFIL						Address receiving field	Status							
$CWDIR	10/B0					Address argument structure	Status							
$GWDIR	10/C0					Address argument structure	Status							
$XPATH	10/D0					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							
$RWREC	11/40, 11/41					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
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							
$PERID	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							
$INSID	14/05					Address receiving field	Status							
$BUID	14/06					Address receiving field	Status					Address receiving field		
$ENTID	14/07		Entry point name length			Address entry point name	Status				Address entry point			

		Contents Before Execution					Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Identification and Information Functions (cont)														
$GRPID	14/08					Address user id	Status		Group id					
$TINFO	14/09	Information code					Status		Information value			Address		
$SYSAT	14/11							Operating system identity (2,4 or 6)	Hardware information (3 or 4)	SIP or CIP context information				
$GINFO	14/0A	Information code					Status		Information value			Address		
$HDIR	14/0B					Address receiving field	Status					Address receiving field		
$TGIN	14/0C					Address receiving field	Status					Address receiving task group		
$GRPST	14/0E	Group id				Address receiving field	Status					Address receiving field		
Intergroup Message Facility Functions														
$MACPT	15/01					Request block address	Status					Request block address		
$MINIT	15/02					Request block address	Status					Request block address		
$MRECV	15/03					Request block address	Status					Request block address		
$MTMG	15/04					Request block request	Status					Request block address		

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Intergroup Message Facility Functions (cont)														
$MSEND	15/05					Request block address	Status					Request block address		
$MCMG	15/07					Request block address	Status					Request block address		
Intergroup and User Terminal Functions														
$USMSG	17/00					Address IGRB	Status					Address IGRB		
$USRSP	17/01					Address IGRB list	Status							
$RQTML	17/03					Request block address	Status					Request block address		
$RLTML	17/04		LRN	Release status code			Status							
Communications Function														
$SDL	1B/00		Channel number or 0		Address device pathname	Address telephone number	Status							
Accounting Functions														
$ACTR1	1E/01					Address receiving area	Status		Record length					
$ACTW1	1E/02		Record length			Record Address	Status							
$ACTD1	1E/03						Status							
$ACTOP	1E/04						Status							
$ACTCL	1E/05						Status							
$ACUPD	1E/06	Field index value	Update value				Status							

CUT ALONG LINE

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

TITLE	SERIES 60 (LEVEL 6) GCOS 6 SYSTEM SERVICE MACRO CALLS ADDENDUM A	ORDER NO.	CB08-02A
		DATED	JULY 1979

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be promptly investigated by appropriate technical personnel and action will be taken as required. If you require a written reply, check here and furnish complete mailing address below. □

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

____

CUT ALONG LINE

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

FOLD ALONG LINE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

FOLD ALONG LINE

Honeywell