

# Honeywell



**LEVEL 6**

SOFTWARE

**SYSTEM SERVICE  
MACRO CALLS**

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 0 of the manual dated January 1978. Except in revised Section 6 and Appendixes A, B and C, change bars indicate new and changed technical information, asterisks denote deletions. Major additions include message group macro calls, new overlay macro calls, and error logging macro calls, as well as descriptions of corresponding data structures.

**SOFTWARE SUPPORTED**

This manual supports Release 0110 of the Series 60 (Level 6) GCOS 6 MOD 400 Operating System. See the Manual Directory of the latest *GCOS 6 MOD 400 System Concepts* manual (Order No. CB20) for information as to later releases supported by this manual.

**ORDER NUMBER**

CB08, Rev. 1

July 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 and how to write a user device driver.

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.

Section 7 discusses trap handling for hardware and software traps.

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

## MANUAL DIRECTORY

The following publications comprise the GCOS 6 manual set. The Manual Directory in the latest GCOS 6 MOD 400 System Concepts manual (Order No. CB20) lists the current revision number and addenda (if any) for each manual in the set.

<u>Order Number</u>	<u>Manual Title</u>
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
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
CB25	GCOS 6 MOD 400 FORTRAN Reference
CB26	GCOS 6 MOD 400 Entry-Level COBOL Reference
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



<u>Order Number</u>	<u>Manual Title</u>
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	IBM 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

The following documents provide general hardware information:

<u>Order Number</u>	<u>Manual Title</u>
AS22	Honeywell Level 6 Minicomputer Handbook
AT04	Level 6 System and Peripherals Operation Manual
AT97	MLCP Programmer's Reference Manual
FQ41	Writable Control Store User's Guide

## CONTENTS

	Page
Section 1	
Introducing and Using the Manual ....	1-1
How to Use the Manual .....	1-1
Macro Call Syntax .....	1-1
Register Conventions and Contents .	1-2
Addressing Conventions .....	1-3
Register Contents at Task	
Activation .....	1-5
Register Contents at Initial Task	
Activation .....	1-5
Return Status Codes in \$R1	
Register .....	1-6
System Service Macro Calls and	
Function Codes .....	1-6
Location of Macro Routines .....	1-6
Task Request Queues .....	1-16
Section 2	
Monitor Service Functions .....	2-1
Batch Functions .....	2-1
Clock Functions .....	2-2
Communications Functions .....	2-2
Date/Time Functions .....	2-3
Error Handling Function .....	2-3
External Switch Functions .....	2-3
Identification and Information	
Functions .....	2-4
Memory Allocation Functions .....	2-4
Message Facility Functions .....	2-5
Operator Interface Functions .....	2-6
Overlay Handling Functions .....	2-6
Physical I/O Functions .....	2-7
Request and Return Functions .....	2-8
Secondary User Terminal Functions .	2-8
Semaphore Functions .....	2-9
Standard System File I/O	
Functions .....	2-10
Task Control Functions .....	2-11
Task Group Control Functions .....	2-12
Trap Handling Functions .....	2-13



## CONTENTS (cont)

		Page
Section 3	File System Functions .....	3-1
	File Information Block (FIB) .....	3-2
	Program View Entry in the FIB ...	3-6
	Offsets Definitions .....	3-11
	Assumptions for File System	
	Examples .....	3-13
	File Management Functions .....	3-14
	Data Management Functions .....	3-16
	Storage Management Functions .....	3-18
	Section 4	Data Structure Generation .....
Monitor Services Data Structures ..		4-1
Request Blocks .....		4-1
Parameter Block and Wait List ...		4-3
Request Block Offsets .....		4-4
File System Data Structures .....		4-5
File Information Block (FIB) ....		4-5
Offsets Definitions .....		4-5
Section 5	Macro Routine/Call Descriptions .....	5-1
	Abort Group (\$ABGRP) .....	5-2
	Abort Group Request (\$ABGRQ) .....	5-4
	Account Identification (\$ACTID) ...	5-6
	Activate Group (\$ACTVG) .....	5-8
	Associate File (\$ASFIL) .....	5-10
	Bound Unit Identification (\$BUID) .	5-13
	Cancel Clock Request (\$CNCRQ) .....	5-15
	Cancel Request (\$CANRQ) .....	5-17
	Cancel Semaphore Request (\$CNSRQ) .	5-19
	Change Working Directory (\$CWDIR) .	5-21
	Clean Point (\$CLPNT) .....	5-24
	Clear External Switches (\$CLRSW) ..	5-26
	Clock Request Block (\$CRB) .....	5-28
	Clock Request Block Offsets	
	(\$CRBD) .....	5-31
	Close File (\$CLFIL) .....	5-32
	Command In (\$CIN) .....	5-36
	Command Line Process (\$CMDLN) .....	5-39
	Console Message Suppression	
	(\$CMSUP) .....	5-42
	Create Directory (\$CRDIR) .....	5-44
	Create File (\$CRFIL) .....	5-47
	Create File Parameter Structure	
	Block - Offsets (\$CRPSB) .....	5-55
	Create Group (\$CRGRP) .....	5-56
	Create Overlay Area Table	
	(\$CROAT) .....	5-60
Create Task (\$CRTSK) .....	5-63	

CONTENTS (cont)

	Page
Section 5 (cont)	
Define Semaphore (\$DFSM) .....	5-67
Delete Group (\$DLGRP) .....	5-71
Delete Record (\$DLREC) .....	5-73
Delete Task (\$DLTSK) .....	5-76
Disable Device on Attention (\$DSDV) .....	5-78
Disable User Trap (\$DSTRP) .....	5-80
Dissociate File (\$DSFIL) .....	5-82
Enable Device (\$ENDV) .....	5-84
Enable User Trap (\$ENTRP) .....	5-86
Error Logging, End (\$ELEND) .....	5-88
Error Logging Information, Exchange (\$ELEX) .....	5-90
Error Logging Information, Get (\$ELGT) .....	5-94
Error Logging, Start (\$ELST) .....	5-96
Error Out (\$EROUT) .....	5-99
Expand Pathname (\$XPATH) .....	5-102
External Date/Time, Convert To (\$EXTDT) .....	5-105
External Time, Convert To (\$EXTIM) .....	5-108
File Information Block (\$FIB) .....	5-111
File Information Block Offsets (\$TFIB) .....	5-119
Get Date/Time (\$GDTM) .....	5-121
Get File (\$GTFIL) .....	5-124
Get File Parameter Structure Block Offsets (\$GTPSB) .....	5-142
Get File Information (\$GIFIL) .....	5-143
Get File Information, File Attribute Block Offsets (\$GIFAB) .....	5-153
Get File Information, Key Descrip- tors Block Offsets (\$GIKDB) .....	5-154
Get File Information, Parameter Structure Block Offsets (\$GIPSB) .....	5-155
Get Memory/Get Available Memory (\$GMEM) .....	5-156
Get Working Directory (\$GWDIR) ....	5-161
Home Directory (\$HDIR) .....	5-163
Input/Output Request Block (\$IORB) .....	5-165
Input/Output Request Block Offsets (\$IORBD) .....	5-168
Internal Date/Time, Convert To (\$INDTM) .....	5-169



CONTENTS (cont)

	Page
Section 5 (cont)	
Message Group, Accept (\$MACPT) ....	5-172
Message Group Control Request Block (\$MGCRB) .....	5-175
Message Group Control Request Block Offsets (\$MGCRT) .....	5-180
Message Group, Count (\$MCMG) .....	5-181
Message Group Initialization Request Block (\$MGIRB) .....	5-184
Message Group Initialization Request Block Offsets (\$MGIRT) ..	5-188
Message Group, Initiate (\$MINIT) ..	5-189
Message Group, Receive (\$MRECV) ...	5-192
Message Group Recovery Request Block (\$MGRRB) .....	5-196
Message Group Recovery Request Block Offsets (\$MGRRT) .....	5-200
Message Group, Send (\$MSEND) .....	5-201
Message Group, Terminate (\$MTMG) ..	5-205
Mode Identification (\$MODID) .....	5-208
New Process (\$NPROC) .....	5-210
New User Input (\$NUIN) .....	5-211
New User Output (\$NUOUT) .....	5-213
Open File (\$OPFIL) .....	5-215
Operator Information Message (\$OPMSG) .....	5-222
Operator Response Message (\$OPRSP) .....	5-225
Overlay Area, Release (\$OVRLS) ....	5-228
Overlay Area Reserve, and Execute Overlay (\$OVRSV) .....	5-230
Overlay, Execute (\$OVEXC) .....	5-234
Overlay, Load (\$OVLD) .....	5-237
Overlay Release, Wait, and Recall (\$OVRCL) .....	5-240
Overlay Status (\$OVST) .....	5-244
Overlay, Unload (\$OVUN) .....	5-247
Parameter Block (\$PRBLK) .....	5-250
Person Identification (\$PERID) ....	5-252
Read Block (\$RDBLK) .....	5-254
Read External Switches (\$RDSW) ....	5-258
Read Record (\$RDREC) .....	5-260
Release Directory (\$RLDIR) .....	5-266
Release File (\$RLFIL) .....	5-269
Release Semaphore (\$RLSM) .....	5-272
Release Terminal (\$RLTML) .....	5-274
Remove File (\$RMFIL) .....	5-276
Rename File/Rename Directory (\$RNFIL) .....	5-279

CONTENTS (cont)

	Page
Section 5 (cont)	
Report Error Condition (\$RPTER) ...	5-282
Request Batch (\$RQBAT) .....	5-285
Request Clock (\$RQCL) .....	5-288
Request Group (\$RQGRP) .....	5-290
Request I/O (\$RQIO) .....	5-294
Request Semaphore (\$RQSM) .....	5-297
Request Task (\$RQTSK) .....	5-300
Request Terminal (\$RQTML) .....	5-303
Reserve Semaphore (\$RSVSM) .....	5-306
Reset Device Attention (\$RDVAT) ...	5-309
Return (\$RETRN) .....	5-311
Return Memory/Return Partial Block of Memory (\$RMEM) .....	5-313
Return Request Block Address (\$RBADD) .....	5-316
Rewrite Record (\$RWREC) .....	5-318
Semaphore Request Block (\$SRB) ....	5-321
Semaphore Request Block Offsets (\$SRBD) .....	5-323
Set Dial (\$SDL) .....	5-324
Set External Switches (\$SETSW) ....	5-328
Set Terminal Characteristics (\$STTY) .....	5-330
Spawn Group (\$SPGRP) .....	5-333
Spawn Task (\$SPTSK) .....	5-339
Status Memory Pool (\$STMP) .....	5-343
Suspend Group (\$SUSPG) .....	5-345
Suspend for Interval (\$SUSPN) ....	5-347
Suspend Until Time (\$SUSPN) .....	5-350
System Identification (\$SYSID) ....	5-353
Task Group Input (\$TGIN) .....	5-355
Task Request Block (\$TRB) .....	5-357
Task Request Block Offsets (\$TRBD) .....	5-361
Terminate Request (\$TRMRQ) .....	5-362
Test Completion Status (\$TEST) ....	5-365
Test File (\$TIFIL) .....	5-367
Test File (\$TOFIL) .....	5-367
Trap Handler Connect (\$TRPHD) ....	5-370
User Identification (\$USRID) .....	5-373
User Input (\$USIN) .....	5-375
User Output (\$USOUT) .....	5-378
Wait (\$WAIT) .....	5-381
Wait Block (\$WTBLK) .....	5-383
Wait File (\$WIFIL) .....	5-385
Wait File (\$WOFIL) .....	5-385
Wait List, Generate (\$WLIST) .....	5-388
Wait on Request List (\$WAITL) .....	5-390



CONTENTS (cont)

		Page
Section 5 (cont)	Write Block (\$WRBLK) .....	5-393
	Write Record (\$WRREC) .....	5-397
Section 6	Input/Output Device Drivers .....	6-1
	Input/Output Drivers .....	6-1
	Device Driver Conventions .....	6-2
	Driver Functions and Function Codes .....	6-2
	Wait Online Function (fc=0) ...	6-4
	Write Function (fc=1) .....	6-4
	Read Function (fc=2) .....	6-5
	Read Disabled Device Function (fc=E) .....	6-5
	Write Tape Mark Function (fc=3) .....	6-6
	Position Block Function (fc=4) .....	6-6
	Position Tape Mark Function (fc=6) .....	6-6
	Break Notification Function (fc=9) .....	6-6
	Communications Function Codes ...	6-6
	Connect Function (fc=A) .....	6-7
	Disconnect Function (fc=B) ....	6-7
	Data Structures .....	6-7
	Input/Output Request Block .....	6-7
	Resource Control Table (RCT) ....	6-12
	Caller Interface With Device Driver .....	6-14
	Device Drivers .....	6-16
	Card Reader/Card Reader-Punch Driver .....	6-16
	ASCII Mode .....	6-17
	Verbatim Mode .....	6-19
	Card Reader/Card Reader-Punch Device-Specific IORB Fields .	6-19
	Card Reader/Card Reader-Punch Device-Specific RCT Fields ..	6-20
	Card Reader/Card Reader-Punch RCT/IORB Status Code Mapping .....	6-20
	Printer Driver .....	6-22
	Print Control Byte .....	6-22
	Printer Device-Specific IORB Fields .....	6-24
	Printer Device-Specific RCT Fields .....	6-25

CONTENTS (cont)

	Page
Section 6 (cont)	
Printer RCT/IORB Hardware/ Software Status Code Mapping .....	6-25
Disk Driver .....	6-26
Disk Driver Processing for Diskette .....	6-26
Diskette Device-Specific IORB Fields .....	6-27
Diskette Device-Specific RCT Fields .....	6-28
Diskette RCT/IORB Hardware/ Software Status Code Mapping .....	6-28
Disk Driver Processing for Cartridge Disk .....	6-29
Cartridge Disk Device- Specific IORB Fields .....	6-29
Cartridge Disk Device- Specific RCT Fields .....	6-30
Cartridge Disk RCT/IORB Hardware/Software Status Code Mapping .....	6-31
Disk Driver Processing for Mass Storage Unit .....	6-32
Mass Storage Unit Device- Specific IORB Fields .....	6-33
Mass Storage Unit Device- Specific RCT Fields .....	6-33
Mass Storage Unit RCT/IORB Hardware/Software Status Code Mapping .....	6-34
ASR/KSR Drivers .....	6-34
Keyboard Input .....	6-35
Printer Output .....	6-36
ASR/KSR Device-Specific IORB Fields .....	6-36
ASR/KSR Device-Specific RCT Fields .....	6-37
ASR/KSR RCT/IORB Hardware/ Software Status Coding Mapping .....	6-38
Magnetic Tape Driver .....	6-39
Magnetic Tape Device-Specific IORB Fields .....	6-42
Magnetic Tape Device-Specific RCT Fields .....	6-43
Magnetic Tape RCT/IORB Hardware/Software Status Code Mapping .....	6-43

CONTENTS (cont)

	Page
Section 7	
Trap Handling .....	7-1
Trap Conditions During Task	
Execution .....	7-2
Trap Enabled .....	7-2
Trap Not Enabled .....	7-2
Contents of Trap-Related Memory	
Areas .....	7-2
Pointer to Next Available Trap	
Save Area (NATSAP) .....	7-5
Trap Vector .....	7-6
Trap Save Areas .....	7-6
Interrupt Vector .....	7-6
Interrupt Save Area (ISA) .....	7-6
Honeywell-Supplied Trap Handlers ..	7-12
Trap Handling by the Debug	
Program .....	7-12
Commercial Simulator .....	7-12
Floating-Point Simulator .....	7-13
Scientific Branch Simulator .....	7-14
Software Generated Traps .....	7-14
User-Written Trap Handlers .....	7-15
Trap Handlers Designed as Monitor	
Extensions .....	7-15
Programming Considerations for	
User-Written Trap Handlers .....	7-16
Appendix A	
Data Structure Formats .....	A-1
Clock Request Block Format .....	A-2
File Information Block Format .....	A-4
Input/Output Request Block Format ..	A-6
Semaphore Request Block Format .....	A-10
Task Request Block Format .....	A-11
Parameter Block Format .....	A-13
Wait List Format .....	A-14
Message Facility Message Group	
Request Blocks .....	A-15
Appendix B	
Writing a Peripheral I/O Driver .....	B-1
System Building Considerations in	
Writing a Driver .....	B-1
Driver Interface in Writing a	
Driver .....	B-2
User-Written Driver	
Initialization .....	B-2
Driver Usable System Functions .....	B-3
I/O Subroutines (ZIOSUB) for	
User-Written Drivers .....	B-3
Initialize Function (Code 0) ..	B-3

## CONTENTS (cont)

		Page
Appendix B (cont)	Wait On Line Function	
	(Code 1) .....	B-3
	Stop I/O Function (Code 2) ....	B-4
	Wait for Interrupt (Code 3) ...	B-4
	Read/Modify Status Function	
	(Code 4) .....	B-5
	Locate RCT for Device (ZXSRTC) ..	B-5
	Driver Terminate (ZXD_TR) .....	B-6
	Output Address and Range	
	(ZIOLD) .....	B-6
Appendix C	General I/O Requirements for User	
	Device Driver .....	B-8
Appendix C	Summary of Register Contents for System Service Macro Calls .....	C-1
Appendix D	ASCII and EBCDIC Character Sets .....	D-1
	Control Characters .....	D-1
	Special Graphic Characters .....	D-2

## ILLUSTRATIONS

Figure 3-1.	Life Cycle of a File .....	3-17
Figure 6-1.	Format of I/O Request Block .....	6-8
Figure 6-2.	LRN as Pointer to RCT .....	6-15
Figure 6-3.	ASCII Card-to-Memory Code Formatting ..	6-17
Figure 6-4.	Verbatim Mode Formatting .....	6-19
Figure 6-5.	Packed and 6-Bit Modes on 7-Track Tape .....	6-39
Figure 7-1.	Trap Handling Mechanism .....	7-3
Figure A-1.	First Four Items of Request Blocks ....	A-2
Figure A-2.	Format of Clock Request Block .....	A-2
Figure A-3.	Format of File Information Block (FIB) .....	A-4
Figure A-4.	Format of I/O Request Block .....	A-7
Figure A-5.	Format of Semaphore Request Block .....	A-10
Figure A-6.	Format of Task Request Block .....	A-12
Figure A-7.	Format of Parameter Block .....	A-14
Figure A-8.	Format of Wait List .....	A-14
Figure B-1.	Typical Device Driver .....	B-7

## TABLES

		Page
Table 1-1.	System Service Macro Calls .....	1-7
Table 3-1.	Contents of File Information Block (FIB) .....	3-2
Table 3-2.	Contents of Program View Entry in FIB .	3-7
Table 3-3.	Offsets Definition Macro Calls .....	3-12
Table 5-1.	User-Generated Table for Error Logging Macro Calls .....	5-91
Table 5-2.	MGIRB Argument Values for \$MACPT Macro Call .....	5-173
Table 5-3.	Argument Values for \$MGCRB Macro Call .	5-176
Table 5-4.	MGIRB Argument Values for \$MCMG Macro Call .....	5-182
Table 5-5.	Argument Values for \$MGIRB Macro Call .	5-185
Table 5-6.	MGIRB Argument Values for \$MINIT Macro Call .....	5-190
Table 5-7.	MGCRB Argument Values for \$MRECV Macro Call .....	5-193
Table 5-8.	Argument Values for \$MGRRB Macro Call .	5-197
Table 5-9.	MGCRB Argument Values for \$MSEND Macro Call .....	5-202
Table 5-10.	MGRRB Argument Values for \$MTMG Macro Call .....	5-207
Table 5-11.	Tape File Search Rules for \$OPFIL Macro Call .....	5-217
Table 6-1.	Input/Output Function Code .....	6-3
Table 6-2.	Return Status Codes .....	6-5
Table 6-3.	Contents of I/O Request Block .....	6-8
Table 6-4.	IORB Software Status Word (I ST) .....	6-11
Table 6-5.	Resource Control Table (RCT) Field Definitions .....	6-12
Table 6-6.	Hollerith-ASCII Code Table .....	6-18
Table 6-7.	Card Reader Card/Reader-Punch Device- Specific IO RB Fields .....	6-19
Table 6-8.	Card Reader IO RB Hardware/Software Status Code Mapping .....	6-21
Table 6-9.	Card Reader/Punch Hardware/Software Status Code Mapping .....	6-21
Table 6-10.	Printer Device-Specific IO RB Fields ...	6-24
Table 6-11.	Printer Device-Specific RCT Fields ....	6-25
Table 6-12.	Printer RCT/IO RB Hardware/Software Status Code Mapping .....	6-26
Table 6-13.	Diskette Device-Specific IO RB Fields ..	6-27
Table 6-14.	Diskette Device-Specific RCT Fields ...	6-28
Table 6-15.	Diskette RCT/IO RB Hardware/Software Status Code Mapping .....	6-28
Table 6-16.	Cartridge Disk Device-Specific IO RB Fields .....	6-30

TABLES (cont)

		Page
Table 6-17.	Cartridge Disk Device-Specific RCT Fields .....	6-31
Table 6-18.	Cartridge Disk RCT/IORB Hardware/ Software Status Code Mapping .....	6-31
Table 6-19.	Mass Storage Unit Device-Specific IORB Fields .....	6-33
Table 6-20.	Mass Storage Unit Device-Specific RCT Fields .....	6-34
Table 6-21.	Mass Storage Unit Status Code Mapping .	6-34
Table 6-22.	ASR/KSR Device-Specific IORB Fields ...	6-36
Table 6-23.	ASR/KSR Device-Specific RCT Fields ....	6-38
Table 6-24.	ASR/KSR RCT/IORB Hardware/Software Status Code Mapping .....	6-38
Table 6-25.	Characteristics of Supported Tape Drives .....	6-40
Table 6-26.	Magnetic Tape Device-Specific IORB Fields .....	6-42
Table 6-27.	Magnetic Tape Device-Specific RCT Fields .....	6-43
Table 6-28.	Magnetic Tape RCT/IORB Hardware/ Software Status Code Mapping .....	6-44
Table 7-1.	Contents of Selected Words of Trap Save Area When Trap Occurs .....	7-7
Table A-1.	Contents of Clock Request Block .....	A-3
Table A-2.	Contents of File Information Block (FIB) .....	A-5
Table A-3.	Contents of I/O Request Block .....	A-7
Table A-4.	Summary of IORB Fields for Operator Interface .....	A-9
Table A-5.	Contents of Semaphore Request Block ...	A-10
Table A-6.	Contents of Task Request Block .....	A-12
Table A-7.	Message Group Control Request Block (MGCRB) .....	A-15
Table A-8.	Message Group Initialization Request Block (MGIRB) .....	A-18
Table A-9.	Message Group Recovery Request Block (MGRRB) .....	A-22
Table C-1.	Macro Calls, Function Codes, and Register Contents .....	C-2
Table D-1.	ASCII/Hexadecimal Equivalents .....	D-2
Table D-2.	EBCDIC/Hexadecimal/Binary Equivalents .	D-3





## SECTION 1

### INTRODUCING AND USING THE MANUAL

This manual describes the function and use of GCOS 6 system service macro routines, used by the assembly language programmer to obtain monitor and input/output services and to build control structures, for applications programs.

#### HOW TO USE THE MANUAL

Table 1-1 is an alphabetic list of all macro calls and their functions, arranged alphabetically by macro call name (column 1).

Sections 2, 3, and 4 contain brief descriptions of the functional groupings for the macro calls, together with a list of the macro calls arranged alphabetically by function grouping. The lists also include the macro call names shown in column 1 of Table 1-1. Section 2 summarizes monitor services macro calls, Section 3 the file system macro calls related to I/O services, and Section 4 the macro calls to generate and define system data structures.

Section 5 describes the use, functions, structures, and error return codes for all macro routines/calls, with one example for most. For easy reference, these detailed descriptions are arranged in alphabetic order by specific function, (see column 2 in Table 1-1).

This section also describes macro call syntax and register conventions. The Assembly Language Reference manual discusses the use of labels and address formats in detail.

#### MACRO CALL SYNTAX

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

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

### REGISTER CONVENTIONS AND CONTENTS

Macro call arguments are often loaded into registers for access by the system services. An argument of a macro call can specify that the corresponding system service argument is either contained in memory or in a register. If an argument is omitted from the macro call the system assumes that the register normally used to provide the value or address to the system service routine contains the required value or address. For this reason it is important to know how the system service routines use the registers, as well as the conventions that exist for saving register contents.<sup>1</sup>

The system services use the following registers without preserving their contents:<sup>1</sup>

R1	R7
R2	B2
R6	B4

As a general rule, the system services do not alter the contents of the following registers:

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

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

<sup>1</sup>The file system macro calls preserve the contents of all registers except R1. B4 is the only register loaded by the file system macro calls.

## ADDRESSING CONVENTIONS

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

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

`=label`

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

`label`

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

`<label`

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

`=$B4`

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

=SB3

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

\$B3

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

\*\$B3

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

\$B3.\$R2

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

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

!label

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

!\*label

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

Thus, macro call "location address" arguments (which are to be loaded into base registers) can refer to the location of the address of the data or data structure or can refer to the address of the data or data structure. In the first case (location of address), the macro call loads the \$Bn register through an LDB instruction, thus requiring that the "location address" values

in the macro call arguments be the label of a location where the address of the actual argument structure is located. In the second case (address), the macro call loads the effective address of the argument structure into \$Bn directly (through a LAB instruction) when the first argument is a label and is preceded by an exclamation point (!) character.

For Example:

```

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

```

#### REGISTER CONTENTS AT TASK ACTIVATION

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

- Register \$B4: Address of the task request block
- Register \$B5: Address of the system-supplied termination routine (see the return (\$RETRN) macro call)
- Register \$B7: Address of the parameter block containing the request block argument list

#### REGISTER CONTENTS AT INITIAL TASK ACTIVATION

The M registers are set up as follows:

When each task starts, the system establishes the following default values M1, M3, M4, and M5:

- M1 = 00 Trace trap and all R-register overflow traps disabled.
- M3 = 00 CIP overflow trap and truncation trap disabled; CIP is under direct CPU firmware control (i.e., not in software test mode).



- M4 = 03    Truncation mode in effect. Scientific accumulators \$S1 and \$S2 and associated memory operands are two words long; \$S3 and associated memory operands are four words long.
- M5 = 20    Significance error trap enabled; exponent overflow and precision error traps disabled.

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

If the task is in an online task group, then the privileged bit in the S-register is set on. If the task is not in an online task group, the privileged bit is not set.

#### RETURN STATUS CODES IN \$R1 REGISTER

The descriptions of the macro calls in Section 5 include lists of status codes returned in register \$R1, together with an explanation of each code. These lists, while extensive, are not intended to include every possible status return or explanation for each macro call. See the System Messages manual for a list of all \$R1 return status codes, corresponding messages, and added definitions.

#### SYSTEM SERVICE MACRO CALLS AND FUNCTION CODES

Table 1-1 contains an alphabetic list, by macro call name, of the macro calls summarized in Sections 2, 3, and 4, and described in Section 5. The list includes the function codes associated with each macro call (data structure generation macro calls do not have function codes). The first two digits of the function code designate the major function, and are used by the macro call trap-handling routine to locate the entry point of the appropriate system service routine. The last two digits are a subfunction code used by the system service routine to provide the requested subfunction. When a macro call is executed, it generates the following:

```

MCL
DC      Z'mmss'

```

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

#### LOCATION OF MACRO ROUTINES

The macro routines are located either on cartridge disk or on storage module in a library named >LDD>MACRO>EXEC\_LIB. On diskette they are located in ^ZSYS02>LDD>MACRO>EXEC\_LIB. See the Assembly Language Reference manual.

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
\$ACTID	Account identifier	1402	Identification and information
\$ACTVG	Activate group	0D09	Task group control
\$ASFIL	Associate file	1010	File management
\$BUID	Bound unit identification	1406	Identification and information
\$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	Cancel semaphore request	0601	Semaphore handling

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$CRB	Clock request block	-	Data structure generation
\$CRBD	Clock request block offsets	-	Data structure generation
\$CRDIR	Create directory	10A0	File management
\$CRFIL	Create file	1030	File management
\$CRGRP	Create group	0D03	Task group control
\$CROAT	Create overlay area table	070A	Overlay handling
\$CRPSB	Create file parameter structure block - offsets	-	Data structure generation
\$CRTSK	Create task	0C02/0C03	Task control
\$CWDIR	Change working directory	10B0	File management
\$DFSM	Define semaphore	0604	Semaphore handling
\$DLGRP	Delete group	0D04	Task group control
\$DLREC	Delete record	1130/1131	Data management
\$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

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$ELENL	Error logging, end	0209	Physical I/O
\$ELEX	Error logging information, exchange	0207	Physical I/O
\$ELGT	Error logging information, get	0208	Physical I/O
\$ELST	Error logging, start	0205	Physical I/O
\$ENDV	Enable device	0204	Physical I/O
\$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
\$GDTM	Get date/time	0506	Date/time
\$GIFAB	Get file information, file attributes block - offsets	-	Data structure generation
\$GIFIL	Get file information	1060	File management
\$GIKDB	Get file information and create file, key descriptor block - offsets	-	Data structure generation
\$GIPSB	Get file information, parameter structure block - offsets	-	Data structure generation

\*

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$GMEM	Get memory; get available memory	0402/0403	Memory allocation
\$GTFIL	Get file	1020	File management
\$GTPSB	Get file, parameter structure block - offsets	-	Data structure generation
\$GWDIR	Get working directory	10C0	File management
\$HDIR	Home directory, return	140B	Identification and information
\$INDTM	Internal date/time - convert to	0507	Date/time
\$IORB	Input/output request block	-	Data structure generation
\$IORBD	Input/output request block offsets	-	Data structure generation
\$MACPT	Message group, accept	1501	Intergroup message facility
\$MCMG	Message group, count	1507	Intergroup message facility
\$MGCRB	Message group control request block	-	Data structure generation
\$MGCRT	Message group control request block offsets	-	Data structure generation

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$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
\$NPROC	New process	0D0B	Task group control
\$NUIN	New user input file - redefine	0806	Standard system file I/O
\$NUOUT	New user output file - redefine	0807	Standard system file I/O



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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$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
\$RDBLK	Read block	1200-1204	Storage management
\$RDREC	Read record	1110-1116	Data management

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$RDSW	Read external switches	0B00	External switch
\$RDVAT	Reset device attention	0203	Physical I/O
\$RETRN	Return sequence - establish	-	Request and return
\$RLDIR	Release directory	10A5	File management
\$RLFIL	Release file	1035	File management
\$RLSM	Release semaphore	0603	Semaphore handling
\$RLTML	Release terminal	1704	Terminal Function
\$RMEM	Return memory; return partial block of memory	0404/0405	Memory allocation
\$RMFIL	Remove file	1025	File management
\$RNFIL	Rename file/directory	1040	File management
\$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
\$RQIO	Request I/O transfer	0200	Physical I/O
\$RQSM	Request semaphore	0600	Semaphore handling
\$RQTML	Request terminal	1703	Terminal function

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$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
\$SPGRP	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	Status memory pool	0406	Memory allocation
\$STTY	Set terminal file characteristics	1045	File management
\$SUSPG	Suspend group	0D08	Task group control
\$SUSPN	Suspend for interval; suspend until time	0502/0503	Clock
\$SYSID	System identification	1404	Identification and information
\$STEST	Test completion status	0102	Request and return

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$TFIB	File information block - offsets	-	Data structure generation
\$TGIN	Task group input	140C	Identification and information
\$TIFIL	Test file for input	1062	File management
\$TOFIL	Test file for output	1063	File management
\$TRB	Task request block	-	Data structure generation
\$TRBD	Task request block offsets	-	Data structure generation
\$TRMRQ	Terminate request	0103/0104	Request and return
\$TRPHD	Trap handler connect	0A00	Trap handling
\$USIN	User input file - read	0800	Standard system file I/O
\$USOUT	User output file - write	0801	Standard system file I/O
\$USRID	User identification	1400	Identification and information
\$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

\*

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

Macro Call Name (1)	Function Description (2)	Function Code (3)	Function Group (4)
\$WLIST	Wait list	-	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
\$WTBLK	Wait block	1220	Storage management
\$XPATH	Expand pathname	10D0	File management

TASK REQUEST QUEUES

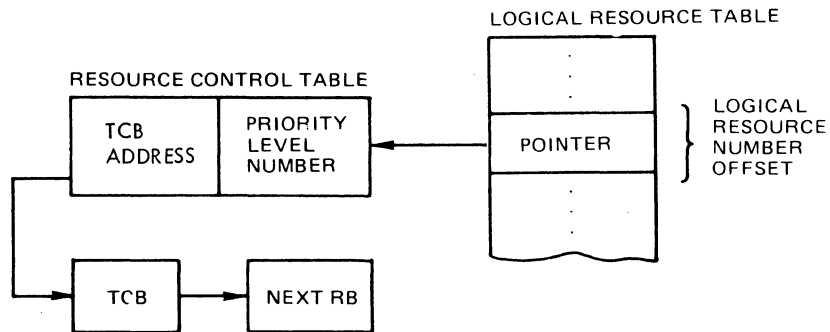
The task manager controls the scheduling and synchronization of tasks, and uses the following data structures.

- o Logical Resource Number (LRN) - An LRN provides a logical task identifier. You assign an LRN to an application task when the application is designed; you assign an LRN to a system device driver task at configuration. Its value is from 0 through 252.
- o Logical Resource Table (LRT) - An LRT associates each LRN with a specific resource control table (RCT). An LRT is created for each user task group and contains an entry for each LRN specified in the task group. Each entry in the LRT points to an RCT that uniquely identifies one of the task group's tasks.

The system has one LRT to associate each configured device with an LRN. The LRT contains an entry for each LRN identified to the Configuration Load Manager at system startup. Each entry in the LRT points to an RCT that uniquely defines the configured device.

- o Resource Control Table (RCT) - A separate RCT is built for each task or device. Each RCT contains the physical priority level number dedicated to that task or device. In addition, each device RCT contains the device-specific characteristics that uniquely describe that device.
- o Request Block (RB) - The RB is the basic control structure that embodies a specific request for the execution of a task. An RB contains the arguments of a task request; it provides the medium of communication between the requesting and requested tasks. Separate RB formats exist for tasks and devices, but the control fields used by the task manager are common to both formats. You may extend the RB to include application-specific information to be passed between the requesting and requested tasks.
- o Request Queue - The request queue is a first-in/first-out queue of request blocks maintained by the task manager in order to serialize requests for task services. A separate request queue is maintained for each task.
- o Task Control Block (TCB) - A TCB is the system control structure for a task and includes the interrupt save area (ISA). When a task is interrupted by a task with a higher priority level, the contents of the ISA are stored in the task control block of the interrupted task.

The task manager controls tasks by manipulating request blocks in task request queues. Using the logical resource number provided in a request block, the task manager consults the logical resource table to locate the TCB of a requested task; the mechanism is shown in the diagram below.



A request for a task causes a request block to be placed in the request queue for the requested task. When the requested task terminates, the task manager removes the request block from the top of its request queue and posts its completion. If desired, one task may wait for the completion of the requested

task; the waiting task is suspended until the requested task signal its completion to the task manager, which will re-activate the waiting task. In the meantime, other tasks may use the level of the waiting task.

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.

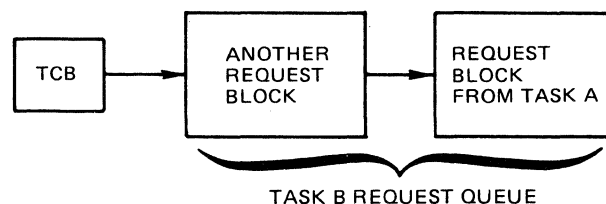


Diagram 1 - Request Block From Task A is Queued in Request Queue for Task B

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.

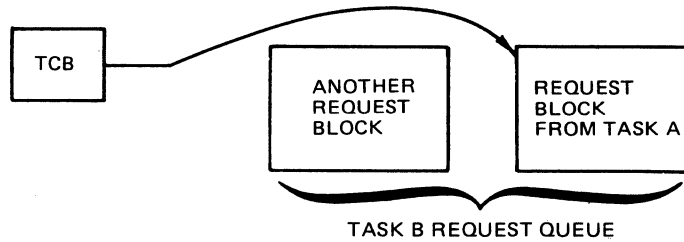


Diagram 2 - First Request Block is Dequeued as Task B Terminates Relative to It

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.



## SECTION 2

### MONITOR SERVICE FUNCTIONS

The Monitor service macro routines summarized and listed in this section provide access to system service functions. The macro routines/calls are arranged and discussed within and in the order of these functional groups (see column 4 in Table 1-1):

Batch	Overlay handling
Clock	Physical I/O
Communications	Request and return
Date/time	Secondary user terminal
Error handling	Semaphore handling
External switches	Standard system file I/O
Identification and information	Task control
Memory allocation	Task group control
Message facility (intergroup)	Trap handling
Operator interface	

Each macro routine/call is described in detail in Section 5, in the alphabetic order of its function description (column 2 of Table 1-1).

#### BATCH FUNCTIONS

The macro call for batch functions facilitates program execution in a way that requires no personal interaction with the system. To use the batch functions, prepare a file that is to act as the command input file and the user input file. All commands and program input are read from this file by the batch task group when your request executes.

The macro routine/call is:

o Request batch execution \$RQBAT

Section 5 describes this macro call in detail.

## CLOCK FUNCTIONS

The macro calls for clock functions allow user control of task execution according to an elapsed time period. These macro calls use the clock manager, a system component whose primary function is initiating task execution based on the passage of time.

The clock manager services interrupts from the real-time clock. At each interrupt, the clock manager ascertains whether the time interval associated with a request to initiate execution of the task has been satisfied. Based on information contained in the clock request block (see Appendix A), the system will:

- o Activate a task
- o Schedule an indicated request block
- o Release a semaphore

The clock macro calls act to:

- o Connect a clock request block to the timer queue
- o Disconnect a clock request block from the timer queue
- o Suspend the issuing task until an interval of time has passed
- o Suspend the issuing task until a particular date/time

The clock function macro routines/calls are:

- o Cancel clock request   \$CNCRQ
- o Request clock           \$RQCL
- o Suspend for interval    \$\$SUSPN
- o Suspend until time      \$\$SUSPN

Section 5 describes these macros in detail and also the macro \$CRB (clock request block) for generating a clock request block.

## COMMUNICATIONS FUNCTIONS

The macro call for communications functions allows you to set a telephone number to be used for automatic dialing. The macro routines/call is:

- o Set dial                                  \$SDL

The Communications Processing manual describes communications processing.

Section 5 describes the above macro in detail.

## DATE/TIME FUNCTIONS

The macro calls for date/time functions allow access and use of the internal date/time value maintained by the system, and conversions of date/time values for internal to external formats, and vice versa. Specifically, the macro calls provide the task with the means to:

- o Obtain the current internal date/time value
- o Convert the internal date/time value to external date/time format
- o Convert the internal date/time value to external time format
- o Convert an external date/time value to internal format

The date/time macro routines/calls are:

External date/time - convert to	\$EXTDT
External time - convert to	\$EXTIM
Get date/time	\$GDTM
Internal date/time - convert to	\$INDTM

Section 5 describes these macros in detail.

## ERROR HANDLING FUNCTION

The macro for error handling enables you to report error conditions and to add error text.

The error handling macro routine/call is:

Report error condition	\$RPTER
------------------------	---------

Section 5 describes it in detail.

## EXTERNAL SWITCH FUNCTIONS

A task group, by using external switch function macro calls, can control its own execution by modifying its own external switches. An external switch operates much like a hardware switch on an operator's control panel. External switches may be set and cleared with the MSW (modify switches) command, or internally with the \$SETSW and \$CLRSW macro calls.

A separate external switch word is associated with each task group. Each bit in the word is an external switch. Thus, each task group can use 16 switches. A user program can contain

instructions or statements to interrogate the settings of one or more of these switches. The program can then use these settings to control its execution logic.

The macro calls allow you to:

- o Set switches
- o Clear switches
- o Read the current values of the switches

The macro routines/calls are:

Clear external switches	\$CLRSW
Read external switches	\$RDSW
Set external switches	\$SETSW

Section 5 describes these macros in detail.

### IDENTIFICATION AND INFORMATION FUNCTIONS

The macro calls for identification and information make available the following information concerning the current task or task group.

<u>Function</u>	<u>Macro Call</u>
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

Section 5 describes these macro in detail.

### 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

Section 5 describes these macros in detail.

#### 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

Section 5 describes these macros in detail.

## 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).

Section 5 describes the operator interface macros in detail.

## 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 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
Overlay, execute	\$OVEXC
Overlay, load	\$OVLD
Overlay status	\$OVST
Overlay, unload	\$OVUN

Section 5 describes these macro in detail.



## 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:

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

Another group of macro calls associated with physical I/O function codes are the error logging macro calls which:

- o Activate error logging for a device
- o Insert current values of error logging information in the user's error log structure
- o End error logging and store logging information in user's error log structure
- o Verify and save user error logging structure.

The macro calls for error logging are:

Error logging, start	\$ELST
Error logging information, get	\$ELGT
Error logging, end	\$LEND
Error logging information, exchange	\$LEX

Section 5 describes these macros in detail.

## 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:

- o Terminate the current execution of a task
- o Wait for the completion of another task
- o Wait for the completion of any of a number of events
- o Test for the completion of an event
- o Return the address of a request block
- o Issue a common return sequence for called subroutines

The macro routines/calls are:

Return request block address	\$RBADD
Return	\$RETRN
Terminate request	\$TRMRQ
Test completion status	\$TEST
Wait for operation to complete	\$WAIT
Wait on request list	\$WAITL

Section 5 describes these macros in detail.

Sections 4 and 5 describe the macro routines for generating request blocks. Appendix A shows request block formats.

## SECONDARY USER TERMINAL FUNCTIONS

The macro calls for secondary terminal functions pertain to a task group's secondary user terminal, which is defined as a terminal given to a task group by the listener component when a secondary login is performed at that terminal. This occurs provided the task group has an outstanding request for a secondary user terminal.

The macro calls for secondary user terminal functions permit:

- o The task group to request a secondary terminal.
- o The task group to release a secondary terminal.
- o The task group to cancel a previous request from the issuing group.

The appropriate macro calls are:

Request terminal	\$RQTML
Release terminal	\$RLTML
Cancel request	\$CANRQ

Section 5 describes these macros in detail.

### SEMAPHORE FUNCTIONS

A semaphore is a mechanism for coordinating the use of resources within task groups. Semaphores are unique to a task group and once defined, are available only to the tasks within that group, to control access to multiple resources and control multiple requests for the same resource.

A semaphore is defined for each resource to be controlled and given a 2-character ASCII semaphore name, which is a system symbol recognized by the Monitor. Every requestor of a resource whose use must be coordinated issues appropriate Monitor calls to the named semaphore to request or release the resource. The task that defines the semaphore assigns the semaphore's initial value. The Monitor maintains its current value to coordinate requestors of the resource being controlled. A requestor obtains use of a resource if the semaphore value is greater than zero at the time of the request. A requestor is either suspended waiting for the resource or notified that no resource is available if the value is zero or negative.

Monitor service macro calls are used to:

- o Define a semaphore and give an initial value.
- o Reserve a semaphore-controlled resource; this macro call subtracts a resource, or signals a waiter for the resource i.e., it decrements the current-value counter.
- o Release a semaphore-controlled resource; this macro call adds a resource, or activates the first waiter on the semaphore queue; i.e., it increments the current-value counter.
- o Request the reservation of a semaphore-controlled resource; this macro call queues a request block (SRB) if the resource is not available. This macro call decrements the current-value counter.
- o Remove a specified semaphore request block from its semaphore request queue.

A semaphore is a gating mechanism, and the initial value given to it depends upon the type of control to be exercised.

For example, assume that you want to restrict access to a particular resource to a one-user-at-a-time order.

1. Task A defines a semaphore by issuing the macro call:

\$DFSM ZZ

Omission of the value parameter causes the initial value to be set to 1.

2. Task B now issues a \$RSVSM call; the counter is decremented to 0, Task B gets the resource for itself knowing that no other task using the semaphore mechanism is using or can obtain the resource.
3. Task C issues a \$RSVSM call; the counter is decremented to -1, Task C is suspended (Task B is still using the resource).
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

Section 5 describes these macros in detail.

#### 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:

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

The macro routines/calls are:

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

Section 5 describes the macros in details.

#### TASK CONTROL FUNCTIONS

The macro calls for task control allow you to:

- o Create, request, spawn, and delete a task
- o Process command lines
- o Unlock locked records for all files attached to the task group (clean point)

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

Section 5 describes these macros in detail.

## 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:

- o Create a task group
- o Request a task group
- o Delete a task group
- o Spawn a task group
- o Suspend a task group
- o Activate a suspended task group
- o Terminate current task group and restart task group request
- o Abort a task group request
- o Abort a task group

A task executing under one task group can initiate another task group. A task group must first be defined to create task group control structures and load the bound-unit root segment as the lead task. A group request must be issued to activate the lead task for execution. Tasks can be executed concurrently in this task group with the use of task control functions or commands.

The task group can be deleted; no more requests can be made against this task group after it has been marked for deletion. When all tasks in the task group terminate and become dormant, all memory associated with the task group is returned to its memory pool, making that memory available to other task groups.

The above effects of create, request, and delete a task group occur in sequence when a spawn task group macro call is issued.

A task can suspend a task group's execution and then activate that task group.

A task can terminate the current task group request and then restart the processing of the original task group request.

A task can abort the current request for the activation of a specified task group. In this case the next request, if any, against that task group, will be processed.

To delete the task group immediately, before all its tasks terminate and become dormant, the group can be aborted.

Some macro calls listed below use parameter blocks.

The macro call to generate parameter blocks is discussed and described in Sections 4 and 5; block format is shown in Appendix A.

The macro routines/calls for task group control are:

Abort group	\$ABGRP
Abort group request	\$ABGRQ
Activate group	\$ACTVG
Create group	\$CRGRP
Delete group	\$DLGRP
New process	\$NPROC
Request group	\$RQGRP
Spawn group	\$SPGRP
Suspend group	\$SUSPG

Section 5 describes these macros in detail.

#### TRAP HANDLING FUNCTIONS

The macro calls for trap functions allow you to indicate the kind of trap handling functions within the task group of the issuing task. The macro calls allow you to:

- o Connect a user-written, generalized trap handling routine to a task
- o Enable a specific trap or all traps
- o Disable a specific trap or all traps

Section 7 describes traps and trap handling in detail.

The macro routines/calls for trap handling are:

Disable user trap	\$DSTRP
Enable user trap	\$ENTRP
Trap handler connect	\$TRPHD

Section 5 describes these macros in detail.





## SECTION 3

### FILE SYSTEM FUNCTIONS

The macro routines summarized and listed in this section enable you to use the file system functions, which are organized according to the following major functional groups: (see column 4 in Table 1-1):

- o File/directory management
- o Data management
- o Storage management

The file/directory management macro routines provide service functions at the file level (i.e., creating files, reserving files, specifying concurrency control, opening and closing files, creating directories, etc.). Data management macro routines supply the service functions required at the record level, such as read, write, delete, and rewrite. The storage management macro routines furnish service functions at the block (unit of transfer) level, such as read and write.

Every macro routine/call is described in detail in Section 5 in the alphabetic order of its function description (see column 2 of Table 1-1).

File management services use argument structures, which vary in content from macro call to macro call. The content of each argument structure is described with the appropriate macro call.

"Addressing Conventions" in Section 1 describes the address forms that are valid for address and data registers.

A pointer in the file system structures consists of two words. In SAF, the pointer resides in the first word; in LAF the pointer is a double-word address.

Some macros, particularly the data and storage management services, use a data structure called a file information block (FIB), which is generated with a \$FIB macro call (see Sections 4 and 5). The following are detailed descriptions of the structure and contents of the file information block (FIB).

#### FILE INFORMATION BLOCK (FIB)

There must be one file information block for each file, in order for the file to be accessed. The FIB, which must start on a word boundary, provides the interface between your program and the system for performing data- and storage-management functions. Table 3-1 describes each of the FIB entries, Table 3-2 describes the second entry in the FIB (i.e., program view) in detail.

The FIB can be generated by a \$FIB macro call (see Sections 4 and 5).

Table 3-1. Contents of File Information Block (FIB)

Entry	Size (bytes)	Description
Logical file number (LFN)	2	Specifies the logical file number (LFN) by which the file is referenced. The LFN is the common element linking the FIB and the external file; this connection is made via the \$ASFIL, \$CRFIL, or \$GTFIL macro call (or equivalent command).
Program view	2	Describes the user visibility of the file and its functional capabilities; it contains information specific to each of the function groups (see Table 3-2) as follows:  Bits 0-9 File/directory management Bits 10-14 Data management Bits 13-15 Storage management  As described in Table 3-2, the setting of bit 0 determines whether bits 13 and 14 are data management specific or storage management specific.

Table 3-1 (cont). Contents of File Information Block (FIB)

Entry	Size (bytes)	Description
User record pointer (for data management functions)	4	<p>Identifies the start of the user-record area as follows:</p> <p>\$RDREC - Identifies the storage area into which records are delivered by the system.</p> <p>\$RWREC, \$WRREC - Identifies the storage area from which records are taken by the system.</p> <p>The storage area must be large enough to contain the longest record, excluding headers, to be written to or received from the file.</p>
Buffer pointer (for storage management functions)		<p>Identifies the start of the buffer area as follows:</p> <p>\$RDBLK - Identifies the buffer area into which blocks of data are delivered.</p> <p>\$WRBLK - Identifies the buffer area from which blocks of data are taken.</p>
In record length (for data management functions)	2	<p>Specifies the maximum size of the user-record area for \$RDREC operations.</p>
Transfer-size (for storage management functions)		<p>Specifies the size of the data transfer (i.e., the size of the buffer) for storage management operations.</p>
Out record length (for data management functions) or	2	<p>Specifies the actual size of the record to be written or read, as follows:</p> <p>\$RDREC - The system updates this entry to reflect the actual length of the last record delivered into the user-record area.</p>

Table 3-1 (cont). Contents of File Information Block (FIB)

Entry	Size (bytes)	Description
Block size (for storage management functions)		\$RWREC, \$WRREC - Specifies the actual length of the record, excluding the headers, to be written in the file.
		Specifies the size of the block for storage management operations; for disk files the size must be a multiple of physical sector size.
Reserved  Block number (for storage management function)	4	Must specify Z'0000FFFF'.  Specifies the starting block number for the I/O transfers; is relative to the start of the file and to the block size (described above). This entry is incremented by 1 after each I/O transfer; therefore, user's dynamic changes to the block size also require changes to the contents of this entry. The first block in a file is block 0.
Reserved	2	Must specify Z'0000'.
In key pointer	4	Identifies the start of the user-key area in which the key value is stored for the following \$RDREC macro call functions:  Read with key Read position equal Read position greater than Read position greater than or equal Read position forward Read position backward

Table 3-1 (cont). Contents of File Information Block (FIB)

Entry	Size (bytes)	Description
In key pointer (cont.)	4	<p>For the following \$WRREC macro call functions:</p> <p>Write with key  Write position equal  Write position greater than  Write position greater than or equal  Write position forward  Write position backward</p> <p>For the following \$RWREC macro call function:</p> <p>Rewrite with key</p> <p>And for the following \$DLREC macro call function:</p> <p>Delete with key</p> <p>The type of key is specified in the "in key format" entry below.</p>
In key format	1	<p>Identifies the type of key pointed to by the "in key pointer" entry above, as follows:</p> <p>0 - None specified</p> <p>1 - Primary or relative, as specified in the "key type" field in the "program view" entry; see Table 3-2.</p> <p>2 - Simple key</p> <p>The entry is meaningful only for the macro calls specified in the "in key pointer" entry defined above.</p>
In key length	1	<p>Specifies the length of the user-key area identified in the "in key pointer" entry described above. Only meaningful for primary keys; simple and relative keys always assumed to be four bytes.</p>

Table 3-1 (cont). Contents of File Information Block (FIB)

Entry	Size (bytes)	Description
Out record address	4	<p>This field is available for the system to place the media address of the record dealt with by the last data management macro call.</p> <p>Normally, this address is a 32-bit simple key (i.e., it specifies the control interval and logical record within the control interval).</p> <p>However, if the file is accessed via a relative key as specified in the "in key format" field, then this address is a 32-bit relative key (i.e., relative logical record number in the file). If the operation is not performed as specified, this entry is not set (and an error code is returned).</p> <p>For card readers, printers, and terminal devices, this field contains a count of the records transferred; i.e., this field is incremented by 1 for each logical sequential access to the device.</p>
Reserved	4	This entry is reserved for future use.

Program View Entry in the FIB

Table 3-2 shows the contents of the 2-byte FIB program view entry. The program view entry describes to the file system how the file is to be accessed and, to some extent, what it looks like from the programmer's point of view. The contents of this entry are used by the file system to ensure that the file is accessed only as intended.

The bits in the program view entry are read at the time the file is opened. Once the file is opened, you can change only bits 11, 12, and 13. You cannot change the other bits until you close the file and then reopen it.

Table 3-2. Contents of Program View Entry in FIB

Entry	Size (Bits)	Description	Related Function Group										
Access level (Bit 0)	1	<p>Specifies whether file is accessed via data or storage management macro calls as follows:</p> <p>0 - Access via data management macro calls.</p> <p>1 - Access via storage management macro calls.</p>	File Management (\$OPFIL only)										
Process rules (Bits 1-4)	4	<p>Specifies how the file can be processed; that is, it specifies which types of data/storage management macro calls are allowed as follows:</p> <table data-bbox="602 1052 1101 1276"> <thead> <tr> <th data-bbox="602 1087 721 1115"><u>Binary</u></th> <th data-bbox="756 1052 1040 1115"><u>Permitted Macro Calls</u></th> </tr> </thead> <tbody> <tr> <td data-bbox="602 1150 678 1178">1000</td> <td data-bbox="756 1150 1062 1178">\$RDREC or \$RDBLK</td> </tr> <tr> <td data-bbox="602 1182 678 1209">0100</td> <td data-bbox="756 1182 1062 1209">\$WRREC or \$WRBLK</td> </tr> <tr> <td data-bbox="602 1213 678 1241">0010</td> <td data-bbox="756 1213 867 1241">\$RWREC</td> </tr> <tr> <td data-bbox="602 1245 678 1272">0001</td> <td data-bbox="756 1245 867 1272">\$DLREC</td> </tr> </tbody> </table> <p>nnnn Any combination of the binary settings to allow the desired data/storage management macro calls listed above.</p> <p>If a macro call that is not permitted (as specified in this field) is issued, an access violation error results.</p>		<u>Binary</u>	<u>Permitted Macro Calls</u>	1000	\$RDREC or \$RDBLK	0100	\$WRREC or \$WRBLK	0010	\$RWREC	0001	\$DLREC
<u>Binary</u>	<u>Permitted Macro Calls</u>												
1000	\$RDREC or \$RDBLK												
0100	\$WRREC or \$WRBLK												
0010	\$RWREC												
0001	\$DLREC												

\*





Table 3-2 (cont). Contents of Program View Entry in FIB

Entry	Size (Bits)	Description	Related Function Group
Key type (cont).		Sequential Simple (Disk-resident)  Indexed Primary Simple  (Also see the "in key format" entry in Table 3-1.)	File Management (\$OPFIL only)
Record class (Bit 10)	1	Specifies type of logical records that can be present in the file as follows:  0 - Any type (i.e., fixed- or variable-length records allowed).  1 - Only fixed-length records allowed.	Data Management
Record visibility (Bit 11)	1	Specifies whether or not deleted records are skipped during read next record (\$RDREC) operations as follows:  0 - Deleted records not visible (i.e., skip them).  1 - Deleted records are visible (i.e., the system issues the record not found return code when a deleted record is accessed).	

Table 3-2 (cont). Contents of Program View Entry innFIB

Entry	Size (Bits)	Description	Related Function Group
Key storage area alignment (Bit 12)	1	Specifies the boundary alignment of the user-key area (see "in key pointer" entry in Table 3-1) as follows:  0 - Key storage area begins at even-byte boundary.  1 - Key storage area begins at odd-byte boundary.	Data Management
* Record storage area/buffer alignment (Bit 13)	1	Specifies the boundary alignment of the user-record area (see "User Record Pointer" entry in Table 3-1) as follows:  0 - Record storage area begins at even-byte boundary.  1 - Record storage area begins at odd-byte boundary.	Data Management
* *		Same as above, except that the boundary alignment refers to the buffer.	Storage Management
* * Transcription mode (Bit 14)	1	Specifies how data is transferred as follows:  0 - Data is transferred in device-specific native (ASCII) mode.  1 - Data is transferred in binary transcription mode. (See Note 2.)	Data Management

Table 3-2 (cont). Contents of Program View Entry in FIB

Entry	Size (Bits)	Description	Related Function Group
Transcription mode (Bit 14 cont.)		Same as above.	Storage Management
Synchronous/asynchronous indicator (Bit 15)	1	<p>Specifies whether or not \$RDBLK or \$WRBLK macro calls are executed synchronously or asynchronously as follows:</p> <p>0 - \$RDBLK or \$WRBLK macro calls are to be executed synchronously. When synchronous \$RDBLK or \$WRBLK macro calls are issued, a \$WTBLK macro call is not required to synchronize buffer use.</p> <p>1 - \$RDBLK or \$WRBLK macro calls are to be executed asynchronously (i.e., a \$WTBLK macro call is required to synchronize.)</p>	Storage Management
<p>NOTES: 1. Bits 10 through 15 may be set after an \$OPFIL macro call and before any data or storage management macro call.</p> <p>2. Binary transcription mode is meaningful only for card devices and for 7-track magnetic tape. For card devices, this mode is equivalent to verbatim mode (see Section 6). For 7-track magnetic tape, binary transcription mode is usable only with storage management and is equivalent to packed mode (see Section 6).</p>			

Offsets Definitions

You can refer to specific locations in the file information block and the various argument structures by using offsets definition macro calls. These calls, summarized in Section 4 and described in detail in Section 5, define standard offsets tags.

Table 3-3 shows the offsets definition macro calls and the structures for which they define tags.

Table 3-3. Offsets Definition Macro Calls

Macro Call	Affected Structure
\$CRPSB	Argument structure for create file macro call (\$CRFIL)
\$GTPSB	Argument structure for get file macro call (\$GTFIL)
\$GIPSB	Argument structure for get file information macro call (\$GIFIL)
\$GIFAB	File attribute block pointed to by \$GIFIL argument structure
\$GIKBD	Key descriptor block pointed to by the \$GIFIL and \$CRFIL argument structures.
\$TFIB	File information block for the following macro calls:  Open file (\$OPFIL) Close file (\$CLFIL) Test file (\$TIFIL, \$TOFIL) Read record (\$RDREC) Write record (\$WRREC) Rewrite record (\$RWREC) Delete record (\$DLREC) Read block (\$RDBLK) Write block (\$WRBLK) Wait block (\$WTBLK)

Offsets definition macro calls can be specified only once per assembly procedure. They provide tags that are equated to specific offsets in argument structures and FIBs. For example, assuming that the address of an argument structure labeled FILE\_A has been loaded into a base register as follows:

```
LAB $B4,FILE_A
```

and assuming that \$CRPSB has been specified, the following address syllable can be used to refer to the argument structure entry that identifies the control interval size:

```
$B4.R_CISZ
```

This entry effectively points to the displacement FILE\_A+5 in the parameter structure.

Section 5 describes each displacement definition macro routine/call and its tags, displacements, and entry names in detail.

#### ASSUMPTIONS FOR FILE SYSTEM EXAMPLES

The example shown for each file system macro call description in Section 5 is based on the following assumptions.

1. All the following displacement definition macros were specified:

```
$CRPSB
$GTPSB
$GIPSB
$GIFAB
$GIKDB
$TFIB
```

2. The following argument structures were defined:

- a. Argument structure for create file (\$CRPSB)

```
FILE_A  DC      Z'0005'  LFN 5
        DC      <IDX01  PATHNAME PTR.
        RESV    2-$AF
        DC      Z'4900'  FILE ORG = I (INDEXED)
        DC      80      LOG. RCD. SZ. = 80
        DC      512     C.I. SZ. = 512
        DC      5       INIT. ALLOC. SZ. = 5
        DC      10      MAX. ALLOC. SZ. = 10
        DC      320     FREE SPACE = 320
        DC      2       LOCAL OVERFLOW ALLOCATION INCREMENT=2
        DC      1       NO. OF KEY DESCR. = 1
        DC      <KEY    KEY DESCRIPTOR PTR.
        RESV    2-$AF
        RESV    4,0     RESERVED
```

- b. The pathname addressed by the previous structure (FILE\_A)

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

- c. File information block (\$FIB)

MYFIB	DC	Z'0005'	LFN 5
	DC	Z'2000'	PROG. VIEW = ALLOW WRITE
	DC	<INBUF	USER RECORD PTR.
	RESV	2-\$AF	
	DC	256	MAX. INPUT RCD. SZ. = 256
	DC	256	OUTPUT RCD. SZ. = 256
	DC	Z'0000FFFF'	RESERVED
	DC	Z'0000'	RESERVED
	DC	<MYKEY	INPUT KEY PTR.
	RESV	2-\$AF	
	DC	Z'010A'	INPUT KEY=PRIMARY; KEY LENGTH=10
	RESV	2,0	RECORD ADDRESS
	RESV	2,0	RESERVED

When necessary, other structures are defined in the file system macro call examples.

### FILE MANAGEMENT FUNCTIONS

The file/directory management macro calls allow you to manipulate your files within the file system hierarchy (described in the System Concepts manual). Specifically, the calls allow you to:

- o Create a file
- o Get a file (reserve a file for processing)
- o Open a file
- o Close a file
- o Release a file
- o Remove a file from processing
- o Rename a file
- o Associate a logical file number with a pathname
- o Dissociate a logical file number from a pathname
- o Create a directory
- o Release a directory
- o Rename a directory
- o Change the working directory
- o Get the name of the current working directory

- o Expand pathname (develop a full pathname from a relative pathname)
- o Get information about a file
- o Test the status of an I/O activity (terminal)
- o Wait for the completion of an asynchronous I/O activity (terminal)
- o 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
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
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:

- o Open file - Close file
- o Create file - Release file
- o Associate file - Dissociate file
- o Get file - Remove file
- o Create directory - Release directory

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

- o Associate file
- o Dissociate file
- o Get file
- o Remove file
- o Create file
- o Release file
- o Rename file
- o Create directory
- o Change working directory
- o Release directory
- o Get working directory
- o 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 release file (\$RLFIL) and remove file (\$RMFIL). (Associate file and dissociate file provide a way of supplying a pathname as input to create file and get file.)

#### DATA MANAGEMENT FUNCTIONS

The data management macro calls handle only logical records; to do your own blocking and deblocking, you must use the storage management macro calls (see following discussion). If you handle your files at the logical record level (as described in the Data File Organizations and Formats manual), the data management macro routines can be used to perform any of the necessary I/O operations. Specifically, the data management macro calls allow you to:

- o Write a record
- o Rewrite a record
- o Read a record
- o Delete a record

\*

The definitions of arguments in the data management macro calls include identification of required file information block (FIB) entries, which are described at the beginning of this section. The macro calls to generate and change FIBs and to define FIB offsets are discussed in Section 4 and described in detail in Section 5.

Note that before any data management macro calls can be executed, the file must have been reserved and opened with the LFN supplied in the FIB. See "Get File" and "Open File" in Section 5.



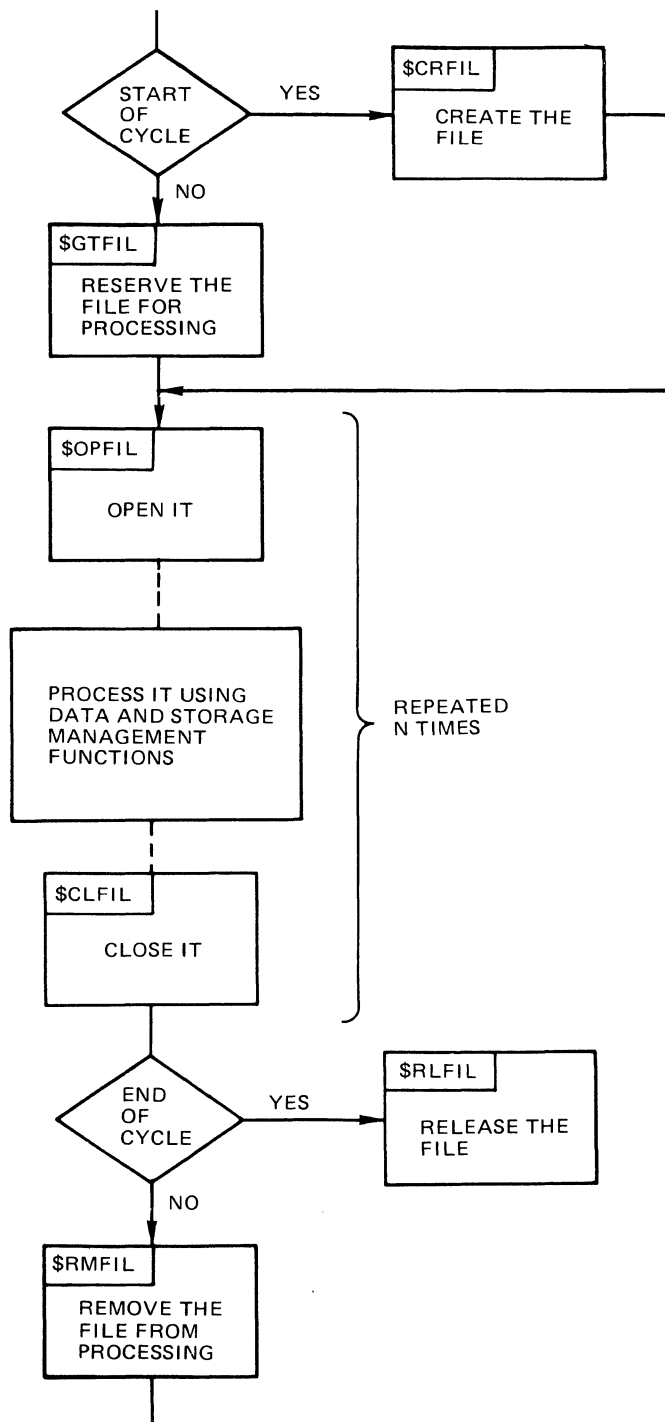


Figure 3-1. Life Cycle of a File

The macro routines/calls for data management functions are:

Delete record     \$DLREC  
Read record       \$RDREC  
Rewrite record    \$RWREC  
Write record      \$WRREC

\*

Section 5 describes each macro in detail.

### STORAGE MANAGEMENT FUNCTIONS

The storage management macro calls provide a primitive I/O interface for transferring blocks directly between your buffer and a file. Storage management itself is used by data management to perform input/output.

The complexities of blocking and deblocking logical records, and conforming at the same time to the various file organizations and formats, recommend against using storage management when dealing with I/O at the logical record level. To ensure maximum efficiency in terms of space and access, let the system (i.e., data management) handle your records.

However, with unblocked records or large blocks with simple fixed-length records that you want to block yourself, the storage management macro calls can be used to perform I/O transfers between your buffer and the file. In addition, the macros offer an asynchronous I/O facility that lets you overlap I/O transfers with task execution. Specifically, the storage management macro calls allow you to:

- o Read a block
- o Write a block
- o Wait for the completion of an I/O activity

Block size for disk files must be some multiple of physical sector sizes.

The definitions of arguments in the storage management macro calls include identification of required file information block (FIB) entries, which are described at the beginning of this section. The macro calls to generate FIBs and define FIB offsets are discussed in Section 4 and described in detail in Section 5.

Note that before any storage management macro calls can be executed, the file must have been reserved and opened with the LFN supplied in the FIB. See "Get File" and "Open File" macro descriptions in Section 5.

The macro routines/calls for storage management functions are:

Read block    \$RDBLK  
Wait block    \$WTBLK  
Write block   \$WRBLK

These macros are described in detail in Section 5.



## 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 structure, 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).

#### MONITOR SERVICES DATA STRUCTURES

Monitor service data structures are the following:

- o Request blocks
- o Parameter block and wait list
- o 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 Program Preparation 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 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, always set)
- o S-bit, or semaphore
- o R-bit, or return request
- o D-bit, or delete

The assignment of these control bits within the request blocks is shown in Appendix A.

Request blocks also carry parameters from the issuing task to the requested task or to the system service. A variable-length area is available in the task request block for intertask communication of application-specific parameters.

The specific request blocks generated by the macro calls listed below are:

- o Task request block (TRB)
- o Input/output request block (IORB)
- o Message group request blocks (MGCRB, MGIRB, MGRRB)
- o Semaphore request block (SRB)
- o Clock request block (CRB)

A field in a request block that is not set by its corresponding argument in the request block macro call is set to zero when the block is generated. You may change these zero fields to any desired value.

The first four words of the request blocks are identical in format (see Appendix A for a diagram of each structure). Additional words carry parameter information specific to the request block type.

There is an offsets macro call for each form of request block. These macro calls, described in Section 5, create explicit labels for request blocks.

The macro routines/calls to generate the request blocks are:

Clock request block	\$CRB
Input/output request block	\$IORB
Message group request blocks	(\$MGCRB, \$MGIRB, \$MGRRB)
Semaphore request block	\$SRB
Task request block	\$TRB

Section 5 describes these macros in detail.

#### Parameter Block and Wait List

The macro routines listed below generate data structures with a format different from that of the request blocks described above. These data structures are:

- o Parameter block
- o Wait list

Their formats are shown in Appendix A.

The macro routines/calls for generating a parameter block and a wait list are:

Parameter block, generate \$PRBLK  
Wait list, generate \$WLIST

Section 5 describes these macros in detail.

### Request Block Offsets

The request block offsets macro routines generate data structure definitions for request blocks that will be constructed at a later time by application code. The request block definitions supplied by the offsets macro calls have explicit labels for each entry in the structure, allowing symbolic displacement references to be made in application code.

NOTE: The request block macro calls previously described generate actual request blocks; the displacement entry labels are not included. The contents of the request block fields are set according to the arguments supplied in the macro calls.

No arguments are specified with the offsets macro calls. Only one request block offsets macro of a particular type is required in an assembly program referencing its entries.

You may include several different templates containing the four common request block fields in your code because the template for each structure begins with a unique identification character prefix. This technique avoids assembly error notices of multiple defined symbols, when, for example, the control word 1 entry label of a TRB and IORB are both included in the application source program.

Note that a program may use a request block macro routine to initially define a desired block, and also include that same type request block offsets to facilitate modification of the initial block by executing code.

The request blocks offsets generated by the offsets macro calls are:

	<u>Generated by macro call:</u>
o Task request block (TRB) offsets	\$TRBD
o Input/output request block (IORB) offsets	\$IORBD
o Clock request block (CRB) offsets	\$CRBD



Generated by  
macro call:

- o Semaphore request block (SRB) offsets           \$SRBD
- o Message group request blocks (MGCRB,  
  MGIRB, MGRRB) offsets                   \$MGCRT  
  \$MGIRT  
  \$MGRRT

Section 5 describes each macro in detail.

#### FILE SYSTEM DATA STRUCTURES

File system data structures are the following:

- o File information block (FIB)
- o Offsets definitions

#### File Information Block (FIB)

The FIB is the principal means of communication between the application and the File System.

Table 3-1 and Appendix A show the format and content of the file information block. The macro routine/call to build a file information block, or alter its contents, or provide labels for its entries, is:

File information block   \$FIB

Section 5 describes this macro in detail.

#### Offsets Definitions

With offsets definition macro calls, you can refer to specific locations in the file information block and in the various argument structures used by certain file system macro calls.

The offsets definition macro calls, which can be specified only once per assembly procedure, provide tags that are equated to specific offsets in the argument structures and in the FIBs.

Macro calls are provided to define tags for the following structures:

- o Create file (\$CRFIL) macro call argument structure
- o Get file (\$GTFIL) macro call argument structure
- o Get file information (\$GIFIL) argument structure

- o File attribute block pointed to by \$GIFIL argument structure
- o Key descriptor block pointed to by \$GIFIL argument structure
- o File information block

Only the macro call name is specified; these macro calls have no arguments.

The macro routines/calls for offsets definition are:

Create file parameter block structure offsets \$CRPSB

File information block offsets \$TFIB

Get file information file attribute block offsets \$GIFAB

Get file information key descriptor block offsets \$GIKDB

Get file information parameter structure block offsets \$GIPSB

Get file parameter structure block offsets \$GTPSB

These macros are described in detail in Section 5.

## SECTION 5

### MACRO ROUTINE/CALL DESCRIPTIONS

This section describes in detail the use, structure, functions, and return status error conditions for all system services macros referred to and listed in Sections 2, 3, and 4, and provides an example for most macros. For easy reference, the descriptions are in alphabetic order by specific function name (see column 2 in Table 1-1).

Each description includes a reference to the command (if there is one) that performs the equivalent function (see the Commands manual). Also included for each description is a representative list of possible return status (error) codes and the corresponding error condition for each return status. (See the System Messages manual for a complete list of GCOS 6 return status codes and corresponding system messages.)

# ABORT GROUP

## ABORT GROUP (MOD 400 ONLY)

Macro Call Name: \$ABGRP

Function Code: 0D/0A

Equivalent Command: Abort Group (ABORT\_GROUP)

Terminate the indicated task group and delete it.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of abort status

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

location of group identifier

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

FUNCTION DESCRIPTION:

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

\*

- NOTES:
1. The abort status code supplied by argument 1 is placed in \$R6; if this argument is omitted \$R6 is assumed to contain the abort status code to be used.
  2. The group identification supplied by argument 2 is placed in \$R2; if the argument is omitted, \$R2 is set to zero to designate that the issuing task group is to be aborted.
  3. If a task group other than the issuing task group was aborted, \$R1 and \$R2 contain the following information upon return to the issuing task.

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

0000 - Abort task group status set  
0806 - Task group not found

\$R2 - Group id of aborted task group

Example:

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

```
$ABGRP    =40
```

# ABORT GROUP REQUEST

## ABORT GROUP REQUEST

Macro Call Name: \$ABGRQ

Function Code: 0D/07

Equivalent Command: Abort Group Request (AGR)

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

FORMAT:

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

ARGUMENT DESCRIPTION:

location of abort status

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

location of group identifier

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

## FUNCTION DESCRIPTION:

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

Files that are open and in use by this task group are closed. The abort process will not complete until all outstanding input/output orders are completed.

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

An abort group request for a spawned group is equivalent to an abort group monitor call.

- NOTES:
1. The abort status code specified by argument 1 is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the abort status code to be used.
  2. The group identification specified by argument 2 is placed in \$R2; if this argument is omitted, \$R2 is set to zero to designate that the issuing task group request is to be aborted.
  3. If the current request of a task group other than the issuing task group was aborted, \$R1 and \$R2 contain the following information upon return to the issuing task.

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

0000 - Abort task group request status set  
0806 - Task group not found

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

## Example:

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

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

# ACCOUNT IDENTIFICATION

## ACCOUNT IDENTIFICATION

Macro Call Name: \$ACTID

Function Code: 14/02

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of account id field address

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

### FUNCTION DESCRIPTION:

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

The entire user id is returned by the user identification (\$USRID) macro call.

NOTES: 1. The address of the receiving account 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

Example:

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

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

# ACTIVATE GROUP

## ACTIVATE GROUP

Macro Call Name: \$ACTVG

Function Code: 0D/09

Equivalent Command: Activate Group (ACTG)

Reactivate a previously suspended task group.

FORMAT:

[label] \$ACTVG [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 reactivated.

FUNCTION DESCRIPTION:

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

If the group id argument is \$B, the previously rolled out batch task group is rolled in when all online task groups have returned memory to the batch pool. Any task group that has explicitly rolled out the batch task group (through a \$SUSPG \$B macro call) should roll in the batch task group before terminating. If the task group does not issue a \$ACTVG \$B macro call before terminating, or if the task group is aborted, the operator must issue an ACTB command to allow batch roll in.

Before it terminates, any online task group that has suspended another online task group (through a \$SUSPG macro call) should reactivate that task group. If the suspending task group does not issue a \$ACTVG macro call, or if the suspended task group is aborted, the operator must issue an ACTG command for the suspended task group to resume.

NOTES: 1. The group id of the task group to be reactivated, supplied by argument 1, is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the correct group id.

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

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

0000 - No error

0806 - Specified task group not currently defined

080D - Specified task group not currently suspended

\$R2 - Group id as supplied

Example:

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

```
ACTGAA    $ACTVG    =G1
```

# ASSOCIATE FILE

## ASSOCIATE FILE

Macro Call Name: \$ASFIL

Function Code: 10/10

Equivalent Command: Associate Path (ASSOC)

Associate a logical file number (LFN) with a specific path-name. 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 ). If you associate an incomplete pathname with the LFN, it can be completed at a later time by a get file macro call using the colon (:) option. 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. Finally, although a single pathname can be associated with several LFNs, a given LFN can be associated with only one pathname at any given time; after a file reservation (see get file) has been established using a specific LFN, subsequent associations of the same LFN will alter the LFN/pathname relationships but will not affect current file reservation. 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

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

In addition to the above, any system service codes received by the file manager are passed on through \$R1.

Example:

This example assumes that \$B4 was loaded with the address of the label FILE A (i.e., LAB \$B4,FILE A); therefore, the macro call to associate the path identified in the create file example (i.e., VOL03 SUBINDEX.A FILE\_A) with LFN 5 is coded as follows:

```
ON1AA $ASFIL
```

FILE A was previously defined in "Assumptions for File System Examples" in Section 3; as a result of issuing the \$ASFIL macro call, the first two entries in that structure are referred to by the system.

# BOUND UNIT IDENTIFICATION

## BOUND UNIT IDENTIFICATION

Macro Call Name: \$BUID

Function Code: 14/06

Equivalent Command: USER BUID

Returns the file 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 name of the currently executing bound unit to a specified field in the issuing task. The name returned is that specified in the Linker NAME statement.

- 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.

Example:

In this example, \$B4 is loaded with the address (BUNAME) of a 6-character 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
```



# CANCEL CLOCK REQUEST

## CANCEL CLOCK REQUEST

Macro Call Name: \$CNCRQ

Function Code: 05/01

Equivalent Command: None

Cancel a previously issued clock request.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of CRB address

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

### FUNCTION DESCRIPTION:

This call removes a no longer needed but previously queued CRB from the timer queue. The CRB must have previously been placed on the queue by a request clock (\$RQCL) macro call.

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

- NOTES:
1. The address of the CRB to be disconnected from the queue, supplied by argument 1, is placed \$B4; if this argument is omitted, \$B4 is assumed to contain the correct address.
  2. On return, \$R1 and \$B4 contain the following information:

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

0000 - No error

0404 - CRB not connected to basic timer  
queue

\$B4 - Address of CRB

Example:

See the example given for the wait on request list macro  
call in this section.

# CANCEL REQUEST

## CANCEL REQUEST

Macro Call Name: \$SCANRQ

Function Code: 0C/01

Equivalent Command: None

Cancel a previously issued request made through a \$RQTML or \$TRB macro call.

### FORMAT:

[label] \$SCANRQ [location of address of request block]

### ARGUMENT DESCRIPTION:

location of address of request block

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

### FUNCTION DESCRIPTION:

This call cancels a previously issued request. The call is used to cancel a request established by a request terminal (\$RQTML) or task request block (\$TRB) macro call.

- NOTES:
1. The address of the request block containing the request to be canceled, supplied by argument 1, is placed in \$B4. If this argument is omitted, the system assumes that \$B4 contains the address of the request block.
  2. On return, \$R1 contains one of the following status codes:
    - 0000 - Terminal request canceled
    - 0803 - Illegal request block address (request block not found)

0817 - Memory access violation

083C - Terminal request already posted

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

Example:

In this example, the \$SCANRQ macro call is used to cancel the request established by a request terminal (\$RQTML) macro call. (See the example for the request terminal macro call.)

```
END_RQ    $SCANRQ    !IORB
```

# CANCEL SEMAPHORE REQUEST

## CANCEL SEMAPHORE REQUEST

Macro Call Name: \$CNSRQ

Function Code: 06/01

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of SRB address

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

### FUNCTION DESCRIPTION:

This call removes a specified SRB from its semaphore request queue. The SRB must have been queued as the result of a previously issued request semaphore macro call. The SRB address specified in argument 1 of the cancel semaphore request call must be the same SRB address used in the request semaphore call.

When executed, this function increments the counter established by the define semaphore macro call, and previously decremented by the request semaphore macro call.

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

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

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

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

0000 - No error

0502 - Invalid SRB

\$B4 - Address of SRB (as supplied)

Example:

In this example, the \$CNSRQ macro call is used to cancel the semaphore request used in the example for the request semaphore macro call. It is assumed that the task did not need the resource.

\$CNSRQ !SRB

# CHANGE WORKING DIRECTORY

## 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

0201 - Illegal pathname

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

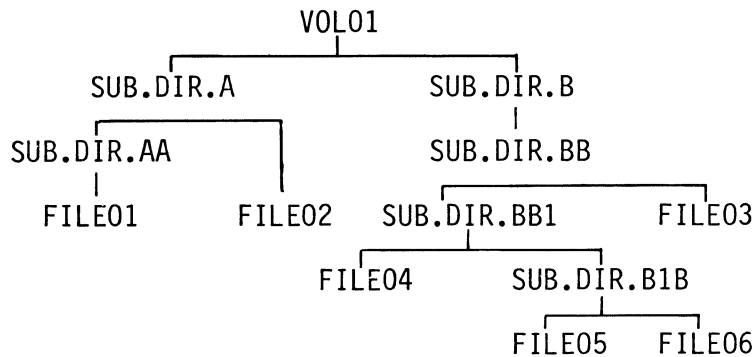
0228 - Illegal file type (not a directory)

In addition to the above, any system service codes received by the file manager are passed on through \$R1.

Example:

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





The current working directory is SUB.DIR.B1B and you want to access FILE01 from subdirectory SUB.DIR.AA. You need not specify the absolute pathname to FILE01 if you specify the macro call \$CWDIR to SUB.DIR.AA as shown below. The file can then be accessed with the simple pathname FILE01.

To change to this working directory, you can use the \$CWDIR macro call:

```
$CWDIR !CHGPTH
```

to identify the path:

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

or

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

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

# CLEAN POINT

## CLEAN POINT

Macro Call Name: \$CLPNT

Function Code: 0C/13

Equivalent Command: None

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

FORMAT:

[label] \$CLPNT

ARGUMENT DESCRIPTION:

None

FUNCTION DESCRIPTION:

This macro call results in the following:

1. The buffers of updated records of files accessed by the task group are written to disk.
2. If the end of data record for a disk file accessed by the task group is altered, the directory record for that file is updated.
3. All record locks set by this task group are unlocked, thus allowing other users to continue processing.

Record locking, a file system mechanism, provides multi-user interface protection for shared file access. A record, when accessed by a user, is locked by a lock applied to the control interval(s) where the record is located. Locking is on a first-come first-served basis. Another user (task group)

sharing this file is denied access to that record and any other record in the same control interval, until the previous user unlocks the record.

The only limit to the number of locks at one time is the amount of memory dedicated to the lock pool at system building. (The lock pool is that memory area where locked records are recorded.)

Record locks for a file must be specifically requested when the file is reserved through a get file (\$GTFIL) macro call or by a GET command. Once record locking for a file is requested, any access (read or write) causes a lock. Once locked, records are unlocked only when a clean point (\$CLPNT) macro call is issued or the file is closed. (Abnormal task group termination also causes records to be unlocked.)

Records should be unlocked when there is no further need to lock them. Otherwise, when records remain locked, lock pool overflow or deadlock record contention may result. The description of the get file (\$GTFIL) macro call later in this section has more details about record locking.

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

- NOTES:
1. To perform the clean point function in a COBOL program, the user must call an assembly language subroutine that contains the \$CLPNT macro call(s).
  2. On return, \$R1 contains one of the following status codes:
    - 0000 - No error
    - 0225 - Insufficient system memory for buffer or structures
    - 0226 - Insufficient user memory for buffer or structures
  3. Any system service error codes received by data management are passed on through \$R1.

# CLEAR EXTERNAL SWITCHES

## CLEAR EXTERNAL SWITCHES

Macro Call Name: \$CLRSW

Function Code: 0B/02

Equivalent Command: Modify External Switches (MSW)

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

external switch name ... external switch name

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

### FUNCTION DESCRIPTION:

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

\$R2 is the mask word. Each bit that is 1 in \$R2 causes the corresponding bit in the external switch word to be set off; each bit that is 0 causes the corresponding bit to remain unchanged.

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

- o If bit 11 is 0, no switch set off had previously been set on.
- o If bit 11 is 1, at least one switch set off had previously been set on.

NOTES: 1. The bits corresponding to the external switches in the arguments are set on in \$R2; if no arguments are supplied, \$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 - External switch word after modification

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

0 - No switch set off was on

1 - At least one switch set off was on

Example:

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

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

# CLOCK REQUEST BLOCK

## CLOCK REQUEST BLOCK

Macro Call Name: \$CRB

Function Code: None

Equivalent Command: None

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

### FORMAT:

```
[label] $CRB [CRB type],  
            [issuing task suspension option],
```

or

```
[termination action],  
[interval value]
```

### ARGUMENT DESCRIPTION:

#### CRB type

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

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

#### issuing task suspension option

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

- WAIT - Suspend the issuing task until the clock request has been satisfied (set w-bit to 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 (termination action) must be omitted.

#### termination action

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

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

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

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 Program Preparation 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

## CLOCK REQUEST BLOCK OFFSETS (MOD 400 ONLY)

Macro Call Name: \$CRBD

Generated Label Prefixes:

```
CRB label  C_RRB/C_SEM  
           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 Program Preparation manual for more information about SAF/LAF independent code.

# CLOSE FILE

## 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:

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

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.

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

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

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

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

NORMAL is the default value for this macro call.

{LEAVE}  
{LEV }

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.

{UNLOAD}  
{UNL }

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. Either an output close (write permission granted) or an input close (write permission denied) can be performed. Note that when a tape volume is opened for storage management access, and both volume and file names were not specified, then no trailer labels nor tape marks were written; in that case it is the user's responsibility.

1. Output close (write permission):

- 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 and write operations were performed, the trailer label group and EOD tape mark are written. Data and/or files located in front of the current position of the tape are destroyed.

If no write operations were performed, or an input close is performed (as described below), existing data and/or files located in front of 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.

2. Input close (no write permission)

- 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

- 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

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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

## 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 task group of 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).

\*

## FUNCTION DESCRIPTION:

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

- NOTES:
1. The address of the command input 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 area size supplied by argument 2 is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the correct size.
  3. If argument 3 is L, \$R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, \$R7 is set to 1 to designate that the record area begins in the right byte of the specified address. Any other value for argument 3 is assumed to designate the location of the byte offset to be used, and is placed in \$R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address, and \$R7 is set to zero.
  4. On return, \$R1, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error  
0817 - Memory access violation

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

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

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

<u>Value</u>	<u>File Type</u>
02	Fixed relative
10	Line/serial printer
11	Card reader
12	KSR (MDC-connected)
1A	Bidirectional MLCP

<u>Value</u>	<u>File Type</u>
1B	BSC
1E	Output-only MLCP
30	Variable sequential (spanned records)
32	Relative
33	Indexed (data)
34	Indexed (index)

\$B4 - Input record area address

Example:

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

```

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

```



# COMMAND LINE PROCESS

## 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 on behalf of the macro call is provided with a request block that has been constructed by the system to contain the edited arguments in system standard task request block format. The task that issues this macro call waits for the completion of the spawned task before

continuing its own processing. The spawned task passes the completion status (\$R1) to the issuing task.

- NOTES:
1. The 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

160A - Invalid bound unit pathname for first argument

160B - 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

MYPROG.0 in the working directory. If MYPROG.0 does not already exist, it will be created.

\$CMDLN !LINE,=LENGTH

·  
·  
·

LINE	TEXT	'ASSEM MYPROG	-SZ	5	-COUT	>SPD>LPT01'
LENGTH	EQU	2*(\$-LINE)				

# CONSOLE MESSAGE SUPPRESSION

## CONSOLE MESSAGE SUPPRESSION

Macro Call Name: \$CMSUP

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

Equivalent Command: None

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

FORMAT:

[label] \$CMSUP [keyword]

ARGUMENT DESCRIPTION:

keyword

One of the following values:

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

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

If this argument is omitted, OFF is assumed.

FUNCTION DESCRIPTION:

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

When console message suppression is turned on, operating system components such as storage management will not issue error messages to the operator terminal - either directly (through the facility offered by the operator information message macro call) or indirectly (through the facility offered by the report error condition macro call, described later in this section). Turning on console message suppression does not disable these facilities; rather it prevents

the system components from using the facilities to report anything other than catastrophic errors.

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

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

NOTE: On return, \$R1 contains one of the following subfunction codes:

0002 - Turn on suppression  
0003 - Turn off suppression

Example:

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

SUPON      \$CMSUP    ON

# CREATE DIRECTORY

## CREATE DIRECTORY

Macro Call Name: \$CRDIR

Function Code: 10/A0

Equivalent Command: Create Directory (CD)

Creates a new directory in the file system hierarchy. This function is typically done outside of program execution.

### FORMAT:

[label] \$CRDIR [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.

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, when expanded, identifies the directory in the hierarchy in which to create the new directory and the name of the new directory itself.

reserved

A 4-byte entry containing "zeros."

### FUNCTION DESCRIPTION:

This request can be used only to create new directories, which are created with:

- o 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.
- o 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).
- o 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

0201 - Illegal pathname

0205 - Illegal argument

0209 - Same 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

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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.AΔ'
```

The macro call can be specified as follows:

```
$CRDIR  !SUBDIR
```



# CREATE FILE

## CREATE FILE

Macro Call Name: \$CRFIL

Function Code: 10/30

Equivalent Command: Create File (CF)

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

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

#### reserved

This 1-byte field must contain zeros.

#### logical record size

A 2-byte value that specifies the length of the longest logical record in the file. If the file organization entry, above, specified R, S, or I, this size does not include headers. If the file organization entry specified 5, the size includes the 2-byte record header. There are no headers for a file organization shown as 2.

#### control interval size

A 2-byte value that specifies the unit of file space allocation, as follows:

For fixed-relative files: defines only the unit of space allocation and can be specified as any multiple of 128 bytes which includes both CI and logical record header information.

For all other files: defines the size of a data transfer to/from main memory (and thus the buffer size); must be specified as a multiple of 256 bytes, including CI and logical record header information.

Zeros in this entry result in a size of 512 bytes.

#### initial allocation size

A 2-byte value that specifies the number of control intervals to be allocated to the file at file-creation time; zeros in this entry indicate that no space is to be allocated initially.

#### allocation increment size

A 2-byte value that specifies the number of additional control intervals to be dynamically allocated to the file at load time if the number specified in the "initial allocation size" entry are filled. Zeros in this entry indicate a value of 40 physical sectors.

#### maximum allocation size

A 2-byte value that specifies the maximum number of control intervals that can be allocated to the file. Zeros in this entry indicate that there is no limit.

free space per control interval

A 2-byte value, as follows:

For indexed files: The number of bytes to be left free in each control interval at file-loading time; this permits records to be inserted in the file without causing overflow.

For all other file organizations: Contains zeros.

local overflow allocation increment

A 2-byte value that sets the frequency at which a local overflow control interval will be allocated when an indexed file is loaded. For example, if this value is 10, one local overflow control interval will be allocated after every ten data control intervals are allocated.

number of key descriptors

A 2-byte value, as follows:

For indexed files: Must contain Z'0001'

For all other file organizations: Contains zeros.

pointer to key descriptor

A 4-byte address, as follows:

For indexed files: Any address form valid for an address register; points to a key descriptor structure that defines the key field in records stored in an indexed file. This structure is described below.

For all other file organizations: Contains zeros.

reserved

An 8-byte entry containing zeros.

The key-descriptor structure pointed to by the pointer to key descriptors entry in the argument structure described above must contain the following entries in the order shown:

record type range

A 4-byte value that must contain Z'0000FFFF'.

number of key components

A 1-byte value that must contain the value 1.

reserved

A 9-byte entry containing zeros.

key component data type

A 1-byte entry that contains an ASCII C for character data or D for decimal data.

key component size

A 1-byte binary value that specifies the length of the key field in the record.

key component offset

A 2-byte binary value that specifies the number of 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:

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

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

- o If only an LFN is specified, it must previously have been associated with a pathname (see the associate file macro call).

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

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

Initial allocation and allocation increment sizes (although stated in terms of control intervals) cannot resolve to a value greater than 8191 logical sectors for mass storage units, and 8191 physical sectors for diskettes and cartridge disks. After the space is allocated, the system reserves it with "exclusive" concurrency control; as a result, it is not necessary to issue a get file macro call before an open file macro call in order to access the file exclusively. If the file being created is a temporary file (see the "pathname pointer" entry described in the argument structure description), it can be released (i.e., deleted) through the remove remove file macro call.

Offset tags for the parameter structure can be defined by the \$CRPSB macro call.

- 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

- 0205 - Illegal argument
- 0206 - Unknown or illegal LFN
- 0208 - LFN or file already open
- 0209 - Same named subdirectory not found
- 020C - Volume not found
- 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

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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'0000FFFF'	RECORD TYPE RANGE
	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_AΔ'
```

With the preceding definitions having been made, the following macro call will create FILE\_A:

```
DOMYAA    $CRFIL    !FILE_A
```



# CREATE FILE PARAMETER STRUCTURE BLOCK-OFFSETS

## CREATE FILE PARAMETER STRUCTURE BLOCK - OFFSETS

Macro Call Name: \$CRPSB

Associated Macro Call: \$CRFIL

Generated Offsets Tags:

<u>Tag</u>	<u>Corresponding Offsets (in Words)</u>	<u>Entry Name</u>
R_LFN	0	Logical file number (LFN)
R_PTHP	+1	Pointer to path
R_ORG	+3	File organization (first byte)
R_RFU	+3	Reserved (second byte)
R_LRSZ	+4	Logical record size
R_CISZ	+5	Control interval size
R_IASZ	+6	Initial allocation size
R_AISZ	+7	Allocation increment size
R_MASZ	+8	Maximum allocation size
R_FREE	+9	Amount of free space per C.I.
R_LOV	+10	Local overflow allocation increment
R_NKD	+11	Number of key descriptors
R_KDP	+12	Pointer to key descriptors (see \$GIKBD macro call)
R_SZ	+18	Size of structure (in words); not a field in the block

# CREATE GROUP

## CREATE GROUP

Macro Call Name: \$CRGRP

Function Code: 0D/02

Equivalent Command: Create Group (CG)

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

### FORMAT:

```
[label] $CRGRP [location of group identifier],  
               [location of memory pool 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 does not have the \$ character as its first character.

location of memory pool identifier

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

#### location of base level

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

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

#### location of high logical resource number

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

#### FUNCTION DESCRIPTION:

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

- NOTES:
1. The group identifier supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the group identifier to be used.
  2. The memory pool identifier supplied by argument 2 is placed in \$R4; if this argument is omitted, \$R4 is set to zero to indicate that the memory pool of the issuing task group should be used by the newly created task group.
  3. The base priority level supplied by argument 3 is placed in \$R5; if this argument is omitted, \$R5 is assumed to contain the base priority level to be used.
  4. The high LRN value supplied by argument 4 is placed in \$R6; if this argument is omitted, \$R6 is set to zero to indicate that the value of the highest LRN created for the system task will be used.
  5. The high LFN value specified by argument 5 is placed in \$R7; if this argument is omitted, \$R7 is set to 15.
  6. The address of the root entry name supplied by argument 6 is placed in \$B2; if this argument is omitted, \$B2 is assumed to contain the address of the bound unit to be executed by the lead task.
  7. On return, \$R1 and \$R2 contain the following information:

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

- 0000 - No error
- 0601 - Insufficient memory
- 0602 - Insufficient memory
- 0804 - Group id in use
- 0806 - Invalid group id
- 0807 - Invalid memory pool identifier
- 0808 - Invalid base level
- 0809 - Invalid high LRN
- 080A - Invalid high LFN
- 080C - Unresolved start address
- 160A - Invalid pathname
- 160B - Insufficient memory

\$R2 - Group id of created group

Example:

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

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

# CREATE OVERLAY AREA TABLE

## 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 Programmer's Reference 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

1602 - Invalid argument (size or number of overlay areas)

160A - Insufficient memory

\$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 TASK

## CREATE TASK

Macro Call Name: \$CRTSK

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

Equivalent Command: Create Task (CT)

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

### FORMAT:

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

### 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.

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 within the bound unit of 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

0809 - LRN too large

0813 - Referenced LRN already in use or  
invalid

0827 - Bound unit file not fixed-relative

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 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
          .
          .
          .
```

# DEFINE SEMAPHORE

## DEFINE SEMAPHORE

Macro Call Name: \$DFSM

Function Code: 06/04

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of semaphore identifier

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

location of initial value of semaphore

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

### FUNCTION DESCRIPTION:

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

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

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

- NOTES:
1. The semaphore identifier supplied by argument 1 is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the identifier to be used.
  2. The initial semaphore value supplied by argument 2 is placed in \$R2; if this parameter is omitted, \$R2 is set to 1.
  3. On return, \$R1 and \$R6 contain the following information:

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

0000 - No error

0503 - Semaphore id previously defined in issuing task group

\$R6 - Semaphore identifier (as supplied)

**Example:**

In this example, the \$DFSM macro calls define two semaphores named TH and LK.

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

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 GROUP

## DELETE GROUP

Macro Call Name: \$DLGRP

Function Code: 0D/04

Equivalent Command: Delete Group (DG)

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

### FORMAT:

[label] \$DLGRP [location of group identifier]

### ARGUMENT DESCRIPTION:

location of group identifier

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

### FUNCTION DESCRIPTION:

This call removes all data structures, built by the create group macro call issued with this group id, when the group becomes dormant. No further enter group request macro calls can be issued for this task group once the delete group macro call has been issued.

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

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

- NOTES: 1. The group id supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is set to zero to designate that the issuing task group is to be deleted.
2. On return, \$R1 and \$R2 contain the following information:

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

0000 - Delete task group status set  
0806 - Task group not found

\$R2 - Group id of deleted task group

Example:

In this example, the \$DLGRP macro call causes the task group in which the macro call is executed to be deleted when the group's tasks are all terminated with no queued group requests.

NOABA \$DLGRP

# DELETE RECORD

## DELETE RECORD

Macro Call Name: \$DLREC

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

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

fib address

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

```
{CURRENT}
{CUR}
```

Indicates that the record read by the immediately preceding read next or read with key (i.e., the last record read; see "Read Record") macro call is to be deleted. (This is the default value for this macro call.) You must code the following FIB entry:

logical file number

## KEY

Indicates that the record identified by the key value pointed to by the FIB is to be deleted. 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 \$TFIB 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  
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

In addition to the above codes, any system service codes received by the data manager are passed on through \$R1.

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 TASK

## DELETE TASK

Macro Call Name: \$DLTSK

Function Code: 0C/04

Equivalent Command: Delete Task (DT)

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of logical resource number

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

### FUNCTION DESCRIPTION:

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

If the task is executing, the macro call causes its definition to be deleted when the task next issues a terminate macro call and there are no request blocks in its request queue. No further request task macro calls can be issued for this task after the delete task macro call has been issued.

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

NOTES: 1. The LRN specified by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is set to -1 to denote that the task issuing the macro call is to be deleted.

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

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

0000 - No error  
0802 - Invalid LRN

\$R2 - LRN of deleted task

Example:

In this example, the \$DLTSK macro call causes the task known as logical resource number 10 (decimal) within the issuing task's task group to be deleted. If the \$DLTSK macro call shown in this example was executed in the same task group as the \$CRTSK macro call used in the first example of the create macro call description, the task created by that example would be deleted.

```
DEL_AA    $DLTSK    =10
```

# DISABLE DEVICE ON ATTENTION

## DISABLE DEVICE ON ATTENTION (MOD 400 ONLY)

Macro Call Name: \$DSDV

Function Code: 02/02

Equivalent Command: None

Disable the specified device when an attention interrupt occurs.

### FORMAT:

[label] \$DSDV [location of LRN]

### ARGUMENT DESCRIPTION:

location of LRN

Any address form valid for a data register; provides the location of the logical resource number (LRN) of the device to be disabled. The LRN must be a system LRN (defined at system building).

### FUNCTION DESCRIPTION:

This call sets the device status to disabled when an attention interrupt occurs. It is typically used to ensure that volume swaps are detected by the application program.

A disabled device is logically unavailable and returns a 0108 status until enabled. To regain use of the device, you must enable it (see the enable device macro call) each time an attention interrupt occurs.

- NOTES:
1. The LRN specified by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the correct LRN.
  2. On return, \$R1 and \$R2 contain the following information:



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

0000 - No error  
0102 - Invalid LRN

\$R2 - LRN of device

Example:

In this example, the \$DSDV macro call is used to disable the device whose LRN is 15 whenever an attention interrupt occurs.

DISPT \$DSDV =15

# DISABLE USER TRAP

## DISABLE USER TRAP

Macro Call Name: \$DSTRP

Function Code: 0A/02

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of trap number

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

### FUNCTION DESCRIPTION:

This macro call disables the hardware trap vector specified by argument 1. All subsequent occurrences of the specified trap are handled by the system's default trap handling routine until an enable user trap macro call is later issued for the trap. (Section 7 describes trap handling.)

- NOTES:
1. The trap number of the trap to be disabled, 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 disabled.
  2. On return, \$R1 and \$R2 contain the following information:

\$R1 - Return status:

0000 - No error

0342 - Illegal trap number

0343 - A previously signalled trap is  
still pending.

\$R2 - Trap number supplied in macro call

Example:

See the example given for "Connect Trap Handler."

# DISSOCIATE FILE

## DISSOCIATE FILE

Macro Call Name: \$DSFIL

Function Code: 10/15

Equivalent Command: Dissociate Path (DISSOC)

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

### FORMAT:

[label] \$DSFIL [parameter structure address]

### ARGUMENT DESCRIPTION:

parameter 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.

logical file number

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

### FUNCTION DESCRIPTION:

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

NOTES: 1. If the argument is coded, the 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  
0206 - Unknown or illegal LFN

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

Example:

In this example, the macro call identifies the same argument structure used in the associate file macro call described earlier (i.e., FILE\_A). The effect of the dissociate macro call is to remove the logical connection between the LFN and the pathname IDX01, as established by the associate file macro call.

```
FILE_A    DC      5          LFN5
           $DSFIL    !FILE-A
```

# ENABLE DEVICE

## ENABLE DEVICE (MOD 400 ONLY)

Macro Call Name: \$ENDV

Function Code: 02/04

Equivalent Command: None

Set the resource control table (RCT) of the specified device to enabled status.

### FORMAT:

[label] \$ENDV [location of LRN]

### ARGUMENT DESCRIPTION:

location of LRN

Any address form valid for a data register; provides the LRN of the device whose RCT is to be set to enabled status. The LRN must be a system LRN (defined at system building).

### FUNCTION DESCRIPTION:

This call turns off the device disabled indicator (bit 10 of the R\_FLGS entry) in the RCT of the specified device. \$ENDV can be used in synchronizing task operation with device availability. A task can issue a disable device on attention macro call (\$DSDV) to request notification of an interrupt. When the interrupt occurs, the device driver will set bit 10 (device disabled) and bit 8 (attention has occurred) of R\_FLGS. When a ready interrupt is generated, the task can clear the disabled status by resetting bit 10 through the \$ENDV macro call.

After clearing bit 8, using the reset device attention (\$RDVAT) macro call, and waiting for the device ready interrupt to occur, a task can use the enable device (\$ENDV) and the reset device attention (\$RDVAT) macro calls to clear bits 8 and 10 to initial states.

NOTES: 1. The LRN supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the correct LRN.

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

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

0000 - No error  
0102 - Invalid LRN

\$R2 - LRN of device

Example:

In this example, the \$ENDV macro call is used to set the RCT of the device whose LRN is 15 to the enabled status. It is assumed that a ready interrupt has been generated for the device.

```
ONDEVA    $ENDV    =15
```

# ENABLE USER TRAP

## 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 "Connect Trap Handler."

# ERROR LOGGING, END

## ERROR LOGGING, END

Macro Call Name: \$ELEND

Function Code: 02/09

Equivalent Command: None

Terminate the error logging function for the named device and provide summary error information.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of device-name

Any address form valid for a data register. Provides the address of the device-name for the peripheral (noncommunications) device for which the logging function is to be terminated.

address of user's logging table

Any address form valid for a data register. Provides the address of the previously generated 27-word user's error logging table. (See Table 5-1 in the discussion of error logging information exchange (\$ELEX) macro call.)

### FUNCTION DESCRIPTION:

This call terminates the error logging function previously activated for this device. The system transfers logging information values from the system logging table into the user's logging table in memory (i.e., delivers to the user information (1) about the current status of the error logging table up to the time of the macro call, and (2) about the last error that occurred. (See Table 5-1.)

- NOTES:
1. When argument 1 is specified, the location of the device-name is placed in \$B2. If the argument is omitted, the system assumes that \$B2 contains a pointer to the device-name.
  2. When argument 2 is specified, the address of the user's logging table is placed in \$B4. When the argument is omitted, the system assumes that \$B4 contains a pointer to that table.
  3. The device name must have been specified (or defaulted) in a previously executed \$ELST macro call for that device.
  4. On return, \$R1 contains one of the following status codes:
    - 0000 - Error logging terminated successfully.
    - 3B01 - Invalid argument (1 or 2).
    - 3B02 - Named device is nonexistent.
    - 3B05 - Logging function for this device is not active.
    - 3B08 - Illegal function code.
    - 3B0A - Device-name refers to a communications device. Macro call cannot be executed.

# ERROR LOGGING INFORMATION, EXCHANGE

## ERROR LOGGING INFORMATION, EXCHANGE

Macro Call Name: \$ELEX

Function Code: 02/07

Equivalent Command: None

Verifies, then saves the values in the user's error logging table; transfers current logging values from system's error logging table to user's error logging table; moves the saved user-supplied error logging values into the system's logging table.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of device-name

Any address form valid for a data register. Provides the address of the device-name (previously coded in \$ELST macro call for this device) for the device whose error logging values are to be exchanged.

address of user's logging table

Any address form valid for a data register; provides the address of the previously generated 27-word user's error logging table. Table 5-1 below defines the user's error logging table, which the user must build and initialize before issuing any error logging macro call.

Table 5-1. User-Generated Table for Error Logging Macro Calls

Word(s)	Value (Signed Binary)	Function
User-Specified in \$ELST and \$ELEX Macro Calls		
1 and 2	Two-word integer $\geq 0$ ; normally initialized to 0.	Counter for number of I/O orders.
3	One-word integer $\geq 0$ ; normally initialized to 0.	Counter for number of device read/write errors.
4	Must be 0.	None.
5	One-word integer from 0 through 1000, represented as a fraction in thousandths; i.e., DC 500 means .500.	Error threshold ratio. A ratio of DC 10 (i.e., 1%) is suggested for magnetic tape.
6	One-word integer $\geq 0$ .	Minimum number of I/O orders processed before error threshold ratio is checked.
Values Returned by \$ELGT, \$ELEX, \$LEND Macro Calls		
7	Reserved for system use.	N/A
8	History counter.	Total number of errors up to time information returned in this macro call.
9	History counter.	Error threshold ratio when information is returned in this macro call.
10-12	History value.	Internal date/time of macro call return.
13-18		Device name.
19	Left byte. Right byte.	LRN for this device. Device type.
20-22		Internal date/time that last error occurred.
23		I/O status word (I ST) when last error occurred.
24-27	Reserved for future use.	

## FUNCTION DESCRIPTION:

This call causes the system to deliver to the user information about (1) the current status of the system's logging table up to the time of the macro call, and (2) about the last error that occurred, as indicated by words 7 through 27 in Table 5-1. The system (1) checks the values of the user's error logging table for errors, and if they are correct, saves those values; (2) executes a \$ELGT macro call to move current values from the system's logging table to the user's logging table; and (3) moves and stores in the system's logging table the new logging values verified and saved from the user's logging table, thus replacing the previous values in the system's logging table. History counters in the system's logging table are reset to 0.

- NOTES:
1. When argument 1 is specified, the location of the device name is placed in \$B2. If the argument is omitted, the system assumes that \$B2 contains a pointer to the device name.
  2. When argument 2 is specified, the address of the user's logging table is placed in \$B4. When the argument is omitted, the system assumes that \$B4 contains a pointer to that table.
  3. The device name must have been specified (or defaulted) in a previously executed \$ELST macro call for that device.
  4. On return, \$R1 contains one of the following status codes:
    - 0000 - Error logging information successfully exchanged.
    - 3B01 - Invalid argument (1 or 2).
    - 3B02 - Named device is nonexistent.
    - 3B03 - Illegal value specified for minimum number of I/O orders.
    - 3B05 - Logging function for this device is not active.
    - 3B06 - Illegal value specified for threshold.
    - 3B07 - Illegal initial value for I/O order counter or device error counter.
    - 3B08 - Illegal function code.

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

# ERROR LOGGING INFORMATION, GET

## ERROR LOGGING INFORMATION, GET

Macro Call Name: \$ELGT

Function Code: 02/08

Equivalent Command: None

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

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of device-name

Any address form valid for a data register. Provides the address of the device-name (previously coded in a \$ELST macro call for this device) for the device whose error logging error information is to be transferred.

address of user's logging table

Any address form valid for a data register. Provides the address of the previously generated 27-word user's error logging table (see Table 5-1 in the discussion of the error logging information exchange (\$ELEX) macro call.)

### FUNCTION DESCRIPTION:

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



- NOTES:
1. When argument 1 is specified, the location of the device-name is placed in \$B2. If the argument is omitted, the system assumes that \$B2 contains a pointer to the device-name.
  2. When argument 2 is specified, the address of the user's logging table is placed in \$B4. When the argument is omitted, the system assumes that \$B4 contains a pointer to that table.
  3. The device-name must have been specified (or defaulted) in a previously executed \$ELST macro call for that device.
  4. On return, \$R1 contains one of the following status codes:
    - 0000 - Error logging values successfully transferred.
    - 3B01 - Invalid argument (1 or 2).
    - 3B02 - Named device is nonexistent.
    - 3B05 - Logging function for this device is not active.
    - 3B08 - Illegal function code.
    - 3B0A - Device-name refers to a communications device; macro call cannot be executed.

# ERROR LOGGING, START

## ERROR LOGGING, START

Macro Call Name: \$ELST

Function Code: 02/05

Equivalent Command: None

Activate error logging for the named device.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

location of device-name

Any address form valid for a data register. Provides the address of the device-name (designated at system building) for the peripheral (noncommunications) device to be monitored. Device name can have up to 12 ASCII characters.

address of user's logging table

Any address form valid for a data register. Provides the address of the user's error logging table, which must have been previously generated. (See Table 5-1 in the discussion of the error logging information exchange (\$ELEX) macro call.) This macro call requires only the first six words of the user's error logging structure.

### FUNCTION DESCRIPTION:

This macro call starts error logging for the named device, and maintains error logging information in memory. The call (1) allocates a block of system memory for the system's logging table; (2) checks parameters in the first six words of the user's logging table and stores the values in the system's logging table in memory; and (3) stores in the

device's RCT a pointer to the system's logging memory area, which activates the logging function. Before this macro call is issued, the user must build and initialize at least a six-word error logging table, as defined in Table 5-1. Whenever an I/O order is issued, the system increments the I/O counter (words 1 and 2 in Table 5-1). When there is a device error, the system increments the device error counter (word 3). When the specified number of I/O orders (word 6) is processed, the system checks the error threshold ratio (word 5) and if the value is exceeded, sends a message to the operator and resets the system's error logging table for this device.

The logging table is reset under any of the following conditions:

1. Designated error threshold ratio exceeded.
2. Either the I/O order counter (words 1 and 2) or device error counter (word 3) overflowed.
3. \$ELEX macro call is executed.

When 1 or 2 occurs, the current value of the I/O order and device error counters are added to the history values in the system's error logging table. (These history values may be later delivered to corresponding history areas in the user's logging table (see Table 5-1)). If there is overflow in the addition, these counters are reset to 0, but the error threshold (word 4) and I/O order minimum (word 5) values are retained. When 3 occurs (\$ELEX executed), the items in the system logging table are reinitialized from the new values supplied in the user's logging table.

- NOTES:
1. When argument 1 is specified, the location of the device-name is placed in \$B2. If the argument is omitted, the system assumes that \$B2 contains a pointer to the device-name.
  2. When argument 2 is specified, the address of the user's logging table is placed in \$B4. When the argument is omitted, the system assumes that \$B4 contains a pointer to that table.
  3. The device-name must be that of a noncommunications peripheral device, i.e., cannot be connected to an MLCP or DLCP.
  4. On return, \$R1 contains one of the following status codes:  
  
0000 - Error logging activated successfully.

- 3B01 - Invalid argument (1 or 2).
  - 3B02 - Named device is nonexistent.
  - 3B03 - Illegal value specified for minimum number of I/O orders.
  - 3B06 - Illegal value specified for threshold.
  - 3B07 - Illegal initial value for I/O order counter or device error counter.
  - 3B08 - Illegal function code.
  - 3B09 - Insufficient system memory for logging table.
  - 3B0A - Device-name refers to communications device; logging cannot be activated.
5. The user can move the latest error logging information values from the system logging table to the user's logging table with a \$SELGT, \$SELEX, or \$LEND macro call.

# 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.

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 begins in the left byte of the address specified in argument 1; if argument 3 is R, the record area begins in the right byte of this address. Any other value for argument 3 is taken to be the location of the byte offset. If argument 3 is omitted, the record area is assumed to begin at the left byte of the address specified in argument 1.

#### FUNCTION DESCRIPTION:

This call allows a task to write the next record (an error message record) to the current error-out file. The error-out file is the same as the initial user-out file defined in the request group (\$RQGRP) macro call, and cannot be changed during execution of the request.

- NOTES:
1. The address of the record to be written, supplied by argument 1, is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the output record.
  2. The output record size, supplied by argument 2, is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the size of the record.
  3. If argument 3 is L, \$R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, \$R7 is set to one to designate that the record area begins in the right byte of the specified address. Any other value is assumed to be the location of the byte offset to be used, and is placed in \$R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address, and \$R7 is set to zero.
  4. On return, \$R1, \$R6, and \$B4 contain the following information:

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

0000 - No error

All data management write-next-record error codes may also be returned. See the System Messages manual.

\$R6 - Residual range (number of bytes not transferred from record area).

\$B4 - Address of record area containing output record.

Example:

In this example, the issuing task is to write an error message record on the error-out file. The record length is 12 bytes (including the slew byte). The output record is located at the record area address RECAD. The record area begins at the leftmost byte of the indicated address.

```
OUTRB    $EROUT    !RECAD,=12
          .
          .
          .
RECAD    TEXT      'AFIELD  ERROR'
```

# EXPAND PATHNAME

## 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

0205 - Illegal argument

0222 - Pathname cannot be expanded, no working directory

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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    <RELDPH
        RESV  2-$AF
        DC    <X_NAME
        RESV  2-$AF
        DC    0
```

The relative pathname is defined as follows:

```
RELPTH  DC    '<<ADF△'
```

The fully expanded pathname ^VOL6>SUB1>SUB2>ADF is developed as a result of the following macro call.

```
$XPATH  !XPND_1
```

# EXTERNAL DATE/TIME, CONVERT TO

## EXTERNAL DATE/TIME, CONVERT TO

Macro Call Name: \$EXTDT

Function Code: 05/04

Equivalent Command: Time (TIME)

Convert an internal format date/time value to an external format date/time value.

### FORMAT:

```
[label] $EXTDT [location of address of internal date/time],  
               [location of address receiving field],  
               [location of size of receiving field]
```

### ARGUMENT DESCRIPTION:

location of address of internal date/time

Any address form valid for an address register; provides the address of the 3-word field containing the internal date/time value to be converted. This value must be in the format returned by the get date/time macro call (\$GDTM).

location of address of receiving field

Any address form valid for an address register; provides the address of a field in the issuing task that is to receive the external format date/time value.

location of size of receiving field

Any address form valid for a data register; provides the size of the receiving field identified by argument 2. The field size must be less than or equal to 22 bytes. If this argument is omitted, the size is set to 20 bytes (the date/time value is resolved to a tenth of a second).

FUNCTION DESCRIPTION:

This call converts an internal date/time value (in the format supplied by the get date/time macro call) to an external date/time format. The date/time value appears in the receiving field as a character string having the format:

<u>Word</u>	<u>Contents</u>
0	yy (Two ASCII numeric characters)
1	yy (Two ASCII numeric characters)
2	/m (Two ASCII characters)
3	m/ (Two ASCII characters)
4	dd (Two ASCII numeric characters)
5	h (Two ASCII characters)
6	hm (Two ASCII numeric characters)
7	m: (Two ASCII characters)
8	ss (Two ASCII numeric characters)
9	.t (Two ASCII characters)
10	tt (Two ASCII numeric characters)

yyyy - Year	mm - Minute
mm - Month	ss - Seconds
dd - Day	ttt - Tenths, hundredths, thousandths of seconds
hh - Hour	

The size of the receiving field cannot be such that the field terminates with a punctuation character (/ , : , or .). Thus argument 3 cannot specify a size of 5, 8, 16, or 19 bytes.

- NOTES:
1. The internal date/time value whose address was supplied by argument 1 is loaded into \$R2, \$R6, and \$R7. If argument 1 is omitted, or is =\$R7, it is assumed that \$R2, \$R6, and \$R7 contain the value to be converted.
  2. The address of the receiving field supplied by argument 2 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the correct address.
  3. The size of the receiving field supplied by argument 3 is placed in \$R5. If this argument is given as =\$R5, \$R5 is assumed to contain the correct size. If this argument is omitted, \$R5 is set to 20 bytes (tenth of a second resolution).

4. On return, \$R1, \$R2, \$R6, \$R7 and \$B4 contain the following information:

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

0000 - No error

0402 - Invalid (negative) receiving field length

040A - Invalid receiving field address

0817 - Memory access violation

\$R2, \$R6, \$R7 - Internal date/time value supplied

\$B4 - Address of receiving field

Example:

See the example given for the get date/time macro call.

# EXTERNAL TIME, CONVERT TO

## EXTERNAL TIME, CONVERT TO

Macro Call Name: \$EXTIM

Function Code: 05/05

Equivalent Command: None

Convert an internal format date/time value to an external format time value.

### FORMAT:

```
[label] $EXTIM [location of address of internal date/time],  
               [location of address of receiving field],  
               [location of length of receiving field]
```

### ARGUMENT DESCRIPTION:

location of address of internal date/time

Any address form valid for an address register; provides the address of a 3-word field containing the internal date/time value to be converted. This value must be in the format returned by the get date/time macro call.

location of address of receiving field

Any address form valid for an address register; provides the address of a field in the issuing task that is to receive the external format time value.

location of length of receiving field

Any address form valid for a data register, provides the size of the receiving field identified by argument 2. The field size must be less than or equal to 11 bytes. If this argument is omitted, the size is set to 9 bytes (the time is resolved to a tenth of a second).

FUNCTION DESCRIPTION:

This call converts an internal date/time value (in the format supplied by the get date/time macro call) to an external time format. The time value appears in the receiving field as a character string having the format hhmm:ss.ttt (see below).

<u>Word</u>	<u>Contents</u>
0	hh (two ASCII numeric characters)
1	mm (two ASCII numeric characters)
2	:s (two ASCII characters)
3	s. (two ASCII characters)
4	tt (two ASCII numeric characters)
5	t (two ASCII characters)

hh - hours                    ss - Seconds  
mm - minutes                ttt - tenths, hundredths,  
   thousandths of seconds

The size of the receiving field cannot be such that the field terminates with a punctuation character (: or .). Thus, the third argument cannot be 5 or 8.

- NOTES:
1. The internal date/time value whose address is supplied by argument 1 is loaded into \$R2, \$R6, and \$R7. If argument 1 is omitted, or is = \$R7, it is assumed that \$R2, \$R6, and \$R7 contain the internal value to be converted.
  2. The address of the receiving field supplied by argument 2 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the correct address.
  3. The size of the receiving field supplied by argument 3 is placed in \$R5. If argument 3 is = \$R5, it is assumed that \$R5 contains the correct size. If this argument is omitted, \$R5 is set to 9 bytes (tenth of a second resolution).
  4. On return, \$R1, \$R2, \$R6, \$R7, and \$B4 contain the following information:  
  
\$R1 - Return status; one of the following:  
  
0000 - No error  
  
0402 - Invalid (negative) receiving field length

040A - Invalid receiving field address

0817 - Memory access violation

\$R2, \$R6, \$R7 - Internal date/time value  
supplied

\$B4 - Address of receiving field

Example:

In this example, the \$GDTM macro call is used to get the current date/time, in internal format, leaving it in registers \$R2, \$R6, and \$R7. The \$EXTIM macro call is then used to format this internal date/time value into a displayable format with a resolution to milliseconds. A message containing the external format date/time is then written on the user-out file.

```
*
* GET THE CURRENT DATE/TIME.
*
*           $GDTM
*
* FORMAT IT FOR DISPLAY.
*
*   $EXTIM   ,!P1_TIM,=11
*
* OUTPUT THE MESSAGE.
*
*   $USOUT   !P1_MSG,=P1_MLN
*           .
*           .
*           .
P1_MSG     TEXT   'APHASE 1 FINISHED ATΔ'
P1_TIM     TEXT   'HHMM:SS.TTT'
P1_MLN     EQU    2*($-P1_MSG)
```



# FILE INFORMATION BLOCK

## FILE INFORMATION BLOCK

Macro Call Name: \$FIB

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, does one of the following:

- o Builds a 16-word file information block (FIB) containing default values for the words.
- o Generates instructions to alter the partial contents of an existing FIB.
- o Calls and expands the \$TFIB macro call to provide labels for the FIB entries.

### FORMAT:

[label] \$FIB [arguments]

### ARGUMENT DESCRIPTION:

There are three types of arguments for this macro call:

- o Keyword only
- o Keyword with expression
- o Keyword with option

The keyword RESV generates a data structure. The \$FIB macro without the keyword RESV generates executable code to modify an existing data structure.

When the call is coded with only the keyword RESV, a 16-word FIB containing default values is built (with tags for the entries). The entries have the following values:

```

DC      0
DC      B'0110010010000000'
DC      0,0
DC      80
DC      80
DC      0
DC      Z'FFFF'
DC      0
DC      0,0
DC      Z'0104'
DC      0,0
DC      0,0

```

The default values generated for this FIB allow access to a file for reading and writing, and allow record access by both primary and relative keys. The default input and output record lengths are 80 characters; the default key format for input records is primary; key length is 4 bytes.

When the keyword RESV is used with other arguments, it preserves all entries in the generated FIB that are not specifically changed by the other arguments.

Arguments coded as keyword=expression apply to the words of the file information block. These arguments can be coded in any order. If a new FIB is to be built and an argument is omitted, the default value (described above) for that word is used. If an existing FIB is to be modified and an argument is omitted, the existing value for that word is used. The diagram below shows the keywords and possible expression values, but does not necessarily correspond to the FIB physical structure. For more detailed information, see Tables 3-1 and 3-2.

Keyword	Expression								
LFN=	A value from 0 through 255 specifying the logical file number by which the file is referenced; or -1; or A'ΔΔ' (two ASCII space characters).								
PVW=	<p>A value specifying the desired program view word (i.e., user visibility), as follows:</p> <table border="0" data-bbox="381 1606 1380 1801"> <thead> <tr> <th data-bbox="381 1606 487 1648"><u>Bits</u></th> <th data-bbox="487 1606 1380 1648"><u>Meaning</u></th> </tr> </thead> <tbody> <tr> <td data-bbox="381 1669 487 1711">0</td> <td data-bbox="487 1669 1380 1711">Access level:</td> </tr> <tr> <td data-bbox="381 1732 487 1774">0</td> <td data-bbox="487 1732 1380 1774">- Access via data management</td> </tr> <tr> <td data-bbox="381 1774 487 1812">1</td> <td data-bbox="487 1774 1380 1812">- Access via storage management</td> </tr> </tbody> </table>	<u>Bits</u>	<u>Meaning</u>	0	Access level:	0	- Access via data management	1	- Access via storage management
<u>Bits</u>	<u>Meaning</u>								
0	Access level:								
0	- Access via data management								
1	- Access via storage management								

Keyword	Expression	
PWW= (cont)	<u>Bits</u> 1-4           5-9           10           11           12           13	<u>Meaning</u>  Process rules: 1000 - \$RDREC or \$RDBLK macro calls permitted 0100 - \$WRREC or \$WRBLK macro calls permitted 0010 - \$RWREC macro call permitted 0001 - \$DLREC macro call permitted nnnn - Any combination of the binary settings to allow the desired macro calls  Key type: 10000 - Primary keys allowed 00010 - Relative keys allowed 00001 - Simple keys allowed n00nn - Any combination of binary settings to allow the desired keys  NOTE: Bits 10-14 are data management specific and can be set before the file is opened.  Record class: 0 - Fixed- and variable-length records allowed 1 - Only fixed-length records allowed  Record visibility: 0 - Deleted records are not visible during a read next record operation 1 - Deleted records are visible  Key storage area alignment: 0 - Area begins at even-byte boundary 1 - Area begins at odd-byte boundary  Record storage area/buffer alignment: 0 - Begins at even-byte boundary 1 - Begins at odd-byte boundary

Keyword	Expression	
PVW= (cont)	<u>Bits</u>	<u>Meaning</u>
	14	Transcription mode: 0 - ASCII mode 1 - Binary transcription mode
	15	Synchronous/asynchronous indicator: 0 - \$RDBLK or \$WRBLK macro calls to be executed synchronously 1 - These calls to be executed asynchronously
URP=	Address of start of user record area (data management) or start of buffer area (storage management)	
IRL=	Maximum size of input record for data management operations	
ORL=	Actual size of output record	
IKP=	Address of user key area in which key value is stored	
IKF=	Type of key: 00 - None specified 01 - Primary or relative (see bits 5-9 of PVW) 02 - Simple	
IKL=	Value specifying the length of the user key area (IKP). A maximum of 256 ASCII characters is allowed for primary keys. (Simple and relative keys are always assumed to be four bytes.) When used with the RESV keyword, the value specified for IKL must be a 1-byte hexadecimal number (e.g., 0A, 01, etc.).	
BFS=	Value specifying size of data transfer (block size) for storage management operations.	
BNL=	Left half of block number; an integer relative to the beginning of the file (storage management)	
BNR=	Right half of block number; an integer relative to the beginning of the file (storage management)	
ADR=	Address of FIB to be modified. Not used when building new FIBs.	

Arguments coded as keyword=option apply only to the program view word of the FIB. Options can be given in any order; more than one option value can be specified per argument. Bits in the program view word that are not explicitly given a value through an option selection retain their previous setting. The diagram below shows the keywords and possible values for the expressions. See Table 3-2 for more detailed information.

Keyword	Option	Meaning
SFN=	RD	File can be read by \$RDREC or \$RDBLK macro calls
	WR	File can be written by \$WRREC or \$WRBLK macro calls
	RW	File can be rewritten by \$RWREC macro call
	DL	File can be deleted by \$DLREC macro call
RFN=	RD	File cannot be read by \$RDREC or \$RDBLK macro calls
	WR	File cannot be written by \$WRREC or \$WRBLK macro calls
	RW	File cannot be rewritten by \$RWREC macro call
	DL	File cannot be deleted by \$DLREC macro call
SKA=	P	Primary keys allowed
	R	Relative keys allowed
	S	Simple keys allowed
RKA=	P	Primary keys not allowed
	R	Relative keys not allowed
	S	Simple keys not allowed
SRA=	FL	Only fixed-length records allowed
	DV	Deleted records are visible
RRA=	FL	Fixed- and variable-length records allowed
	DV	Deleted records are not visible

Keyword	Option	Meaning
SSM=	BT	Binary transcription mode used for data transfer
	AS	\$RDBLK or \$WRBLK macro calls executed asynchronously
	OP	File accessed via storage management
RSM=	BT	ASCII mode used for data transfer
	AS	\$RDBLK or \$WRBLK macro calls executed synchronously
	OP	File accessed via data management
ODD=	KY	Key storage area begins at odd-byte boundary
	RC	Record storage area begins at odd-byte boundary
EVN=	KY	Key storage area begins at even-byte boundary
	RC	Record storage area begins at even-byte boundary

If no arguments are coded, the \$TFIB macro call is expanded.

#### FUNCTION DESCRIPTION:

This call (1) generates a 16-word FIB, or (2) alters the contents of an existing FIB, and (3) calls and expands the \$TFIB offsets macro call.

A FIB must exist for a file if that file is to be operated upon by one of the following macro calls:

- o Open file (\$OPFIL)
- o Close file (\$CLFIL)
- o Test file (\$TIFIL, \$TOFIL)
- o Read record (\$RDREC)
- o Write record (\$WRREC)
- o Rewrite record (\$RWREC)
- o Delete record (\$DLREC)
- o Read block (\$RDBLK)
- o Write block (\$WRBLK)
- o Wait block (\$WTBLK)

If an existing FIB is to be modified and the argument ADR= is not entered, \$B4 is assumed to point to the FIB to be modified.

Registers \$R7 and \$B5 are altered when an existing FIB is modified.

Macro global value GX is used to control expansion of the \$TFIB macro call.

When you use the \$FIB macro call to alter an existing FIB, arguments that use an address follow the convention in which addresses preceded by the ! character cause an LAB instruction to be generated, and addresses not preceded by the ! character cause an LDB instruction to be generated. When you supply values for arguments coded as keyword=expression (IRL=, ORL=, etc.), the address of the value is distinguished by a preceding character. No special character is needed to indicate that the string following the = character is a value. The second example given below uses both values and addresses (IFL=128 and ORL= LENGTH).

The expressions specified with each argument must be in a form suitable for the DC statement. IKF and IKL must specify a 1-byte hexadecimal number.

Example 1:

In this example, the \$TFIB macro call is expanded.

\$FIB

Example 2:

In this example, an existing FIB is modified. This example assumes that the \$B4 register was previously loaded with the address of the FIB to be modified.

```
$FIB    URP=!REC1,RFN=WR,SRA=FL,ODD=RC,IRL=128,LFN=<GETPRM
```

Execution of the macro call generates the following set of instructions:

```
LAB     $B5,REC1
STB     $B5,$B4.F_URP
LBF     $B4,F_PROV,B'0010000000000000'
LBT     $B4,F_PROV,B'0000000000100100'
LDR     $R7,=128
STR     $R7,$B4.F_IRL
LDR     $R7,GETPRM
STR     $R7,$B4.F_LFN
```

Example 3:

This example generates a FIB so that the file can be accessed for reading, writing, rewriting, and deleting records by either primary or relative keys. The rewrite and delete bits (bits 3 and 4) of the program view word are altered from the original values (provided by the RESV parameter) by means of the SFN=RWDL argument.

```
EXTFIB    $FIB    LFN=3, IKF=02, RESV, SFN=RWDL, SKA=PR, IKP=<KEY
```

This macro call generates the following FIB:

```
EXTFIB    DC      3
          DC      B'0111110010000000'
          DC      0,0
          DC      80
          DC      80
          DC      0
          DC      Z'FFFF'
          DC      0
          DC      <KEY
          RESV    2-$AF
          DC      Z'0204'
          DC      0,0
          DC      0,0
```

Example 4:

In this example, a 16-word FIB is generated with default values for all words.

```
EXTFIB    $FIB    RESV
```

The following FIB is generated:

```
EXTFIB    DC      0
          DC      B'0110010010000000'
          DC      0,0
          DC      80
          DC      80
          DC      0
          DC      Z'FFFF'
          DC      0
          DC      0,0
          DC      Z'0104'
          DC      0,0
          DC      0,0
```



# FILE INFORMATION BLOCK OFFSETS

## FILE INFORMATION BLOCK OFFSETS

Macro Call Name: \$TFIB

Associated Macro Calls:

\$OPFIL, \$CLFIL, \$TIFIL, \$TOFIL, \*  
 \$RDREC, \$WRREC, \$RWREC, \$DLREC, \$RDBLK,  
 \$WRBLK, \$WTBLK

Generated Offsets Tags:

<u>Tag</u>	<u>Corresponding Offsets (in Words)</u>	<u>Entry Name</u>	
F_LFN	0	Logical file number (LFN)	
F_PROV	+1	Program view	
F_URP	+2	User record pointer <sup>1</sup>	
F_IRL	+4	Input record length <sup>1</sup>	
F_ORL	+5	Output record length <sup>1</sup>	
F_ORT	+8	Reserved <sup>1</sup>	*
F_IKP	+9	Input key pointer <sup>1</sup>	
F_IKF	+11	Input key format (first byte) <sup>1</sup>	
F_IKL	+11	Input key length (second byte) <sup>1</sup>	
F_ORA	+12	Output record address <sup>1</sup>	
F_ORA1	F_ORA(+12)	(left half of F_ORA) <sup>1</sup>	
F_ORA2	F_ORA+1(+13)	(right half of F_ORA) <sup>1</sup>	
F_UBP	+2	User buffer pointer <sup>2</sup>	
F_BFSZ	+4	Buffer size <sup>2</sup>	
F_BKSZ	+5	Block size <sup>2</sup>	
F_BKNO	+6	Block number <sup>2</sup>	
F_BKNO1	F_BKNO(+6)	(left half of F_BKNO)	
F_BKNO2	F_BKNO+1(+7)	(right half of F_BKNO)	
F_SZ	+16	Size of structure (in words); not a field in the block	

<sup>1</sup>Specific to \$RDREC, \$WRREC, \$RWREC, and \$DLREC macro calls.

<sup>2</sup>Specific to \$RDBLK, and \$WRBLK macro calls.

In addition to the offsets tags listed above, the following program view (F\_PROV tag, above) masks are defined:

<u>Tag</u>	<u>Mask</u>	<u>Meaning</u>
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_CKA	Z'0200'	Reserved
MF_2KA	Z'0100'	Reserved
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

# GET DATE/TIME

## GET DATE/TIME

Macro Call Name: \$GDTM

Function Code: 05/06

Equivalent Command: None

Supply the requesting task with the current internal date/time value maintained by the system.

### FORMAT:

[label] \$GDTM [location of address of receiving field]

### ARGUMENT DESCRIPTION:

location of address of receiving field

Any address form valid for an address register; provides the address of a 3-word field in the issuing task that is to receive the current internal date/time value.

### FUNCTION DESCRIPTION:

This macro call returns to the issuing task the current 3-word internal date/time value. The leftmost word contains the most significant 16 bits; the rightmost word contains the least significant 16 bits. The value supplied is a binary count of the milliseconds since 1 January 1901 at 00:00:00.000 hours.

NOTES: 1. The internal date/time value is returned in \$R2, \$R6, and \$R7 and stored in the receiving field specified by argument 1. If argument 1 is omitted, or is =\$R7, the value is returned only in \$R2, \$R6, and \$R7.

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

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

0000 - No error

040A - Invalid address for receiving field

\$R2, \$R6, and \$R7 - Current 3-word internal date/time value. \$R2 contains the most significant 16 bits and \$R7 the least significant 16 bits.

Example:

In this example the \$GDTM macro call is used to get the starting date/time, in internal format, of a process and store it in the field ST\_TIM. The convert to external date/time macro call (\$EXTDT) is then used to format this internal clock value, contained in registers \$R2, \$R6, and \$R7, into a displayable date/time format with resolution to a tenth of a second. A startup message containing the external format date/time is then written on the user-out file. Later, the get date/time macro call is used again to get the finishing date/time of the process without storing it in memory. The low order two words of the starting date/time are then subtracted from the corresponding words of the finishing date/time, leaving the elapsed time (in milliseconds) in \$R6 and \$R7. The subtraction is performed assuming a central processor that does not have the subtract integer double instruction. The high order word of the starting and finishing date/time values is ignored with the assumption that the elapsed time is less than  $2^{31}$  milliseconds (about 24.855 days).

```
*
*   GET THE STARTING TIME.
*
*       $GDTM    !ST_TIM
*
*   FORMAT IT FOR DISPLAY.
*
*       $EXTDT   ,!GO_TIM,20
*
*   OUTPUT THE START UP MESSAGE.
*
*       $USOUT   !GO_MSG,=GO_MLN
```

```

*
* BEGIN PROCESSING.
*
      .
      .
      .
*
* GET THE FINISHING TIME.
*
      $GDTM
*
* CALCULATE THE ELAPSED TIME.
*
      SUB      $R7,ST_TIM+2
      BCT      >$+2
      ADV      $R6,-1
      SUB      $R6,ST_TIM+1
      .
      .
      .
ST_TIM      RESV      3,0
GO_MSG      TEXT      'APROGX STARTED ATΔ'
GO_TIM      TEXT      'YYYY/MM/DD HHMM:SS.T'
GO_MLN      EQU       2*($-GO_MSG)

```

# GET FILE

## GET FILE

Macro Call Name: \$GTFIL

Function Code: 10/20

Equivalent Command: Get File (GET)

Locate and reserve a file (tape or disk file, disk directory, card reader, printer, or terminal device) for processing with the specified access rights. You identify the file by supplying either a logical file number (LFN) or a pathname. If you supply both an LFN and a pathname, the file is reserved and is assigned to the LFN. Subsequent macro calls (open file, read record, etc.) can then be directed to the file through this LFN. If the file is tape-resident, the get file macro call supplies the necessary tape definition arguments. This function is normally done outside program execution, to assign the LFN to a file that is not known until execution time.

### FORMAT:

[label] \$GTFIL [argument structure address]

### ARGUMENT 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. The size of each entry, whose descriptions follow this list, is as follows:

<u>Argument Structure Entry</u>	<u>Size (in bytes)</u>
logical file number	2
pointer pathname	4
lock/concurrency control	1
mount option	1
tape block size	2
tape logical record size	2
number of buffers	1
tape file sequence number	1
tape label format	1
tape data type	1
tape data format	1
tape parity/BSN indicator	1
tape retention period	4

#### 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, ASCII blanks (2020) if an LFN is not specified, or -1 (FFFF) if the system is to assign an LFN from the pool of available LFNs.

#### pointer pathname

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 when expanded, identifies the file to be reserved. Binary zeros in this entry indicate that a pathname is not specified.

#### lock/concurrency control

A 1-byte code, applicable only to disk files, that specifies the record lock and concurrency control to be established for the file.

If record locking is requested, the records in the file will be locked in shared-read/exclusive-write mode when the file is accessed. Once a file is reserved with locking, it cannot be reserved by another user (task group) unless that user also specifies record locking.

The type of file concurrency chosen indicates how you intend to access the file and in what way you are willing to share access to the file with other users (task groups). There are six types of concurrency control, as follows:

Type 5 - You will write or read; others can write or read (read/write sharing)

Type 4 - You will write or read; others can read but not write (read share, exclusive write)

Type 3 - You will write or read; no others can write or read (exclusive)

Type 2 - You will read; others can read and write

Type 1 - You will read; others can read but not write (read sharing)

Type 0 - If the file is already reserved, the last concurrency specified is used. If the file is not already reserved, type 3 concurrency control is used.

The value of the lock/concurrency control byte is determined as follows:

<u>Bit(s)</u>	<u>Meaning</u>
0	Lock specification:
	0 - Do not lock records
	1 - Lock records
1-4	Must be zero
5-7	Concurrency control specification
	000 - Type 0
	001 - Type 1
	010 - Type 2
	011 - Type 3
	100 - Type 4
	101 - Type 5

\*

The following diagram summarizes the lock/concurrency control specifications for each possible value of the lock/concurrency control byte.



Byte Value (Hexadecimal)	Record Locking?	Type of Concurrency Control
00	NO	0
01	NO	1
02	NO	2
03	NO	3
04	NO	4
05	NO	5
80	YES	0
81	YES	1
82	YES	2
83	YES	3
84	YES	4
85	YES	5

#### mount option

A 1-byte code, applicable to disk files and directories, that specifies whether a mount message is to be issued if the volume is not mounted. This byte can have the following values:

- 0 - Return a 020C error code in \$R1 if the volume containing the file referenced by the pathname or LFN is not mounted.
- n - (Where n is any value other than 0.) Issue a mount request if the volume containing the file referenced by the pathname or LFN is not mounted.

#### tape block size

A 2-byte binary value, applicable only to tape files, that specifies the size of the block in bytes.

For files with fixed length records, the block size is a multiple of the record size plus the 6-character block serial number (if specified).

For files with variable length records, the block size can be any value, but should be at least as large as the maximum record size plus the 4-character logical record header and the 6-character block sequence number (if specified).

The block sequence number is specified by a value of 2 for the tape parity/BSN indicator (see below).

The block size entry is ignored if the file is not tape resident.

The block size entry is ignored if block size was explicitly specified by a previous GET command.

If the file is not currently reserved and block size is not specified (i.e., the field contains all zeros), a value is computed based on the values for logical record size, tape data format, and tape BSN indicator.

If the file is already reserved and block size is not specified, the previously specified or computed value is not changed.

#### tape logical record size

A 2-byte binary value, applicable only to tape files, that specifies the logical record size in bytes.

The logical record size is the size of the longest record in the block, excluding the logical record header (if any).

If this is not a tape file, the tape logical record size entry is ignored.

The logical record size entry is ignored if logical record size was explicitly specified by a previous GET command.

If the file is not currently reserved and logical record size is not specified (i.e., the field contains all zeros), a value is computed based on the values for block size, tape data format, and tape BSN indicator.

If the file is already reserved and logical record size is not specified, the previously specified or computed value is not changed.

\*

#### number of buffers

A 1-byte binary value specifying the number of buffers to be allocated for I/O when the file is opened for access. The possible values are as follows:

0 - None specified; system allocates buffers as follows:

- o Two buffers are allocated for indexed files accessed via data management macro calls
- o One buffer is allocated for other disk files or tape files accessed via data management macro calls
- o No buffers are required for files accessed via storage management macro calls

n - Number of buffers to be used to access the file when other than the above default values are used. When accessing a file, data management first checks all buffers allocated for the file to determine whether the relative block or control interval is already in memory. Thus, increasing the number of buffers for a file being accessed randomly may significantly reduce I/O time.

Buffer space is allocated at open-file time and returned at close file time when the file is accessed via data management macro calls. Buffer space is not required if the file is accessed via storage management macro calls.

This entry does not apply to device files; buffers are allocated according to information specified at system building (buffered or unbuffered devices).

#### tape file sequence number

A 1-byte binary code, applicable only to tape files, that indicates the position of the file on an ANSI tape volume; can have the following values:

00 - The desired file is the next file on the volume

FF - Search for the file in a forward direction only

nn - Relative sequence number of the file on the volume

If a pathname is specified, it is used with the tape file sequence number to perform a file search when an open file macro call is issued. (The maximum file name length is 12 characters.)

See the description of the open file macro call (\$OPFIL) for a discussion of tape search rules.

If FF is specified, the search is performed from the current position on the volume to EOD.

If the file is not tape-resident, this entry is ignored.

#### tape label format

A 1-byte code, applicable only to tape files, that indicates the tape label format.

0 - No label format specified (can be labeled or unlabeled)

1 - Tape has standard labels

2 - Tape is not labeled

If the file is not tape-resident, this entry is ignored.

#### tape data type

A 1-byte code, applicable only to tape files, that specifies the data type.

0 - No data type specified

1 - Honeywell

2 - ANSI Level 3

If the file is not tape-resident, this entry is ignored.

tape data format

A 1-byte code, applicable only to tape files, that indicates the data format.

- 0 - No format specified
- 1 - Fixed length records
- 2 - Variable length records
- 3 - Undefined records

If the file is not tape-resident, this entry is ignored.

tape parity/BSN indicator

A 1-byte code, applicable only to tape files, that indicates whether the tape has odd or even parity, and whether each block on the tape has a six-character block sequence number (BSN) in the first six characters of the block.

<u>Bit(s)</u>	<u>Meaning</u>
0, 1	Must be 0.
2, 3	0 = Parity not specified. 1 = Odd parity. 2 = Even parity.
	These bits are meaningful only for 7-track tapes to be opened for storage management (block level) access.
4, 5	Must be 0.
6, 7	0 = No BSN specified. 1 = BSN not supplied. 2 = BSN supplied.

If 2 is specified, a block sequence number is assumed to be present on input; on output a block sequence number will be inserted.

\*

If the file is not tape-resident, this entry is ignored.

## tape retention period

A 4-byte value, applicable only to tape files, that specifies the tape retention period in days. Zeros in this field indicate that the retention period is not specified.

If the file is not tape-resident, this entry is ignored.

### FUNCTION DESCRIPTION:

\* This macro call reserves the file with proper access rights for use by the data management and storage macro calls. It can also be used to alter concurrency or tape definition arguments established by a previous get file macro call, provided the file is not already open (see the open file macro call) in the task group in which you are executing.

Once you have reserved a file, it cannot be released (see the release file macro call) by a task executing in another task group until you are finished with it (i.e., until you issue a remove file macro call).

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

- o If specified only by an LFN, the LFN must have been previously associated with a pathname (see the associate file macro call), or it must have been previously assigned to the file through the get file or create file function.
- o If only a pathname is specified, the file is reserved without a unique LFN. The only requests that can use the file are those that can reference the file by a pathname only, i.e., \$GTFIL, \$GIFIL, \$RLFIL, \$RMFIL.
- o If a pathname is specified and the LFN field contains a value of -1 (FFFF), the system assigns a unique LFN from the task's LFN pool. In this case it is your responsibility to return the LFN to the pool (by a remove file macro call) when the LFN is no longer needed. In assigning a unique LFN from the pool, the system selects the highest LFN available for assignment and sets it in the LFN entry in the argument structure, overlaying the previous contents (FFFF). You must move this value to other structures (argument structures or FIBs) as required.

- o If both an LFN and a pathname are specified, the file is reserved and assigned to the LFN. This LFN-to-file assignment remains in effect until the file is removed from the task group or another get file macro that specifies the same LFN is issued.

The get file macro call allows you to append ASCII characters to a previously associated pathname or a partial pathname (see the associate file macro call). You do this by prefixing the string of characters to be appended (i.e., pointed to by the pathname pointer entry) with a colon (:). The system replaces the colon with the previously associated pathname, as follows:

<u>Previously Associated Pathname</u>	<u>Characters to be Appended</u>	<u>Result</u>
none	:	Working directory
none	:ABC	ABC
^VOL1>UDD	:>FILE01	^VOL1>UDD>FILE01
^VOL2>	:FILE02	^VOL2>FILE02

As stated above, the get file macro call can be used to alter concurrency control. In doing so, note the following:

- o If you specify type 0 concurrency control the first time the file is reserved in a task group, the system reserves the file for exclusive use (type 3 concurrency).
- o If you specify type 0 concurrency control and the file was previously reserved in this task group, the previous concurrency control does not change. This could occur if you wanted to change the tape file definition argument or address the file through a different LFN.
- o A get file macro call does not alter the concurrency control established through a previously issued GET command. Only by issuing another GET command can you alter the concurrency established through a previous GET command.
- o If device level access is desired (i.e., the pathname is in the form >SPD>dev\_name[>valid]), the following rules apply:
  - Type 3 exclusive concurrency control is set regardless of the value specified in concurrency control entry if the pathname is specified as:

>SPD>dev\_name

No volume label validation is performed. Note that tapes are always reserved with type 3 concurrency.

- For disk volumes type 2 concurrency control is set regardless of the value specified in the concurrency control entry if the pathname is specified as:

>SPD>dev\_name>valid

The volume label is read and validated; if a mismatch occurs, the action specified in the mount option argument occurs.

- To change disk device-level concurrency control, you must first issue a remove file macro call and then issue a new get file macro call.
- o The following rules apply to directories reserved through a get file macro call:
  - If the directory is reserved exclusively (type 3 control), all subdirectories and files inferior to the directory are also held exclusively. For example, a get file macro call having a pathname of ^valid (i.e., only the volume directory supplied) and a concurrency of 3, would reserve the entire volume for exclusive use through normal file, data, and storage management facilities. This is not the same as device level access (>SPD>dev name) since it permits normal access by the user at the file level.
  - If the directory is not reserved exclusively, read/write share concurrency control (type 5) is set regardless of the specified value.
  - Directory-level concurrency cannot be changed by issuing a new get file macro call. To change directory-level concurrency, you must first issue a remove file macro call and then a get file macro call.

The record lock facility is a mechanism that provides multi-user interference protection for shared file access. When a record is accessed by a task group, it is locked (by locking the control interval(s) in which the record is contained) on a first-come first-served basis. If another user is sharing the file, he will be denied access to the record (and any other record contained in the same control interval) until the previous user unlocks the record (through the \$CLPNT macro call). You should consider the following points when using record locking:



- o An LFN within a task group uniquely identifies a user for record locking purposes and thus provides interference protection between task groups. Since tasks within a task group may agree to access a file through different LFNs, interference protection is provided when the cooperating tasks agree to respect the LFN assignments.
- o Lock requests are valid only for disk resident files (a request to lock any other file is ignored). Directories and entire disk volumes cannot be reserved with lock. The primary index of an indexed file is never locked (since once created, it is never updated).
- o Files reserved with lock cannot be modified (written) via storage management access.
- o Records are locked in "shared read/exclusive write" mode, which can be explained as follows:
  - For purposes of record locking, file system users may be classified as "readers" and "updaters." Readers are those who have opened the file, but without update permission, since they need only to read records. They are not concerned if other users are reading the same record, but do not want to read a record while it is being updated.
  - Updaters have opened a file, with update permission, and want to be the only users of a specific record. The record lock facility makes sure that a given record is accessed by only one updater or by n readers at one time.
  - Accordingly, readers set read locks, updaters set write locks. A given record may have any number of read locks, or it may have only one write lock.
- o Once specified, locking is automatic. Any access (read or write) will cause an appropriate lock. The number of locks that can exist at one time is limited only by the amount of memory dedicated to the lock pool (i.e., the area of memory where locked records are recorded). (This area is defined at system building; see the System Building manual.)
- o Any access that encounters a lock conflict is not performed. The caller is notified immediately; no wait is performed.

- o Once a file has been reserved as sharable with record locking, all users who want to share access to the file must also request record locking. Conversely, if a sharable file is initially reserved without record locking, no user who specifies record locking is allowed to access the file. In other words, bit 0 of the lock/concurrency control argument must be identical in the get file macro calls, with the following exceptions:
  - If bits 5 through 7 of the lock/concurrency control argument specify type 2 concurrency control (you read, others read and write), bit 0 need not match in the \$GTFIL macro calls. Thus, the user who specifies type 2 concurrency control can gain access to records that are currently locked by another user. However, this user can only read, and the integrity of the data is not guaranteed.
  - If the entire lock/concurrency control byte is all 0 bits, the lock and concurrency states set by a GET command or a previous \$GTFIL macro call in the same task group are inherited.
- o The \$CLPNT macro call is used to unlock records. If records are not unlocked, lock pool overflow or a deadlock record condition will probably result. (See the clean point macro call for details.)
- o You must provide for all actions to be taken when notified of lock pool overflow or record lock concurrency conflict. When a record deadlock condition occurs, you should restart the current phase by unlocking all records and recycling to the point where the interrupted sequence began. (In so doing some records may be updated, thereby making a simple recycling unsatisfactory.) From a practical standpoint, all records to be updated or deleted should be read first to ensure access; all inserts should be done first to make the unwinding of a transaction easier to manage.

If an operator terminal is not included in the system, or if messages to the operator terminal have been suppressed (through a \$CMSUP macro call), a \$GTFIL macro call issued to reserve a volume that is not mounted results in an 020C (volume not mounted) error return.

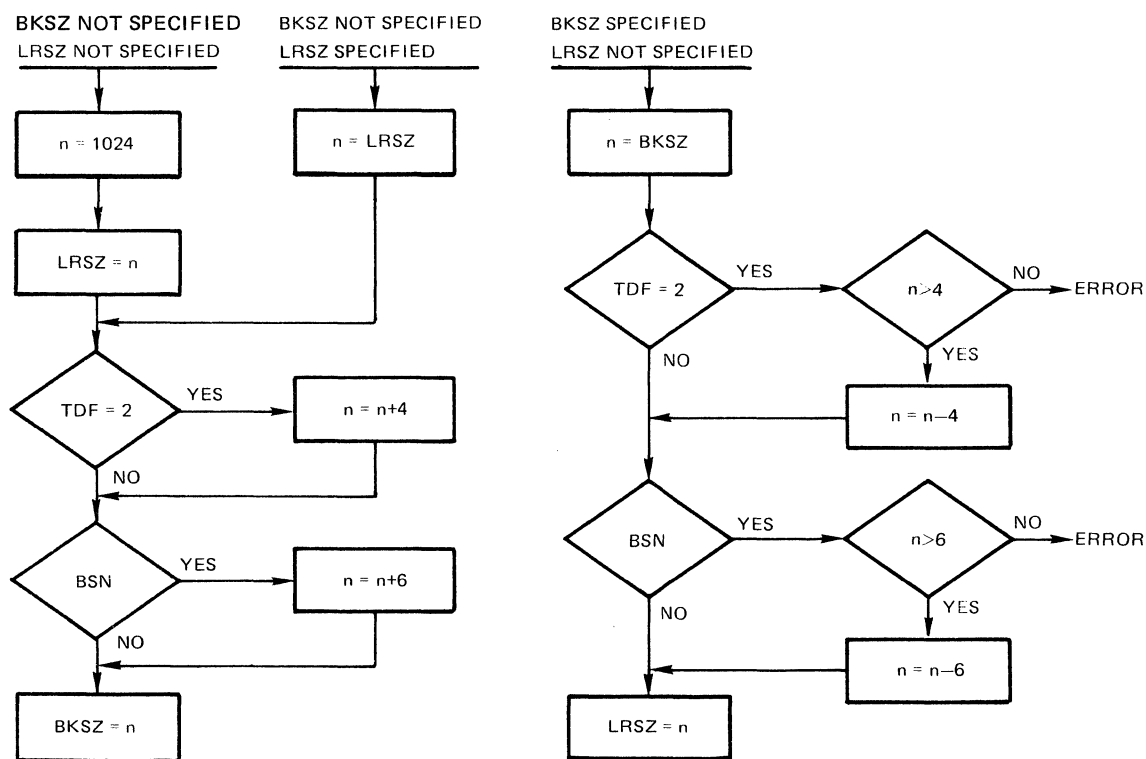
If a file is reserved through an LFN and a subsequent \$GTFIL macro call is issued specifying the same LFN, this LFN becomes associated with the new file. The previously reserved file will remain reserved for the task group until it is removed (through the remove file macro call).

Since the get file macro call performs so many functions, it should be used as infrequently as possible. A \$GTFIL followed by multiple \$OPFIL/\$CLFIL sequences is much more efficient than a \$GTFIL, \$OPFIL, \$CLFIL, \$RMFIL, \$GTFIL, etc.

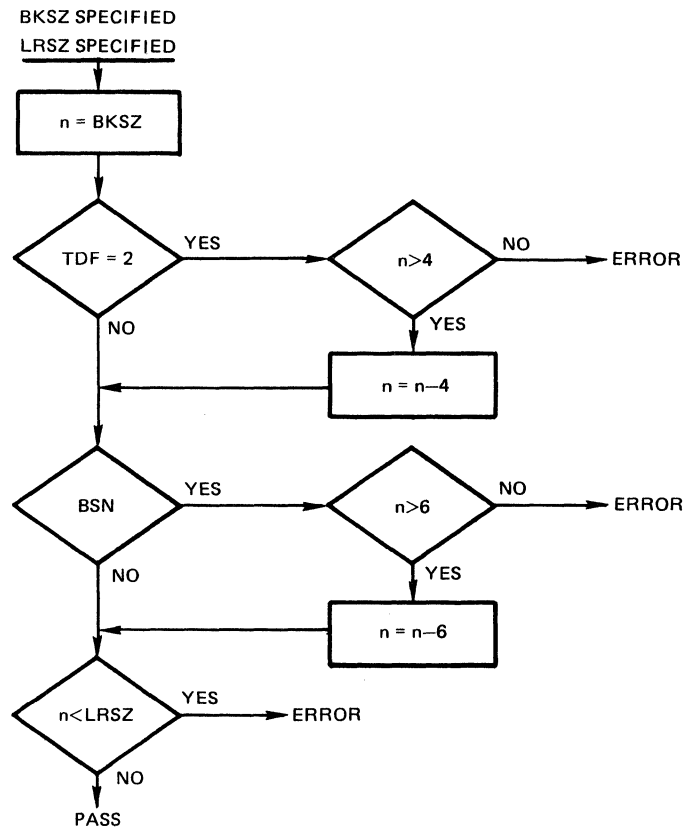
Offsets tags for the argument structure block can be defined by the \$GTPSB macro call.

Tape file arguments are meaningful in only when (1) a labeled tape file is being created (opened) in RENEW mode; and (2) an unlabeled tape file is being processed for input/output. For labeled tapes being opened for input (PRESERVE mode), the various tape parameters are taken from the file header labels.

For tape files, default block size (BKSZ) and logical record size (LRSZ) are computed as shown in the following diagram:



Block and record sizes are checked for validity as shown in the following diagram:



- 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
    - 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
- 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 - Record lock concurrency conflict
- 022A - Record lock area overflow or not defined
- 022C - Access control list violation
- 022E - Record lock concurrency conflict
- 0238 - Invalid file description

In addition to the above codes, any system service codes received through the file manager are passed on through \$R1.

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.AΔ'    (See create direc-
                                         tory macro example)
```

This pathname has been associated with the LFN as follows:

```
$ASFIL    !FILE_X
```

where the argument structure labeled FILE\_X has been defined as follows:

```
FILE_X    DC    Z'00A3'    LFN=163
           DC    <DIRPTH    PATHNAME    '^VOL03 SUBINDEX.AΔ'
           RESV   2-$AF
```

Assuming that the above definitions have been made, the following argument structure identifies the characters to be appended to the incomplete path (DIRPTH):

```
WTFIL2    DC    Z'00A3'    LFN=163
           DC    <IDX02    PATHNAME POINTER
           RESV   2-$AF
           DC    Z'0301'    EXCLUSIVE:    ISSUE MOUNT REQUEST
           RESV   2,0    UNSPECIFIED
           DC    Z'0200'    BUFFERS=2
           RESV   4,0    IGNORED
```

The pathname labeled IDX02 is defined as follows:

```
IDX02     DC    ':>FILE_CΔ'
```

The result of specifying the above structure (WTFIL2) in the following \$GTFIL macro call is to reserve the file identified by the pathname ^VOL03>SUBINDEX.A>FILE\_C with exclusive concurrency control:

```
SGTFIL    !WTFIL2
```

However, before you can open and access FILE\_C, it must exist in the file system hierarchy (i.e., it must have been created as defined in the create file macro call example).

# GET FILE PARAMETER STRUCTURE BLOCK OFFSETS

## GET FILE PARAMETER STRUCTURE BLOCK OFFSETS

Macro Call Name: \$GTPSB

Associated Macro Call: \$GTFIL

Generated Offsets Tags:

<u>Tag</u>	<u>Corresponding Offsets (in words)</u>	<u>Entry Name</u>
G_LFN	0	Logical file number (LFN)
G_PTHP	+1	Pointer to pathname
G_CONC	+3	Concurrency control (first byte)
G_MNT	+3	Mount option (second byte)
G_BLSZ	+4	Tape block size
G_LRSZ	+5	Tape logical record size
G_NBF	+6	Number of buffers (first byte)
G_TFSN	+6	Tape file sequence number (second byte)
G_TLF	+7	Tape label format (first byte)
G_TDT	+7	Tape data types (second byte)
G_TDF	+8	Tape data format (first byte)
G_BSN	+8	Tape BSN indicator (second byte)
G_TRP	+9	Tape retention period
G_SZ	+11	Size of structure (in words); not a field in the block



# GET FILE INFORMATION

## GET FILE INFORMATION

Macro Call Name: \$GIFIL

Function Code: 10/60

Equivalent Command: None

Retrieves information about the specified file. You identify the file by supplying either a logical file number (LFN) or a pathname. This macro call returns information such as file type, device type, and, optionally, other file attributes (logical record size, block or control interval size, space allocation, etc.). In addition, you can receive a description of the keys of an indexed file.

### FORMAT:

[label] \$GIFIL [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. (Entries marked with an asterisk (\*) are provided by the system. You must supply the other entries.) The size of each entry, whose descriptions follow this list, is as follows:

<u>Argument Structure Entry</u>	<u>Size (in bytes)</u>
logical file number	2
*pointer pathname	4
*device type	2
*logical resource number	2
*file type	1
*data format	1
file attribute pointer	4
reserved	4 (all zeros)
key descriptor pointer	4
reserved	4 (all zeros)

#### 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 indicates that an LFN is not specified. If this entry contains blanks, the pathname-pointer entry (below) must point to a pathname.

#### \*pointer pathname

A 4-byte address, which may be any address form valid for an address register. If an LFN is specified in the first entry, this entry (optionally) points to a 58-byte field in main memory into which the system places the full absolute pathname associated with the LFN. If the LFN entry contains ASCII blanks, this entry points to the location where a pathname (which must end with an ASCII space character) is found. This pathname identifies the file for which the system is to retrieve information. Zeros in this entry indicate that the pathname is not to be returned. If zeros are specified, the LFN entry (above) must contain a nonblank value.

#### \*device type

A 2-byte entry into which the system places the 4-digit hexadecimal device code of the device containing the file. The devices, their codes, and marketing identifiers are:

<u>Peripheral Device</u>	<u>Device Type Code</u>	<u>Marketing Identifier</u>
Card Reader	2008	CRU9101/9102/9103/9104
Teleprinter	2018 2019	TTU9102 TTU9101
CRT Keyboard Console	2020	DKU9101
Keyboard Typewriter Console	2018	TWU9101
Diskette	2010	DIU9101/9102
Cartridge Disk	2330 2331 2332 2334	CDU9101 CDU9102 CDU9103 CDU9104
Serial Printer	2004 2006	PRU9101 PRU9102
Line Printer	2000 2001 2002 2003	PRU9104/9106 same as above but with Option PRF9102 PRU9103/9105 same as above but with Option PRF9102
Magnetic Tape	2045 2046 2045 2049 2046 204A 2067 2071 206A 2071	MTU9104 (9-track NRZI, 45-ips) MTU9105 (9-track NRZI, 75-ips) MTU9109 (9-track, 800 bpi, NRZI, 45-ips) MTU9109 (9-track, 1600 bpi PE, 45-ips) MTU9110 (9-track, 800 bpi NRZI, 75-ips) MTU9110 (9-track, 1600 bpi PE, 75-ips) MTU9112 (7-track, 556- bpi) MTU9112 (7-track, 800- bpi) MTU9113 (7-track, 556- bpi) MTU9113 (7-track, 800- bpi)

<u>Peripheral Device</u>	<u>Device Type Code</u>	<u>Marketing Identifier</u>
Magnetic Tape (cont)	2049	MTU9114 (9-track, 1600-bpi, 45-ips)
	204A	MTU9115 (9-track, 1600-bpi, 75-ips)
Mass Storage Unit	2360	MSU9101/9105 (40-megabyte)
	2361	MSU9102/9106 (80-megabyte)
	2362	MSU9103 (143/127-megabyte)
	2363	MSU9104 (288/256-megabyte)
Card Reader/Punch	2088	CCU9101/PCU9101

\*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  
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)  
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:

\*logical record size

A 2-byte entry specifying the maximum size (in bytes) of logical records stored in the file; the size does not include headers.

\*control interval/physical sector size

A 2-byte entry, as follows:

For fixed-relative files: the size (in bytes) of a physical sector.

For all other files: the size (in bytes) of a control interval, including control interval and logical-record headers.

\*current allocation size

A 2-byte entry specifying the number of active control intervals. This value may be less than the currently allocated space.

\*allocation increment size

A 2-byte entry specifying the number of additional control intervals to be allocated to the file when it is necessary to do so. This value is the size of an additional extent to be added to the file.

\*maximum allocation size

A 2-byte entry specifying the maximum number of control intervals that can be allocated to the file; indicates the limit to which the file can grow. Zeros indicate there is no defined limit.

\*amount of free space per control interval

A 2-byte entry, as follows:

For indexed files: the number of bytes to be left free in each control interval at file loading time; this value supplies space for records to be inserted without causing overflow.

For all other files: contains zeros.

\*local overflow allocation increment

For indexed files: a 2-byte value that sets the frequency at which a local overflow control interval will be allocated when the file is loaded. For example, if this value is 10, one local overflow control interval will be allocated after every ten data control intervals allocated.

For all other files: contains zeros.

\*number of keys

A 2-byte entry that contains a 1 for indexed files, and a 0 for all other files.

\*reserved

A 16-byte field containing zeros.

For tape-resident files, the structure contains the following information:

\*logical record size

A 2-byte entry specifying the maximum size (in bytes) of logical records stored in the file; the size does not include headers.

\*block size

A 2-byte entry specifying the maximum size (in bytes) of a block, including block and logical-record headers.

\*reserved

A 28-byte entry containing zeros.

For device files, the structure contains the following information:

\*logical record size

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

\*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 Z'0000FFFF'.

\*reserved

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
- 0201 - Illegal pathname
- 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
- 0238 - Invalid file description

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

Example:

In this example, the get file information macro call is used to obtain information about the file reserved in the example for the get file macro call. The argument structure is defined as follows:

F_INFO	DC	5	LFN=5
	DC	<PATH5	POINTER TO PATHNAME
	RESV	2-\$AF	
	RESV	3,0	DEV. TYPE, LRN, FILE/RECORD TYPE INFOR AREA
	DC	<FILATT	POINTER TO FILE ATTRIBUTE AREA
	RESV	2-\$AF	
	RESV	6,0	RESERVED
PATH5	RESV	29,0	FIELD TO RECEIVE PATH
FILATT	RESV	16,0	FIELD TO RECEIVE FILE ATTRIBUTE INFO

Since, as stated under "Assumptions for File System Examples" (in Section 3), the \$GIPSB and \$GIFAB macro calls have been included in your procedure, you can reference any entry in F\_INFO and FILATT after executing the following macro call:

```
$GIFIL !F_INFO
```

The following instructions allow the reference to be made:

```
LAB $B6,F INFO  
LAB $B7,FILATT
```

Then, for example, to reference the system-supplied logical resource number and control interval size, respectively, you would specify the following address syllables in your instructions:

```
$B6.I_LRN      SYSTEM-SUPPLIED LRN  
$B7.K_CISZ    SYSTEM-SUPPLIED C.I. SIZE
```

# GET FILE INFORMATION, FILE ATTRIBUTE BLOCK OFFSETS

GET FILE INFORMATION, FILE ATTRIBUTE BLOCK OFFSETS

Macro Call Name: \$GIFAB

Associated Macro Call: \$GIFIL

Generated Offsets Tags:

For tape-resident and device files:

<u>Tag</u>	<u>Corresponding Offsets (in words)</u>	<u>Entry Name</u>
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:

<u>Tag</u>	<u>Corresponding Offsets (in words)</u>	<u>Entry Name</u>
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

\*

# GET FILE INFORMATION, KEY DESCRIPTORS BLOCK OFFSETS

## GET FILE INFORMATION, KEY DESCRIPTORS BLOCK OFFSETS

Macro Call Name: \$GIKDB

Associated Macro Call: \$GIFIL

Generated Offsets Tags:

<u>Tag</u>	<u>Corresponding Offsets (in words)</u>	<u>Entry Name</u>
Y_RFU1	0	Reserved
Y_RT	+1	Record-type
Y_NKC	+2	Number of key components
Y_RFU2	+3	Reserved
Y_KLEN	+7	Key type and size in bytes
Y_KOFF	+8	Key offset in bytes
Y_SZ	+9	Size of structure (in words); not a field in the block

# GET FILE INFORMATION, PARAMETER STRUCTURE BLOCK OFFSETS

## GET FILE INFORMATION, PARAMETER STRUCTURE BLOCK OFFSETS

Macro Call Name: \$GIPSB

Associated Macro Call: \$GIFIL

Generated Offsets Tags:

<u>Tag</u>	<u>Corresponding Offsets (in words)</u>	<u>Entry Name</u>
I_LFN	0	Logical file number (LFN)
I_PTHP	+1	Pointer to pathname
I_DTYP	+3	Device type
I_LRN	+4	Logical resource number
I_FTYP	+5	File type (first byte)
I_RTYP	+5	Data format (second byte)
I_FABP	+6	Pointer to file attributes (see \$GIFAB description)
I_FSTP	+8	Reserved
I_KDP	+10	Pointer to key descriptors (see \$GIKDB description)
I_RFU2	+12	Reserved
I_SZ	+14	Size of structure (in words); not a field in the block

# GET MEMORY/GET AVAILABLE MEMORY

## GET MEMORY/GET AVAILABLE MEMORY

Macro Call Name: \$GMEM

Function Code: 04/02 (get memory), 04/03 (get available memory)

Equivalent Command: None

Allocate to the issuing task the requested amount of contiguous memory. The memory is allocated as a block from the memory pool of the task group to which the issuing task belongs. If the specified amount of contiguous memory is not available, perform one of the following actions:

- o Return immediately to the issuing task without performing any allocation (get memory with DENY specified).
- o Suspend the issuing task until the required memory becomes available (get memory with WAIT specified).
- o Allocate the largest contiguous block of memory currently available in the memory pool and return to the issuing task (get available memory (AVAIL) specified).

### FORMAT:

```
[label] $GMEM location of maximum number of words required  
           , [ { DENY }  
             { WAIT }  
             { AVAIL } ]
```

### ARGUMENT DESCRIPTION:

location of maximum number of words required

Any address form valid for a data register; provides the maximum number of words of memory to be allocated as a block to the issuing task. The value used cannot exceed the size of the pool minus the memory block header. (Each bit in the bit map represents a 32-word allocation.) The value for the number of words cannot exceed 1,048,575 (minus the memory block header).

## DENY

If the number of words of memory specified in argument 1 is not available either in the task group's or, if it can extend into it, the batch group's memory pool, return immediately to the issuing task. If argument 2 is omitted, DENY is the default value.

## WAIT

If the number of words of memory specified in argument 1 is not available either in the task group's or, if it can extend into it, the batch group's memory pool, suspend the issuing task until the memory becomes available. Activate the task, allocate the memory, and return to the task.

## AVAIL

If the number of words of memory specified in argument 1 is not available in the task group's memory pool, allocate to the issuing task the largest contiguous block of memory currently available.

This function will not obtain memory from the batch pool, even if the online pool may extend into the batch pool.

## FUNCTION DESCRIPTION:

This call allows the issuing task to dynamically obtain a block of memory from the task group's memory pool. If argument 2 is DENY, the task obtains a block of the specified size or no block at all. If argument 2 is WAIT, the task is suspended until the requested amount of memory becomes available. If the online pool extended into the batch pool, allocated memory may be from the batch pool, which may have been rolled out because of this request.

If argument 2 is AVAIL, the task obtains a block of the specified size or the largest block (less than the specified size) that is currently available.

When AVAIL (get available memory) is specified, the actual size of the memory block allocated may be much smaller than the desired size. This situation occurs because the memory manager does not wait for memory to become available. Rather, it checks for contiguous memory of the specified size and if none is available, allocates the largest contiguous block of memory that is available. If no memory is available, the system returns a status code of 0602.

NOTE: When AVAIL is specified, all of available memory may be removed from the pool. Other functions (including the command processor) that require memory from that pool then will not be able to execute until memory becomes available.

When a return is made to the issuing task, the actual size of the supplied contiguous memory block is placed in \$R6 and \$R7. "Actual size" has the following meaning: Memory is allocated in units of 32 words; a block of memory contains an integral number of 32-word allocation units. A memory block also contains a header; its size is two words in SAF mode and three words in LAF mode. The value returned in \$R6 and \$R7 is the specified number of words rounded up to the next higher allocation unit, minus the size of the memory block header.

NOTE: If AVAIL is specified and a block of the requested size could not be found, the actual size of the block is that of the largest contiguous memory block available, minus the size of the header.

The maximum size of a memory block that can be obtained is 1,048,575 words, minus the memory block header. The block size cannot exceed the pool size.

On return to the issuing task, \$B4 contains the address of the first usable word in the block (first word after the block header).

The get memory/get available memory functions enable the task to dynamically acquire additional memory in response to processing needs. When a memory block is no longer required, it must be returned to the task group's memory pool (by a return memory or return partial block of memory macro call). If a task repeatedly acquires memory blocks and does not return them, the task group memory area will become empty (or nearly so), denying other tasks the opportunity to obtain memory blocks.

- NOTES:
1. The number of contiguous words of memory required, supplied by argument 1, is placed in \$R6 and \$R7; if this argument is =\$R7, it is assumed that \$R6 and \$R7 contain the number of words desired.
  2. If argument 2 is DENY, \$R2 is set to 0. If argument 2 is WAIT, \$R2 is set to -1. If argument 2 is AVAIL, \$R2 is not set. If argument 2 is omitted, \$R2 is set to 0 (DENY).



3. On return to the issuing task, \$R1, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - If the call specified WAIT or DENY, a successful allocation was made; if the call specified AVAIL, at least one memory unit was allocated

0601 - If the call specified WAIT or DENY, requested contiguous memory exceeds defined pool size; not applicable if the call specified AVAIL

0602 - If the call specified WAIT or DENY, the requested contiguous memory could not be obtained; if the call specified AVAIL, no memory allocation units were available

The following codes could be returned if WAIT or DENY was specified.

0818 - No task group with specified group id exists (system software error)

081A - Suspend in progress (system software error)

081B - Rollout of online task group attempted (system software error)

081D - Batch task group already rolled out (system software error)

081E - Unrecoverable media error during rollout

\$R6, \$R7 - Actual size of contiguous memory blocks supplied, rounded up to the nearest multiple of 32 words

\$B4 - If \$R1 was 0000, address of first available word in memory block

Examples:

In this example, the \$GMEM macro call is used to attempt to obtain 2500 words of memory from the issuing task group's memory area. If the memory is available, the system will return with a status of 0000 in \$R1, the actual size of the memory area obtained in \$R6 and \$R7, and the address of the first usable word of the area in \$B4. The example saves the address of the memory area in the field labeled M\_PTR and continues processing. If 2500 contiguous words of memory are not available, the system will return with a status of 0602 in \$R1. If the pool size is less than 2500 words, the system will return error code 0601 in \$R1.

```
                $GMEM    =2500
                BNEZ     $R1,NO_MEM
                STB      $B4,M_PTR
                .
                .
                .
M_PTR          DC        <$
```

In this example, the \$GMEM macro call is used to attempt to obtain the largest contiguous area of memory, not exceeding 5000 words, available in the issuing task group's memory area. If any memory is available, the system will return with a status of 0000 in \$R1, the actual size of the memory area obtained in \$R6 and \$R7, and the address of the first usable word of the area in \$B4. If all of the memory in the task group's memory area is in use at the time, the system returns with a status of 0602 in \$R1.

```
$GMEM    =5000,AVAIL
```

# GET WORKING DIRECTORY

## 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

0201 - Illegal pathname

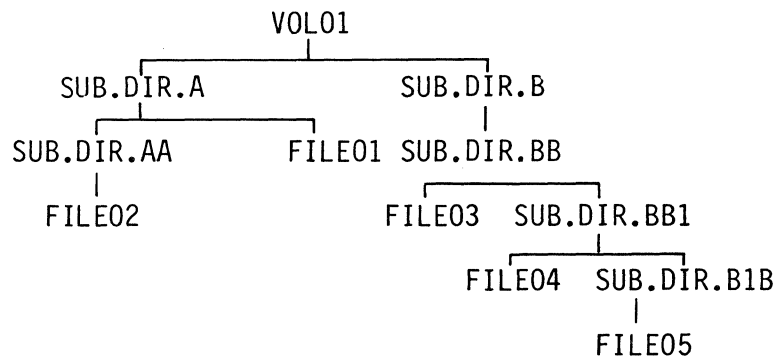
0205 - Illegal argument

0222 - Pathname cannot be expanded, no working directory

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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.BB1AAAAAAAAAAAAAAAAAAAA
```

# HOME DIRECTORY

## HOME DIRECTORY

Macro Call Name: \$HDIR

Function Code: 14/0B

Equivalent Command: None

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, non-varying 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

On return, \$B4 contains the address of the receiving field.

Example:

In this example, the pathname of the initial working directory of the calling task group is stored in the 45-character field labeled DEF\_WD.

	\$HDIR	!DEF_WD
	.	
	.	
DEF_WD	RESV	22,0
	DC	0

# INPUT/OUTPUT REQUEST BLOCK

## INPUT/OUTPUT REQUEST BLOCK

Macro Call Name: \$IORB

Function Code: None

Equivalent Command: None

Generates an input/output request block (IORB). The length of the IORB is eight or nine words in SAF (short address form), and ten to twelve words in LAF (long address form).

### FORMAT:

```
[label] $IORB [logical resource number],  
              [issuing task suspension option],
```

or

```
[issuing task termination option],  
[buffer address],  
[buffer byte alignment],  
[buffer range]
```

### ARGUMENT DESCRIPTION:

logical resource number

A value from 0 through 252 specifying the LRN of the device involved in the request. The value specified must be that of a system LRN. If this argument is omitted, the left byte of the I\_CTL word (see Appendix A) is set to zero.

issuing task suspension option

One of the following values is specified to indicate whether the requesting task is to be suspended until the completion of the request:

WAIT - Suspend the issuing task until the request is complete (set the w-bit to zero)

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

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

If WAIT is specified, argument 3 must be omitted.

#### issuing task termination option

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.



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 Program Preparation 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
```

# INPUT/OUTPUT REQUEST BLOCK OFFSETS

## INPUT/OUTPUT REQUEST BLOCK OFFSETS

Macro Call Name: \$IORBD

Counterpart: \$IORB (see "Input/Output Request Block")

Generated Label Prefixes:

IORB label	I_RRB/I_SEM offset 0 I_CT1 I_CT2 I_ADR I_RNG I_DVS I_RSR I_ST
------------	---

See Appendix A for the format of the input/output request block.

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

# INTERNAL DATE/TIME, CONVERT TO

## INTERNAL DATE/TIME, CONVERT TO

Macro Call Name: \$INDTM

Function Code: 05/07

Equivalent Command: None

Convert the external format date/time value to an internal format date/time value.

### FORMAT:

```
[label] $INDTM [location of address of external date/time],  
               [location of address of receiving field],  
               [location of size of external date/time]
```

### ARGUMENT DESCRIPTION:

location of address of external date/time

Any address form valid for an address register; provides the address of a field containing an external date/time value. This value must be in the format returned by the convert to external date/time macro call.

location of address of receiving field

Any address form valid for an address register; provides the address of a 3-word field into which the system places the internal format date/time value.

location of size of external date/time

Any address form valid for a data register; provides the size of the external date/time value identified by argument 1. The size must be less than or equal to 22 bytes. If this argument is omitted, the size is set to 20 bytes (tenth of a second resolution).

The size must be such that the date/time value does not end with the characters : (colon) or . (period).

FUNCTION DESCRIPTION:

This call converts an external date/time value (as supplied by the convert to external date/time macro call) to internal format (as supplied by the get date/time macro call). The internal date/time value appears in the receiving field as a binary count of the milliseconds that have elapsed from 1 January 1901 at 00:00:00.000 hours.

- NOTES:
1. The address of the external date/time value supplied by argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the correct external value.
  2. The internal date/time value returned is loaded into \$R2, \$R6, and \$R7, and is placed in the receiving field specified by argument 2. If argument 2 is omitted, or is = \$R7, the internal date/time value is returned only in \$R2, \$R6, and \$R7.
  3. The size of the external date/time value supplied by argument 3 is placed in \$R5. If this argument is = \$R5, it is assumed that \$R5 contains the correct size. If this argument is omitted, \$R5 is set to a value of 20 (tenth of a second resolution).
  4. On return, \$R1, \$R2, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error  
0407 - Invalid external date/time value  
040A - Invalid input field address

\$R2, \$R6, \$R7 - Generated internal date/time value

\$B4 - Address of supplied external date/time value

Example:

In this example, the \$GDTM macro call is used to get the current date/time, in internal format, leaving it in registers \$R2, \$R6, and \$R7. The \$EXTDM macro call is then used to convert this internal format to an external format, replacing the date portion (first 10 characters) of the field labeled TODAY. The TODAY field now contains the external format date/time for 0800 hours of today. The \$INDTM macro call then converts this date/time value back to an internal format contained in \$R2, \$R6, and \$R7. One day (86,400,000 milliseconds) is then added to this internal date/time giving the internal date/time for 0800 hours tomorrow, which is stored in the 3-word field labeled MORROW. The addition is programmed with the assumption that the central processor does not have the add integer double instruction.

```
*
*   GET THE CURRENT DATE/TIME VALUE.
*
*           $GDTM
*
*   CONVERT IT TO AN EXTERNAL FORMAT DATE.
*
*           $EXTDT    , !TODAY,=10
*
*   NOW CONVERT THE EXTERNAL DATE/TIME
*   BACK TO THE INTERNAL FORMAT.
*
*           $INDTM    !TODAY,,=15
*
*   ADD IN ONE DAY.
*
*           ADD    $R7,A_DAY+1
*           CAD    =$R6
*           CAD    =$R2
*           ADD    $R6,A_DAY
*           CAD    =$R2
*
*   NOW STORE THE RESULT.
*
*           STR    $R2,MORROW
*           SDI    MORROW+1
*
*           .
*           .
*           .
*   TODAY      TEXT    'YYYY/MM/DD 0800 '
*   A_DAY      DC      86400000B(31,0)
*   MORROW     RESV    3,0
```

# MESSAGE GROUP, ACCEPT

## 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 System Concepts manual for a discussion about the message facility.)

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

2. The system places the address of the MGIRB in \$B4. If the argument is omitted, the system assumes that \$B4 contains a pointer to the MGIRB.
3. Before the \$MACPT macro call is executed, the user must generate the MGIRB (see Table A-8) with the argument values shown in Table 5-2.

Table 5-2. MGIRB Argument Values for \$MACPT Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
<p>synchronous/asynchronous indicator</p> <p>Indicates whether macro call execution is to be synchronous or asynchronous.</p>	MI_MAJ (bit 9)	<p>0 - Synchronous; task waits until all specified message group conditions are met before macro call is executed.</p> <p>1 - Asynchronous; task does other processing while checking whether the message group conditions have been met.</p>
Reserved for future use.	MI_MPD	Must be 0001.
<p>acceptor identification</p> <p>Describes the following characteristics of the acceptor at a mailbox that is the destination of the message group:</p> <p>address type:</p> <p>Specifies that this is an acceptor's address.</p> <p>Reserved for future use.</p> <p>Reserved for future use.</p> <p>acceptor mailbox</p>	<p>As shown below.</p> <p>MI_ADT (bits 8-F)</p> <p>MI_NWA</p> <p>MI_NDA</p> <p>MI_MBA</p>	<p>As shown below.</p> <p>Must be hexadecimal 01.</p> <p>Must be 0.</p> <p>Must be 0.</p> <p>Must be from 1 to 12 ASCII characters, blank-filled, left justified.</p>

4. At successful macro execution, the system returns the following MGIRB output argument values:

message group identifier:

MG\_MGI field: is the message group identifier of the "accepted" message group. A valid identifier is returned for all requests even when a message group is not available.

initiator identification:

Designates the following characteristics of the initiator at a mailbox from which the acceptor will accept the message group:

address type:

MI\_ADT field (bits 0 through 7), indicates an initiator address.

mailbox name:

MI\_MBI field; is the name (from 1 to 12 ASCII characters, blank-filled, left justified) of the mailbox designated by the initiator task group as the initiator mailbox.

5. On return, \$R1 contains the following 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
  - 0C62 - Normal message group termination
6. On return, register \$B4 will point to the application's MGIRB, which is updated according to the specifications in the macro call.



# MESSAGE GROUP CONTROL REQUEST BLOCK

## MESSAGE GROUP CONTROL REQUEST BLOCK

Macro Call Name: \$MGCRB

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, does one of the following:

- o Builds a message group control request block (MGCRB) of 24 words (for SAF) and 28 words (for LAF) that contains default values for all fields not explicitly specified in the call. See Table A-7 in Appendix A.
- o Generates instructions to alter the partial contents of an existing MGCRB.

FORMAT:

[label] \$MGCRB , [arguments]

ARGUMENT DESCRIPTION:

There are three types of arguments for this macro call:

- o Keyword only (i.e., RESV).
- o Keyword with expression (expression is a user-selected variable whose literal value is used by the system).
- o Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGCRB. When the macro call is issued with RESV as its only argument, an MGCRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGCRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGCRB, in which case the keyword with expression argument ADR=address is used to specify the address of the MGCRB to be changed. When ADR=address is omitted, the system assumes that register \$B4 points to that MGCRB. The argument ADR=address is not used in building a new MGCRB, i.e., when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 5-3 below.

The first argument position is reserved for system use, and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by a comma.

Table 5-3 describes the arguments for the \$MGCRB macro call, and indicates the fields in the MGCRB into which the system inserts the argument values.

Table 5-3. Argument Values for \$MGCRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGCRB
Keyword only				
1	None	None	Reserved by system, must be a comma.	N/A
2	WAIT <sup>a</sup>	None	Issuing task suspension option: Suspend the issuing task until the request is completed (set W-bit (wait) to 0).	MC_MAJ
	NWAIT (default)	None	Do not suspend the issuing task (set W-bit to 1).	
Any	RESV	None	Generates MGCRB.	

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

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGRB
Keyword with expression				
3	SM=  RB=	aa  label	Issuing task termination <sup>b</sup> option:  When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.  When requested task is complete, do not suspend the issuing task; issue a request for the request block identified by label.	N/A
Any	ADR=	address	When existing MGRB is to be changed (RESV omitted), specifies address of MGRB to be changed.	N/A
Any	BUF=	buffer address  (default is 0)	Address of the buffer location in the task where sent or received record is to be placed.	MC_BUF
Any	RANGE=	number of bytes  (default is 0)	Length, in bytes of the buffer.	MC_BSZ

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

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGRB
Keyword with option				
Any	ALIGN=	R  L (default value)	Buffer byte alignment:  Buffer begins in right-most byte of address specified by BUF= argument.  Buffer begins in left-most byte of address specified by BUF= argument.	MC_OPT
Any	WTI=	WAIT  DENY (default value)	Wait test indicator (for \$MRECV only):  Do not process request until data is available.  Return error status when there is no data available.	MC_WTI
Any	ENC=	EOR  EOQ (default value)  EOM	Enclosure level that delimits send or receive message unit.  End-of-record.  End-of-quarantine-unit.  End-of-message.	MC_LVL
<p><sup>a</sup> When WAIT is specified, argument 3 must be omitted.</p> <p><sup>b</sup> When this argument is omitted, or argument 3 is WAIT, the generated MGRB contains no termination option. In that case, the user must issue a \$WAIT, \$WAITL, or \$TEST macro call.</p>				

#### 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 Program Preparation 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 Program Preparation 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.

2. The system places the address of the MGIRB in register \$B4. If this argument is omitted, the system assumes that \$B4 contains a pointer to the MGIRB.
3. Before the macro call is executed, the user must generate the MGIRB (see Table A-8) with the argument values shown in Table 5-4).

Table 5-4. MGIRB Argument Values for \$MCMG Macro Call

Argument Name and Description	Field in MGIRB	Argument Value
<p>synchronous/asynchronous</p> <p>Indicates whether macro call execution is to be synchronous or asynchronous</p>	<p>MI_MAJ (bit 9)</p>	<p>0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed.</p> <p>1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.</p>
<p>acceptor identification</p> <p>Describes the following characteristics of the acceptor at a mailbox that contains the message groups.</p>	<p>As shown below</p>	<p>As shown below.</p>
<p>address type:</p> <p>Specifies that this is an acceptor's address</p>	<p>MI_ADT (bits 8-F)</p>	<p>Must be hexadecimal 01.</p>
<p>Reserved for future use.</p>	<p>MI_NWA</p>	<p>Must be 0.</p>
<p>Reserved for future use.</p>	<p>MI_NDA</p>	<p>Must be 0.</p>
<p>acceptor mailbox name</p>	<p>MI_MBA</p>	<p>Must be from 1 to 12 ASCII characters, blank-filled, left justified.</p>



4. At successful macro execution, MI\_CNT will contain the count of "unaccepted" completed message groups remaining in the mailbox.
5. On return, \$R1 contains the following return status codes:
  - 0000 - No error
  - 0C02 - Invalid message group identification
  - 0C03 - Abnormal termination received
  - 0C09 - Invalid enclosure level specified
  - 0C25 - Acceptor mailbox may not be accessed by the initiator
  - 0C26 - Acceptor mailbox or acceptor mailbox node not known
  - 0C34      User-coded reason for abnormal message  
through - group termination
  - 0C44
  - 0C62 - Normal message group termination
6. On return, register \$B4 will point to the application's MGIRB, which is updated according to the specifications in the macro call.

# MESSAGE GROUP INITIALIZATION REQUEST BLOCK

## MESSAGE GROUP INITIALIZATION REQUEST BLOCK

Macro Call Name: \$MGIRB

Function Code: None

Equivalent COmmand: None

Depending on the arguments supplied in the call, does one of the following:

- o Builds a message group initialization request block (MGIRB) of 42 words (for SAF) and 46 words (for LAF) that contains default values for all fields not explicitly specified in the call. See Table A-8 in Appendix A.
- o Generates instructions to alter the partial contents of an existing MGIRB.
- o When modifying an existing MGIRB, calls and expands the corresponding \$MGIRT template macro call to provide labels for the MGIRB's fields.

FORMAT:

[label] \$MGIRB ,[arguments]

ARGUMENT DESCRIPTION:

There are three types of arguments for this macro call:

- o Keyword only (i.e., RESV).
- o Keyword with expression (expression is a user-selected variable whose literal value is used by the system).
- o Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGIRB. When the macro call is issued with RESV as its only argument, an MGIRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGIRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGIRB, in which case the keyword with expression argument ADR=address is used to specify the address of the MGIRB to be changed. When ADR=address is omitted, the system assumes that register \$B4 points to that MGIRB. The argument ADR=address is not used in building a new MGIRB, i.e., when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 5-5 below.

The first argument position is reserved for system use, and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by a comma.

Table 5-5 describes the arguments for the \$MGIRB macro call, and indicates the fields in the MGIRB into which the system inserts the argument values.

Table 5-5. Argument Values for \$MGIRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGIRB
Keyword only				
1	None	None	Reserved by system, must be a comma.	N/A
2	WAIT <sup>a</sup>	None	Issuing task suspension option: Suspend the issuing task until the request is completed (set W-bit (wait) to 0).	MI_MAJ
	NWAIT (default)	None	Do not suspend the issuing task (set W-bit to 1).	
Any	RESV	None	Generates the MGIRB.	

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

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGIRB
Keyword with expression				
3	SM=  RB=	aa  label	Issuing task termination <sup>b</sup> option:  When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.  When requested task is complete, do not suspend the issuing task; issue a request for the request block identified by label.	N/A
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 ANSII charactrs, blank-filled, left justified. Default is 12 blanks.		MI_MBI
Any	MBA=	Acceptor mailbox name.  From 1 to 12 ANSII characters, blank-filled, left justified. Default is 12 blanks.		MI_MBA
<sup>a</sup> When WAIT is specified, argument 3 must be omitted.  <sup>b</sup> When this argument is omitted, or argument 3 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 Program Preparation 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 Program Preparation manual for information about SAF/LAF independent code.

# MESSAGE GROUP, INITIATE

## 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 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
<p>synchronous/asynchronous indicator</p> <p>Indicates whether macro call execution is to be synchronous or asynchronous.</p>	MI_MAJ (bit 9)	<p>0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed.</p> <p>1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.</p>
Reserved for future use.	MI_MPD	Must be 0001.
<p>initiator identification</p> <p>Describes the following characteristics of the initiator at a mailbox where the message group originates.</p>	As shown below.	As shown below.
<p>address type</p> <p>Specifies that this is an initiator's address.</p>	MI_ADT (bits 0-7)	Must be hexadecimal 1.
Reserved for future use.	MI_NWI	Must be 0.
Reserved for future use.	MI_NDI	Must be 0.
initiator mailbox name	MI_MBI	Must be from 1 to 12 ASCII characters, blank-filled, left-justified.



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      User-coded reason for abnormal message  
through - group
  - 0C44
  - 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, RECEIVE

## MESSAGE GROUP, RECEIVE

Macro Call Name: \$MRECV

Function Code: 15/03

Equivalent Command: None

Requests that this task group receive a message group via a named mailbox, from another task group, specifies how much message data is to be received, and detects when there is no more data to be received.

### FORMAT:

[label] \$MRECV [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 the \$MACPT macro call to open the receive function of the message facility can issue one or more \$MRECV macro calls in order to receive message data from the sending task group, via named mailboxes. The message group identifier returned in the \$MACPT macro call is used by the \$MRECV macro call to identify the message groups of the sending and receiving task groups. A receive message can be any unit, not necessarily exactly as defined by the sender. A portion of a message group cannot be received unless designated as a quarantine unit by the sender. The \$MRECV macro call can request that the message be received in record sizes other than those with which it was sent. It can specify how much data is to be received in terms of numbers of bytes (range), and by "enclosure level" (see below). Every receive unit is an enclosure. The receiving task group can delimit the amount of received data as

end-of-quarantine-unit (see description of quarantine unit under the message, send (\$MSEND) macro call) or as end-of-message.

- NOTES:
1. Mailboxes must have been created before this macro call is issued. (See the create mailbox (CMBX) command in the Commands manual.)
  2. The system places the address of the MGRB in register \$B4. If the argument is omitted, the system assumes that \$B4 contains a pointer to the MGRB.
  3. Before issuing the macro call, the user must generate the MGRB (see Table A-7) with the argument values shown in Table 5-7.

Table 5-7. MGRB Argument Values for \$MRECV Macro Call

Argument Name and Description	Field in MGRB	Argument Values
message group identifier  Identifies the message group within whose enclosures the record is to be received.	MC_MGI	Value returned in \$MACPT macro call.
buffer area identifier  Defines the location within the task where the received record is to be placed.	MC_BUF	Buffer pointer.
range  Defines the maximum number of bytes to be placed into the buffer area in one execution of the macro call. When the specified range is exceeded, the transfer of message text is terminated.	MC_BSZ	User-specified.

Table 5-7 (cont). MGCRB Argument Values for \$MRECV Macro Call

Argument Name and Description	Field in MGCRB	Argument Values
<p>requested enclosure level</p> <p>Amount of data, in text units, that the receiving task group is to receive. When the buffer range is exceeded, text transfer terminates.</p>	<p>MC_LVL</p> <p>(bits 0-7)</p>	<p>ASCII values:</p> <p>1 - End-of-record, but not last record in quarantine unit.</p> <p>2 - End-of-quarantine-unit.</p> <p>5 - End-of-message.</p>
<p>wait test indicator</p> <p>Specifies whether user waits for data, even if none now available; or whether request is terminated when there is no data.</p>	<p>MC_WTI</p> <p>(bits 8-F)</p>	<p>0 - Terminate the request.</p> <p>1 - Wait for data to become available.</p>
<p>synchronous/asynchronous indicator</p> <p>Indicates whether macro call execution is to be synchronous or asynchronous.</p>	<p>MC_MAJ</p> <p>(bit 9)</p>	<p>0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed.</p> <p>1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.</p>

4. After successful receipt of a complete message (i.e., value of detected enclosure level in bits 8-F in MC\_LVL is ASCII 5) the receiving task group must issue a message group terminate (\$MTMG) to terminate the message group. (See "Message Group, Terminate" later in this section for a discussion of normal and abnormal termination.)

5. At successful macro execution, the system returns the following MGCRB output argument values:

test length (range):

MC\_RSR field, reports the number of bytes of text not transferred into the buffer area. When a record has no text associated with it, the value will equal buffer size.

detected user enclosure level:

MC\_LVL field (bits 8-F), reports the enclosure level detected at end of the transfer. Possible values (ASCII):

- 0 - No enclosure detected
- 1 - End-of-record
- 2 - End-of-quarantine-unit
- 5 - End-of-message

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

0000 - No error

0C02 - Invalid message group identifier

0C03 - Abnormal termination received

0C09 - Invalid enclosure level specified

0C16 - Message quarantine unit exceeded capacity

0C33 - Invalid user-coded abnormal termination

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

0C62 - Normal message group termination

0C64 - Terminate request rejected

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

# MESSAGE GROUP RECOVERY REQUEST BLOCK

## MESSAGE GROUP RECOVERY REQUEST BLOCK

Macro Call Name: \$MGRRB

Function Code: None

Equivalent Command: None

Depending on the arguments supplied in the call, does one of the following:

- o Builds a message group recovery request block (MGRRB) of 24 words (for SAF) and 27 words (for LAF) that contains default values for all fields not explicitly specified in the call. See Table A-9 in Appendix A.
- o Generates instructions to alter the partial contents of an existing MGRRB.
- o When modifying an existing MGRRB, calls and expands the corresponding \$MGRRT template macro call to provide labels for the MGRRB's fields.

FORMAT:

[label] \$MGRRB , [arguments]

ARGUMENT DESCRIPTION:

There are three types of arguments for this macro call:

- o Keyword only (i.e., RESV).
- o Keyword with expression (expression is a user-selected variable whose literal value is used by the system).
- o Keyword with option (option is a prescribed ASCII string that is interpreted by the system).

The keyword-only argument RESV generates an MGRRB. When the macro call is issued with RESV as its only argument, an MGRRB is built with system-assigned default values. When RESV is specified with other arguments, all entries in the MGRRB that are not specifically changed by other arguments are defaulted.

Omitting the RESV argument generates executable code to modify an existing MGRRB in which case the keyword with expression argument ADR=address is used to specify the address of the MGRRB to be changed. When ADR=address is omitted, the system assumes that register \$B4 points to that MGRRB. The argument ADR=address is not used in building a new MGRRB, i.e., when RESV is specified, the system ignores any ADR=address argument.

The other keyword-only arguments are WAIT and NWAIT, which are described in Table 5-8 below.

The first argument position is reserved for system use, and must be specified by the user as a comma. The second and third arguments are positional, and when omitted, each must be replaced by comma.

Table 5-8 describes the arguments for the \$MGRRB macro call, and indicates the fields in the MGRRB into which the system inserts the argument values.

Table 5-8. Argument Values for \$MGRRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGRRB
Keyword only				
1	None	None	Reserved by system, must be a comma.	N/A
2	WAIT <sup>a</sup>	None	Issuing task suspension option:  Suspend the issuing task until the request is completed (set W-bit (wait) to 0).	MR_MAJ
	NWAIT (default)	None	Do not suspend the issuing task. (Set W-bit bit to 1.)	
Any	RESV	None	Generates the MGRRB.	

Table 5-8 (cont). Argument Values for \$MGRRB Macro Call

Argument Position	Keyword	Keyword Value	Argument Description	Field in MGRRB
Keyword with expression				
3	SM=	aa	Issuing task termination <sup>b</sup> option:  When requested task is completed, do not suspend issuing task; release the semaphore identified by the two ASCII characters aa.	N/A
	RB=	label	When requested task is complete, do not suspend the issuing task; issue a request for the request block identified by label.	
Any	ADR=	address	When existing MGRRB is to be changed (RESV omitted), specifies address of MGRRB to be changed.	N/A
Any	TERM=	0, or 34 through 44	Message group termination code:  0 - Indicates normal termination of this message group.  34 through 44 - User-coded reason for abnormal termination.	MR_RSN
<sup>a</sup> When WAIT is specified, argument 3 must be omitted. <sup>b</sup> When this argument is omitted, or argument 3 is WAIT, the generated MGRRB contains no termination option. In that case, the user must issue a \$WAIT, \$WAITL, or \$TEST macro call.				



FUNCTION DESCRIPTION:

The message group recovery request block is used for communication between task groups, and is the means for passing arguments among task groups in connection with the message group terminate (\$MTMG) macro call of the message facility. This macro call makes it possible to modify an existing MGRRB by generating executable instructions that use registers \$R7 and \$B5 (as appropriate). The modifying process always uses \$B4 to point to the MGRRB.

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

# 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 Program Preparation manual for information about SAF/LAF independent code.

# MESSAGE GROUP, SEND

## 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 visible 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 the message connection, issues one or more \$MSEND macro calls to send message data via that connection. A task group sending a message to another task group sends through a named mailbox. The message group identifier returned in the \$MINIT macro call is used by the \$MSEND macro call to identify the message group of the task group that initiated it. A message is one or more records. Each \$MSEND sends one record, which is the basic unit of data exchange. Each transmission is a buffer of data, which must point to the MGCRB that describes the buffer. Only when designated by the sender as a "quarantine unit," is a sender's record or group of records interpreted at the destination (i.e., is visible).

Every send unit is an enclosure. The last or intermediate records in the message are identified by an enclosure level which delimits sent data as end-of-record, end-of-quarantine-unit, or end-of-message. End-of-message implies the end-of-quarantine-unit; end-of-quarantine-unit implies end-of-record.

- NOTES:
1. Mailboxes must have been created before this macro call is issued. (See the create mailbox command (CMBX) in the Commands manual.)
  2. The system places the address of the MGCRB in register \$B4. If the argument is omitted, the system assumes that \$B4 contains a pointer to the MGCRB.
  3. Before issuing the \$MSEND macro call, the user must generate the MGCRB (see Table A-7) with the argument values shown in Table 5-9.

Table 5-9. MGCRB Argument Values for \$MSEND Macro Call

Argument Name and Description	Field in MGCRB	Argument Value
<p>synchronous/asynchronous indicator</p> <p>Indicates whether macro call execution is to be synchronous or asynchronous</p>	MC MAJ (bit 9)	<p>0 - Synchronous; task waits until all specified message group conditions are met before the macro call is executed.</p> <p>1 - Asynchronous; task continues with other processing while checking whether the message group conditions have been met.</p>
<p>message group identifier</p> <p>Identifies the message group within whose enclosure the record is to be sent</p>	MC_MGI	Value returned in \$MINIT macro call.
<p>buffer area identifier</p> <p>Defines the location within the task where the message text to be sent is located.</p>	MC_BUF	Buffer pointer.

Table 5-9 (cont). MGRB Argument Values for \$MSEND Macro Call

Argument Name and Description	Field in MGRB	Argument Value
<p>user-requested enclosure level</p> <p>Defines the unit of text i.e., how much data, is contained in an "enclosure level."</p>	<p>MC_LVL</p> <p>(bits 0-7)</p>	<p>ASCII values:</p> <p>1 - End-of-record, but not last record in a quarantine unit.</p> <p>2 - End-of-quarantine unit.</p> <p>5 - End-of-message.</p>
<p>range</p> <p>Indicates the number of bytes of message text to be sent from the buffer area. A zero value indicates no text is to be sent; even then the other argument values are examined and a record enclosure is opened if not already open.</p>	<p>MC_BSZ</p>	<p>User-specified.</p>

4. To complete sending a message group, the sending task group must terminate the message group by either:
  - o Specifying an ASCII 5 (end-of-message) enclosure level in MC\_LVL of the MGRB (see Table 5-9) supplied on an \$MSEND macro call, or
  - o Issuing the macro call message group terminate (\$MTMG), with the value 0 in MR\_CNC of the MGRB. (See "Message Group, Terminate" later in this section for a discussion of normal and abnormal termination.)
5. On return, register \$R1 contains the following return status codes:
  - 0000 - No error
  - 0C02 - Invalid message group identification
  - 0C03 - Abnormal termination received

0C09 - Invalid enclosure level specified

0C16 - Message/quarantine unit exceeded  
capacity

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

0C62 - Normal message group termination

6. On return, \$B4 will point to the application's  
MGCRB, which is updated according to the  
specification in the macro call.

# MESSAGE GROUP, TERMINATE

## MESSAGE GROUP, TERMINATE

Macro Call Name: \$MTMG

Function Code: 15/04

Equivalent Command: None

Terminates a message group, either normally or abnormally.

### FORMAT:

[label] \$MTMG [location of MRRB address]

### ARGUMENT DESCRIPTION:

location of MRRB address

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

### FUNCTION DESCRIPTION:

This macro call, issued by a sending or receiving task group, causes normal or abnormal termination of a message group. A sending task group, after normal transmission of a message, must terminate the message group with either a \$MSEND macro call that specifies an end-of-message enclosure level (5 in MC\_LVL of the MRCRB), or with a \$MTMG macro call having a termination value of 0 in bits 0 through 7 of MR\_RSN (see Table 5-10 below). The sending task group can specify abnormal termination of the message group by inserting a user-coded value from 34 through 44 in bits 0-7 of MR\_RSN. This code informs the receiving task group of the reason for abnormal termination.

NOTE: When a receiving task group terminates a message group abnormally before the sending task group does, the sending task group will receive an 0C34 through 0C44 error return on his next \$MSEND macro call. At that point the sending task group can issue only a \$MTMG macro call for normal termination.

A receiving task group, after issuing a \$MACPT macro call and \$MRECV macro call(s), and after receiving complete message data (detected user enclosure level 5 in MC\_LVL), will receive a return status of 0000 in \$R1. The receiving task group must then issue a \$MTMG macro call to terminate the message group. When the receiving task group wants to terminate the message group before receiving the last completed data, it requests abnormal termination by specifying a user-coded termination value from 34 through 44 in bits 0 through 7 of MR\_RSN in a \$MTMG macro call (see Table 5-10 below). This causes the message group to be removed from the receiving task group and any remaining data to be purged. The sending task group, if still active, will receive the same return status code (from 34 through 44) in \$R1 when it next issues a \$MSEND macro call.

- NOTES:
1. The system places the address of the MGRRB in register \$B4. If the argument is omitted, the system assumes that \$B4 contains a pointer to the MGRRB.
  2. Before issuing this macro call, the user must generate the MGRRB (see Table A-9) with the argument values shown in Table 5-10 below.
  3. When the message group is terminated, its message group identifier is available for reuse.
  4. On return, register \$R1 contains the following status codes:
    - 0000 - No error
    - 0C02 - Invalid message group identifier
    - 0C33 - Invalid user-coded abnormal termination
    - 0C34 - User-coded reason for abnormal message through - group termination
    - 0C44
    - 0C62 - Normal message group termination
    - 0C64 - Terminate request rejected
  5. On return, register \$B4 will point to the application's MGRRB, which is updated according to the specifications in the macro call.



Table 5-10. MGRRB Argument Values for \$MTMG Macro Call

Argument Name and Description	Field in MGRRB	Argument Values
synchronous/asynchronous indicator	MR_MAJ (bit 9)	<p>0 - Synchronous; task waits until all specified message group conditions are met before macro call is executed.</p> <p>1 - Asynchronous; task does other processing while checking whether the message group conditions have been met.</p>
<p>message group identifier</p> <p>Identifies the message group to be terminated. This message group must have been identified in a \$MINIT or \$MACPT macro call by the task group issuing this macro call.</p>	MR_MGI	Value returned on \$MINIT and \$MACPT macro calls.
<p>termination type</p> <p>Specifies whether message termination is normal or abnormal.</p>	MR_RSN  (bits 0-7)	<p>0 - Indicates normal termination.</p> <p>Any value from 34 to 44 indicates abnormal termination, and issued in \$R1 as user-coded error codes. Any value other than 0 or from 34 through 44 is transmitted as 33.</p>

# MODE IDENTIFICATION

## MODE IDENTIFICATION

Macro Call Name: \$MODID

Function Code: 14/03

Equivalent Command: User Mode

Returns the mode component of the calling task group's user identification to a 3-character receiving field.

### FORMAT:

[label] \$MODID [location of mode id field address]

### ARGUMENT DESCRIPTION:

location of mode id field address

Any address form valid for an address register; provides the address of a 3-character, aligned, nonvarying field into which the system will place the mode component of the user identification associated with the issuing task group.

### FUNCTION DESCRIPTION:

This call returns the mode component of the task group's user identification to a field in the issuing task. The mode identification returned is that entered as part of the LOGIN command that established the user as a primary or secondary user of this task group. See the Commands manual for details.

The entire user id is returned by the user identification (\$USRID) macro call.

NOTES: 1. The address of the receiving mode 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 mode id field.

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

0000 - No error  
0817 - Memory access violation

3. On return, \$B4 contains the address of the receiving field.

Example:

In this example, \$B4 is loaded with the address (MODFL) of a 3-character field and the \$MODID macro call is issued to place the mode identification of the task group in that field.

```
MODFL    RESV    3,0
          LAB     $B4,MODFL
          .
          .
          .
          $MODID
```

# NEW PROCESS

## NEW PROCESS

Macro Call Name: \$NPROC

Function Code: 0D/0B

Equivalent Command: New Process (NPROC)

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

# NEW USER INPUT

## NEW USER INPUT

Macro Call Name: \$NUIN

Function Code: 08/04

Equivalent Command: None

Redefine, reset, or set the user-in file for the issuing task. The user-in file can be redefined by a new pathname, reset to the initial user-in file, or set to the file currently defined as the command-in file. The action taken applies only to the task that issues the macro call.

### FORMAT:

[label] \$NUIN [location of pathname address]

### ARGUMENT DESCRIPTION:

location of pathname address

Any address form valid for an address register; provides the pathname of the file that is to be used as the new user-in file for the issuing task. If \$CIN is specified for this argument, the file currently defined as the task's command-in file is also used as the new user-in file. If this argument is omitted, the file defined by the request group (\$RQGRP) macro call as the user-in file for tasks in this task group is again used for this task.

### FUNCTION DESCRIPTION:

This call allows the issuing task to use another file as the user-in file.

If a pathname is specified in the macro call, input will be read from the file identified by the pathname.

If \$CIN is specified as argument 1, the file that is currently the task's command-in file will be used as the source of input for the task.

If the call is written without an argument, the user-in file is identified as the initial user-in file for this task. (The request group macro call identifies this user-in file.)

When the macro call has been executed, \$R6 will contain the record length of the new user-in file, and \$R7 will contain the file status.

- NOTES:
1. If argument 1 is a pathname address, \$R2 is set to zero and the pathname supplied by argument 1 is placed in \$B4. If argument 1 is \$CIN, \$R2 is set to two. If argument 1 is omitted, \$R2 is set to one.
  2. On return, \$R1, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error

All file management get-file and open-file error codes may also be returned. See the System Messages manual.

\$R6 - Record length of redefined file

\$R7 - File status of redefined file (see "Command In")

\$B4 - Address of pathname string of new user-in file (if pathname was supplied in argument 1)

Example:

In this example, the issuing task is to read its input from a new user-in file name, ^V1124>UDD>TEST>JONES.

```
INAA      $NUIN      !NEWIN
           .
           .
           .
NEWIN     DC        '^V1124>UDD>TEST>JONESΔ'
```

# NEW USER OUTPUT

## NEW USER OUTPUT

Macro Call Name: \$NUOUT

Function Code: 08/05

Equivalent Command: File Out (FO)

Redefine or reset the user-out file for the task group of the issuing task. The user-out file can be redefined by a new pathname or reset to the user-out file initially defined for the issuing task group. The action taken applies to all tasks in the task group from which the command is issued.

### FORMAT:

[label] \$NUOUT [location of pathname address]

### ARGUMENT DESCRIPTION:

location of pathname address

Any address form valid for an address register; provides the pathname of the file to be used as the new user-out file for the issuing task group. If this argument is omitted, the file defined by the request group macro call (\$RQGRP) is used as the user-out file for tasks in this task group.

### FUNCTION DESCRIPTION:

This call allows the issuing task group to use another file as the user-out file.

If a pathname is specified in the macro call, the tasks in this task group will write their output to the file identified by the pathname.

If the call is written without an argument, the user output file identified as the initial output file for this task group is used for task output. (The request group macro call identifies the initial user-out file.)

When the macro call has been executed, \$R6 will contain the record length of the new user-out file, and \$R7 will contain its file status.

- NOTES:
1. The address of the pathname supplied by argument 1 is placed in \$B4, and \$R2 is set to zero. If this argument is omitted, \$R2 is set to one.
  2. On return, \$R1, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error

All file management get-file, create-file, and open-file error codes may also be returned. See the System Messages manual.

\$R6 - Record length of redefined file

\$R7 - File status of redefined file (see "Command In")

\$B4 - Address of pathname string of new user-out file (if a pathname was specified in argument 1)

Example:

In this example, the user-out file is reset to its initial definition.

OUTBK \$NUOUT



# OPEN FILE

## OPEN FILE

Macro Call Name: \$OPFIL

Function Code: 10/50 (preserve), 10/51 (renew)

Equivalent Command: None

Initializes and establishes addressability to a file (which can be used by any task in the group). You identify the file to be opened by supplying its logical file number (LFN).

### FORMAT:

[label] \$OPFIL [fib address] [ {,PRESERVE} ]  
[ {,RENEW} ]

### 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 and program view.

{PRESERVE}  
{PRE

Specifies that this is an existing data file, and that labels and data already in the file are to be preserved. Reading starts from the first logical record; writing starts at the current logical end-of-file. PRESERVE is the default value for this macro call.

For indexed files only, specifying PRESERVE means that a file when opened cannot be opened by anyone else in RENEW mode.

{RENEW}  
{REN }

Specifies that this is a new file, and that no attempt should be made to read it (i.e., the file should be treated as though it was empty).

Specifying RENEW means that the file cannot be opened by anyone else.

For disk files, both writing and reading start from the first logical record (except for indexed sequential files, which cannot be read in this mode).

For tape files, RENEW is used to rewrite an existing file or add a new file to a volume. Write permission must be granted in the FIB program view word.

#### FUNCTION DESCRIPTION:

Before this macro call can be issued, the following actions must have occurred:

1. The specified file must physically exist (i.e., it must have been created through a create file macro call).
2. The LFN must have been associated with the external file through an associate file, get file, or create file macro call (or through an equivalent command).

If a file is currently opened elsewhere in the system (by any LFN in the requesting task group or any other task group), the following rules apply:

1. Opening the file in RENEW mode is not allowed.
2. Opening an indexed file in PRESERVE mode is not allowed if the file is currently open in RENEW mode.
3. Opening a tape file in any mode is not allowed.

If an associate file macro call was executed, but a get file macro call was not, the open file macro call will attempt to reserve the file with exclusive concurrency control. (This method of opening a file is not recommended.)

A file cannot be opened directly through its pathname. If you issue a get file or create file macro call with only a pathname (no LFN specified), the system will assign an LFN, which you can then use to open the file.

If an indexed sequential file is empty (i.e., has been created but never opened in RENEW mode), and this file is opened in PRESERVE mode, the system converts the open to an open in RENEW mode and provides no notification of this change. Only write operations will be allowed. A subsequent read operation will result in a 0203 (illegal function) error code.

The following discussion and rules apply only to magnetic tape files.

1. Certain tape search rules are used when the file is opened to locate the required tape file. These rules are applied when the tape is opened for data management (record-level) access, or when a file name is specified and the tape is opened for storage management (block level) access. Table 5-11 defines these rules.

Table 5-11. Tape File Search Rules for \$OPFIL Macro Call

File Label Type and Open Mode	FSN <sup>a</sup> Value in \$GTFIL Call	Result
Labeled tapes opened in PRESERVE mode:		
File name not specified	0/FF	Tape positioned to next file.
	n	Tape positioned to nth file.
File name is specified	0	Tape positioned to next file; file name in header label is compared against specified file name.
	n	Tape positioned to nth file; file name in header label is compared against specified file name.
	FF <sub>16</sub>	Tape searched in <u>forward</u> direction only for a header label with a matching file name.

Table 5-11 (cont). Tape File Search Rules for \$OPFIL Macro Call

File Label Type and Open Mode	FSN <sup>a</sup> Value in \$GTFIL Call	Result
Labeled tape opened in RENEW mode (file name is always required)	0	Tape positioned to next file.
	n	Tape positioned to nth file.
	FF <sub>16</sub>	Tape positioned in <u>forward</u> direction only to a <u>file</u> with a matching file name. If no match is found, the new file is appended after the end of all existing files on the last tape volume.
Unlabeled tapes opened in PRESERVE mode (file or volume name cannot be specified)	0/FF <sub>16</sub>	Tape positioned to the next file (past the next tape mark).
	n	Tape positioned to the nth file (past the nth tape mark).
Unlabeled tapes opened in RENEW mode (file or volume name cannot be specified)	0	Tape positioned to the next file (past the next tape mark).
	n	Tape positioned to the nth file (past the nth tape mark).
	FF <sub>16</sub>	Tape positioned <u>forward</u> only to the end of existing data; the new file is appended after the end of all existing files on the tape.
<sup>a</sup> FSN = Tape file sequence number argument in \$GTFIL macro call.		

2. For tapes opened in PRESERVE mode, the position of data within the file is determined as follows:
  - a. If only read permission is granted (FIB program view word allows read but not write), the header label group is processed and the file positioned directly in front of the first data record.

- b. If only write permission is granted (FIB program view word allows write but not read) the header label group is processed and the file 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

0205 - Illegal argument

0206 - Unknown or illegal LFN

0208 - LFN or file already open

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 - Record lock concurrency conflict

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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.

FILE_A	DC	Z'0005'	}	(See "Assumptions for File System Examples" in Section 3.)	
.	.	.			
MYFIB	DC	Z'0005'			
.	.	.			
.	.	.			
KEY	DC	Z'0000FFFF'			
.	.	.			
.	.	.			
IDX01	DC	'^VOL03>SUBINDEX.A>FILE_AΔ'			(See create file macro call)
WRTFIL	DC	Z'0005'			(See get file macro call)
.	.	.			
.	.	.			
.	.	.			
\$CRFIL		!FILE_A or \$GTFIL !WRTFIL			
\$OPFIL		!MYFIB,RENEW			
\$WRREC		!MYFIB	(See write record macro call)		

# OPERATOR INFORMATION MESSAGE

## OPERATOR INFORMATION MESSAGE

Macro Call Name: \$OPMSG

Function Code: 09/00

Equivalent Command: Message (MSG)

Display an information message on the terminal designated as the operator terminal.

### FORMAT:

[label] \$OPMSG [location of IORB address]

### ARGUMENT DESCRIPTION:

location of IORB address

Any address form valid for an address register; provides the address of the input/output request block (IORB) that describes the location and range of the output information message. See Appendix A for a description of the IORB.

### FUNCTION DESCRIPTION:

This call enables the issuing task to send a message to the system operator. The location of the message and its range are specified in the IORB (which is generated by the \$IORB macro call, or coded by the user). The IORB also specifies whether control is to be returned to the issuing task immediately or the task is to wait until the message is displayed.

- NOTES:
1. The address of the IORB 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 and \$B4 contain the following information:



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

0000 - No error

0801 - IORB in use (T-bit on)

0802 - Invalid LRN; or console message  
suppression in effect

0803 - Illegal wait or R, S, D, bit in  
IORB is still on

(The following could occur if the IORB  
specified the issuing task was to wait for  
the message to be displayed.)

0104 - Invalid arguments

0105 - Device not ready

0106 - Device timeout

0107 - Hardware error (check IORB status  
word)

0108 - Device disabled

0109 - File mark encountered

010A - Controller unavailable

010B - Device unavailable

010C - Inconsistent request

\$B4 - Address of IORB

Example:

In this example, the \$OPMSG macro call is used to write the message labeled OP\_MSG on the operator terminal. The wait macro call (\$WAIT) is later used to block the task until the message has been received.

\$OPMSG !IORB THIS CODE EXECUTES WHETHER OR NOT OPERATOR'S MESSAGE WAS PHYSICALLY WRITTEN TO THE TERMINAL.

\$WAIT !IORB THIS CODE EXECUTES ONLY AFTER THE MESSAGE IS PHYSICALLY WRITTEN.

\*  
\*  
\*

DEFINE THE IORB.

```

IORB    RESV    $AF, 0      RSU

TEXT    Z'00';          RETURN STATUS
        B'0';          T (IN USE) BIT
        B'1';          W (DON'T WAIT) BIT
        B'0';          U (USER) BIT
        B'0';          MBZ
        B'0';          MBZ
        B'0';          MBZ
        B'01'          MUST BE 1
        }
        I_CT1

TEXT    Z'00';          LRN
        B'0';          MBZ
        B'0';          B (BYTE INDEX) BIT
        B'0';          MBZ
        B'0';          MBZ
        Z'1'          FUNCTION CODE
        }
        I_CT2

DC      <OP_MSG        BUFFER ADDRESS
DC      OP_MLN         RANGE (IN BYTES)

TEXT    B'0000000';
        B'0';          B (BREAK) BIT
        B'0';          D BIT (MBZ)
        B'0';          K BIT (MBZ)
        B'0';          E (KEYBOARD ECHO) BIT
        B'1';          L (LF) BIT
        B'0';          C (NO CR) BIT
        B'000'        MODE
        }
        I_DVS

DC      0              RESIDUAL RANGE
DC      0              STATUS WORD

```

\*  
\*  
\*

END OF THE IORB.

```

OP_MSG  TEXT    'A MESSAGE TO THE OPERATOR.'
OP_MLN  EQU     2*($-OP_MSG)

```

# OPERATOR RESPONSE MESSAGE

## OPERATOR RESPONSE MESSAGE

Macro Call Name: \$OPRSP

Function Code: 09/01

Equivalent Command: None

Display a message on the operator terminal and place the operator's response to that message in a buffer specified by the input IORB.

### FORMAT:

[label] \$OPRSP [location of IORB list address]

### ARGUMENT DESCRIPTION:

location of IORB list address

Any address form valid for an address register; provides the address of a list specifying the IORBs to be used. The format of the IORB list is as follows:

entry 1 - Address of IORB describing output message  
(to operator terminal)

entry 2 - Address of IORB describing input message  
(for operator response)

### FUNCTION DESCRIPTION:

This call enables the issuing task to send a message to the system operator and receive the operator's response to that message.

Two IORBs are needed; an IORB describing the output message and an IORB describing the input buffer for the response. Both IORBs are generated through a \$IORB macro call or coded by the user.

The output message IORB describes the location of the output message and the size of the output message.

The input IORB describes the location of the input buffer for the response, the size of the buffer, and whether control is to be returned to the issuing task immediately or, by setting the W-bit of the input IORB, after the response has been received.

- NOTES: 1. The address of the IORB list 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 and \$B4 contain the following information:

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

0000 - No error

0801 - IORB in use (T-bit on)

0802 - Invalid LRN; or console message suppression in effect

0803 - Illegal wait, or the R, S, D bit in the IORB is not zero

0817 - Memory access violation

(The following could occur if the IORB describing the input buffer specified that the issuing task was to wait for the response.)

0104 - Invalid argument

0105 - Device not ready

0106 - Device timeout

0107 - Hardware error (check IORB status word)

0108 - Device disabled

0109 - File mark encountered

010A - Controller unavailable

010B - Device unavailable

010C - Inconsistent request

010D - EOT on magnetic tape detected

\$B4 - Address of input IORB

Example:

In this example, the \$OPRSP macro call causes the message labeled OP\_QRY to be written on the operator terminal. A reply from the operator terminal will then be read into the buffer labeled OP\_ANS. The issuing task will remain blocked until the above actions have been completed.

```

                $OPRSP    !IORB_L
                .
                .
                .
*
*       DEFINE THE IORB LIST.
*
IORB_L    DC          <OUT_RB,<IN_RB
*
*       DEFINE THE IORBS.
*       OUTPUT IORB:
*
OUT_RB    RESV        $AF, 0
          TEXT        Z'00', B'00000001'
          TEXT        Z'00', B'0000', Z'1'
          DC          <OP_QRY
          DC          OP_QLN
          TEXT        B'00000000000110000'
          DC          0, 0
*
*       INPUT IORB:
*
IN_RB     RESV        $AF, 0
          TEXT        Z'00', B'00000001'
          TEXT        Z'00', B'0000', Z'2'
          DC          <OP_ANS
          DC          OP_ALN
          TEXT        B'0000000000110000'
          DC          0, 0
*
*       END OF IORBS.
*
OP_QRY    TEXT        'A QUERY TO THE OPERATOR?'
OP_QLN    EQU         2*($-OP_QRY)
OP_ANS    RESV        40, 0
OP_ALN    EQU         2*($-OP_ANS)
```

# OVERLAY AREA, RELEASE

## OVERLAY AREA, RELEASE

Macro Call Name: \$OVRLS

Function Code: 07/06

Equivalent Command: None

Exit from the calling overlay, decrement the count of users maintained for this overlay, and transfer control to the supplied return point. (The overlay must have been requested through an overlay area reserve and execute overlay (\$OVRSV) macro call.)

### FORMAT:

[label] \$OVRLS [location of return point address]

### ARGUMENT DESCRIPTION:

location of return point address

Any address form valid for an address register; provides the address of the return point to which control is to be transferred.

### FUNCTION DESCRIPTION:

This call causes an exit from the calling overlay and a return to a specified point. The identity of the overlay area table (OAT) controlling the overlay is extracted from the task control block of the issuing task. The identity of the OAT is cleared from the TCB and the count of the number of users of this overlay is decremented in the defining OAT. When the count drops to zero (i.e., the task is the last to use the reserved area), the overlay area is marked as available (i.e., released) and can be reused by a reserve area and execute overlay function. Control is transferred to the return point supplied by argument 1.

- 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.

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 defined by the specified overlay area table (OAT), increment the user count for that overlay area, load the specified floatable overlay, and transfer control to the overlay at the specified (or default) entry point. (The overlay area must have been 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.



location of OAT address

Any address form valid for an address register; provides the address of the OAT that defines this overlay area. This address was returned by the system when the OAT was created through the create overlay area table macro call (\$CROAT).

FUNCTION DESCRIPTION:

This call causes the system to perform the following:

1. Determine if the issuing task already has an area reserved. If so, an illegal overlay nesting (160B) is returned.
2. If the issuing task has no area reserved, determine whether the specified overlay of the bound unit being executed by the issuing task is currently resident in any of the overlay areas defined by the OAT referred to in the call.
3. If the overlay is resident, the system increments the area's user count and transfers control to the overlay at the specified (or default) entry point.

NOTE: The default OAT is the first OAT in the OAT queue of the current bound unit having areas of sufficient size to contain the requested overlay.

4. If the overlay is not resident, the system attempts to reserve an overlay area defined by the specified OAT. If the overlay area is successfully reserved, the system increments the user count for the area, loads the specified overlay, and transfers control to the overlay at the specified (or default) entry point.
5. If no overlay area defined by the specified OAT is available, the system suspends the issuing task until an area becomes available. When an area becomes available, the system reserves it, increments its user count, loads the specified overlay, and transfers control to its specified (or default) entry point.
6. When control is transferred to the overlay, the system records the identity of the defining OAT in the task control block of the issuing task.

The overlay to be loaded must be a user segment with proper access rights and the proper size.

- NOTES:
1. The overlay id supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the overlay id.
  2. The relative displacement of the entry point from the overlay base, supplied by argument 2, is placed in \$R6; if this argument is omitted, a value of -1 is placed in \$R6 to designate that the default entry point established through the language processor or the Linker is to be used.
  3. The address of the overlay area table (OAT), supplied by argument 3, is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the OAT to be used.
  4. On normal entry to the overlay, \$B4 contains the address of the overlay area table (OAT). All other registers are preserved as they existed at the execution of the call.
  5. On return, \$R2 and \$B4 contain the following:  
\$R2 - Overlay id (as supplied)  
\$B4 - Overlay area table address (as supplied)
  6. If an error occurs, return is made to the caller. \$R1 contains one of the following status codes:  
01xx - Media error  
0602 - Insufficient system memory  
1602 - Invalid overlay id  
1605 - Illegal start address  
160A - Insufficient memory  
160B - Illegal overlay nesting  
1610 - Named OAT cannot be found

Example:

In this example, the \$CROAT macro call is used to create an overlay area table of three 512-word entries. (It is assumed that no existing OAT controls 512-word entries.) The address of the controlling OAT will be stored in the field OAT\_A. Later, the \$OVRSV macro call is used to cause the overlay named OVLY2 to be loaded into one of the areas controlled by the OAT (if it is not already available in one of the OAT areas) and then executed at its default entry point.

```

XVAL      OVLY2
*
*   CREATE AN OAT IF ONE DOES NOT ALREADY EXIST
*
*   $CROAT   !OAT_A, =512, =3
*
*   CHECK FOR ERRORS
*
*       BNEZ   $R1, ERROR1
*       .
*       .
*
*   LOAD OVLY2 (IF NECESSARY) AND EXECUTE IT
*
*       $OVRSV  =OVLY2,, !OAT_A
*
*   CHECK FOR ERRORS
*
*       BNEZ   $R1, ERROR2
*       .
*       .
*
*   DEFINE NORMAL RETURN ADDRESS FOR OVERLAY
*
*       XDEF   (OV2_RA, $)
*       .
*       .
OAT_A     DC      <$

```

# OVERLAY, EXECUTE

## OVERLAY, EXECUTE

Macro Call Name: \$OVEXC

Function Code: 07/00

Equivalent Command: None

Load the specified overlay of the bound unit being executed by the issuing task. Transfer control to the overlay at the specified entry point or at the start address declared to the language processor or to the Linker.

### FORMAT:

```
[label] $OVEXC [location of overlay id],  
               [location of entry point offset],  
               [location of overlay base address]
```

### ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the overlay to be executed. (The overlay id is a binary value generated by the Linker.)

location of entry point offset

Any address form valid for 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.

### FUNCTION DESCRIPTION:

This call causes the named overlay to be loaded at the fixed (virtual) address established at link time.

If argument 2 is specified, the value it provides is the offset from the overlay load base. Control is transferred to that location. (Note that the offset must be less than the overlay size.) If argument 2 is not specified, control is transferred to the start address declared to the language processor or the Linker (default start address). If argument 3 is specified, it provides the location of the overlay load base.

The overlay to be loaded and executed must have the proper access rights and must not be of zero length.

- NOTES:
1. The overlay id supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the overlay id.
  2. The relative displacement of the entry point from the overlay load base, supplied by argument 2, is placed in \$R6; if this argument is omitted, a value of -1 is placed in \$R6 to designate that the default entry point is to be used.
  3. The overlay base address supplied by argument 3 is placed in \$B4; if this argument is omitted, the system assumes a null address.
  4. The address where the overlay is actually loaded is ascertained as follows:
    - a. If the overlay is nonfloatable, it is loaded at the fixed address established at link time.
    - b. If the referenced overlay is floatable and null pointer value is specified as the base address, the overlay is loaded into memory obtained by the loader from the memory pool of the issuing task's task group.
    - c. If the overlay is floatable, and a nonnull pointer value is specified as the base address, the overlay is loaded at the specified base address.
  5. On overlay entry, \$R1, \$R2, \$R6, and \$B4 contain the following information:  
\$R1 - 0000  
\$R2 - Overlay id  
\$R6 - Entry point offset

6. If an error is made in the calling sequence, return is to the caller. \$R1 contains one of the following status codes:

01xx - Media error

0602 - Insufficient system memory

0817 - Memory access violation

0829 - Group available memory exceeded

1602 - Invalid overlay id

1604 - Illegal start address (offset greater than or equal to overlay size)

160A - Insufficient memory

Example:

In this example, the \$OVEXC macro call causes the overlay named DPOSIT (of the bound unit being executed) to be loaded and started at the entry point whose offset is named ENTRY2. The example assumes that ENTRY2 has been defined as an external value when the bound unit was linked (or possibly when its source unit was assembled or compiled).

```
XVAL      DPOSIT, ENTRY2
$OVEXC    =DPOSIT, =ENTRY2
```

# OVERLAY, LOAD

## OVERLAY, LOAD

Macro Call Name: \$OVLD

Function Code: 07/01

Equivalent Command: None

Load the specified overlay of the bound unit being executed by the issuing task. Return control to the issuing task.

### FORMAT:

```
[label] $OVLD [location of overlay id],  
              [location of overlay base address]
```

### ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the overlay to be loaded. (The overlay id is a binary value generated by the Linker.)

### FUNCTION DESCRIPTION:

This macro call causes the loading of the named overlay at the fixed (virtual) address established at Link time. When the overlay is successfully loaded, control is returned to the issuing task with the overlay base address in \$B4 and the overlay default start address offset in \$R6. (The overlay default start address is that address specified to the language processor or the Linker.)

- NOTES:
1. The location of the overlay id, supplied by argument 1, is placed in \$R2. If argument 1 is omitted, \$R2 is assumed to contain the location of the overlay id.
  2. The location of the overlay base address, supplied by argument 2, is placed in \$B4; if this argument is omitted, a null address is assumed.

3. The address where the overlay is actually loaded is ascertained as follows:

- a. If the overlay is nonfloatable, it is loaded at the fixed address established at link time.
- b. If the referenced overlay is floatable and null pointer value is specified as the base address, the overlay is loaded into memory obtained by the loader from the memory pool of the issuing task's task group.
- c. If the overlay is floatable, and a nonnull pointer value is specified as the base address, the overlay is loaded at the specified base address.

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

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

0000 - No error

01xx - Media error

0601 - Requested contiguous memory exceeds defined pool size

0602 - Insufficient system memory

0817 - Memory access violation

0829 - Group available memory exceeded

1602 - Invalid overlay id

160A - Insufficient memory

\$R2 - Overlay id (on a successful return)

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

\$B4 - Overlay base address



Example:

In this example, the \$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, \$B4 will contain the overlay base address or a null pointer value for floatable overlays. For nonfloatable overlays, \$B4 is not applicable, and \$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 RELEASE, WAIT, AND RECALL

## OVERLAY RELEASE, WAIT, AND RECALL

Macro Call Name: \$OVRCL

Function Code: 07/07

Equivalent Command: None

Exit from the calling overlay. When completion status has been posted to the specified request block, perform an overlay area reserve and execute overlay function for the specified overlay. Use the current definition of the overlay control table (OCT) and overlay area table (OAT). The calling overlay must have been loaded through the overlay area reserve and execute overlay macro call (\$OVRSV).

### FORMAT:

```
[label] $OVRCL [location of overlay id],  
               [location of entry point offset],  
               [location of request block address]
```

### ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the overlay to be called when the specified request block has been posted as complete. (The overlay id is a binary value generated by the Linker.) If this argument is omitted, the overlay that issued this macro call is recalled.

location of entry point offset

Any address form valid for 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.

location of request block address

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

#### FUNCTION DESCRIPTION:

This call enables the task to exit from the calling overlay and then to load the same or another overlay when the specified request block is posted as complete. The call releases the overlay previously reserved by the overlay area reserve and execute overlay (\$OVRSV) macro call. This overlay is identified by the currently reserved overlay area field in the task control block of the issuing task. After the identity of the currently reserved area is retrieved, the field in the task control block is cleared and the usage count of the area is decremented. If this task is the last task using the area, the area is released as a resource of the associated overlay area table (OAT), and tasks waiting for an area are posted where applicable.

The identity of the associated OAT is saved. The issuing task is then forced to wait on the specified request block (RB). When the requested block is posted as completed, the overlay area table is restored and the overlay area reserve and execute overlay (\$OVRSV) function is performed to recall the specified overlay. The address and completion status of the request block are returned to the called overlay.

If argument 1 specifies an overlay that is resident in the area defined by this OAT, the area's user count is incremented and control is transferred to the overlay at the specified (or default) entry point (argument 2). If the overlay is not resident, an attempt is made to reserve an overlay area controlled by the OAT. If the area is successfully reserved, the user count is incremented and the overlay is loaded and executed. If no overlay area defined by the OAT is available, the issuing task is suspended until an area becomes available. When the area is available, the count is incremented and the overlay is loaded and executed.

- NOTES:
1. The overlay id supplied by argument 1 is placed in \$R2; if this argument is omitted, a value of -1 is placed in \$R2 to designate that the issuing overlay is to be recalled.
  2. The relative displacement of the entry point from the overlay base, supplied by argument 2, is placed in \$R6; if this argument is omitted, a value of -1 is placed in \$R6 to designate that the default entry point established through the language processor or the Linker is to be used.

3. The address of the request block to be waited on, supplied by argument 3, is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the request block.
4. On entry to the called overlay (no error in calling sequence), \$R1 and \$B4 contain the following information:
  - \$R1 - Posted completion status of specified request block, as follows:
    - 0000 - No error
    - 0000-FFFF - Request-specific posted completion status
  - \$B4 - Start address of called overlay (used by debugger) or address of OAT (if no debugging in effect)
5. If the calling sequence is in error, return is made to the calling overlay. \$R1, \$R2, and \$B4 contain the following information:
  - \$R1 - Return status; one of the following:
    - 0802 - Invalid LRN
    - 0803 - Illegal wait
    - 1602 - Invalid overlay id
    - 1605 - Illegal start address
    - 1617 - OAT has no overlay to release
  - \$R2 - Overlay id value (as supplied)
  - \$R6 - Overlay entry point offset (as supplied)
  - \$B4 - Request block address (as supplied)

Example:

In the following example, the task is to exit from the current overlay and wait for the task request block named TRB1 to be marked as complete before loading overlay OVLY2 and executing it at its default entry point. Note that the overlay to be exited from and the overlay to be loaded and executed are controlled by the OAT whose identity was stored in the task control block of the issuing task by a previously issued overlay area reserve and execute overlay macro call.

	XVAL	OVLY2
	.	
	.	
TRB1	\$TRB	17
	.	
	.	
	\$OVRCL	=OVLY2,, !TRB1

# OVERLAY STATUS

## OVERLAY STATUS

Macro Call Name: SOVST

Function Code: 07/03

Equivalent Command: None

Return the current status of the specified overlay. Among the items of status information returned are:

- o Sharable or nonsharable bound unit
- o Patched or nonpatched overlay

FORMAT:

[label] SOVST [location of overlay id]

ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the overlay whose status is desired. (The overlay id is a binary value generated by the Linker.)

FUNCTION DESCRIPTION:

This call causes the system to return an overlay status word in \$R2. The description of \$R2, below, contains the detail of the overlay status word. Bits 5 and 6 are meaningful only to the call/cancel/exit controller.

When this call is executed, the overlay entry point is returned in \$R6, the overlay size in \$R7, and the overlay base address in \$B4.

NOTES: 1. The overlay id supplied by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the required overlay id.

2. On return, \$R1, \$R2, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error  
1601 - Invalid overlay id

\$R2 - Overlay status indicator word:

- Bit 0 - Set to 1 if bound unit sharable; otherwise 0.
- Bit 1 - Set to 1 if overlay permanently loaded; otherwise 0.
- Bit 2 - Set to 1 if slow load section present; otherwise 0.
- Bit 3 - Set to 1 if overlay floatable; otherwise 0.
- Bit 4 - Set to 1 if bound unit can be executed in system task group; otherwise 0.
- Bit 5 - Set to 1 if overlay resident in memory; otherwise 0; meaningful only to call/cancel/exit controller.
- Bit 6 - Set to 1 if overlay has been called but has not exited; otherwise 0; meaningful only to call/cancel/exit controller.
- Bits 7 through 9 - Reserved for system use.
- Bit 10 - Set to 1 if overlay contains initialized relocatable pointers; otherwise 0.
- Bit 11 - Set to 1 if overlay contains symbolic references; otherwise 0.
- Bit 12 - Set to 1 if overlay defines symbolic names; otherwise 0.
- Bit 13 - Set to 1 if overlay is patched; otherwise 0.

Bit 14 - Set to 1 if overlay must be  
executed in SAF mode; otherwise  
0.

Bit 15 - Set to 1 if overlay must be  
executed in LAF mode; otherwise  
0.

\$R6 - Overlay default entry point (as specified  
by the language processor or Linker);  
given as a word offset from the overlay  
base address.

\$R7 - Overlay size in words (0000 is returned if  
the size is 64K words).

\$B4 - Base address of overlay if permanently  
loaded or nonfloatable.

Example:

In this example, the \$OVST macro call is used to determine  
the status of the overlay named DPOSIT, which is an overlay  
of the bound unit being executed. If the overlay is float-  
able, the get memory macro call (\$GMEM) obtains memory for  
the overlay. The overlay execute macro call (\$OVEXC) then  
loads the overlay and starts it at its default entry point.  
To simplify the example, the return status will not be  
checked for possible errors.

```
*
*   NAME THE STATUS INDICATORS TO BE USED.
*
FLOAT          EQU      B'0001000000000000'
*
*   DECLARE THE OVERLAY'S NAME.
*
                XVAL    DPOSIT
*
*   GET THE OVERLAY'S STATUS.
*
                $OVST   DPOSIT
*
*   GET MEMORY FOR IT IF IT IS FLOATABLE.
*
                LB      = $R2, FLOAT
                BBF     > LOAD
                LDV     $R6, 0
                $GMEM   = $R7, WAIT
*
*   LOAD AND EXECUTE THE OVERLAY.
*
LOAD           $OVEXC  DPOSIT
```



# OVERLAY, UNLOAD

## OVERLAY, UNLOAD

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],  
              [location of return point address]
```

### ARGUMENT DESCRIPTION:

location of overlay id

Any address form valid for an address register; provides the overlay id of the overlay to be unloaded. (The overlay id is a binary value generated by the Linker.)

location of overlay base address

Any address form valid for an address register; provides the base address of the overlay to be unloaded. The load overlay and execute overlay macro calls cause the overlay to be loaded at the fixed (virtual) address established at Link time.

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. The overlay base address supplied by argument 2 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the base address.
  3. The return point address supplied by argument 3 is placed in \$B5; if this argument is omitted, the return point address is assumed to be the address of the first word following the generated monitor call sequence.
  4. The overlay being unloaded must be floatable, and the memory it occupies must have been obtained by a get memory macro 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.
  5. On return, \$R1 contains one of the following status codes:
    - 0000 - No error
    - 0817 - Memory access violation
    - 0818 - No task group with specified id exists (system software error)
    - 081B - Roll-in of online task group attempted (system software error)
    - 081C - Roll-in attempted when the batch group was not rolled out (system software error)
    - 081E - Unrecoverable media error during roll-in
    - 081F - Group not suspended when roll-in attempted (system software error)

Example:

See the example given for the overlay load (\$OVLD) macro call.

# PARAMETER BLOCK

## PARAMETER BLOCK

Macro Call Name: \$PRBLK

Function Code: None

Equivalent Command: None

Generate a parameter block that is equivalent to the argument list portion of the task request block.

### FORMAT:

```
[label] $PRBLK [user argument 1],  
              [user argument 2],  
              .  
              .  
              [user argument n]
```

### ARGUMENT DESCRIPTION:

user argument 1 ... user argument n

User argument values; a parameter block is generated containing the specified user argument values in the parameter positions that correspond to the argument positions in the \$PRBLK macro call. Pathname arguments that include a trailing blank should be enclosed in single or double (' or ") quotation marks.

If an argument value of zero is specified before the last argument, an argument pointer of zeros is generated in the corresponding position in the argument list.

### FUNCTION DESCRIPTION:

A parameter block is equivalent to the argument list portion of the task request block (see the task request block macro call).

A parameter block is the standard means of providing additional arguments to the request group, spawn group, and request batch macro calls.

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

Example:

In this example, the \$PRBLK macro call is used to specialize a control file for the command processor by replacing &1 and &2 by -XREF and -PRINT. (The parameter block format is given in Appendix A.)

```
ARGS1  $PRBLK  -XREF, -PRINT
```

# PERSON IDENTIFICATION

## PERSON IDENTIFICATION

Macro Call Name: \$PERID

Function Code: 14/01

Equivalent Command: None

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

### FORMAT:

[label] \$PERID [location of person id field address]

### ARGUMENT DESCRIPTION:

location of person id field address

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

### FUNCTION DESCRIPTION:

This call returns the person component (i.e., the user's personal identification) of the task group's user identification to a field in the issuing task. The person identification returned is that entered as part of the LOGIN command that established the user as a primary or secondary user of this task group. See the Commands manual for details.

The entire user id is returned by the user identification (\$USRID) macro call.

NOTES: 1. The address of the receiving person 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.

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

0000 - No error  
0817 - Memory access violation

3. On return, \$B4 contains the address of the receiving field.

Example:

In the following example, a 12-character field is set up in the issuing task and the \$PERID macro call is issued to place the person identification of the task group in that field.

ID02	\$PERID	!PRIDFL
	.	
	.	
	.	
PRIDFL	RESV	6,0

# READ BLOCK

## READ BLOCK

Macro Call Name: \$RDBLK

Function Codes: 12/00 (normal), 12/01 (tape mark), 12/02  
(beginning of tape), 12/03 (space), 12/04 (end  
of tape)

Equivalent Command: None

Read (i.e., transfer) a block from a file to a buffer in main memory; you must supply a buffer and specify both the size of the block and its relative location in the file.

FORMAT:

[label] \$RDBLK [fib address]  $\left[ \begin{array}{l} (, \text{NORMAL}) \\ (, \text{TM}) \\ (, \text{BOT}) \\ (, \text{SPACE}) \\ (, \text{EOT}) \end{array} \right]$

ARGUMENT DESCRIPTION:

fib address

Any address form valid for an address register; provides the location of the file information block (FIB). The following FIB entries are required.

logical file number

program view

Should include buffer alignment and whether the next read operation is synchronous or asynchronous.

user buffer pointer

transfer size



block size

Must be a multiple of the physical sector size.

block number

{NORMAL}  
{NOR }

For disk resident files this mode argument indicates that the block identified in the block number entry in the FIB is transferred from the file to the buffer area.

For nondisk files this mode argument indicates that the next block is to be transferred from the file to the buffer.

NORMAL is the default value for this macro call.

TM

(For tape-resident files only.) This mode argument indicates that the tape is to be moved forward or backward the number of tape marks specified in the block number entry in the FIB. Positioning is to a point immediately following the nth tape mark. A positive value indicates forward movement; a negative value indicates backward movement.

{SPACE}  
{SPA }

(For tape-resident files only.) This mode argument indicates that the tape is to be moved forward or backward the number of blocks specified in the FIB block number entry. Positioning is to a point immediately following the nth block. A positive value in the block number entry indicates forward movement; a negative value indicates backward movement.

EOT

(For tape-resident files only.) This mode argument causes the tape to be positioned to its logical end, which is defined as the occurrence of two tape marks in succession. Positioning is to a point immediately following the second tape mark.

## FUNCTION DESCRIPTION:

Before this macro call is executed, the LFN must have been opened (see open file macro call) with a FIB program view word that allows access via storage management (bit 0 is 1) and allows read operations (bit 1 is 1). In order to read the file sequentially, it is necessary only to issue successive read block macro calls in NORMAL mode, which causes the block-number entry to be incremented by 1 after each transfer. If there is not sufficient data in the block being transferred to fill the buffer, the transfer size entry in the FIB is set by the system to the number of bytes read and a return code of 0000 is delivered.

After completion of a TM, BOT, or EOT operation, the block-number entry in the FIB is automatically reset to 0; however, a SPACE operation causes the system to specify the actual relative number of the next block that would be read by a read block macro call. If a tape mark is encountered during a SPACE operation, the operation is terminated and a return-status code of 021F is delivered. In addition, if the end-of-reel is reached, a 0105 error code (device not ready) is delivered; however, if the end-of-tape is reached, it is treated like a normal operation and a return code of 0000 is delivered on successful completion.

\* Only one asynchronous I/O operation per file can be outstanding at any given time.

The file information block can be generated by a \$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. Upon return, \$R1 contains one of the following return codes:

- 0000 - No error
- 0203 - Illegal function
- 0205 - Illegal argument
- 0206 - Unknown or illegal LFN
- 0207 - LFN not open
- 020A - Address out of file
- 0217 - Access violation
- 021F - End of file

In addition to the above codes, any system service codes received by the storage manager are passed on through \$R1.

Example:

In this example the FIB is defined as follows:

```
BLKFIB  DC      Z'0005'      LFN=5
        DC      Z'E000'      PROGRAM VIEW = ALLOW READ/WRITE
                                SYNCHRONOUS PROCESSING
        DC      <BLKBUF      BUFFER POINTER
        RESV    2-$AF
        DC      256          TRANSFER SIZE = 256
        DC      256          BLOCK SIZE = 256
        DC      Z'00000000'
```

Based on the above FIB, block 0, which is 256 bytes long, is transferred to a buffer, labeled BLKBUF, in main memory.

```
SRDBLK  !BLKFIB,NORMAL
```

# READ EXTERNAL SWITCHES

## READ EXTERNAL SWITCHES

Macro Call Name: \$RDSW

Function Code: 0B/00

Equivalent Command: None

Return the current value of the specified switches in the task group's external switch word; return the inclusive logical OR of the current settings.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

external switch name ... external switch name

A single hexadecimal digit specifying the external switch in the task group's external switch word to be read. A maximum of 16 external switch names (0 through F) can be specified. If no arguments are supplied, \$R2 is assumed to contain the switches to be read. If ALL is specified, all switches are read.

### FUNCTION DESCRIPTION:

This call provides a mask by which the current setting of selected switches in the task group's external switch word can be read.

\$R2 is the mask word. Each bit that is 1 in \$R2 causes the corresponding bit in the external switch word to be read.

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

## READ RECORD

Macro Call Name: \$RDREC

Function Code: 11/10 (next), 11/11 (key), 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:

[label] \$RDREC [fib address]  $\left[ \begin{array}{l} , \text{NEXT} \\ , \text{KEY} \\ , \text{POSEQ} \\ , \text{POSGR} \\ , \text{POSGREQ} \\ , \text{POSFWD} \\ , \text{POSBWD} \end{array} \right]$

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; simple key unless relative access specified at open time).

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.

{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.



{POSGREQ}  
{PGE }

(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.

{POSFWD}  
{PFD }

(For tape-resident 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.

{POSBWD}  
{PBD }

(For tape-resident 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 \$TFIB macro call.

The following illustrate the effects of read actions according to file organizations.

<u>File Organizations</u>	<u>Effects of Read Actions</u>
Sequential	Read next causes sequential read. Read with key causes direct read. <sup>1</sup> A simple key is used.
Relative	Read next causes a sequential read. Read with key causes a direct read. <sup>1</sup> A relative or simple key can be used.
Indexed	Read next causes a sequential read. The records returned are in logical 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. <sup>1</sup> A primary key or simple key can be used.
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.

- 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
  - 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

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

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

In addition to the above codes, any system service codes received by the data manager are passed on through \$R1.

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)
```

# RELEASE DIRECTORY

## RELEASE DIRECTORY

Macro Call: \$RLDIR

Function Code: 10/A5

Equivalent Command: Release (RL)

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] \$RLDIR [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
    - 0201 - Illegal pathname
    - 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
    - 022C - Access control list (ACL) violation

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

Example:

In this example, the \$RLDIR macro call releases the directory created in the create directory example (i.e., SUBINDEX.A). The system uses the first entry to identify the directory to be released. The release directory macro call is coded as:

```
SUBDIR    DC    <DIRPTH
DIRPTH    DC    '^VOL03>SUBINDEX.AΔ'
          $RLDIR !SUBDIR
```

# RELEASE FILE

## RELEASE FILE

Macro Call Name: \$SRLFIL

Function Code: 10/35

Equivalent Command: Release (RL)

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 released by supplying either a logical file number (LFN) or a pathname. This function is usually done outside program execution.

### FORMAT:

[label] SRLFIL [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 released 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 release 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
    - 0201 - Illegal pathname
    - 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
    - 0222 - Pathname cannot be expanded, no working directory
    - 0225 - Not enough system memory for buffers or structures
    - 0226 - Not enough user memory for buffers or structures



022C - Access control list (ACL) violation

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

Example:

In this example, the macro call releases 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 released. The release file macro call is coded as:

```
$R1FIL !FILE_A
```

# RELEASE SEMAPHORE

## 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.

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

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

0000 - No error

0502 - Semaphore not defined

\$R6 - Semaphore identifier (as supplied)

Example:

See the example given for the define semaphore macro call.

# RELEASE TERMINAL

## RELEASE TERMINAL

Macro Call Name: \$RLTML

Function Code: 17/04

Equivalent Command: None

Issued by a task group to release a secondary user terminal back to the listener component after the terminal file has been closed and removed.

### FORMAT:

[label] \$RLTML [location of terminal LRN],  
[location of status code]

### ARGUMENT DESCRIPTION:

location of terminal LRN

Any address form valid for a data register; provides the logical resource number (LRN) of the terminal to be released.

location of status code

Any address form valid for a data register; provides a completion status code that will be posted when the request is marked as terminated. Acceptable status codes are:

0000 - Normal release  
3920 - Secondary login rejected

### FUNCTION DESCRIPTION:

This call is used to return a secondary user terminal that was previously obtained by the calling task group through a request terminal (\$RQTML) macro call. Until this call is issued, the terminal is reserved for the task group that issued the \$RQTML call.

- NOTES:
1. The LRN of the addressed terminal, supplied by argument 1, is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the terminal's LRN.
  2. The status code supplied by argument 2 is placed in \$R7; if this argument is omitted, \$R7 is assumed to contain the status code.
  3. On return, \$R1 contains one of the following status codes:
    - 0000 - Terminal successfully released
    - 3902 - Invalid LRN
    - 3921 - Terminal not assigned to task group
    - 3928 - Unable to release terminal; file not removed

Example:

In this example, the SRLTML macro call is used to release a terminal previously reserved through a request terminal call. Note that the terminal LRN is stored in word 0 of the area that received the login parameters (see the request terminal macro call). In this example, the LRN was later stored in the field LRN\_STR. A status code of 0000 is to be used.

```
REL_TA    SRLTML    =LRN_STR, =0
```

# REMOVE FILE

## REMOVE FILE

Macro Call Name: \$RMFIL

Function Code: 10/25

Equivalent Command: Remove (REMOVE)

Cancels the file reservation previously established by a get file macro call. You identify the file to be removed by supplying either a logical file number (LFN) or a pathname. This function is usually done outside program execution.

### FORMAT:

[label] SRMFIL [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 from 0 through 255, or ASCII blanks (2020), which indicate that an LFN is not specified.

pathname pointer

A 4-byte address, which may be any address form valid for an address register; it points to a pathname (which must end with an ASCII space character) that identifies the directory in the file hierarchy in which the file to be removed is found (as well as the name of the file itself). Binary zeros in this entry indicate that a pathname is not specified.

## 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 release 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
    - 0201 - Illegal pathname
    - 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

- 0222 - Pathname cannot be expanded, no working directory
- 0225 - Not enough system memory for buffers or structures
- 0226 - Not enough user memory for buffers or structures
- 0229 - File not known to task group

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

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 Commands manual).

The file can be renamed by specifying (1) an LFN only or (2) a pathname only. If only an LFN is specified, the file must have been reserved (through a create file or get file macro call, or equivalent command) with that LFN.

- 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
    - 0201 - Illegal pathname
    - 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
    - 0226 - Not enough user memory for buffers or structures

0228 - Illegal file type

022C - Access control list (ACL) violation

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

Example:

In this example, assume a file has been created in the directory SUB.INDEX.A by the name of FILEA. Its full path-name is ^VOL03>SUB.INDEX.A>FILEA. In addition this file is reserved with LFN=2. User executes this code:

```
ANEWAA  $RNFIL  !NEWNM1
          .
          .
          .
NEWNM1  DC      2          LFN = 2
        RESV   2,0      NO PATHNAME POINTER
        DC     'OLDF_1Δ'
```

The result is that FILEA in the directory SUB.INDEX.A is renamed to OLDF\_1.

# REPORT ERROR CONDITION

## REPORT ERROR CONDITION

Macro Call Name: \$RPTER

Function Code: 0F/00

Equivalent Command: None

Report error conditions that follow the standard error code format and provide optional error message expansion text.

### FORMAT:

```
[label] $RPTER [location of component error code],  
               [location of category/specific error codes],  
               [location of message expansion text size],  
               [location of message expansion text]
```

### ARGUMENT DESCRIPTION:

location of component error code

Any address form valid for a data register; provides the 2-byte hexadecimal error code of the software component that reports the error. The first byte must be zero; the second byte is the component error code (reporting component). The symbolic name, if any, specified in a Linker EDEF directive for the task start address is included with the message.

location of category/specific error codes

Any address form valid for a data register; provides the 2-byte hexadecimal error category code and specific error condition. Byte 0 is the error code of the component that detects the error; byte 1 is the specific error condition within that component.

#### location of message expansion text size

Any address form valid for a data register; provides the size of the message expansion text that further explains the error. The text size must include the slew byte (format control byte), which is the first byte in the message expansion text (the actual text begins with the second byte); see "Printer Driver" in Section 6. If this argument is omitted, no expansion message text has been provided.

#### location of message expansion text

Any address form valid for an address register; provides the address of the message expansion that further defines the error. The message expansion text must include the slew byte as the first byte. If argument 3 is omitted, and this argument is omitted, there is no expansion text.

#### FUNCTION DESCRIPTION:

This macro call enables the task to report error conditions that follow the standard system error code format (see the System Messages manual). Error conditions are recorded on the error-out file unless the category/specific error code (argument 2) is 01, in which case they are recorded on the operator terminal. The component, category, and specific error codes must be provided; message text is optional.

- NOTES:
1. The component error code supplied by argument 1 is placed in \$R3; if this argument is omitted, \$R3 is assumed to contain the component error code.
  2. The category and specific error codes supplied by argument 2 are placed in \$R7; if this argument is omitted, \$R1 is assumed to contain the category and specific error codes and \$R7 is set to the value contained in \$R1.
  3. The expansion text size supplied by argument 3 is placed in \$R6; if this argument is omitted, \$R6 is set to zero to indicate that no expansion text is provided.
  4. The address of the expansion text supplied by argument 4 is placed in \$B3; if this argument is omitted, \$B3 is assumed to contain the address of the expansion text (if argument 3 was specified).

5. On return, \$R7 contains the following:

\$R7 - Codes: byte 0 is error category code;  
byte 1 is specific error code.

Example:

In this example, the user in macro call (\$SUSIN) is used to attempt to read a line from the user input file. If the attempt is unsuccessful, the \$RPTER macro call is used to report this fact. The error message produced will include the expansion text PROCESS ABORTED. The abort group request macro call (\$ABGRQ) is then used to abort processing of the current task group request with a completion status of one. Processing will begin with the next group request, if any.

```
*
*      NAME MY OWN COMPONENT CODE.
*
MY_CC   EQU      X'85'
*
*      READ THE USER INPUT FILE.
*
*      $SUSIN    !IN_BUF,=IN_BLN
*
*      IF UNSUCCESSFUL GO TO BAD_RD.
*
*      BNEZ      $R1,BAD_RD
*
*      .
*      .
*
*      REPORT AN UNSUCCESSFUL USER INPUT READ
*      AND ABORT THE CURRENT GROUP REQUEST.
*
BAD_RD  $RPTER   =MY_CC;      MY COMPONENT CODE
*      ;          ERROR CODE IS IN $R1
*      =X_TLNG;   EXPANSION TEXT LENGTH
*      !X_TEXT    LOCATION OF EXPANSION TEXT
*
*      $ABGRQ    =1          TERMINATION STATUS
X_TEXT  TEXT     'APROCESS ABORTED'
X_TLNG  EQU      2*($-X_TEXT)
```

# REQUEST BATCH

## REQUEST BATCH

Macro Call Name: \$RQBAT

Function Code: 0E/00

Equivalent Command: Enter Batch Request (EBR)

Add a request to the queue of command processor files to be processed by the command processor executing in the batch task group.

### FORMAT:

[label] \$RQBAT [location of address of argument list],  
[location of address of fixed parameter block]

### ARGUMENT DESCRIPTION:

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the parameter block macro call (\$PRBLK), to be used to build the batch request block. The batch request block is built in the system area of memory, and is used by the command processor to specialize commands read from the command-in file.

location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block, which can be generated by the parameter block macro call. This parameter block has the following arguments:

Argument 1

A string specifying the user id to be associated with this batch request (for system use). The user id currently associated with the issuing task group will be used when the call is executed from a user task group.

#### Argument 2

A pathname string specifying the command-in and the initial user-in files for the batch request. A nonzero value is required.

#### Argument 3

A pathname string specifying the error-out and initial user-out files for this batch request. If this entry is zero, one of the following assumptions is made:

- o If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.AO.
- o If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.
- o If in-path specifies an input-only device, the pathname for the output files is null.

#### Argument 4

A pathname string specifying the initial value of the working directory for this batch request.

#### FUNCTION DESCRIPTION:

This call causes a request to execute the commands contained in the file identified by the second byte in the fixed parameter block (argument 2) to be queued against the batch task group. The batch task group has a first-in/first-out queue of command processor files.

If the batch task group is dormant when the \$RQBAT macro call is issued, execution begins immediately; otherwise, the request is queued.

The command processor is executed as the lead task of the batch task group. Since the command processor obtains its commands from the file named in the second entry of the fixed parameter block, the file must begin with a command.

Batch requests cannot be waited upon.



- NOTES:
1. The address of the argument list to be used to build the request block, supplied by argument 1, is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the correct address.
  2. The address of the fixed parameter block, supplied by argument 2, is placed in \$B5; if this argument is omitted, \$B5 is assumed to contain the correct address.
  3. On return, \$R1 contains one of the following status codes:
    - 0000 - No error
    - 0209 - Invalid pathname

Example:

In this example, the \$RQBAT macro call causes a request to execute the command contained in the file ^V1124>UDD>TEST>JONES>ASM\_TST to be queued against the batch task group. This file will also be used as the user-in file. Since argument 3 is null, the user-out and error-out files will default to ^V1124>UDD>TEST>JONES ASM\_TST.AO. The user id and the initial working directory will be JONES.TEST.B and ^V1124>UDD>TEST>JONES, respectively. The arguments -XREF and -PRINT will be passed to the command processor to specialize the control file ASM\_TST (&1 and &2 in the control file will be replaced by -XREF and -PRINT, respectively). The \$PRBLK macro call used in this example is described earlier in this section.

```

      $RQBAT    !ARGS, !INFO
      .
      .
      .
INFO    $PRBLK    ^V1124>UDD>TEST>JONES>ASM_TST,; NULL USER-OUT
          ^V1124>UDD>TEST>JONES
ARGS    $PRBLK    -XREF, -PRINT

```

# REQUEST CLOCK

## REQUEST CLOCK

Macro Call Name: \$RQCL

Function Code: 05/00

Equivalent Command: None

Request the clock manager to mark the specified clock request block (CRB) as complete when the interval specified in that CRB has elapsed.

### FORMAT:

[label] \$RQCL [location of CRB address]

### ARGUMENT DESCRIPTION:

location of CRB address

Any address form valid for an address register; provides the address of the clock request block to be posted when its specified time interval has elapsed.

### FUNCTION DESCRIPTION:

This call connects the specified CRB to the timer queue.

If the clock request block is not cyclic (see "Clock Request Block" earlier in this section), when the specified interval elapses, the CRB is dequeued from the timer queue. Another \$RQCL macro call must be issued to requeue the CRB. Note that a noncyclic CRB can specify an absolute time value rather than an interval.

If the CRB is cyclic, when the specified interval elapses, the CRB is posted and a new request for the originally specified interval is automatically initiated. The automatic resetting continues until a cancel clock request (\$CNCRQ) macro call is issued. A cyclic CRB cannot have a time interval of zero, and cannot specify an absolute time value.

- NOTES:
1. The address of the CRB to be connected, 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 and \$B4 contain the following information:
    - \$R1 - Return status; one of the following:
      - 0000 - No error
      - 0401 - Illegal time value (zero value for cyclic CRB)
      - 0402 - Invalid LRN
      - 0403 - Invalid basic timer specified
    - \$B4 - Address of CRB

Example:

See the example given for the wait on request list macro call later in this section.

# REQUEST GROUP

## REQUEST GROUP

Macro Call Name: \$RQGRP

Function Code: 0D/00

Equivalent Command: Enter Group Request (EGR)

Request the execution of the lead task of a specified task group. The request is placed in the first-in/first-out request queue maintained for that task.

### FORMAT:

```
[label] $RQGRP [location of group identifier],  
              [location of address of argument list],  
              [location of address of fixed parameter block]
```

### ARGUMENT DESCRIPTION:

location of group identifier

Any address form valid for a data register; provides the group identification of the task group to be requested. This task group must have been previously defined by a create group macro call.

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the parameter block macro call to be used to specialize a request block that will be used to request the lead task.

location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block (which can be generated by the parameter block macro call). This parameter block has the following arguments:

#### Argument 1

A string specifying the user id to be associated with this request (for system use). If this entry is zero, the user id currently associated with the issuing task group will be used at the time the call is executed from a user task group.

#### Argument 2

A pathname string specifying the command-in and initial user-in files for this request for the lead task of the referenced task group. If this entry is zero, no command-in and initial user-in files will be available to the group. However, the group can later obtain a user-in file by means of the new user input macro call. A non-zero entry is required if the command processor is the lead task.

#### Argument 3

A pathname string specifying the error-out and initial user-out files for this request of the task group. If this entry is zero, one of the following assumptions is made when the call is executed:

- o If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.AO.
- o If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.
- o If in-path specifies an input-only device, the pathname for the output files is null.

#### Argument 4

A pathname string specifying the initial value of the working directory for this request of the referenced task group.

## FUNCTION DESCRIPTION:

This call initiates the execution of the lead task of a task group previously created by a create group macro call. If the task group is dormant at the time the request group macro call is issued, execution begins immediately. If the task group has been activated by a previous request group function and has not yet terminated, execution of this request group macro call begins when the group becomes dormant.

Execution begins with the lead task specified in the create group macro call. The second argument of the request group macro call provides an argument list to be used to specialize a request block that is, in turn, used to request the lead task. (This request block is built in space taken from the memory pool of the requested group.)

It is not possible to wait on the execution of a request group macro call.

- NOTES:
1. The group id supplied by argument 1 is placed in \$R2; if argument 2 is omitted, \$R2 is assumed to contain the group id to be used.
  2. The address of the argument list supplied by argument 2 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the list.
  3. The address of the fixed parameter block supplied by argument 3 is placed in \$B5; if this argument is omitted, \$B5 is assumed to contain the address of the fixed parameter block to be used.
  4. On return, \$R1 contains one of the following status codes:
    - 0000 - No error
    - 0601 - Insufficient memory
    - 0602 - Insufficient memory
    - 0806 - Group id not currently defined
    - 160B - Insufficient memory

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: SRQIO

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] SRQIO [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 [valid]).

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.



- NOTES: 1. The address of the IORB supplied by argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the IORB to be used.
2. On return, \$R1 and \$B4 contain the following information:

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

0000 - No error

0801 - IORB in use (t-bit on)

0802 - Invalid LRN

0803 - Illegal wait or the R/S/D bit in the IORB is nonzero

If the IORB specifies that the issuing task is to wait for the completion of the request, one of the following codes could be returned:

0104 - Invalid arguments

0105 - Device not ready

0106 - Device timeout

0107 - Hardware error (check IORB status word)

0108 - Device disabled

0109 - File mark encountered

010A - Controller unavailable

010B - Device unavailable

010C - Inconsistent request

010D - Magnetic tape EOT marker (reflective strip) detected

0817 - Memory access violation

\$B4 - Address of IORB

Example:

In this example, the \$RQIO macro call is used to request an I/O transfer involving a device whose logical resource number is 143. The device has been reserved by a get file macro call; its LRN has been obtained by a get file information macro call. In addition to the LRN, the IORB provides the following information about the I/O transfer:

- o The issuing task is to be suspended until the request is complete.
- o The address of the buffer to be used in the I/O transfer is BUFAD.
- o The buffer begins in the left byte of BUFAD.
- o The buffer is 326 bytes long.

```
AF001    $RQIO    !IORB21
          .
          .
          .
IORB21   $IORB    143, WAIT, , BUFAD, L, 326
```

# REQUEST SEMAPHORE

## REQUEST SEMAPHORE

Macro Call Name: \$RQSM

Function Code: 06/00

Equivalent Command: None

Request the reservation of a resource controlled by the semaphore specified in the indicated semaphore request block (SRB). If it is available, reserve the resource. If the resource is not available, queue the SRB until the resource becomes available.

### FORMAT:

[label] \$RQSM [location of SRB address]

### ARGUMENT DESCRIPTION:

location of SRB address

Any address form valid for an address register; provides the address of the semaphore request block to be queued if the resource is not available. See the semaphore request block (\$SRB) macro call later in this section.

### FUNCTION DESCRIPTION:

This call is an asynchronous request for a resource controlled by the semaphore identified in the semaphore request block (SRB). The semaphore itself must have been defined by a define semaphore macro call. The semaphore request block can be generated by a \$SRB macro call.

When the request semaphore macro call is executed, the counter whose initial value was established by the define semaphore macro call is decremented by 1.

If the resource is available, it is reserved. If the resource is not available, the SRB is queued until the resource becomes available.

If WAIT was specified in argument 2 of the \$SRB macro call, the issuing task is suspended until the resource becomes available. The resource is then reserved, the SRB is marked as terminated, and control is returned to the issuing task.

If argument 2 of the \$SRB macro call is not WAIT, control is immediately returned to the issuing task, which can then perform other processing. When the resource becomes available, it is reserved and the SRB is marked as terminated. The issuing task can then use the test, wait, or wait on request list macro calls to check the completion status of the SRB. (Alternatively, the task can use the request-task or post-semaphore termination options.)

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

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

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

0000 - No error  
0502 - Illegal SRB

\$B4 - Address of SRB

Example:

In this example the \$RQSM and \$WAIT macro calls are used to replace the P-op on semaphore TH used in the example given for the define semaphore macro call. This technique allows the requesting task to start the process of reserving a resource before it is actually needed and continuing concurrent processing until the resource is required (at which time the requesting task will wait for the semaphore request block). Processing then continues as in the define semaphore example.

```

*
* START THE PROCESS OF CAPTURING A RESOURCE BY ISSUING
* A REQUEST SEMAPHORE CALL TO RESERVE A RESOURCE
*
*           $RQSM    !SRB
*
* NOW CONTINUE NORMAL PROCESSING
*
*           .
*           .
*           .
*
* ROUTINE TO FINISH GETTING A RESOURCE
*
* WAIT FOR THE REQUEST SEMAPHORE CALL TO FINISH
*
*           $WAIT    !SRB
*
* NOW LOCK THE FREE RESOURCE LIST
*
*           $RSVSM   ='LK'
*
* NOW TAKE A RESOURCE FROM THE FREE RESOURCE LIST
*
*           .
*           .
*           .
*
* THEN UNLOCK THE FREE RESOURCE LIST
*
*           $RLSM    ='LK'
*
* NOW THE RESOURCE IS RESERVED
*
*           .
*           .
*           .
SRB           $SRB    TH, WAIT

```

# REQUEST TASK

## REQUEST TASK

Macro Call Name: \$RQTSK

Function Code: 0C/00

Equivalent Command: Enter Task Request (ETR)

Request the execution of a previously created task within the same task group from which this request is issued.

### FORMAT:

[label] \$RQTSK [location of request block address]

### ARGUMENT DESCRIPTION:

location of request block address

Any address form valid for an address register; provides the address of the task request block that identifies the requested task and specifies whether the issuing task is to wait for the completion of the request.

### FUNCTION DESCRIPTION:

This call activates a task that was previously defined by a create task macro call. The request task macro call allows a running task to request the execution of another task. The issuing task must supply a task request block that identifies the requested task and the characteristics of the request.

A task request block is constructed through the \$TRB macro call. The first argument of the \$TRB macro call specifies the logical resource number (LRN) of the requested task. The second and third arguments specify whether or not the issuing task is to be suspended until the request is complete. The fourth argument specifies the start address of the task.

Using the LRN supplied in the request block, the task manager ascertains the task control block of the requested task. The task manager then places the request block in the request queue of the requested task. If the request queue was previously empty, the task is queued to its priority level. If the priority level was empty, it is activated. In addition, if the newly activated priority level is higher than that of the calling task, the task manager (operating at the priority level of the calling task) is interrupted and the requested task begins execution.

When the priority level of the calling task again becomes the highest active priority level, the task manager checks the task request block to ascertain if the calling task is to wait for the completion of this request (for the requested task) before continuing. If the calling task is to wait (and the requested task has not already signaled its completion relative to the request), the task manager associates the identity of the calling (and now waiting) task with the request block for the requested task. The task manager then removes the calling task from its priority level and activates the next task in the queue. If the calling task is not to wait for completion of this request for the requested task, the task manager returns control to the calling task. \*

The calling task can explicitly supply the address of the requested task's entry point in the request block it uses. If it does not, the requested task's entry point, derived when the task was created or last terminated, is used.

When a requested task is entered, the task manager provides the address of the request block that is being honored. This address is that of the first request block in the request queue for the priority level of the requested task.

If a calling task waits for the completion of its request for a requested task, the task manager returns the completion status of the request to the calling task when the latter regains control. (See also the wait and wait on request list macro calls.)

- NOTES:
1. The address supplied by argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the task request block for the task.
  2. On return, \$R1 and \$B4 contain the following information:

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

0000 - No error

0801 - Request block in use

0802 - Invalid LRN used in request block

0803 - Illegal wait (a task cannot wait on a request for itself)

If wait specified:

0000-FFFF - Completion status

\$B4 - Address of task request block

Example:

In this example, the \$RQTSK macro call is used to request the execution of the task created in the first example for the create task macro call (assuming that both macro calls are executed in the same task group). The task request block used is generated so that the issuing task will not be suspended awaiting completion of the requested task, the semaphore named TD will be V-oped at completion of the requested task, and the requested task will be started at entry point ENTRY3 instead of the address specified when the task was created. The task request block is also to contain the argument -PRINT, and by default will contain no additional space for use by the requested task.

```
          $RQTSK    !TRB
          .
          .
          .
TRB      $TRB      10,NWAIT,SM=TD;
          ENTRY3,, -PRINT
```



# REQUEST TERMINAL

## REQUEST TERMINAL

Macro Call Name: \$RQTML

Function Code: 17/03

Equivalent Command: None

Permit the issuing task group to accept a user who is logging into that task group through the listener component.

### FORMAT:

[label] \$RQTML [location of terminal IORB]

### ARGUMENT DESCRIPTION:

location of terminal IORB

Any address form valid for a data register; provides the address of the input/output request block (IORB) of the terminal associated with this task group.

### FUNCTION DESCRIPTION:

This call enables the task group of the issuing task to be notified when a terminal user logs in as a secondary user of the task group. Notification is made at the terminal associated with the task group issuing the call.

If a secondary user logs in after this call has been issued, the terminal from which the user logs in is reserved by the task group issuing the call. The issuing task group can use the \$RLTML macro call to release the terminal. The issuing task group can cancel the request by a \$CANRQ macro call.

The buffer address field of the terminal IORB specifies an area that is to receive some or all of the login parameters in the format specified below. (The actual amount of data transferred is determined by the IORB buffer range field.)

<u>Word(s)</u>	<u>Contents</u>
0	Terminal LRN
1-6	Terminal symbolic peripheral device name (e.g., TTY0)
7-12	Person identification from login line or the default account
13-18	Account name for login line or the default account
19-22	Encrypted password from login sequence (networking use only)
23-xx	Entire login line as entered from terminal

The setting of the IROBs W-bit determines whether control is returned immediately or is returned after a login has occurred.

The IROB's I/O bit must be set; the D-bit is reset. The S- and R-bits specify how the task group is to be notified when the request is satisfied. The requesting task group must issue a get file macro call to the terminal file to reserve the file.

- NOTES:
1. The address of the terminal IORB supplied by argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the current address.
  2. On return, \$R1 contains one of the following return status codes:
    - 0000 - If no wait specified, request was issued successfully; if wait specified, successful login
    - 0817 - Memory access violation
    - 0824 - Request canceled
    - 082E - Parameter error (invalid control bits in IORB)
  3. On return, \$B4 contains the request block address.
  4. This macro call modifies item I\_CT2 of the IORB.

Example:

In this example, the \$RQTML macro call is used to ensure that the issuing task group is notified when a terminal user logs in as a secondary user of the task group. The information returned to the task group consists only of the terminal LRN, terminal symbolic peripheral device name, person identification, and account name. Note that control is returned immediately to the issuing task group; the group does not wait for a login to occur.

```
CHK_1      $RQTML      !IORB
*
*      DEFINE IORB
*
IORB       RESV        $AF,0          RSU
          TEXT        Z'00';         RETURN STATUS
          B'0';        T-BIT (IN USE)
          B'1';        W-BIT (DON'T WAIT)
          B'0';        U-BIT (USER)
          Z'0';        MBZ
          B'1';        MUST BE ONE
          TEXT        Z'03';         LRN
          B'0';        MBZ
          B'0';        B-BIT (BYTE INDEX)
          B'00';       MBZ
          Z'1';        FUNCTION CODE
          DC          <SEC_USR       BUFFER ADDRESS
          DC          IN_LNG         RANGE
          DC          0
          DC          0              RESIDUAL RANGE
          DC          0              STATUS WORD
*
*      END IORB
*
SEC_USR    RESV        18,0
IN_LNG     EQU         2*($-SEC_USR)
*
```

# RESERVE SEMAPHORE

## RESERVE SEMAPHORE

Macro Call Name: \$RSVSM

Function Code: 06/02

Equivalent Command: None

Reserve a resource controlled by the specified semaphore if the resource is available (i.e., do a P-op or P-test). If the resource is not available, perform one of the following actions, depending on the value of argument 2:

- o Return immediately to the issuing task (do a P-test).
- o Suspend the issuing task until the resource becomes available. Then reserve the resource and return to the issuing task (these three actions are known collectively as a P-op).

### FORMAT:

[label] \$RSVSM [location of semaphore identifier], [ {DENY} ]  
[ {WAIT} ]

### ARGUMENT DESCRIPTION:

location of semaphore identifier

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

### DENY

Specifies that if the resource is not available for reservation, an immediate return to the issuing task is to be made (i.e., a P-test is to be done).

## WAIT

Specifies that if the resource is not available for reservation, the issuing task is to be suspended until the resource becomes available; then the resource is to be reserved and a return to the issuing program is to be made (i.e., a P-op is to be done).

WAIT is assumed if the argument is omitted.

### FUNCTION DESCRIPTION:

This call is a synchronous request for a resource controlled by the semaphore identified in argument 1. This semaphore must have been defined by a define semaphore macro call.

When a P-op is performed, the counter whose initial value was established by the define semaphore macro call is decremented by 1.

Since the reserve function does not queue a semaphore request block (see request semaphore macro call), the reserve resource macro call must be reissued when DENY is specified for argument 2.

- NOTES:
1. The semaphore identifier supplied by argument 1 is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the identifier of the semaphore to be tested.
  2. If DENY was specified for argument 2, \$R2 is set to 0 (P-test to be done); if WAIT is specified for argument 2, or if the argument is omitted, \$R2 is set to -1 (P-op to be done).
  3. On return, \$R1 and \$R6 contain the following information:

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

0000 - No error

0501 - Unsuccessful reservation (only if DENY specified)

0502 - Semaphore not defined

0505 - Illegal return condition indicator

\$R6 - Semaphore identifier (as supplied)

Example:

For an example of the reserve semaphore (\$RSVSM) macro call see the example given for the define semaphore macro call.

# RESET DEVICE ATTENTION

## RESET DEVICE ATTENTION (MOD 400 ONLY)

Macro Call Name: \$RDVAT

Function Code: 02/03

Equivalent Command: None

Turn off the attention status indicator in the resource control table (RCT) of the specified device.

### FORMAT:

[label] \$RDVAT [location of LRN]

### ARGUMENT DESCRIPTION:

location of LRN

Any address form valid for a data register; provides the LRN of the device whose attention status indicator is to be reset. The LRN must be a system LRN (defined at system building).

### FUNCTION DESCRIPTION:

This call turns off the attention status indicator (bit 8 of the R\_FLGS entry) in the RCT of the specified device. \$RDVAT can be used in synchronizing task operation with device availability.

A task can issue a disable device on attention macro call (\$DSDV) to request notification of an interrupt. When the interrupt occurs, the device driver will set bit 10 (device disabled) and bit 8 (attention has occurred) of R\_FLGS. When a ready interrupt is generated, the task can clear the disabled status by resetting bit 10 through the \$ENDV macro call.

After clearing bit 8, using the reset device attention (\$RDVAT) macro call, and waiting for the device ready interrupt to occur, a task can use the enable device (\$ENDV) and the reset device attention (\$RDVAT) macro calls to clear bits 8 and 10 to initial states.

NOTES: 1. The LRN specified by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the correct LRN.

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

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

0000 - No error  
0102 - Invalid Error

\$R2 - LRN of the device

Example:

In this example, the \$RDVAT macro call is used to turn off the attention status indicator in the RCT of the device whose LRN is 15. If the interrupt occurs, the device will be set to the disabled state. When a ready interrupt occurs, the device disabled condition is cleared through the \$ENDV (enable device) macro call.

```
ATTOFF  $RDVAT  =15
        .
        .
        .
CLRDIS  $ENDV   =15
```



# RETURN

## RETURN

Macro Call Name: \$RETRN

Function Code: None

Equivalent Command: None

Issues a standard return sequence for tasks or called subroutines.

### FORMAT:

```
[label] $RETRN [location of completion status],  
               [location of return address]
```

### ARGUMENT DESCRIPTION:

location of completion status

Any address form valid for a data register; provides the user-selected status code to be returned when the subroutine or system service routine finishes processing. Any code can be selected.

location of return address

Any address form valid for an address register; provides the address in the calling task to which the subroutine or system service routine will return when it has finished processing.

### FUNCTION DESCRIPTION:

This macro call allows a procedure (which can be called as a subroutine or invoked to service a task request) to have a common return interface to the calling task.

If the procedure was statically linked with its caller, the return address supplied in argument 2 is placed in \$B5 and a JMP \$B5 instruction is issued. The completion status is placed in \$R1.

If the procedure was invoked as a subtask, the procedure's task is terminated and its request block is marked as complete. (See "Terminate Request" for further information about task termination.)

Note that \$B5 is set to the address of a system-supplied termination routine when any of the following occur:

- o A task is initially activated to service a request.
- o A return request block macro call is issued.

- NOTES:
1. The status code specified by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the intended status code.
  2. The address supplied by argument 2 is placed in \$B5 and a JMP \$B5 instruction is executed. If this argument is omitted, \$B5 is assumed to contain the return address.

Example:

In this example, the RETRN macro call is used by a semaphore to return to its caller with a completion status of zero. The example assumes the procedure was entered at the entry pointed named BEGIN, and that the contents of SAV\_B5 are not altered within the procedure - other than at its entry point. If the procedure was statically linked with its caller, the macro call causes a JMP \$B5 return to the caller, with the completion status in \$R1. If the procedure was invoked as a subtask, the macro call causes the procedure's task to be terminated and its request block marked as complete.

	EDEF	BEGIN
BEGIN	STB	\$B5, SAV_B5
	.	
	.	
	\$RETRN	=0, SAV_B5
	.	
	.	
SAV_B5	RESV	2

# RETURN MEMORY/RETURN PARTIAL BLOCK OF MEMORY

## RETURN MEMORY/RETURN PARTIAL BLOCK OF MEMORY

Macro Call Name: \$RMEM

Function Code: 04/04 (return memory),  
04/05 (return partial block)

Equivalent Comand: None

Return all or part of the previously allocated memory block to the memory pool of the task group of the issuing task. If argument 2 is omitted, return all of the memory block; if argument 2 is specified, return the number of words it indicates.

### FORMAT:

[label] \$RMEM [location of memory block address],  
[location of number of words to be returned]

### ARGUMENT DESCRIPTION:

location of memory block address

Any address form valid for an address register; provides the location of the address of the leftmost word (excluding the block header) of the memory block to be returned (either partially or totally).

location of number of words to be returned

Any address form valid for a data register; provides the number of words to be returned (starting at the rightmost part of the block). If this parameter is omitted, the entire memory block is returned.

### FUNCTION DESCRIPTION:

The return memory and return partial block of memory macro calls are the means by which a task returns a previously allocated memory block to the task group's memory area. If the entire block is to be returned, argument 2 is omitted. If a part of the block is to be returned, argument 2 specifies the number of words to be returned.

When a partial block of memory is returned, the return is done in 32-word increments of memory; the actual amount of memory returned is the specified amount rounded down to the next lower 32-word increment.

The memory block address referred to by argument 1 is the same address as that returned in \$B4 when the task issued a get memory or get available memory macro call and was allocated this block.

- NOTES:
1. The memory block address derived from argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the memory block to be returned.
  2. The number of words to be returned (partial return only) derived from argument 2 is placed in \$R6 and \$R7. If argument 2 is =\$R7, it is assumed that \$R6 and \$R7 contain the number of words to be returned. If argument 2 is omitted, the entire memory block is returned.
  3. On return, \$R1, \$R6, \$R7, and \$B4 contain the following information:

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

0000 - No error

0603 - Illegal memory block address specified

0604 - Size of memory to be returned is greater than size of memory block (partial return only)

0818 - No task group with specified group id exists (system software error)

081B - Roll-in of online task group attempted (system software error)

081C - Roll-in attempted when batch group not rolled out (system software error)

081E - Unrecoverable media error during roll-in

081F - Group not suspended when roll-in attempted (system software error)

\$R6, \$R7 - Partial return only; remaining size  
of block still allocated

\$B4 - Partial return only; address of first  
(leftmost) word of allocated memory block  
(excluding header word).

Examples:

In this example, the \$RMEM macro call is used to return all of the memory obtained in the first example for the get memory/get available memory macro calls. The \$RMEM macro call is assumed to be contained in the same procedure as the coding shown in that example.

```
$RMEM M_PTR
```

In this example, the \$RMEM macro call is used to return 100 words of the memory obtained in the first example for the get memory/get available memory macro calls. Upon return from the system \$B4 will contain the address of the first usable word of the memory area and \$R6 and \$R7 will specify the number of words still remaining in the memory area. The \$RMEM macro call is assumed to be contained in the same procedure as the coding shown in the get memory example.

```
$RMEM M_PTR,=100
```

# RETURN REQUEST BLOCK ADDRESS

## RETURN REQUEST BLOCK ADDRESS

Macro Call Name: \$RBADD

Function Code: 01/07

Equivalent Command: None

Return the address of the request block currently at the head (top) of the issuing task's request queue.

FORMAT:

[label] \$RBADD

ARGUMENT DESCRIPTION:

No arguments are used with this call.

FUNCTION DESCRIPTION:

This call returns the address of the first request block in the request queue for the task. The request block is not removed or altered.

The address of the request block is placed in \$B4. The address of the argument list (if any) associated with the request block is placed in \$B7.

Upon return to the issuing task, \$B5 contains the address of the system-supplied termination routine.

NOTE: On return, \$R1, \$B4, \$B5, and \$B7 contain the following information:

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

0000 - No error

0801 - No request block found (no dispatched request block exists for the issuing task)

\$B4 - Address of current request block (if \$R1 is  
0000)

\$B5 - Address of system-supplied termination routine

\$B7 - Address of request block argument list (if \$R1  
is 0000)

Example:

In this example, the \$RBADD macro call returns the address of the issuing task's request block in \$B4. The address of the argument list contained within the request block is returned in \$B7 and the address of a system-supplied termination routine is returned in \$B5.

```
CHEK_L    $RBADD
```

# REWRITE RECORD

## REWRITE RECORD

Macro Call Name: \$RWREC

Function Code: 11/40 (current), 11/41 (key)

Equivalent Command: None

Change the contents of the specified logical record in the file; this macro call is valid for all file organizations except tape-resident sequential files and device files.

### FORMAT:

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

### ARGUMENT DESCRIPTION:

fib address

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

{CURRENT}  
{CUR}

This mode argument indicates that the record read by the immediately preceding read next or read with key (i.e., the last record read; see "Read Record") macro call is to be rewritten by the record defined in the FIB. This is the default value for this macro call. You must code the following FIB entries:

```
logical file number  
user record pointer  
output record length
```

This mode is referred to as rewrite current record.



## KEY

This mode argument indicates that the position in the file associated with the key value specified in the FIB is to be written over by the record identified by the FIB. You must code the following FIB entries:

logical file number

input key pointer (unless this is an indexed file that contains the key embedded in the logical record)

input key length

This mode is referred to as rewrite with key.

## 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 rewrite operations (bit 3 is 1). The file must have been reserved (see the get file call) with write access concurrency control (type 3, 4, or 5). The rewrite record macro call has no effect on the read or write pointer. If the file is an indexed file, the embedded key must not be altered.

The file information block can be generated by a \$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 this 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  
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  
021D - Attempt to change the symbolic key value

021E - Key length or location error  
022A - Record lock area overflow or not defined  
022B - Requested record is locked

In addition to the above codes, any system service codes received by the data manager are passed on through \$R1.

Example:

In this example, it is assumed that the file is reserved with write access concurrency control and opened. The FIB identified in the first parameter is defined in "Assumptions for File System Examples" in Section 3. The macro call is specified as follows:

```
BACREC $RWREC !MYFIB,CURRENT
```

# 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 Program Preparation 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 (MOD 400 ONLY)

Macro Call Name: \$SRBD

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

Generated Label Prefixes:

SRB label	S_RRB/S_SEM
	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 Program Preparation manual for more information about SAF/LAF independent code.

# SET DIAL

## 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 autodial telephone number list for the specified line. This telephone number will be used first when the autodial 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.)

{ CHANNEL }  
{ CHAN }

Indicates 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.)

{ PATHNAME }  
{ PATH }

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 autodial facility 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 autodial 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 - Autodial 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.

\$R2 is the mask word. Each bit in \$R2 that is 1 causes the corresponding bit in the external switch word to be set on; each bit that is 0 causes the corresponding bit to remain unchanged.

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

- o If bit 11 is 0, no switch set on had previously been set on.
- o If bit 11 is 1, at least one switch of those set on had previously been set on.

NOTES: 1. The bits corresponding to the external switches in the arguments are set on in \$R2; if no arguments are supplied, \$R2 is assumed to contain the mask to be used. If ALL is specified, all bits are set on in \$R2.

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

\$R2 - External switch word after modification

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

0 - No switch set on was on

1 - At least one switch of those set on was on

Example:

In this example, the \$SETSW macro call is used to turn on external switches 2, 4, and B of the task group in which the issuing task is executing.

```
SET_AA $SETSW 2,4,B
```

# SET TERMINAL CHARACTERISTICS

## SET TERMINAL CHARACTERISTICS

Macro Call Name: \$STTY

Function Code: 10/45

Equivalent Command: Set Terminal Characteristics (STTY)

Set the file characteristics of a terminal.

### FORMAT:

[label] \$STTY [location of parameter structure address]

### ARGUMENT DESCRIPTION:

location of parameter structure address

Any address form valid for an address register; the format of the parameter structure is:

<u>Word</u>	<u>Field</u>	<u>Content</u>
0-2	sympd	Symbolic peripheral device name of the terminal to be affected.
3	line length	Binary integer specifying the line length (not including the slew byte) of the terminal. If this word is zero, the terminal's line length is not changed.
4	device-specific word	An aligned 16-bit string that specifies the device specific word. If all zero bits, the terminal device-specific word is not changed. (See the <u>Communications Processing manual for possible values.</u> )

\*

<u>Word</u>	<u>Field</u>	<u>Content</u>
5	file indicator	An aligned 16-bit string that specifies the characteristics of the terminal to be changed (0 means do not change the characteristic; 1 means change the characteristic).

<u>Bit</u>	<u>Meaning</u>
0	Input-only device type
1	Output-only device type
2	Bidirectional device type
3	Detab on
4	Detab off
5	Asynchronous input
6	Asynchronous output
7	Synchronous input
8	Synchronous output
9	Synchronous nonbuffered input
10	Synchronous nonbuffered output
11-15	Must be zero

The device type (bits 0, 1, and 2) cannot be changed if the file is open.

6-16 Reserved for later use

#### FUNCTION DESCRIPTION:

This call allows the issuing task to dynamically alter terminal file characteristics. The original file characteristics, established at system generation, can be altered to reflect the needs of the issuing task.

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

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

0000 - No error  
1302 - Device name not found  
1304 - Parameter missing or incomplete  
1306 - Parameter not consistent with device type

Example:

In this example, the terminal whose symbolic peripheral device name is TTY4 is changed to an output-only device for asynchronous output. Neither the line length nor the device-specific word is changed. (It is assumed that the file is not open.)

```
TER_AA  DC      'TTY4ΔΔ'  
        RESV    2,0  
        DC      B'0100001000000000'  
        .  
        .  
        .  
SETAA   $STTY  !TER_AA  
        (10)Z'0000'
```

# SPAWN GROUP

## SPAWN GROUP

Macro Call Name: \$SPGRP

Function Code: 0D/05

Equivalent Command: Spawn Group (SG)

Create the definition of a new task group within the system. Request the execution of the group's lead task. Delete the group from the system when the group request terminates.

### FORMAT:

```
[label] $SPGRP [location of group identifier],  
               [location of address of argument list],  
               [location of address of fixed parameter block],  
               [location of memory pool 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 task group to be spawned. The group identification must be a two-character (ASCII) name that does not have the \$ character as its first character.

location of address of argument list

Any address form valid for an address register; provides the address of the argument list, which can be generated by the parameter block macro call to be used to specialize a request block that will be used to request the lead task of the spawned task group.

## location of address of fixed parameter block

Any address form valid for an address register; provides the address of a fixed parameter block, which can be generated by the parameter block macro call. This parameter block has the following arguments:

### Argument 1

A string specifying the user id to be associated with the spawned task group (for system use). If this entry is zero, the user id currently associated with the issuing task group will be used when the call is executed from a user task group.

### Argument 2

A pathname string specifying the command-in and initial user-in files for this request of the lead task of the spawned task group. If this entry is zero, no command-in and initial user-in files will be available to the spawned group. However, the spawned group can later obtain a user-in file by means of the new user input macro call. A nonzero entry is required if the command processor is the lead task.

### Argument 3

A pathname string specifying the error-out and initial user-out files of the spawned task group. If this entry is zero, one of the following assumptions is made when the call is executed:

- o If the pathname string specifying the command-in and initial user-in files (in-path) specifies a disk device, the pathname for the output files is in-path.A0.
- o If in-path specifies an interactive terminal, the pathname for the output files is the same as in-path.
- o If in-path specifies an input-only device, the pathname for the output files is null.

### Argument 4

A pathname string specifying the initial value of the working directory to be used by the spawned task group.



#### location of memory pool identifier

Any address form valid for a data register; provides the identifier of the memory pool to be used to service all memory requests emanating from the spawned task group. The memory pool identifier consists of two ASCII characters that name a pool defined at system generation. If this argument is omitted, the spawned task group will use the memory pool associated with the issuing task group.

#### location of base level

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

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

#### location of high logical resource number

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

#### location of high logical file number

Any address form valid for a data register; specifies the highest logical file number (LFN) to be used by any task in the spawned task group. The LFN can be a value from 0 through FF (hexadecimal). If this argument is omitted, the value 15 is assumed. (Refer to the associate file macro call.)

## location of root entry name address

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

### FUNCTION DESCRIPTION:

This call combines the create group, enter group request, and delete group macro calls. Spawn group implicitly causes the execution of these calls in sequence (i.e., it (1) allocates and creates the data structures required to define and control the execution of the task group, (2) places a request against the group, thereby activating it and, (3) when execution terminates, removes all controlling data structures and returns memory used by the task group to the appropriate memory pool).

Spawned task groups cannot be requested, nor can they be waited upon.

The request block generated according to the second argument in the macro call is constructed in space taken from the memory pool of the spawned task group.

A spawn group macro call can be issued from a task group that was itself spawned.

- NOTES:
1. The group identifier specified by argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the group id to be used.
  2. The address of the argument list supplied by argument 2 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the argument list to be used to build the request block.
  3. The address of the fixed parameter block supplied by argument 3 is placed in \$B5; if this argument is omitted, \$B5 is assumed to contain the address of the block to be used.

4. The memory pool identifier specified by argument 4 is placed in \$R4; if this argument is omitted, \$R4 is set to zero to indicate that the memory pool of the issuing task group is to be used by the spawned task group.
5. The base priority level specified by argument 5 is placed in \$R5; if this argument is omitted, \$R5 is assumed to contain the base priority level to be used.
6. The high LRN value specified by argument 6 is placed in \$R6; if argument 6 is omitted, \$R6 is set to zero to indicate that the value of the highest LRN created for the system task group will be used.
7. The high LFN value specified by argument 7 is placed in \$R7; if this argument is omitted, \$R7 is set to 15.
8. The address of the root entry name supplied by argument 8 is placed in \$B2; if this argument is omitted, \$B2 is assumed to contain the address of the root entry name of the bound unit to be executed by the lead task of the spawned group.
9. On return, \$R1 and \$R2 contain the following information:

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

- 0000 - No error
- 0601 - Insufficient memory
- 0602 - Insufficient memory
- 0804 - Group id in use
- 0806 - Invalid group id
- 0807 - Invalid memory pool identifier
- 0808 - Invalid base level
- 0809 - Invalid high LRN
- 080A - Invalid high LFN
- 080C - Unresolved start address
- 160A - Invalid pathname
- 160B - Insufficient memory

\$R2 - Group id of spawned task group

Example:

In this example, the SSPGRP macro call is used to create a task group, execute a task in that group, and then delete the group. The task group is created with a group id of Q2, use of memory pool P2, and a base level of 40 (decimal). Both the high LRN and the high LFN are defaulted (only system logical resources will be available and the highest logical file number available will be 15, decimal). The task group's lead task will be the command processor. The request part of the spawn is the same as the request given in the example for the request group macro call.

```
        SSPGRP    ='Q2',!ARGS,!INFO;  
                ='P2',=40,,,!ROOT  
        .  
        .  
        .  
INFO    $PRBLK    ,^V1124>UDD>TEST>JONES>ASM_TST;  
                ^V1124>UDD>TEST>JONES>L>ASM_TST.AC;  
                ^V1124>UDD>TEST>JONES  
ARGS    $PRBLK    -XREF,-PRINT  
ROOT    TEXT      'EC?ZXECLΔ'
```

# SPAWN TASK

## SPAWN TASK

Macro Call Name: \$SPTSK

Function Code: 0C/05 (different bound unit),  
0C/06 (same bound unit)

Equivalent Command: Spawn Task (ST)

Create, request the execution of, and then cause a task to be deleted within the task group of the issuing task.

### FORMAT:

```
[label] $SPTSK [location of task request block address],  
              [location of relative priority level],  
              [location of start address],  
              [location of root entry name address]
```

### ARGUMENT DESCRIPTION:

location of task request block address

Any address form valid for an address register; provides the location of the address of the request block for the spawned task. The request block indicates whether the issuing task is to wait for the execution of the spawned task; the request block may contain parameters to be passed to the spawned task.

location of relative priority level

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

#### location of start address

Any address form valid for an address register; provides the location of the task start address to be used when the spawned task is to execute the same bound unit as the issuing task. Note that the bound unit must have been declared sharable at the time it was linked. (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)
  - 0lxx - Media error
  - 0209 - Bound unit not found
  - 0602 - Insufficient memory
  - 0801 - Request block in use (t-bit on)
  - 0817 - Access violation on request block
  - 0827 - Bound unit is not a fixed-relative file
  - 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).

Example:

In this example, the \$SPTSK macro call creates a task, requests its execution, and then deletes the task. The task creation part of the spawn is the same as that given in the first example for the create task macro call, except that there is no LRN. The request part of the spawn is the same as that given in the example for the request task macro call, except that a synchronous request is made instead of an asynchronous request and no semaphore is V-oped (see "Semaphore Functions" in Section 2). The delete part of the spawn is the same as given in the example for the delete task macro call.

```
                $SPTSK    !TRB,=2,,!ROOT
                .
                .
                .
TRB             $TRB     ,,,ENTRY.3,,-PRINT
ROOT           TEXT     'PROG10 Δ'
```



# STATUS MEMORY POOL

## STATUS MEMORY POOL

Macro Call Name: \$STMP

Function Code: 04/06

Equivalent Command: None

Determine the amount of memory available in a specified memory pool.

### FORMAT:

[label] \$STMP [location of memory pool id]

### ARGUMENT DESCRIPTION:

location of memory pool id

Any address form valid for a data register; provides the memory pool id of the memory pool to be examined. If this argument is omitted, the memory pool examined is that associated with the task group of the issuing task.

### FUNCTION DESCRIPTION:

This call allows the issuing task to determine the amount of memory currently available in a specified memory pool. The amount of available memory is returned to the issuing task both as the actual number of words now available in the pool and as the percentage of the pool's total memory now available. The total available memory may not be contiguous.

If the memory pool being examined has the preempt batch option, the statistics returned are for the specified memory pool combined with the batch task group's memory pool.

NOTES: 1. The memory pool id of the memory pool to be examined, supplied by argument 1, is placed in \$R2; if this argument is omitted, \$R2 is set to -1 to indicate that the memory pool of the task group of the issuing task is to be examined.

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

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

0000 - No error

0606 - 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    >+SA
                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:

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.

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

If the group id argument is \$B, the \$\$SUSPG macro call forces the rollout of the current batch task group. Rollin cannot occur until the \$ACTVG \$B macro call is issued.

If the suspended task group is aborted, or if no other task group issues a \$ACTVG macro call to enable the suspended group, the operator must issue an ACTB or ACTG command to allow the suspended group to continue.

NOTES: 1. The group id of the task group to be suspended, supplied by argument 1, is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the correct group id.

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

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

0000 - No error

0806 - Specified group id not currently defined

\$R2 - Group id as supplied

Example:

In this example, the \$SUSPG macro call is used to suspend the task group whose group id is G1. Task group G1 will not be suspended until all its tasks have exited from critical areas of the Monitor and all external requests have been satisfied.

SUSGAA \$SUSPG =G1

# SUSPEND FOR INTERVAL

## SUSPEND FOR INTERVAL

Macro Call Name: \$SUSPN

Function Code: 05/02

Equivalent Command: None

Remove the issuing task from the active queue for its priority level until the specified interval has elapsed.

### FORMAT:

[label] \$SUSPN interval unit designator,  
[location of interval value]

### ARGUMENT DESCRIPTION:

interval unit designator

One of the following codes must be specified to indicate the manner in which the interval is to be measured.

<u>Code</u>	<u>Interval Measurement</u>
MS	Milliseconds
T	Tenths of a second
S	Seconds
M	Minutes
C	Clock resolution units

location of interval value

Provides the interval for which the issuing task is to be suspended; can be one of the following:

=\$R7

Interval value is in \$R6 and \$R7

=hexadecimal string

String specifies the interval value

fieldname

fieldname represents the first word of a 2-word field containing the interval value

FUNCTION DESCRIPTION:

This call causes the issuing task to be suspended for the period of time specified in the call arguments.

The suspend until time macro call (\$SUSPN, function code 05/03) also suspends the issuing task, but the suspension exists until a particular date/time is reached.

NOTES: 1. The interval unit designator supplied by argument 1 is placed in \$R2. The contents of \$R2 depend on the interval designator chosen, as follows:

<u>Interval Unit Designator</u>	<u>Contents of \$R2</u>
MS	1
T	2
S	3
M	4
C	5

2. The interval value supplied by argument 2 is placed in \$R6 and \$R7; if this argument is omitted, or is =\$R7, it is assumed that \$R6 and \$R7 contain the correct interval value.

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

0000 - Specified time has elapsed  
0401 - Invalid time interval specified

4. Periodic use of this call by central processor bound tasks will allow other tasks with the same hardware priority level to obtain CPU time more often.

Example:

In this example, the \$SUSPN macro call suspends the issuing task for one unit of time measured in units of clock resolution.

```
ASTPA  $SUSPN  C,=1
```

# SUSPEND UNTIL TIME

## SUSPEND UNTIL TIME

Macro Call Name: \$SUSPN

Function Code: 05/03

Equivalent Command: None

Remove the issuing task from the active queue for its priority level until the date/time specified in the call.

### FORMAT:

[label] \$SUSPN [TIME],  
[location of internal date/time value]

### ARGUMENT DESCRIPTION:

#### TIME

Optional keyword; explicitly notes that date/time value will be used to govern the suspension of the issuing task.

location of internal date/time value

Any address form valid for an address register; provides the address of a 3-word internal date/time value until which the task is to be suspended. The value is a binary count of milliseconds since January 1, 1901.

### FUNCTION DESCRIPTION:

This call causes the issuing task to be suspended until the date/time value indicated by argument 2 is reached.

The suspend for interval macro call (\$SUSPN, function code 05/02) also suspends the issuing task, but for a particular interval of time.

NOTES: 1. If argument 1 is omitted, date/time format is assumed.



2. The internal date/time value supplied by argument 2 is placed in \$R2, \$R6, and \$R7. If this argument is omitted, or is either =\$R2 or =\$R7, these registers are assumed to contain the correct internal date/time value.
3. On return, \$R1 contains one of the following status codes:
  - 0000 - Specified date/time has been reached
  - 0401 - Invalid internal date/time value

Example:

The get date/time macro call \$GDTM is used to get the current date/time (in internal format), leaving it in registers \$R2, \$R6, and \$R7. The convert to external date/time macro call (\$EXDTM) is then used to convert this internal format to an external format, replacing the date portion (first ten characters) of the field labeled TODAY. The convert to external time macro call (\$EXTIM) is then used to convert the internal format date/time to an external format, storing the hour of the day in the field labeled HOUR. The convert to internal date/time macro call (\$INDTM) converts the contents of the field TODAY back to internal format contained in \$R2, \$R6, and \$R7. The field HOUR is then compared to the constant 08. If HOUR is greater than or equal to 08, one day (86,400,000 milliseconds) is added to \$R2, \$R6, and \$R7. Thus \$R2, \$R6, and \$R7 now contain the internal format date/time value for the next time, either today or tomorrow, that 0800 hours will occur. The suspend until time macro call (\$SUSPN) then suspends the issuing task until the next time the clock reads 0800 hours. The addition of one day to \$R2, \$R6, and \$R7 is programmed assuming a central processor that has the add integer double (AID) instruction. (See the example given for the convert to internal date/time macro call for the same addition performed without the use of the AID instruction.)

```

*
*   GET THE CURRENT DATE/TIME VALUE.
*
*       $GDTM
*
*   CONVERT IT TO AN EXTERNAL FORMAT DATE.
*
*       $EXTDT    ,!TODAY,=10
*
*   CONVERT IT TO AN EXTERNAL FORMAT HOUR OF DAY.
*
*       $EXTIM    ,!HOUR,=2
*
*   NOW CONVERT THE EXTERNAL FORMAT DATE/TIME
*   BACK TO THE INTERNAL FORMAT.
*
*       $INDTM    !TODAY,,=15
*
*   IF 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)

```

# SYSTEM IDENTIFICATION

## SYSTEM IDENTIFICATION

Macro Call Name: \$SYSID

Function Code: 14/04

Equivalent Command: None

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.

2. On return, \$R1 contains following status code:

0000 - No error  
0817 - Memory access violation

Example:

In this example, the \$SYSID macro call is used to return the identification of the currently executing system to a field whose address is SYIDFL.

ASYSID	\$SYSID	!SYIDFL
	.	
	.	
	.	
SYIDFL	RESV	15,A'ΔΔ'

# TASK GROUP INPUT

## TASK GROUP INPUT

Macro Call Name: \$TGIN

Function Code: 14/0C

Equivalent Command: None

Returns the pathname of the initial command-in file of the calling task group.

### FORMAT:

[label] \$TGIN [location of task group input address]

### ARGUMENT DESCRIPTION:

location of task group input address

Any address form valid for an address register; provides the address of a 58-character, aligned, non-varying field into which the system will place the pathname.

### FUNCTION DESCRIPTION:

This macro call returns the pathname of the initial command-in file of the calling task group into a 58-character, aligned, nonvarying field whose address is provided by argument 1.

- NOTES:
1. When the argument is entered, the task group input address is loaded into \$B4. When the argument is omitted, \$B4 is assumed to contain the address of the receiving home directory field.
  2. On return, \$R1 and \$B4 contain the following information:

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

0000 - No error

0817 - Memory access violation

\$B4 - Address of the receiving task group

# 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],  
             [issuing task suspension option],  
             or  
             [termination option],  
             [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.

issuing task suspension option

One of the following values is specified to indicate whether the requesting task is to be suspended until the completion of the request:

WAIT

Suspend the issuing task until the request is complete (set w-bit to 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 option

One of the following values is specified to indicate the action to be taken upon the completion of the request.

SM=aa

\* Release (V-op) the semaphore identified by aa (two ASCII characters), when requested task is completed.

RB=label

\* Issue a request for the request block identified by label, when requested task is completed.

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

task start address

Any address form valid for an address register; provides the start address to be used when the requested task is turned on to service the request. If this argument is omitted, the implicit task start address is to be used (bit 15 of the T\_CTL word is set to 1; see Appendix A).



size of argument to request block

Value specifying the number of words in the added portion of the task request block. If this argument is omitted, the generated request block will be large enough to contain only the user arguments specified in the macro call. If no user arguments are specified, the request block will be generated to contain only the standard fixed format request block fields (arguments 1 through 4). If this argument is specified in addition to user arguments, an area is reserved following those arguments.

user argument 1 ... user argument n

Begins the optional, variable-sized area containing user arguments to be passed to the requested task in response to a spawn task or request task macro call or command. This variable portion of the task request block is built in the following standard format.

entry 1 - One-word count of number of argument pointers

entry 2 - Address of first argument length field

entry 3 - Address of second argument length field

·  
·  
·

entry n - Address of nth argument length field

entry z - Length (in bytes) of first argument (one word)

entry y - First argument value (of specified size)

entry x - Length (in bytes) of second argument (one word)

entry w - Second argument value (of specified size)

·  
·  
·

entry p - Length (in bytes) of nth argument (one word)

entry o - nth argument value (of specified size)

## 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).

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

### 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,X
```

# TASK REQUEST BLOCK OFFSETS

## TASK REQUEST BLOCK OFFSETS (MOD 400 ONLY)

Macro Call Name: \$TRBD

Generated Label Prefixes:

TRB label	T_RRB/T_SEM
	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 Program Preparation manual for more information about SAF/LAF independent code.

# TERMINATE REQUEST

## TERMINATE REQUEST

Macro Call Name: \$TRMRQ

Function Code: 01/04, 01/03

Equivalent Command: None

Terminate the current request being processed by the issuing task. End the current request for the execution of the task and mark its associated request block as terminated.

### FORMAT:

```
[label] $TRMRQ [location of completion status],  
              [location of new task error address]
```

### ARGUMENT DESCRIPTION:

location of completion status

Any address form valid for a data register; provides the user-selected status code that is to be returned when the current request and its associated request block are terminated. Completion status codes 0801, 0802, and 0803 should not be used; they will be indistinguishable from error codes with the same values.

location of new task start address

Any address form valid for an address register; provides the new task start address for the terminating task. This address will subsequently be requested by a request block that does not explicitly specify a start address.

### FUNCTION DESCRIPTION:

This call is used to end a request for the execution of a task. The \$TRMRQ function marks a current request block as terminated and removes it from the appropriate request queue.

If there are no other request blocks on the request queue affected by the terminate function, the task manager places the task in a dormant state. If there are one or more request blocks in the affected queue, the task manager immediately uses the next request block to begin execution of the task at the indicated start address. If the task is requested for deletion and there is no other request for it the task is deleted; if this is a spawned task, it is deleted.

The task manager will do one of the following:

1. Activate a task if that task is awaiting completion of the current request block being terminated.
2. Release (V-op) the semaphore indicated by the current request block.
3. Schedule the task request block indicated by the current request block being terminated.

If the terminating task will subsequently be requested by a request block that does not explicitly specify a task start address, the terminating task can specify the new task address through argument 2.

- NOTES:
1. The completion status code supplied through argument 1 is placed in \$R2; if this argument is omitted, \$R2 is assumed to contain the completion status code.
  2. If argument 2 contains the location of the new task start address, that address is placed in \$B4 and an MCL 01/04 is issued. If argument 2 specifies =\$B4, \$B4 is assumed to contain the new start address and an MCL 01/04 is issued. If argument 2 is omitted, \$B4 is not modified and an MCL 01/03 is issued (no new task start address).
  3. On return, \$B4, \$B5, and \$B7 contain the following information:
    - \$B4 - Address of request block for new request.
    - \$B5 - Address of system supplied termination routine.
    - \$B7 - Address of the request block parameter list.

Example:

In this example, the \$TRMRQ macro call labeled TRM\_NM terminates the issuing task with a completion status of zero without changing the task's start address. The \$TRMRQ macro call labeled TRM\_AB terminates the issuing task with a completion status of one and changes the task's start address to RETRY.

```
TRM_NM    $TRMRQ    =0
          .
          .
          .
TRM_AB    $TRMRQ    =1,!RETRY
```

# TEST COMPLETION STATUS

## TEST COMPLETION STATUS

Macro Call Name: \$TEST

Function Code: 01/02

Equivalent Command: None

Return the completion status of any type of specified request block (e.g., task, clock, I/O, or semaphore).

### FORMAT:

[label] \$TEST [location of request block address]

### ARGUMENT DESCRIPTION:

location of request block address

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

### FUNCTION DESCRIPTION:

This call permits a running task to ascertain whether a specified request block has been marked as terminated by another task. When the call is executed, control is returned to the issuing task with \$R1 containing a return status that shows whether the request block has been terminated and \$B4 containing the address of the tested request block.

The test macro call does not cause a wait for the request block to be terminated; that function is performed by the wait macro call.

A given request block can be tested by any number of tasks.

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

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

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

yyzz - Where yy can be 00, or 00 through EE for user status, or as defined for other yy values in the System Messages manual.

\$B4 - Address of tested request block.

Example:

In this example, the \$TEST macro call is used to determine the status of a task that was requested using a request block labeled TRB. If the requested task has not run to completion yet, a status of 0801 (hexadecimal) will be returned in \$R1 and the t-bit in the request block will be on. If the requested task has run to completion, or has so indicated by posting the request block through a terminate request macro call, the posted completion status will be returned in \$R1 and the t-bit in the request block will be off.

```
$TEST !TRB
```



# TEST FILE

## TEST FILE

Macro Call Names: \$TIFIL (input), \$TOFIL (output)

Function Codes: 10/62 (\$TIFIL), 10/63 (\$TOFIL)

\*

Equivalent Command: None

Test the status of any outstanding I/O activity. These macro calls are used in conjunction with I/O operations where the device to/from which the data transferred is a terminal. You identify the file by supplying its logical file number (LFN) in the file information block (FIB).

### FORMAT:

[label] { \$TIFIL } [,fib address]  
          { \$TOFIL }

### ARGUMENT DESCRIPTION:

fib address

Any address form valid for an address register; provides the location of FIB. The FIB must contain a valid LFN.

### FUNCTION DESCRIPTION:

When a terminal file is opened (see "Open File"), a physical connect is issued asynchronously; a wait-until-complete (i.e., line physically connected) is not done. Issuing a test file macro call after the file is opened causes a busy code to be returned until the connect is complete. If the connect is not complete within 5-minute standard time, a physical I/O time-out code (0106) is returned by any file or data management function (e.g., \$TIFIL, \$RDREC) issued by your program. Since the time-out code does not cause the file to be closed, any further access to the terminal (i.e., reopen to raise the connect again) must be preceded by a close file macro call. Once the connect is satisfied, the first test file macro call issued to an input or bidirectional LFN (after a successful connect) causes a read-ahead

(i.e., anticipatory read) to be issued. Furthermore, a read is requeued following a successful read record macro call.

\*

When the terminal file is closed, the sytem waits for the completion of any outstanding write orders, dequeues any anticipatory reads, and issues a disconnect.

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 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  
0204 - File busy  
0205 - Illegal argument  
0206 - Unknown or illegal LFN  
0207 - LFN not open

In addition to the above codes, any system service codes received by the file manager are passed on through \$R1.

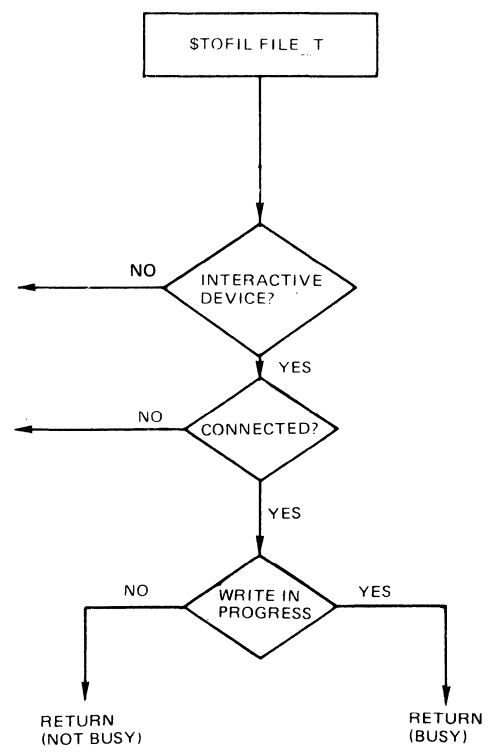
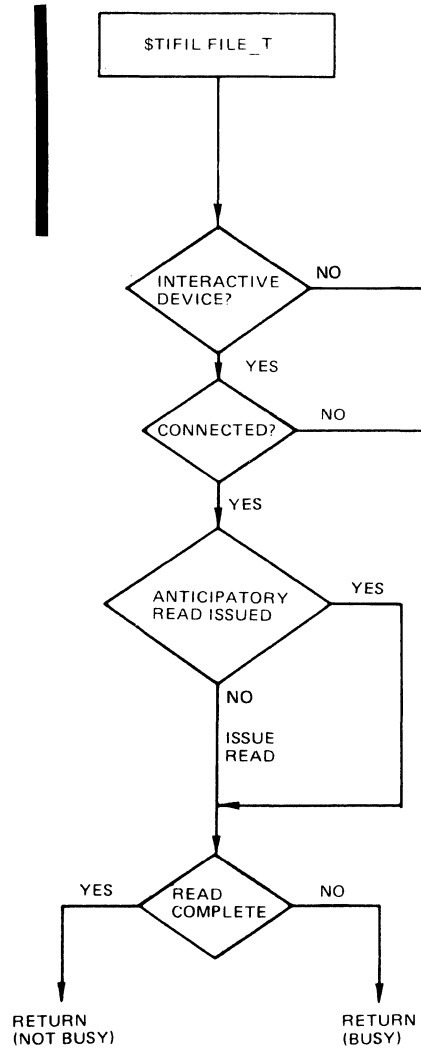
Example:

In this example, a terminal file, FILE T, associated with LFN 0006, has been reserved (see "Get File") and opened (see "Open File"). The following macro calls function as shown in the flowchart below. The FIB for FILE\_T is defined as:

```
FILE_T  DC  Z'0006'
```

```
  .  
  .  
  .
```

(Remainder is the FIB)



# TRAP HANDLER CONNECT

## TRAP HANDLER CONNECT

Macro Call Name: \$TRPHD

Function Code: 0A/00

Equivalent Command: None

Connect a user-written generalized trap handling routine entry point to the issuing task.

### FORMAT:

[label] \$TRPHD [location of trap handling routine address]

### ARGUMENT DESCRIPTION:

location of trap handling routine address

Any address form valid for an address register; provides the address of the user-written trap handling routine. This address (entry point) is entered at each occurrence of a user trap that has been enabled for that task.

### FUNCTION DESCRIPTION:

The connect trap function identifies a user-written routine that provides an alternative to the system default trap handler's response to user trap conditions. If user trap conditions are handled by the system default trap handler, the task in which the condition occurs is aborted.

Since trap conditions are handled in a task context, each task must identify the trap handler and enable the particular trap numbers to be serviced on behalf of the task (see enable user trap macro call). When an enabled user trap condition occurs, control is transferred to the user-written trap handling routine rather than the system default routine. See Section 7 for more information about trap handling.

- NOTES:
1. The address of the user-written trap handling routine 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  
0341 - Invalid trap handling routine address
  3. This macro call is required in order to enable a software simulated trap in a task that the user interrupts with the break function, and for which a PI or UW break response is entered.

Example:

In this example, the connect trap handler (\$TRPHD) macro call connects the routine labeled TRAPS as the issuing task's trap handler. The enable user trap macro call (\$ENTRP) enables the program interrupt and unwind traps for the task. All program interrupt and unwind traps for the issuing task will be directed to the routine labeled TRAPS. The disable user trap macro call (\$DSTRP) disables all user traps for the issuing task.

The remaining code illustrates the basic techniques used to write a user trap handler; it is not meant to be typical.

```

*
*      NAME THE PROGRAM TRAPS
*
PI_T      EQU      1
UW_T      EQU      49
*
*      NAME THE PERTINENT TSA FIELDS
*
TSA_W     EQU      $B3.4
*
*      CONNECT "TRAPS" AS THE TRAP HANDLER.
*
          STRPHD    !TRAPS
*
*      ENABLE PROGRAM INTERRUPT AND UNWIND.
*
          $ENTRPF   =PI_T
          $ENTRPF   =UW_T
          .
          .
          .
*
*      READ A NEW DIRECTIVE FROM USER INPUT.
*
GETLIN     $USIN    !LINE,=80
          .
          .
          .
*
*      DISABLE ALL TRAPS.
*
          $DSTRP    =-1
          .
          .
          .
FINISH     $TRMRQ   =$R2
*
*      TRAP HANDLER FOR THIS TASK:
*      SEND PROGRAM INTERRUPT TO "GETLIN"
*      SEND UNWIND TO "FINISH".
*
TRAPS      CMN      +$B3          INCREMENT B3 BY POINTER SIZE
          STB      $B4,TSA_W     SAVE B4
          LAB      $B4,GETLIN
          CMV      $R3,PI_T
          BE       >+$A
          LAB      $B4,FINISH
$A         STB      $B4,$B3
          LDB      $B4,TSA_W     RESTORE B4
          RTT

```

# USER IDENTIFICATION

## USER IDENTIFICATION

Macro Call Name: \$USRID

Function Code: 14/00

Equivalent Command: None

Returns the user identification of the calling task group to a 29-character receiving field, plus a terminating space.

### 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 29-character, aligned, non-varying field, followed by a blank, 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, person.account, or person.account.mode, depending on the login id used. 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 15-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      15,A'ΔΔ'
```



# USER INPUT

## USER INPUT

Macro Call Name: \$USIN

Function Code: 08/00

Equivalent Command: None

Read the next record from the current input file for the task group of 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 begins in the right byte of the specified address. Any other argument 3 value is assumed to designate the location of the byte offset from the address specified by argument 1 and is placed in \$R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address and \$R7 is set to zero.
  4. On return, \$R1, \$R6, \$R7, and \$B4 contain the following information:  
  
\$R1 - Return status; one of the following:  
  
0000 - No error  
0817 - Memory access violation

All data management read-next-record error codes may also be returned 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 64-byte record area whose address is in RECAD. The record area begins at the left byte of the indicated address.

```
INAA    $USIN    !RECAD,=128
          .
          .
          .
RECAD    RESV    64,0
```

# USER OUTPUT

## USER OUTPUT

Macro Call Name: \$USOUT

Function Code: 08/01

Equivalent Command: None

Write the next output record to the current user-out file for the task group of the issuing task.

### FORMAT:

[label] \$USOUT [location of record area address],  
[location of record size],  
[byte offset of beginning of record area]

### ARGUMENT DESCRIPTION:

location of record area address

Any address form valid for an address register; provides the address of the output record area containing the record to be written to the user-out file. The first byte of the record must be a slew byte (print file format control byte; see "Printer Driver" in Section 6). The record text begins in the second byte.

location of record size

Any address form valid for a data register; provides the size (in bytes) of the record area whose address is given in argument 1. The size value must include the slew byte.

byte offset of beginning of record area

Any address form valid for a data register; provides the byte offset of the beginning of the record area (from the address provided in argument 1). If argument 3 is L, the record area begins in the left byte of the address specified in argument 1; if argument 3 is R, the record area begins in the right byte of this address. Any other value is taken to be the location of the byte offset of the beginning of the record area from the address specified in argument 1. If argument 3 is omitted, the record area is assumed to begin at the left byte of the address specified in argument 1 and \$R7 is set to 0.

#### FUNCTION DESCRIPTION:

This call allows a task to write the next record to the current user-out file. Unless it has been changed by a new user output (\$NUOUT) macro call, the user-out file is as identified by the request group (\$RQGRP) or enter batch request (\$RQBAT) macro call.

- NOTES:
1. The address of the record to be written, supplied by argument 1, is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the output record.
  2. The output record size, supplied by argument 2, is placed in \$R6; if this argument is omitted, \$R6 is assumed to contain the size of the output record.
  3. If argument 3 is L, \$R7 is set to zero to designate that the record area begins in the left byte of the specified address. If argument 3 is R, \$R7 is set to one to designate that the record area begins in the right byte of the specified address. Any other value is assumed to be the location of the byte offset to be used, and is placed in \$R7. If argument 3 is omitted, the record area is assumed to begin in the left byte of the specified address and \$R7 is set to zero.
  4. On return, \$R1, \$R6, and \$B4 contain the following information:

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

0000 - No error

All data management write-next-record error codes may also be returned. See the System Messages manual.

\$R6 - Residual range (number of bytes not transferred from record area).

\$B4 - Address of record area containing output record.

Example:

In this example, the issuing task is to write the next record to the current user-out file for its task group. The record length is 137 bytes (including the slew byte). The output record begins at the right byte of the word labeled REC\_AR.

```
                $USOUT    !REC AR,  =137,R
                .
                .
                .
REC_AR    TEXT    'ΔA', (136) 'Δ'
```

# WAIT

## WAIT

Macro Call Name: \$WAIT

Function Code: 01/00

Equivalent Command: None

Wait for the completion of the operation that uses the specified request block (task, I/O, semaphore, clock, or overlay).

### FORMAT:

[label] \$WAIT [location of request block address]

### ARGUMENT DESCRIPTION:

location of request block address

Any address form valid for an address register; provides the address of the request block whose termination is to be awaited by the issuing task.

### FUNCTION DESCRIPTION:

This call permits a running task to indicate, as a separate action, that it wishes to wait for the completion of a particular request for the execution of another task. (The capability of the synchronous wait function is available through the request task function.)

When a wait macro call is issued, the issuing task must supply the address of the request block to be waited upon. If the task manager discovers that this request block is marked as terminated, it immediately returns control to the calling task, supplying the completion status code of the terminated request. If the request block is not marked as terminated, the task manager stores the identity of the calling (and now waiting) task in the request block and then suspends the calling task - another task can run at this task's level. Later, when the task manager is notified of the completion of the request being waited upon, it activates the waiting

task and reports the completion status code of the terminated request.

A given request block can be waited upon by only one task.

- NOTES:
1. The request block address derived from argument 1 is placed in \$B4; if this argument is omitted, \$B4 is assumed to contain the address of the request block.
  2. On return, \$R1 and \$B4 contain the following information:

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

yyzz - Where yy can be 00 or 80 through EE for user status, or as defined for other yy values in the System Messages manual.

0000-FFFF - Posted completion status

0802 - Invalid LRN

0803 - Illegal wait (request block already waited on, waiting on self, or request block not pending for this task).

\$B4 - Address of request block being waited upon

Example:

In this example the \$WAIT macro call is used to block the issuing task until a task that was requested using the request block labeled TRB posts its completion to that request block. See "Terminate Request" for information about task termination. When the issuing task is returned to the ready state, the task's posted completion status will be in \$R1.

```
WAIT_1  $WAIT  !TRB
```



# WAIT BLOCK

## WAIT BLOCK

Macro Call Name: \$WTBLK

Function Code: 12/20

Equivalent Command: None

Wait for the completion of the I/O operation associated with the specified buffer. This macro call is used only when the asynchronous bit is set in the program view entry in the file information block (FIB) for the preceding read block or write block macro call.

### FORMAT:

[label] \$WTBLK [fib address]

### ARGUMENT DESCRIPTION:

fib address

Any address form valid for an address register; provides the location of the FIB. The following FIB entries are required.

logical file number  
program view (must specify asynchronous I/O)  
user buffer pointer  
transfer size  
block size  
block number

### FUNCTION DESCRIPTION:

This macro call immediately follows a read block or write block macro call. The buffer identified by the user buffer pointer entry in the FIB must not be accessed between the read block or write block macro call and the wait block macro call, as shown below:

```
$RDBLK (block 0)
$WTBLK (block 0)
$RDBLK (block 1)
(process block 0)
$WTBLK (block 1)
$RDBLK (block 2)
(process block 1)
.
.
.
```

Furthermore, only one asynchronous operation per file can be outstanding at any given time.

The file information block (FIB) 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 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
0203 - Illegal function
0205 - Illegal argument
0206 - Unknown or illegal LFN
0207 - LFN not open
020A - Address out of file
0217 - Access violation
021F - End of file
```

In addition to the above codes, any system service codes received by the storage manager are passed on through \$R1.

Example:

In this example, it is assumed that the read block macro call was coded as described above, except that the program view entry specified Z'E001'. Based on this assumption, the wait block macro call would be coded as follows:

```
WAITAA $WTBLK !BLKFIB
```

# WAIT FILE

## WAIT FILE

Macro Call Name: \$WIFIL (input), \$WOFIL (output)

Function Codes: 10/64 (\$WIFIL), 10/65 (\$WOFIL)

Equivalent Command: None

Wait for the completion of an asynchronous I/O activity. \$WIFIL and \$WOFIL are used in conjunction with I/O operations in which the device to/from which data is transferred is a terminal. You specify a list of logical file numbers (LFNs) identifying the files to be checked by the wait function. If the \$WIFIL macro call is used, the function waits until at least one anticipatory read to one of the specified files is complete. If the \$WOFIL call is used, the function waits until at least one write to one of the specified files is complete. The first LFN for which an anticipatory read (\$WIFIL) or an asynchronous write (\$WOFIL) is complete is placed in a field that you specify.

### FORMAT:

[label] { \$WIFIL } [parameter structure address]  
          { \$WOFIL }

### ARGUMENT DESCRIPTION:

parameter 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.

out-LFN

A 2-byte field into which file management places the LFN (logical file number) that was the first LFN in the list for which I/O was complete.

## list length

A 2-byte field containing a binary number that specifies the number of LFNs in the list. If this field is zero (meaning no list of LFNs is specified), file management assumes a list of LFNs consisting of all LFNs in the task group that are currently associated with opened, interactive devices.

## LFN entries

A series of 2-byte fields, each containing the 2-byte logical file number (LFN) used to refer to the file. LFN is a binary number in the range 0 through 255. Each referenced file must have been reserved (see "Get File") and opened (see "Open File") through this LFN. The LFNs in the list are considered to be in order of priority; the first LFN specified for which I/O has completed will be returned in the out-LFN field.

## FUNCTION DESCRIPTION:

A wait-file-input function (\$WIFIL) is meaningful only for interactive device files that allow asynchronous input; this function gives up control of the central processor until at least one anticipatory read to any of the specified files is complete. A wait-file-output function (\$WOFIL) is meaningful only for interactive device files that allow asynchronous output; this function gives up control of the central processor until output to one of the specified files is complete.

When a wait-file-input function is executed, the out-LFN field is set to identify the first LFN in the list for which an anticipatory read is complete. Since more than one read may be completed at the same time, a \$TIFIL (see "Test File") macro call can be used after the \$WIFIL call to ascertain those LFNs for which input is complete. Note that the first \$WIFIL call issued after the file has been opened waits for the connect termination (initiated at the time of the open) in addition to waiting for the completion of the first read.

When a wait-file-output function is executed, the out-LFN field is set to identify the first LFN in the list for which an asynchronous write operation is complete. This function returns the status of the write operation. If the write terminated normally, the file can be considered as available for output.

The LFNs in the list are considered to be ordered by priority; thus the first LFN for which I/O has completed will be returned in the out-LFN field. You can ignore the output LFN and establish your own priority by using the test-file-input and test-file-output functions (see "Test File"). For example you could:

1. Issue a \$WIFIL for LFNs 1, 2, and 3.
2. Issue a \$TIFIL for LFN 2; read and process if not busy.
3. Do the same for LFN 1 and LFN 3.
4. Return to 1.

NOTES: 1. If the argument is coded, the 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.

0205 - Illegal argument (duplicate LFN)  
 0206 - Unknown or illegal LFN  
 0207 - LFN not open  
 0217 - Access violation

In addition to the above codes, any system service codes received by file manager are passed on through \$R1.

Example:

In this example, a wait-file-input function is executed to wait for the completion of a write operation on any of three interactive files whose LFNs are 1, 2, and 3. The completion of a write operation on the file associated with LFN 3 is checked first; if the write is complete, LFN 3 is placed in the out-LFN field. If the write is not complete, LFN 1's file is checked; if not complete, the file associated with LFN 2 is checked. If none of the write operations are completed, the task is placed in the wait state.

```

IWTLSL  DC      0
         DC      3
         DC      3
         DC      1
         DC      2
ONWTBB  $WIFIL  !IWTLSL
  
```

# WAIT LIST, GENERATE

## 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. A maximum of 35 request block pointers can be specified.

### FORMAT:

```
[label] $WLIST [request block label 1],  
               [request block label 2],  
               .  
               .  
               [request block label 35]
```

### ARGUMENT DESCRIPTION:

request block label 1 ... request block label 35

Label of the request block to be placed in the wait list. A maximum of 35 blocks can be specified.

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 0lnn (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 Program Preparation 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

## 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 35]
```

### ARGUMENT DESCRIPTION:

request block label 1 ... request block label 35

Label of the request block to be placed in the wait list. A maximum of 35 blocks can be specified.

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. (Note that only 35 request blocks can be specified directly in the macro call.)



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 Program Preparation manual for more information about SAF/LAF independent code.

Example:

In this example, the request clock macro call (\$RQCL) is issued to start a 5-second timer using the clock request block labeled TIMER. Then the wait on request list macro call (\$WAITL) is used to block the issuing task until either the task that was requested using a request block labeled TRB posts its completion or the clock manager posts completion of the 5-second interval on the clock request block labeled TIMER. If the task goes to completion first, the cancel clock request macro call (\$CNCRQ) will cancel the request on TIMER, thus freeing it for later reuse. To simplify the example, the return status will not be checked for errors that might occur.

```

                $RQCL    !TIMER
                $WAITL   TRB,TIMER
                CMB      $B4,=TIMER
                BE       T_OUT
*
*   THE SUBTASK COMPLETED FIRST
*
                $CNCRQ   !TIMER
                .
                .
*
*   THE CLOCK TIMED OUT FIRST
*
T_OUT EQU      $
                .
                .
TIMER  $CRB    R, NWAIT,,MS=5000
```

# WRITE BLOCK

## WRITE BLOCK

Macro Call Name: \$WRBLK

Function Code: 12/10 (normal), 12/11 (tape mark)

Equivalent Command: None

Write (i.e., transfer) a block from a buffer in main memory to a file. The user must supply a buffer and specify both the size of the block and its relative location in the file.

### FORMAT:

[label] \$WRBLK [fib address] [ { ,NORMAL }  
{ ,TM } ]

### ARGUMENT DESCRIPTION:

fib address

Any address form valid for an address register; provides the location of the file information block (FIB). The following FIB entries are required.

logical file number

program view (should include buffer alignment and whether the next write operation is synchronous or asynchronous)

user buffer pointer

transfer size

block size (must be a multiple of the physical sector size)

block number

{ NORMAL }  
{ NOR }

For disk-resident files this mode argument indicates that the contents of the buffer are to be written in a control interval whose block number is specified in the block number entry in the FIB.

For nondisk-resident files this mode argument indicates that the block is to be transferred from the buffer to the next sequential position on the file.

NORMAL is the default value for this macro call.

TM

(For tape-resident files only.) This mode argument indicates that a tape mark is to be written in the next sequential position on the tape.

#### FUNCTION DESCRIPTION:

Before this macro call can be executed, the LFN must have been opened (see open file macro call) with a FIB program view word that allows access via storage management (bit 0 is 1) and allows write operations (bit 2 is 1). To write a file sequentially, it is necessary only to issue successive write block macro calls in NORMAL mode, which causes the block number entry to be incremented by 1 after each transfer. The system extends the file space up to the limit specified in the create file macro call when required to do so as a result of a write block macro call. In addition, the system updates the logical end-of-file as the file is extended.

The following end-of-file/end-of-tape considerations must be noted:

- o Tape-resident files. If logical end-of-tape (i.e., EOT reflector) is detected during a write block macro call, all data is written and status code 0231 is returned. If physical end-of-tape is reached before all data is written, a code of 0231 is also returned.
- o Disk-resident files. If there is insufficient space to contain the data defined by the transfer size entry in the FIB (i.e., the file has reached its maximum size), a status code of 0223 is returned. If the file has not reached its maximum size but no more sectors are available to be allocated to it, a code of 0215 is returned. No data is written.
- o All files. Partial blocks are not written.

Only one asynchronous I/O operation per LFN can be outstanding at any given time.

The file information block (FIB) 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

0203 - Illegal function

0205 - Illegal argument

0206 - Unknown or illegal LFN

0207 - LFN not open

020A - Address out of file

0215 - Not enough contiguous logical sectors available

0217 - Access violation

0223 - File space limit reached or file not expandable

0224 - Directory space limit reached or not expandable

0231 - Unexpected tape EOT encountered

In addition to the above codes, any system service codes received by the storage manager are passed on through \$R1.

Example:

This example assumes that the FIB was defined as follows:

```
BLKFIB  DC      Z'0005'      LFN = 5
        DC      Z'E000'      PROGRAM VIEW = ALLOW READ/WRITE;
                                SYNCHRONOUS PROCESSING
        DC      <BLKBUF      BUFFER POINTER
        RESV    2-$AF
        DC      256          TRANSFER SIZE = 256
        DC      256          BLOCK SIZE = 256
        DC      Z'00000000'   BLOCK NUMBER
```

When the above FIB is defined, and assuming the appropriate open file and get file macro calls are specified, the following macro call can be executed to write the contents of the buffer into the first block (i.e., block 0) in the file:

```
$WRBLK  !BLKFIB,NORMAL
```

# WRITE RECORD

## WRITE RECORD

Macro Call Name: \$WRREC

Funtion Code: 11/20 (next), 11/21 (key), 11/22 (position equal),  
11/23 (position greater than), 11/24 (position  
greater than or equal), 11/25 (position forward),  
11/26 (position backward)

Equivalent Command: None

Transfers logical records to a file from your record area or merely positions the write pointer to a desired record. Whether to transfer or position is specified by the second (i.e., mode) parameter.

FORMAT:

[label] \$WRREC [fib address]  $\left[ \begin{array}{l} \left\{ \begin{array}{l} ,NEXT \\ ,KEY \\ ,POSEQ \\ ,POSGR \\ ,POSGREQ \\ ,POSFWD \\ ,POSBWD \end{array} \right\} \end{array} \right]$

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 is to be written into the position in the file identified by the write pointer. The write pointer is set to the next logical record in the file after the write is complete. The system ensures that the position pointed to is unused or contains a deleted record. Records are written in the file as described in the Data File Organizations and Formats manual. This is the default for this macro call. You must code the following FIB entries:

logical file number  
program view (record area alignment)  
user record pointer  
output record length

After the record is transferred to the file, the system updates the following FIB entry:

output record address (serial sequence number if device file; BSN if tape file; simple key unless relative access specified at open time).

This mode is referred to as write next.

#### KEY

(For disk files accessed by key, only.) This mode argument indicates that the record is to be written into a position in the file, based upon the key value. The write pointer is set to the next logical record in the file after the write is complete. Records are written as described in the Data File Organizations and Formats manual. You must code the following FIB entries:

logical file number  
program view (record and key area alignment)  
user record pointer  
output record length  
input key pointer  
input key format  
input key length



After the record is transferred to the file, the system updates the following FIB entry:

output record address

This mode is referred to as write with key.

{ POSEQ }  
{ PEQ }

(For disk files accessed by key, only.) This mode argument positions the write pointer to the first position in the file whose key value is equal to the one specified in the FIB. It is normally followed by write next macro calls to load the file starting from that position. 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 write pointer to the first position in the file whose key value is greater than the one specified in the FIB. It is normally followed by write next macro calls to load the file starting from that position. The same FIB entries as for POSEQ above must be coded. This mode is referred to as write position greater than.

{ POSGREQ }  
{ PGE }

(For disk files accessed by key only.) This mode argument positions the write pointer to the first position in the file whose key value is greater than or equal to the one specified in the FIB. It is normally followed by write next macro calls to load the file starting from that position. The same FIB entries as for POSEQ above must be coded. This mode is referred to as write position greater than or equal.

{ POSFWD }  
{ PFD }

(For tape-resident 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 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 \$TFIB 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, or POSGREQ 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.

### 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 concurrency 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.

## File Organization

## Effects of Write Action

### Indexed (cont)

If the file is being initially loaded, it should be opened in RENEW mode, with the data to be written sequenced in ascending order by primary key. Data management will verify that the supplied keys are in order, and will generate a key out of sequence error if they are not. When inserts are to be made, the existing file should be opened in PRESERVE mode.

### Fixed Relative

Fixed relative with nondeletable records:  
Write next, issued immediately after an open file, appends a record to the end of an existing file. In RENEW mode, this action can be used to create the file sequentially. When issued after a write next, write with key, or any position mode, it inserts a record in the next logical record position.

Write with key inserts a record in the space addressed by the relative key. All position modes set write concurrency to the specified record.

Fixed relative with deletable records:  
Write next, issued immediately after an open file, appends a record to the end of an existing file. In RENEW mode, this action can be used to create the file sequentially. Issued after a write next, write with key, or any position mode, write next inserts a new record in the next available (unused or deleted) space. This write next operation searches for the next available space in which to place the record.

Write with key uses a relative key that must address a deleted record or an unused space.

All position modes use a relative key that addresses (sets write concurrency to) an active record, deleted record, or an unused record.

### Device Files

Write next allows sequential writing, provided the device can be written to and has been so defined.

NOTES: 1. If the first argument is coded, the address of the FIB is loaded into \$B4; if this 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

0203 - Illegal function

0205 - Illegal argument

0206 - Unknown or illegal LFN

0207 - LFN not open

020A - Address out of file

0217 - Access violation

0219 - No current record pointer

021A - Record length error

021B - Duplicate key

021C - Key out of sequence

021E - Key length or location error

0223 - 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

022B - Requested record is locked

In addition to the above codes, any system service codes received by the data manager are passed on through \$R1.

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, 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 call the drivers directly or can use them indirectly by calling the file manager. If you want to write a peripheral driver for a nonstandard device, or modify the function of an existing peripheral driver, refer to Appendix B.

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 CONVENTIONS

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

- o The I/O request block (IORB) is the standard control structure used by a driver (see "Data Structures," later in this section for definition).
- o The \$RQIO macro call is used to request a driver.
- o The B4-register contains the address of the IORB supplied by the caller; the IORB contains the LRN of the device to be used.
- o The I/O-specific words of the IORB (I\_CT2 through I\_DVS) are not modified by the driver.
- o If a device becomes inoperable, it can be disabled with an operator command and another device can be substituted.
- o Drivers are reentrant and interrupt driven; one driver supports many devices of the same type.
- o Synchronous and asynchronous I/O are supported.
- o Work space, if required, is located in the device's related resource control table (RCT).
- o 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

IORB Function Code	Device					
	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 <sup>a</sup>	Write (punch)	Write	Write	Write
2	Read	Read	NA	NA	Read	Read
3	NA	NA	Write file mark (punch)	NA	Write deleted data	Write file mark
4	NA	NA	NA	NA	Read deleted data	Position block <sup>b</sup>
5	NA	NA	NA	NA	Format write	NA
6	NA	NA	NA	NA	Format read	Position file <sup>c</sup>
9	Break Notification (KSR only)	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

<sup>a</sup>Not applicable.

<sup>b</sup>Positive range of one is forward space to start of next block.  
Negative range of one is backspace to beginning of previous block.

<sup>c</sup>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 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). When the ready state of a device changes, the attention bit set in the RCT (R\_FLGS, bit 8) may be reset by a reset device attention (\$RDVAT) macro call.

A task can ask to be notified about the occurrence of an interrupt by a device by setting bit 9 of R\_FLGS of the RCT (see the disable device (\$DSDV) macro call.) When such an interrupt occurs, the driver will set bit 10 of R\_FLGS, and place the value 8 in I\_ST of subsequent IORB's supplied by the requesting task. Subsequently, when a ready interrupt for the device is generated, the using task can clear the disabled status by resetting bit 10 of R\_FLGS (see \$ENDV (enable device) macro call).

## WRITE FUNCTION (fc=1)

The write function is available for use by 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.

Table 6-2. Return Status Codes

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
7	Hardware error, check IORB status word <sup>a</sup>
8	Device disabled <sup>b</sup>
9	File mark encountered
A	Controller unavailable
B	Device unavailable <sup>a</sup>
C	Inconsistent request <sup>c</sup>

<sup>a</sup>When these codes are found in I\_CT1 (IORB), or in \$R1 on a resume after wait, look at I\_ST (IORB) to identify the specific error. The status B is re-returned with every read or write IORB that has been aborted by a disconnect request with queue abort.  
<sup>b</sup>This status will be returned on an I/O request if the user program has disabled the logical resource. This event is indicated in the device resource control table (See Figure 6-2) by bit 10 of R\_FLGS.  
<sup>c</sup>This status indicates illogical peripheral driver requests: read or write before connect; duplicate connect or disconnect requests; write after disconnect.

READ FUNCTION (fc=2)

The read function is available for use by all devices for 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.

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.

#### 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 only for KSR devices, 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.

## DATA STRUCTURES

Two data structures control the interactions among an application program, its device drivers, and the devices it uses. These structures are the input/output request block (IORB) and the resource control table (RCT). The IORB is the interface between the application task and the device driver; the RCT is the interface between the driver and its device(s).

### 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.

The format of the IORB is shown in Figure 6-1; Tables 6-3 and 6-4 define the individual IORB entries; these entries are common to drivers for all device types. 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.

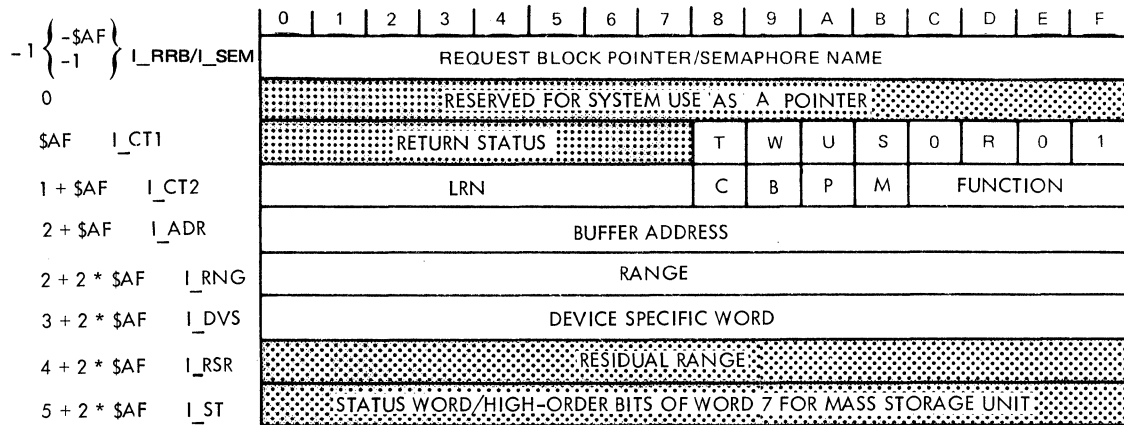


Figure 6-1. Format of I/O Request Block

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 condition of the S- or R-bits of I_CT1, this word contains a 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 through 15; 0 through 31	Reserved for system use. 1- or 2-word pointer to indirect request block.
\$AF	I_CT1	0 through 7  8 (T)	Return status  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.

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

Item	Label	Bits	Contents
\$AF (cont)	I_CTL (cont)	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; system does not change it.
		B (S)	Release semaphore indicator. Values: 0=No release, 1=Release on time-out of item named in I_RRB.
		C	Must be zero.
		D (R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in I_RRB, after time-out of this request. (System executes \$RQTSK, using I_RRB, upon task termination.
		E (D)	Delete IORB. Values: 0=No delete; 1=return IORB to dynamic memory pool where IORB is first entry of memory block.
		F	I/O bit - must be set.
1+\$AF	I_CT2	0 through 7	Logical resource number (LRN); identifies device to be used.
		8 (C)	IBM-type request. Changes interpretation of I_DVS to task word, and 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.
		A (P)	Reserved for system use.
		B (M)	0 = Standard IORB; 1 = IORB is extended at least 6+2*\$AF items.

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

Item	Label	Bit	Contents
1+\$AF (cont)	I_CT2 (cont)	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; used as input field for cartridge disk or disk storage unit.
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 <u>not</u> 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.
5+2*\$AF	I_ST	0 through 15	Status word. Reflects the mapping of the hardware status into software status format; used as input field high-order bits of sector number for mass storage unit. (See Table 6-4)
<p>NOTE: For break notification requests, the KSR driver stores the TCB of the issuing task in this word. When break occurs, the contents of this word are transferred to the RCT.</p>			



Table 6-4. IORB Software Status Word<sup>1</sup> (I\_ST)

Bit Position	ASR/KSR	Card Reader	Card Reader/Punch	Printer	Diskette	Cartridge Disk	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
<p>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.</p> <p>2. The online drivers will flag, in the RCT, corrected memory errors and driver or hardware corrected errors.</p>								

<sup>1</sup>Equivalent to a modified status word 1.

## Resource Control Table (RCT)

The resource control table (RCT) is a system-defined table that contains task or device parameters. A task RCT consists of a pointer to the interrupt save area (ISA) servicing the task. A device RCT contains a pointer to the interrupt save area servicing the task and the device-specific information shown in table 6-5. An entry in the logical resource table (LRT) points to the resource control table, thus establishing a correspondence between the logical resource number (LRN) and the interrupt save area.

Table 6-5 shows specific entries in the RCT.

Table 6-5. Resource Control Table (RCT) Field Definitions

Field (bits)	Word Offset	Content
R_TCB	-\$AF	Pointer to TCB (and ISA) of the driver.
*	0	Channel of the referenced device; set before driver initialization.
R_TYP	1	Device type as read from the device by the initialization routine.
R_FLGS	2	Flags for use by the drivers. The first 5 bits (0-4) are device specific. <u>Flags that are dynamically modified by the drivers are indicated by an asterisk (*)</u> .
R_TRCV(0)		*(Tape) - Tape recovery requested.
R_DDEN(0)		*(Diskette) - Double density (set by AVR).
R_IN(0)		*(KSR) - Input attention to process.
R_CRED(1)		*(Tape) - Tape recovery successful.
R_IBM(1)		*(Diskette) - IBM type volume (set by AVR)
R_BNVL(2)		*(Tape) - Block count invalid.
R_NFS(2)		* Not connected to file system.
R_FLSM(3)		(For disk driver) - Large disk address (mass storage unit).
R_FSCM(4)		(For KSR) Not single character mode.

Table 6-5 (cont). Resource Control Rable (RCT) Field Definitions

Field (bits)	Word Offset	Content
R_AVR(4)	2 (cont)	(Disk) Do AVR when set.
R_FLDT(5)		Disk type device.
R_FLCM(6)		Communication connected device.
R_FLSL(7)		Slew type device.
R_FATN(8)		*Attention has occurred.
R_FDOA(9)		Disable device on attention.
R_DSAB(A)		*Device disabled.
R_ELBZ(B)		*Error log busy.
(C-F)		*The last four bits of status word one.
R_STTS	3	Last hardware status word one.
R_ERLG	4	Pointer to the error log structure or null (if error log is not active).
R_TIOT	4+\$AF	Timeout value (seconds).
R_FMSK	5+\$AF	The 32-bit mask defining the global action to be taken for a given function (see I_CT2). The first word has a bit corresponding to function, set if the function for this device is illegal or no-operation. The second word will have the corresponding bit set for illegal function (e.g., read for printer) or special, globally handled functions (wait on line.)
R_DRQ	7+\$AF	Pointer to driver's request handler.
R_DRAT	7+2\$AF	Pointer to driver's attention handler.

Table 6-5 (cont). Resource Control Table (RCT) Field Definitions

Field (bits)	Word Offset	Contents
The following fields are device-specific, for device shown:		
Disk Specific		
R_SCCL	7+3\$AF	Number of sectors per cylinder. For devices, depending on the media, it must be set by AVR.
R_SCTR	8+3\$AF	Number of sectors per track.
R_SRCH	9+3\$AF	Z'FFFF' for devices with the automatic track crossing Z'FEFF' for diskette.
R_FXPL	10+3\$AF	Z'0800' for fixed cartridge disk, zero for others.
KSR Specific		
R_CWA	7+3\$AF	Configuration words A, B
R_BTDB	9+3\$AF	Two-word unique identifier of task for which "break" must inhibit messages.
R_BUF	11+3\$AF	Pointer to single character mode buffer and its control structures.
Magnetic Tape Specific		
R_SW2	7+3\$AF	Status words two
R_TMCT	8+3\$AF	Tape mark count
R_BLCT	9+3\$AF	Block count

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 should 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. If there was an error it 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).

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.

The channel number, defined at system building, is located in the device's resource control table (RCT) entry. (See the GCOS 6 MOD 400 System Building manual). The LRN supplied by the caller in the IORB serves as an index into a system table (LRT) which in turn contains the pointer to the RCT entry defining the device as in Figure 6-2.

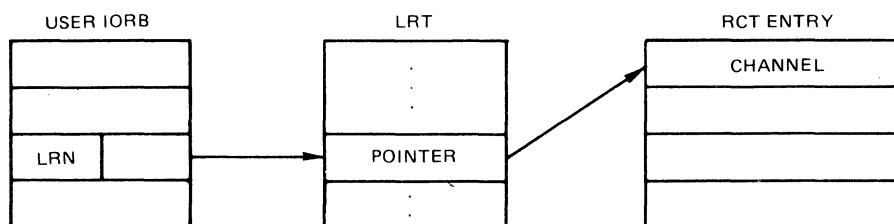


Figure 6-2. LRN as Pointer to RCT

Thus, with the LRN, the driver can indirectly access the RCT entry which defines the particular device the calling task wishes to access.

The rest of the information needed to perform the I/O request is found in the IORB itself. 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 (e.g., read function code from disk translated by the disk driver into a "seek" to the correct track, and a "read" of the correct physical sector).

Finally a timer is started in order to protect against a device fault prohibiting the sending of a completion interrupt to the central processor.

At the completion of any central processor-to-I/O instruction the driver can test the I/O flag in the indicator (I) regis-

ter in order to tell if the central processor instruction was accepted by the device.

When the I/O operation terminates, the device interrupts the central processor at the level of the driver. The device knows which level to interrupt at since the corresponding driver transmitted the level value to the device itself prior to any I/O being performed. (Within the device ISA, the DEV field (see Figure 7-1) contains the channel number of the device and the interrupting level. Thus, the driver can explicitly tell which device is interrupting, and can perform the I/O completion processing.)

## DEVICE DRIVERS

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

- o Card reader/Card reader-punch driver
- o Printer driver
- o Disk driver
- o ASR/KSR driver
- o 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 only three function codes; i.e., read, write, 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 6-7.

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:

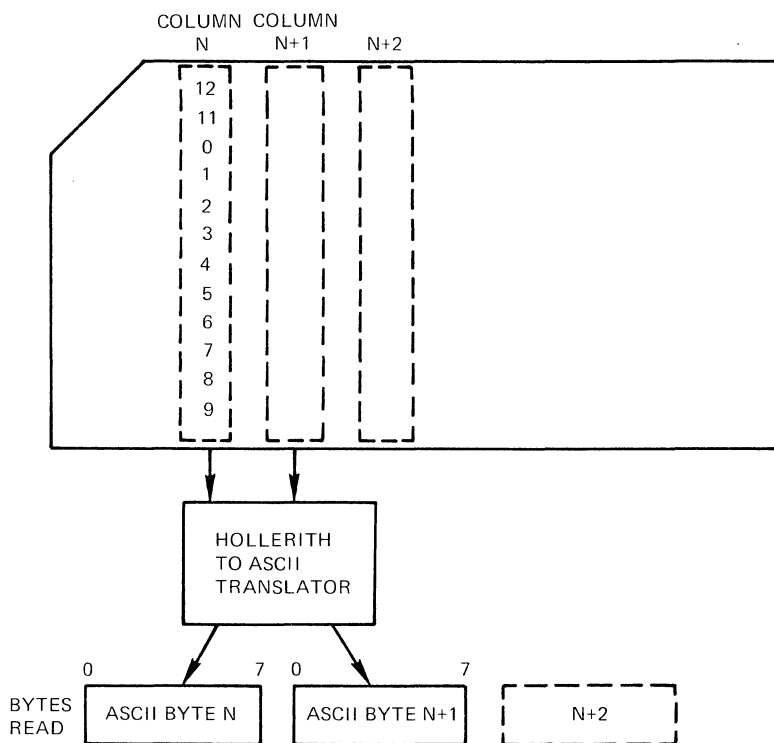
- o Detects and discards unsolicited interrupts (\$B4=0 upon entry to driver).
- o Detects an end-of-file condition and sets the appropriate return status (ASCII GS character in column 1 of any card=EOF).
- o Detects "device not ready" condition and sets appropriate error condition.

## ASCII MODE

In this mode, punched cards are processed as shown in Figure 6-3. 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 6-6. 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 7 in R1 register if any card column read had a hole pattern which was not one of the legal hole patterns given in Table 6-6. The illegal card code causes an ASCII-E0 (all 1's) code to be loaded in the main memory.



- NOTES:
1. This translator will provide a status indicator which will be set whenever an illegal Hollerith code is read.
  2. The translator shown above is in the card reader attachment.

Figure 6-3. ASCII Card-to-Memory Code Formatting

Table 6-6. Hollerith-ASCII Code Table

b4b3b2b1	COL ROW	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	COL
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ROW
0000	0	NUL 12-0-9-8-1	DLE 12-11-9-8-1	SP NO PCH	0 0	a 8-4	P 11-7	8-1	p 12-11-7	11-0-9-8-1	12-11-0-9-8-1	12-0-9-1	12-11-9-8	12-11-0-9-6	12-11-8-7	12-11-0-8	12-11-9-8-4	0
0001	1	SCH 12-9-1	DC1 11-9-1	!	1 12-8-7	A 12-1	Q 11-8	a 12-0-1	q 12-11-8	0-9-1	9-1	12-0-9-2	11-8-1	12-11-0-9-7	11-0-8-1	12-11-0-9	12-11-9-8-5	1
0010	2	STX 12-9-2	DC2 11-9-2	"	2 8-7	B 12-2	R 11-9	b 12-0-2	r 12-11-9	0-9-2	11-9-8-2	12-0-9-3	11-0-9-2	12-11-0-9-8	11-0-8-2	12-11-0-8-2	12-11-9-8-6	2
0011	3	ETX 12-9-3	DC3 11-9-3	#	3 8-3	C 12-3	S 0-2	c 12-0-3	s 11-0-2	0-9-3	9-3	12-0-9-4	11-0-9-3	12-0-8-1	11-0-8-3	12-11-0-8-3	12-11-9-8-7	3
0100	4	ECT 9-7	DC4 9-8-4	\$	4 11-8-3	D 12-4	T 0-3	d 12-0-4	t 11-0-3	0-9-4	9-4	12-0-9-5	11-0-9-4	12-0-8-2	11-0-8-4	12-11-0-8-4	11-0-9-8-7	4
0101	5	ENQ 0-9-8-5	NAK 9-8-5	%	5 0-8-4	E 12-5	U 0-4	e 12-0-5	u 11-0-4	11-9-5	9-5	12-0-9-6	11-0-9-5	12-0-8-3	11-0-8-5	12-11-0-8-5	11-0-9-8-3	5
0110	6	ACK 0-9-8-6	SYN 9-2	&	6 12	F 12-6	V 0-5	f 12-0-6	v 11-0-5	12-9-6	9-6	12-0-9-7	11-0-9-6	12-0-8-4	11-0-8-6	12-11-0-8-6	11-0-9-8-4	6
0111	7	BEL 0-9-8-7	ETB 0-9-6	'	7 8-5	G 12-7	W 0-6	g 12-0-7	w 11-0-6	11-9-7	12-9-8	12-0-9-8	11-0-9-7	12-0-8-5	11-0-8-7	12-11-0-8-7	11-0-9-8-5	7
1000	8	BS 11-9-6	CAN 11-9-8	(	8 12-8-5	H 12-8	X 0-7	h 12-0-8	x 11-0-7	0-9-8	9-8	12-8-1	11-0-9-8	12-0-8-6	12-11-0-8-1	12-0-9-8-2	11-0-9-8-6	8
1001	9	HT 12-9-5	EM 11-9-8-1	)	9 11-8-5	I 12-9	Y 0-8	i 12-0-9	y 11-0-8	0-9-8-1	9-8-1	12-11-9-1	0-8-1	12-0-8-7	12-11-0-1	12-0-9-8-3	11-0-9-8-7	9
1010	10	IF 0-9-5	SUB 9-8-7	*	11-8-4 8-2	J 11-1	Z 0-9	j 12-11-1	z 11-0-9	0-9-8-2	9-8-2	12-11-9-2	12-11-0	12-11-8-1	12-11-0-2	12-0-9-8-4	12-11-0-9-8-2	10
1011	11	VT 12-9-8-3	ESC 0-9-7	+	12-8-6 11-8-6	K 11-2	[ 12-8-2	k 12-11-2	[ 12-0	0-9-8-3	9-8-3	12-11-9-3	12-11-0-9-1	12-11-8-2	12-11-0-3	12-0-9-8-5	12-11-0-9-8-3	11
1100	12	FF 12-9-8-4	FS 11-9-8-4	,	0-8-3 12-8-4	L 11-3	\ 0-8-2	l 12-11-3	, 12-11	0-9-8-4	12-9-4	12-11-9-4	12-11-0-9-2	12-11-8-3	12-11-0-4	12-0-9-8-6	12-11-0-9-8-4	12
1101	13	CR 12-9-8-5	GS 11-9-8-5	-	8-6 11	M 11-4	^ 11-8-2	m 12-11-4	^ 11-0	12-9-8-1	11-9-4	12-11-9-5	12-11-0-9-3	12-11-8-4	12-11-0-5	12-0-9-8-7	12-11-0-9-8-5	13
1110	14	SO 12-9-8-6	RS 11-9-8-6	.	0-8-6 12-8-3	N 11-5	^ 11-8-7	n 12-11-5	~ 11-0-1	12-9-8-2	9-8-6	12-11-9-6	12-11-0-9-4	12-11-8-5	12-11-0-6	12-11-9-8-2	12-11-0-9-8-6	14
1111	15	SI 12-9-8-7	US 11-9-8-7	/	0-1 0-8-7	O 11-6	_ 0-8-5	o 12-11-6	DEL 12-9-7	11-9-8-3	11-0-9-1	12-11-9-7	12-11-0-9-5	12-11-8-6	12-11-0-7	12-11-9-8-3	EO 12-11-0-9-8-7	15



## VERBATIM MODE

In this mode, punched cards are processed as shown in Figure 6-4. The card column pattern is stored in bits 4 through 15 of the main memory word with bits 0 through 3 set to zero. All 2 hole patterns will be valid during a verbatim mode operation. The device-specific fields in the IORB and RCT are given below.

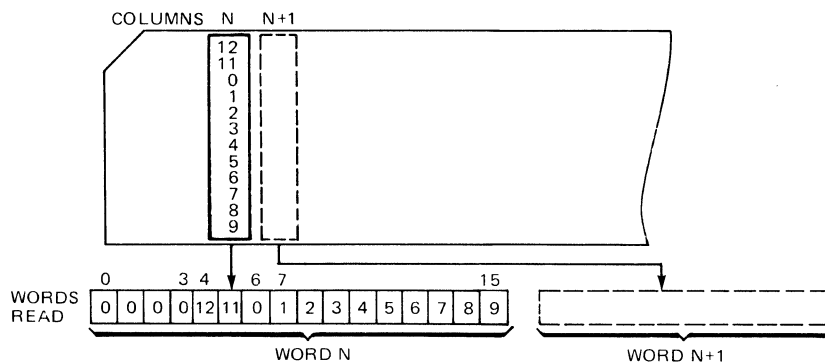


Figure 6-4. Verbatim Mode Formatting

### CARD READER/CARD READER-PUNCH DEVICE-SPECIFIC IORB FIELDS

Table 6-7 defines the device-specific fields in the IORB not previously defined.

Table 6-7. Card Reader Card/Reader-Punch Device-Specific IORB Fields

Item	Field	Definition	Use
I_CT2	Function code	0=wait-on-line 1=write  2=read  3=write file mark	Refer to "Driver Functions and Functions Codes" earlier in this section.  Driver "reads" cards for "range" number of bytes.  Driver "writes" cards for "range" number of bytes.

Table 6-7 (cont). Card Reader/Card Reader-Punch Device-Specific IORB Fields

Item	Field	Definition	Use										
I_RNG	Range	$0 \leq \text{range} \leq 32K-1$	If range is greater than card size, residual range will reflect the difference.										
I_DVS	Device-specific	<table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td style="padding: 2px;">0</td> <td style="padding: 2px;">12</td> <td style="padding: 2px;">13</td> <td style="padding: 2px;">14</td> <td style="padding: 2px;">15</td> </tr> <tr> <td style="padding: 2px;">0</td> <td style="padding: 2px;">0</td> <td colspan="3" style="padding: 2px;">mode</td> </tr> </table> <p style="margin-left: 40px;">mode: 0=ASCII 2=verbatim</p>	0	12	13	14	15	0	0	mode			Defines character set of data being read.
0	12	13	14	15									
0	0	mode											
I_RSR	Residual range	$0 \leq \text{initial range}$	Detects device malfunction.										

CARD READER/CARD READER-PUNCH DEVICE-SPECIFIC RCT FIELDS

The following RCT fields contains device-specific values for the card reader and card reader-punch.

<u>Item</u>	<u>Field</u>	<u>Definition</u>	<u>Use</u>
R_TYP	Device Type	2008-200F 2088-208F	Card reader device type Card reader-punch device type
R_STTS	Device Status	See Table 6-8 6-9.	Status as read from device

CARD READER/CARD READER-PUNCH RCT/IORB STATUS CODE MAPPING

The card reader/card reader-punch controller returns to the driver various codes that are placed in the RCT. The status code is then made visible to the application by placing it in the IORB as shown in Tables 6-8 and 6-9.

Table 6-8. Card Reader IORB Hardware/Software Status Code Mapping

RCT Word R_STTS	IORB Word I_ST	Meaning If Bit Set
0	-	Device ready
1	-	Attention
2	2	Data service rate error
3	3	Mark sense mode
4	4	40-column card mode
5	5	51-column card mode
6	6	External clock track
7	7	Read check error
8	8	ASCII code error
-	-	
-	-	
-	-	
12	-	Corrected memory error
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Uncorrectable memory error/fatal error

Table 6-9. Card Reader/Punch Hardware/Software Status Code Mapping

RCT Word R_STTS	IORB Word I_ST	Meaning If Bit set
0	-	Device ready
1	-	Attention
2	2	Data service rate error
3	3	Invalid ASCII code
4	4	Punch echo or read registration
5	5	Light/dark check
6	6	Card jam
7	-	
8	-	
-	-	
-	-	
-	-	
12	-	Corrected memory error
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Uncorrectable memory error/fatal error

## Printer Driver

The printer driver performs all data transfers to all line and serial printers as well as terminal print devices. Format control of printing is achieved by supplying a control byte as the first entry in a data buffer.

### PRINT CONTROL BYTE

A print control byte can precede text data sent to output "print" devices (e.g., console, printer, VIP, etc.). This byte must be the first character in the data buffer and is included in the range count of the IORB for the request.

Forms control differs between printers (which uses prespace, then print format conventions), and consoles (terminals), (which use print, then postspace format conventions). These are indicated separately below.

The contents and meanings of the control byte are:

BIT:	0	1	2	3	4	7
FIELD:	Y	PP	V	COUNT		

Field	Action Caused	
	Line Printer (Space Before Print)	Terminal Printer (Space After Print)
Y	Not Used	0=Use IORB word I_DVS for device-specific information  1=Ignore IORB word I_DVS
PP	00 Print; ignore V and count/fields; single space except at end-of-form, skip to head-of-form.  01 Don't print; perform actions defined in V and count fields.  10 Print; perform actions defined in V and count fields.  11 Reserved for system use.	Not Used.

Field	Action Caused	
	Line Printer (Space Before Print)	Terminal Printer (Space After Print)
V	0 Prespace according to Count Field.  1 If count = 0, skip to head-of-form. If count is between 1 and 12, and the VFU option is present, skip to the VFU channel defined by the count field.  If count is greater than 12, or there is no VFU option, do one prespace.	0=No prespace.  1=Prespace three lines, count field must be 0.

The control byte summary is as follows:

Code		
Line/Serial Printers		
Hexadecimal	ASCII	Resulting Action
00-1F	NUL-US	Single space, then print; skip to head-of-form at end-of-form.
20-2F	Δ - /	Space Count lines, don't print.
30-3F	0 - ?	Skip to VFU channel number in Count, don't print.
40-4F	@ - 0	Space Count number of lines, print.
50-5F	P - _	Skip to VFU channel number in Count, print; 50 = skip to head-of-form.
60-6F	\ - o	Reserved for future use.
70-7F	p - DEL	Reserved for future use.

Code		
Hexadecimal	ASCII	Resulting Action
Terminal Printers		
00-0F		
20-2F	Δ - /	No prespace, print.
40-4F	@ - 0	
10-1F } 30-3F } 50-5F }	0̄ - ? } P - - }	Prespace three lines, print.
60-6F	' - o	Reserved for future use.
70-7F	p - DEL	Reserved for future use.

#### PRINTER DEVICE-SPECIFIC IORB FIELDS

Table 6-10 defines the IORB fields whose contents are specific to the printer driver.

Table 6-10. Printer Device-Specific IORB Fields

IORB Item	Field	Definition	Use
I_CT2	Function code	0=wait-on-line 1=write	See "Driver Function and Function Codes." Driver will "write" from I_ADR "range" number of bytes.
I_RNG	Range	$0 \leq \text{range} \leq 32K-1$	If range is greater than line size residual range will reflect the difference.
I_DVS	Device-specific	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 0 0 0 F 0 0 0 0 0 0 0 0 0 0 0 0 F: 0=Assumes line printer format control 1=Assumes terminal format control All other bits must be zero.	

TABLE 6-10 (cont). Printer Device-Specific IORB Fields

IORB Item	Field	Definition	Use
I_ST	Software Status word	Shown below	Mapped from RCT hardware status.
I_RSR	Residual range	See Note	
<p>NOTE: For cases where original range is less than or equal to line length, the value in the residual range has the following meanings:</p> <p>0 - Completed space/print operation.                      original range - Neither space nor print occurred.                      original range-1 - Spacing occurred but no printing.                      Other residual range values imply spacing and partial printing.                      The preceding implies modulo line length.</p>			

PRINTER DEVICE-SPECIFIC RCT FIELDS

The fields in Table 6-11 were not defined previously since they are device-dependent.

Table 6-11. Printer Device-Specific RCT Fields

Item	Field	Definition	Use
R_TYP	Device type	2000-2007	Identifies device type.
R_STTS	Device status	Shown below	Status as read from device.

PRINTER RCT/IORB HARDWARE/SOFTWARE STATUS CODE MAPPING

Table 6-12 indicates the hardware/software status code mapping for printers.

Table 6-12. Printer RCT/IORB Hardware/Software Status Code Mapping

RCT R_STTS	IORB I_ST	Meaning If Bit Set
0	-	Device ready
1	-	Attention
-	-	
3	3	End-of-form
-	-	
-	-	
-	-	
-	-	
-	-	
-	-	
12	-	Corrected memory error
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Uncorrectable memory error/fatal error

### Disk Driver

A single disk driver supports the following disk devices: diskette, cartridge disk, and mass storage unit.

#### DISK DRIVER PROCESSING FOR DISKETTE

Disk driver processing for diskette is as follows:

- o The driver does not explicitly reference the volume id of the diskette; therefore, you must ensure that volumes addressed are on the proper drives.
- o All sector addresses used in the IORB are relative to track 0/sector 0.
- o The driver converts the volume relative sector number, defined in the IORB, into physical track and sector numbers, which it then sends to the device to define the operation. It does this by dividing the sector number by 26, which is the number of sectors/track on a diskette.
- o The driver can support more than one diskette device, as long as each device is configured at a different level. The reason for this is that each device requires an RCT and there is only one RCT per level.



- o Note that a diskette sector is 128 bytes long. If range is less than 128 bytes, a write command will zero fill the rest of the sector. If range is greater than 128 bytes on either a read or a write, the driver will read/write multiple sectors including switching to the next adjacent track, if necessary. Thus, from a user point of view, a read or write request for "range" number of bytes will take place regardless of the number of physical sectors or track switches involved.
- o If hardware errors occur, the operation (seek or read/write) will be retried up to eight times (four rereads and four rereads with recalibrate).
- o If the device is not ready, a return status of "device not ready" (5) will be returned.

#### Diskette Device-Specific IORB Fields

Table 6-13 defines diskette device specific IORB fields not previously defined. Other IORB fields are as already defined.

Table 6-13. Diskette Device-Specific IORB Fields

IORB Item	Field	Definition	Use
I_CT2	Function code	0=wait-online 1=write data 2=read data E=read disabled device	Previously defined.  Driver performs appropriate I/O commands to transfer the data.
I_DVS	Device-specific	Relative sector number	Driver converts this to physical track number and physical sector number on the track.
I_ST	Software status	Shown below	Hardware status word from diskette.
I_RSR	Residual range	$0 \leq$ original range	Residual range will always be equal to zero (i.e., transfer completed) unless there is either a hardware malfunction, or invalid track number accessed during a extended read or write operation.

## Diskette Device-Specific RCT Fields

The fields in Table 6-14 were not defined earlier due to their device-specific nature.

Table 6-14. Diskette Device-specific RCT Fields

Item	Field	Definition	Use
R_TYP	Device type	2010-2017	Identification of device.
R_STTS	Device status	Shown below	Status word 1 as read from device.

## Diskette RCT/IORB Hardware/Software Status Code Mapping

Table 6-15 indicates the hardware/software status code mapping for diskette.

Table 6-15. Diskette RCT/IORB Hardware/Software Status Code Mapping

RCT R_STTS	IORB I_ST	Meaning If Bit Set
0	-	Device ready
1	-	Attention
2	2	Data service rate error
3	3	Deleted file
4	4	Read error
5	5	Device fault
6	6	Missed data sync
7	7	Unsuccessful search
-	-	
-	-	
10	10	Seek error
12	-	Corrected memory error
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Uncorrectable memory error/fatal error

## DISK DRIVER PROCESSING FOR CARTRIDGE DISK

Disk driver processing for cartridge disk is as follows:

- o Sector size is 256 bytes.
- o The driver does not explicitly refer to the volume id of the disk; you must ensure that the volumes addressed are on the proper drives.
- o All sector addresses used in the IORB are relative to cylinder 0, track 0, sector 0.
- o The driver converts the volume relative sector number, defined in the IORB, into physical cylinder, track, and sector numbers, which it then sends to the device to define the operation. It does this by dividing the sector number by 24, which is the number of sectors/track on the disk.
- o Cartridge disk requires two RCTs, one for the fixed and one for the removable platter; each platter has its own LRN.
- o Cartridge disk driver logic combines seek and data transfer functions. When errors occur, eight attempts are made to correct an error, four seek/data transfers and four seek/data transfers with recalibrate.
- o Offset read (not write) capability is provided by specifying the desired displacement in the I\_RSR field of the IORB.
- o The offset read and offset write capabilities are not provided.
- o When the driver notes a change in the ready state, it -is disabled the device (by a software switch) and notifies the file manager, which executes the automatic volume recognition procedures.

See the Data File Organizations and Formats manual about data format on cartridge.

### Cartridge Disk Device-Specific IORB Fields

The fields in Table 6-16 are specific to cartridge disk; all other fields are previously defined.

Table 6-16. Cartridge Disk Device-Specific IORB Fields

IORB ITEM	Field	Definition	Use
I_CT2	Function Code	0=Wait online  1=Write  2=Read  E=Read disabled device	Specifies I/O operation. A read with range of zero performs a verify of the selected sector, with no data transfer to memory.
I_DVS	Device specific	Relative sector number	Driver converts this to physical cylinder, track, and sector number in order to locate the data needed.
I_ST	Software status word	See Table 6-22	Hardware status from disk.
I_RST	Residual range	0 ≤ original range	Prior to a read, an offset value may be specified here so that reading can begin at other than the physical sector boundary; after I/O operation the field contains the number of bytes not transferred in the operation.

Cartridge Disk Device-Specific RCT Fields

Table 6-17 defines the RCT fields whose contents are specific to the online cartridge disk driver.

Table 6-17. Cartridge Disk Device-Specific RCT Fields

RCT Item	Definition
R_TYP	Device type. The values are: 2330-2333, and have these meanings:  2330 Low density, removable only 2331 Low density, removable and fixed 2332 High density, removable only 2333 High density, removable and fixed
S_STTS	Device status; see Table 6-18.
R_FXPL	Bit 15 is set if this RCT is for a fixed platter.

Cartridge Disk RCT/IORB Hardware/Software Status Code Mapping

Table 6-18 indicates the hardware/software status code mapping for cartridge disk.

Table 6-18. Cartridge Disk RCT/IORB Hardware/Software Status Code Mapping

RCT R_STTS	IORB I_ST	Meaning If Bit Set
0	-	
1	-	
2	2	Over or underrun
3	3	Write protect error
4	4	Read error
5	5	Illegal seek
6	6	Missed data synchronization
7	7	Unsuccessful search
8	8	Missed clock pulse
9	9	Missed sector pulse
10	10	Seek error
11	-	
12	-	
13	-	
14	-	
15	15	Fatal error

## DISK DRIVER PROCESSING FOR MASS STORAGE UNIT

Disk driver processing for the mass storage unit is as follows:

- o Sector size is 256 bytes; there are 64 sectors per track.
- o The driver does not explicitly refer to the volume id of the disk pack, so you must ensure that the volumes addressed are on the correct drives.
- o All sector addresses in the OIRB are relative to cylinder 0, track 0, sector 0. There are four models:

5 tracks per cylinder

411 cylinders

823 cylinders

19 tracks per cylinder

411 cylinders

823 cylinders

The driver converts the volume relative sector number, defined in the IORB, into physical cylinder, track, and sector numbers, which it then sends to the device to define the operation.

It does this by dividing the sector number by 64, i.e., the number of sectors/track on a disk.

- o The mass storage unit requires only one RCT, and one LRN.
- o The driver combines seek and data functions. When errors occur, eight attempts are made to correct the error, four seek/data transfers, and four seek/data transfers with recalibrate.
- o Offset read (not write) capability is provided by specifying the required displacement in the I\_RSR field of the IORB.
- o The offset read and offset write capabilities are not provided.
- o When the driver notes a change in the ready state, it disables the device (by a software switch) and notifies the file manager to execute the automatic volume recognition procedures.

See Data File Organizations and Formats manual about data formats on the mass storage unit.

## Mass Storage Unit Device-Specific IORB Fields

The fields in Table 6-19 are specific to the mass storage unit; all other fields are as previously defined.

Table 6-19. Mass Storage Unit Device-Specific IORB Fields

IORB Item	Field	Definition	Use
I_CT2	Function Code	0=Wait online 1=Write 2=Read E=Read disabled device	Specifies I/O operation.
I_DVS	Device-specific	Relative sector number	Driver converts this to the physical cylinder, track, and sector number in order to locate the data needed.
I_RSR	Residual range	$0 \leq$ original range	After an I/O operation, the field contains the number of bytes not transferred.
I_ST	Software status word	See Table 6-21	Prior to an order, this field contains the high-order sector bits of the I_DVS field. After the operation, it contains the hardware status from device.

## Mass Storage Unit Device-Specific RCT Fields

Table 6-20 defines the RCT fields whose contents are specified to the disk driver for mass storage units.

Table 6-20. Mass Storage Unit Device-Specific RCT Fields

RCT Item	Description
R_TYP	Device type. The values are: 2360-2363, and have these meanings:  2360 411 cylinders, 5 tracks/cylinder 2361 823 cylinders, 5 tracks/cylinder 2362 411 cylinders, 19 tracks/cylinder 2363 823 cylinders, 19 tracks/cylinder
R_STTS	Device status; see Table 6-21.

Mass Storage Unit RCT/IORB Hardware/Software Status Code Mapping

Table 6-21 indicates the hardware/software status code mapping for the mass storage unit.

Table 6-21. Mass Storage Unit Status Code Mapping

RCT R_STTS	IORB R_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

ASK/KSR Drivers

The standard directive DEVICE is used at system building to configure a KSR or ASR terminal. The keyboard/printer functions of an ASR are supported; the paper tape reader/punch functions are not. Thus, the k-field within I\_DVS word (Table 6-22) 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 available for the keyboard/printer are described below.

#### KEYBOARD INPUT

- o 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.
- o Keyboard control (line feed, carriage return, etc.) is definable in the IORB.
- o Editing characters can control input:
  - @ Deletes the previous character entered.  
(control X) (an ASCII CAN code)
  - \ Deletes all the previous characters entered on the same input line.

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

- o Printer output is accepted until end-of-range.
- o Time-out period for keyboard/printer operation is 5 minutes.

ASR/KSR DEVICE-SPECIFIC IORB FIELDS

Table 6-22 shows the values of device-specific fields for ASR/KSR devices.

Table 6-22. ASR/KSR Device-Specific IORB Fields

IORB Item	Field	Definition Keyboard/ Printer	Use
I_CT2	Function code	1=write            A=Connect 2=read             B=Disconnect 3=break notifica- tion	Used by driver to complete the description of the requested I/O function.
I_DVS	Device-Specific	3 4 5 6 7 8 9 10 11 12 13 14 15 S F T 0 Q D K E L C 0 A H  S=0 Must be zero.  F=0 Assumes line printer format control. =1 Assumes terminal format control.  T=0 Use control characters as defined to adapter. =1 Treat all characters as data; perform no special character action.  Q=0 Stop output operation immediately upon "attention" detection if the detected character has a No Stop Bit status (e.g., a "Break" key). =1 Post "attention" but allow output data transfer to complete.  D=0 Read attention character with input (if present).	

Table 6-22 (cont). ASR/KSR Device-Specific IORB Fields

IORB Item	Field	Definition Keyboard/ Printer	Use
I_DVS (cont)	Device (cont)	=1 Discard attention character on input.  K=0 Transfer to keyboard/printer.  E=0 Do not echo keyboard input.  =1 Echo keyboard input.  L=0 No line feed at end of transfer.  =1 Line feed at end of transfer.  C=0 Carriage return at end of transfer.  =1 No carriage return at end of transfer mode.  A=0 Must be zero.  H=0 Disconnect without phone hang up.  =1 Disconnect with 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.

ASR/KSR DEVICE-SPECIFIC RCT FIELDS

The fields in Table 6-23 were not defined earlier due to their device-specific nature.

Table 6-23. ASR/KSR Device-Specific RCT Fields

Item	Field	Definition	Use
R_TYP	Device Status	2018-201F	Identifies device type; value read from device at initialization time.
R_STTS	Device Status	Shown below	Last status word read from device.

The following kinds of processing data are kept in the RCT workspace for the terminal.

- o Configuration word A
- o Configuration word B
- o Identifier for last task to receive a "break" from this terminal

ASR/KSR RCT/IORB HARDWARE/SOFTWARE STATUS CODING MAPPING

Table 6-24 indicates the status code mapping for ASR/KSR devices.

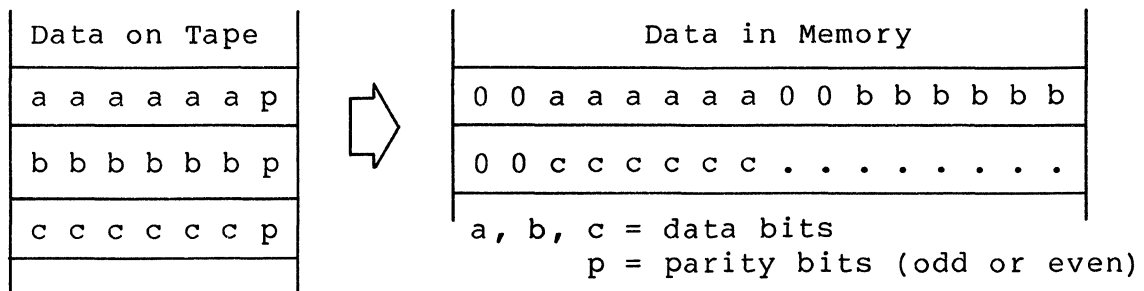
Table 6-24. ASR/KSR RCT/IORB Hardware/Software Status Code Mapping

RCT R_STTS	IORB I_ST	Meaning If Bit Set
0	-	Device ready
1	-	Attention
2	2	Data service rate error
3	3	Parity error (even)
-	-	
5	5	No stop bit
-	6	Long record
-	7	Checksum error
8	8	Control character <u>number 2</u> termination
9	9	Control character <u>number 3</u> termination
-	-	
-	-	
12	-	Corrected memory error
13	15	Nonexistent resource/fatal error
14	15	Bus parity error/fatal error
15	15	Uncorrectable memory error/fatal error

## Magnetic Tape Driver

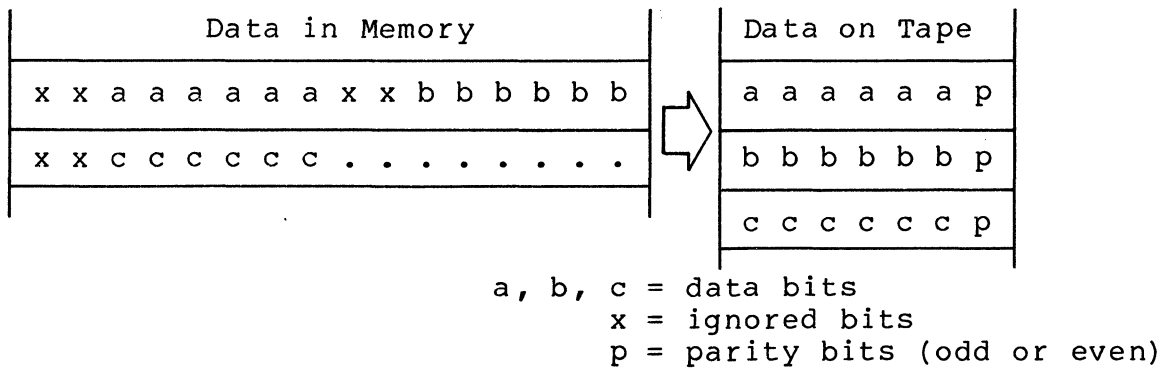
The magnetic tape driver manages all standard data transfer requests to and from 9-track phase encoded (PE), and 7- and 9-track nonreturn to zero inverted (NRZI) tape drives on one or more magnetic tape controllers. The tape drive characteristics supported by this tape driver are shown in Table 6-25.

Figure 6-5 illustrates 6-bit and packed modes on 7-track tape and in memory in the transfer of data between the tape device and memory.



6-bit mode on 7-track (read tape into memory)

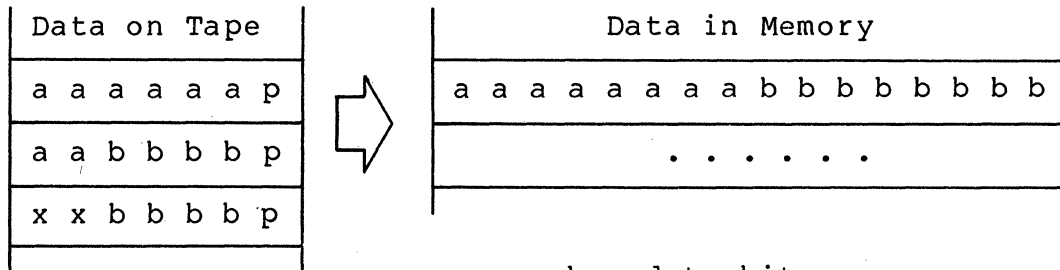
---



6-bit mode on 7-track tape (write tape from memory)

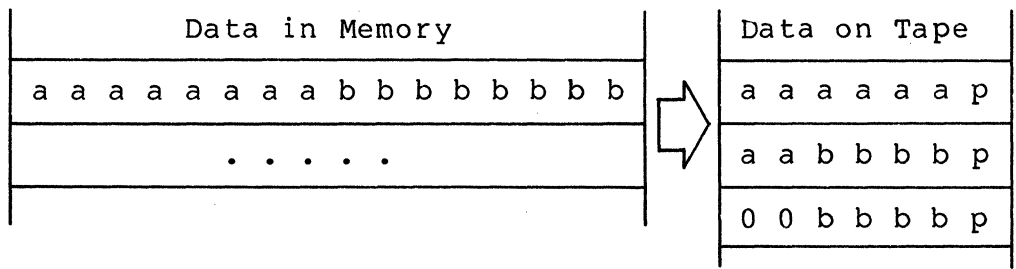
---

Figure 6-5. Packed and 6-Bit Modes on 7-Track Tape



a, b = data bits  
 x = ignored bits  
 p = parity bit (odd)

Packed mode on 7-track tape (read tape into memory)



a, b = data bits  
 p = parity bits (odd)

Packed mode on 7-track tape (write tape from memory)

Figure 6-5 (cont). Packed and 6-Bit Modes on 7-Track Tape

Table 6-25. 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 <sup>a</sup>	X	X	-	X	X	X	X	X	X	X

<sup>a</sup>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 eight 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 6-26 are specific to magnetic tape devices. All other IORB fields are defined in previous subsections.

Table 6-26. Magnetic Tape Device-Specific 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 = Position file (see range)								
I_DVS	Device specific	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%; text-align: center;">0</td> <td style="width: 50%;"></td> <td style="width: 15%; text-align: center;">12</td> <td style="width: 25%; text-align: center;">13-15</td> </tr> <tr> <td style="text-align: center;">0 0 0 0 0 0 0 0 0 0 0 0 0 0</td> <td style="text-align: center;">I</td> <td colspan="2" style="text-align: center;">mode</td> </tr> </table> <p>I: 0=Normal American National Standard block sizes</p> <p>1=Inhibit sensing for American National Standard block size</p> <p>mode: 0 = 9-track tape; or 7-track in odd parity 6-bit mode</p> <p>1 = 7-track tape in even parity 6-bit mode</p> <p>2 = 7-track tape in packed mode</p>	0		12	13-15	0 0 0 0 0 0 0 0 0 0 0 0 0 0	I	mode	
0		12	13-15							
0 0 0 0 0 0 0 0 0 0 0 0 0 0	I	mode								



Table 6-26 (cont). Magnetic Tape Device-Specific IORB Fields

Item	Field	Definition
I_RSR	Range	<p>Write: 1 through 7FFF</p> <p>Read: 0 means verify; 1 through 7FFF<sub>16</sub> is valid</p> <p>Position by block: Negative is backspace 0 is illegal</p> <p>Positive is forward space</p> <p>Position by file: -2 = Rewind unload -1 = Rewind 0 = Backspace to tapemark 1 = Forward space to tapemark</p>
I_RNG	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 DEVICE-SPECIFIC RCT FIELDS

The device-specific fields in an RCT for magnetic tape devices are given in Table 6-27.

Table 6-27. Magnetic Tape Device-Specific RCT Fields

Item	Definition
R_TYP	Device type; values are 2045 - 207A
R_STTS	Status word
R_SW2	Tape status word 2
R_TMCT	Tape mark count
R_BLCT	Block count

#### MAGNETIC TAPE RCT/IORB HARDWARE/SOFTWARE STATUS CODE MAPPING

The hardware/software status code mapping for magnetic tape devices is shown in Table 6-28.

Table 6-28. Magnetic Tape RCT/IORB 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

## SECTION 7

### TRAP HANDLING

A trap is a special software- or hardware-related condition that may occur during execution of a task. The Level 6 hardware/firmware responds to many trap conditions. The design of any application program should provide that when a trap occurs, the hardware/software response will include calling a dedicated software routine (a trap handler) to react to the trap. When trap handlers are provided, handling the trap is invisible to the task that caused the trap.

See the System Building manual for detailed information about the system building directives that are referred to below.

Using the trap facilities of Level 6 hardware/firmware and the Honeywell trap handlers (Floating-point and Scientific Branch Simulators) require that you:

- o Provide enough trap save areas (TSA's) in the system memory pool, by coding a value for the TSA parameter of the CLM directive SYS. The default value for this parameter results in six TSA's.
- o Include any Honeywell-supplied trap handlers, and any user-written generalized trap handling routine, in your configured software. These are the initial trap handlers.
- o Provide a generalized trap handling routine (see Table 7-1). The code for each task must identify this trap handler by providing its address in the \$TRPHD macro call.
- o On a task-by-task basis, enable all the trap numbers to be handled for the task, by including a \$ENTRP macro call in the code for each task.

## TRAP CONDITIONS DURING TASK EXECUTION

A trap handler that was configured at initialization time (by a SYS directive with an SSIP, CSIP, or DSIP argument, or by an LDBU directive) will handle a trap as follows. If more than one trap handler is connected to the same trap, the order of their execution is: (1) LDBU-created trap handlers are executed first, in reverse order of their definition. (2) SYS directive-created trap handlers are executed following execution of any LDBU-created trap handlers.

The trap may then be handled (i.e., passed along) by a series of trap handlers. If those trap handlers do not handle the trap, one of the following conditions (trap enabled or trap not enabled) occurs, according to whether the trap number of the trap condition was enabled by the task.

### Trap Enabled

\* If the task's code contains the \$ENTRP macro call with the pertinent trap number for this condition, the trap handler is invoked to execute at the priority level of the task in which the trap occurred. The trap handler responds to the trap, and exits via an RTT instruction to return to the task's code at the next sequential instruction following the one that caused the trap. The trap handler always runs in privileged mode.

Should another trap condition occur during the execution of the handler the sequence is repeated unless the nested traps are the same type, in which case the sequence is as described for traps that are not enabled.

### Trap Not Enabled

When a trap condition occurs in task code that has not enabled this particular trap, an error message is written to the error-out file; the delete bit in the task control block is reset, the task is terminated, but the task's resources (memory and peripherals) are not released. Thus, a memory dump can be taken so that the error condition can be examined.

The usual way to continue processing after a trap-not-enabled condition is to issue a new process (NPROC) command against the group containing the terminated task.

## CONTENTS OF TRAP-RELATED MEMORY AREAS

In examining a dump to determine the nature of a trap condition, check particularly the contents of the TSA. The trap handling mechanism is illustrated in Figure 7-1; the contents of the trap save area are described in the following pages.

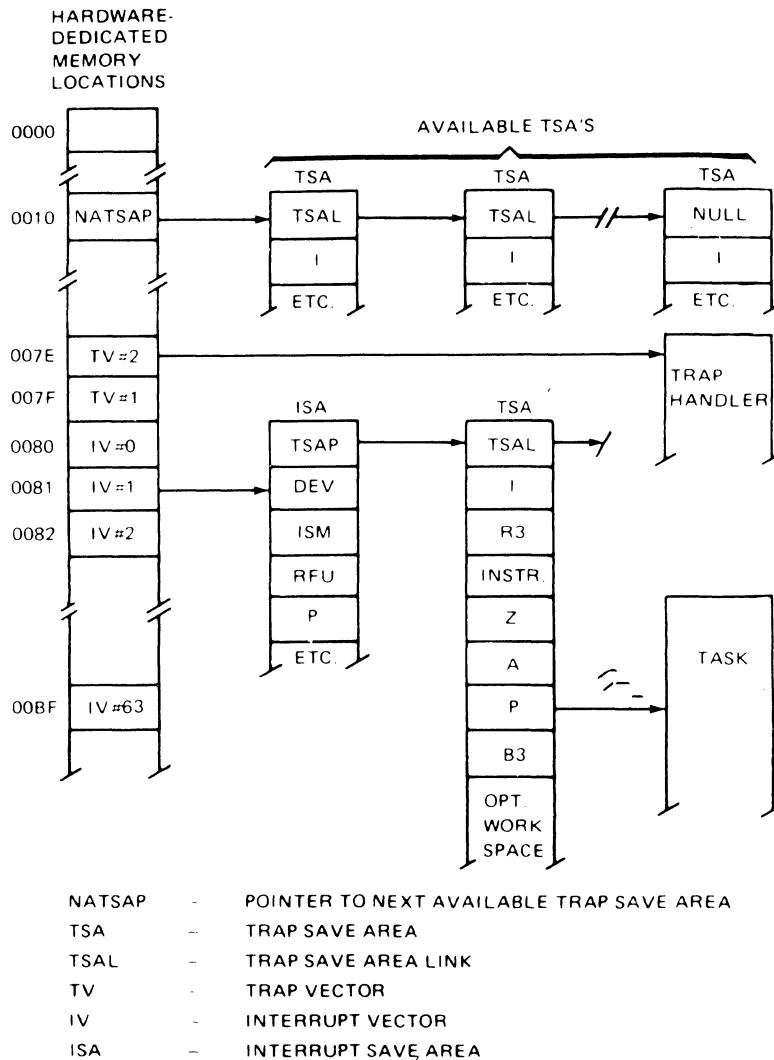
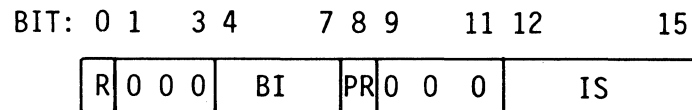


Figure 7-1. Trap Handling Mechanism

- o Trap Save Area Link (TSAL) - When the trap save area resides in the "available" pool, TSAL points to the next trap save area in the pool; the TSAL of the last trap save area in the pool contains a null pointer. When the trap save area is in use (i.e., connected to an interrupt save area), TSAL contains a null pointer (if this is the only or last, trap save area connected to this interrupt save area) or it points to the next trap save area connected to this interrupt save area.

- o I-Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler. The high-order byte contains the quantity ( $40_{16}$  - trap number). This byte is stored by software on Models 33, 6/34, 6/36 central processors, and by firmware on Models 43 and 47 processors.
- o R3-Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler.
- o Instruction - The hardware/firmware stores the instruction associated with the trap. If a multiword instruction is involved, the first word is saved.
- o Z-Word - This word contains miscellaneous information relative to the trap. The format of this word is shown below:



R

If R=0, the saved contents of the A-word are meaningful relative to this trap condition; if R=1, the saved contents of the A-word are not meaningful.

BI

4-bit field that is meaningful only when an indexed bit or byte instruction is associated with the trap. If an indexed bit instruction is involved BI indicates the four low-order bits of the associated index register; bit 7 of BI stores the least significant bit. If an indexed byte instruction is involved, bit 4 of BI indicates the least significant bit of the associated index register and bits 5 through 7 are zeros.

PR

The privilege state of the task was running when the trap occurred. 0 = nonprivileged state; 1 = privileged state. The value is taken from the P-bit of the S-register.

IS

The length (in words) of the instruction associated with the trap. If a multiword instruction is involved and the trap occurs before the entire instruction has been fetched, IS indicates the number of words that were fetched, before the trap.

- o A-Word. In many cases, this word contains an address associated with the trap. (This word is not meaningful if bit 0 of the Z-word contains a 1.) The nature of the saved address is governed by the specific trap condition and the specific instruction associated with the trap. Details relative to each trap condition are in Table 7-1.
- o Program Counter - The contents of the program counter are saved by the hardware/firmware when a trap occurs. This is the address to which a return is made when the trap handler completes. In most cases the program counter will point to the instruction or location following the instruction associated with the trap. However, when an input/output instruction is involved, the program counter may point to an address within the instruction; in this case, the trap handler must modify this word before issuing a return to "normal" task processing.
- o B3-Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler; as the trap handler is entered, the B3-register points to the A-word in the trap save area.

Trap save areas (TSA's) are 64 words long in SAF mode, and 104 words in LAF mode. The discussions about memory dumps in the Program Execution and Checkout manual include discussions about optional work space used by trap handlers and about saving registers for a Monitor macro call.

#### Pointer to Next Available Trap Save Area (NATSAP)

Memory location 0010<sub>16</sub> (NATSAP in Figure 7-1) points to the first trap save area in the "available" pool. If there are no trap save areas in this pool, NATSAP contains a null pointer.

When a trap occurs, hardware/firmware examines the pointer in NATSAP to ascertain the address of the start of the trap save area in which information will be stored for use by the trap handler (if any) that reacts to the trap.

## Trap Vector

A trap vector is a hardware-dedicated memory location that can be used to point to the entry address of a trap handler. Each trap vector is associated with a specific trap condition and can point to only one trap handler's entry address.

## Trap Save Areas

Initially, when processing of an application begins, all trap save areas exist in a linked "pool", which is pointed to by memory location 0010<sub>16</sub> (NATSAP in Figure 7-1). All trap save areas remain in this pool until a trap condition occurs within a running task, at which point the hardware/firmware (1) stores information in the first available trap save area in the pool, (2) links this trap save area to the interrupt save area for the priority level of the task that was running when the trap occurred, and (3) unlinks this trap save area from the pool. Later, after the trap handler (if any) has completed its work, the trap save area is returned to the pool of available trap save areas. Thus, at any time, a given trap save area is either in the pool of available trap save areas or in use because of a trap condition

The trap save areas reside in the system pool.

## Interrupt Vector

An interrupt vector is a hardware-dedicated memory location that (if "connected") points to the second entry of the interrupt save area for a specific priority level. (See Figure 7-1, where interrupt vector 1 is pointing to an interrupt save area.)

## Interrupt Save Area (ISA)

An interrupt save area is a block of memory in which hardware/firmware performs a context save when an interrupt occurs.

When a trap occurs, if the appropriate trap handler is available in the application, the first word (TSAP) of the interrupt save area (for the current priority level) is set to point to the link word (TSAL) of the trap save area in which hardware/firmware has just stored information relative to the trap (see Figure 7-1). TSAP is subsequently used by the trap handler to gain access to the trap save area.



Table 7-1. Contents of Selected Words of Trap Save Area When Trap Occurs

Trap Number and Condition	Specific Event	Saved Instruction	Saved Z-Word	Saved A-Word	Saved Program Counter
1 Monitor call; implicitly handled by the operating system	MCL instruction	0001	80xl	Unspecified	Next location
1 Software trap	PI	Unspecified	Unspecified	Unspecified	Unspecified
2 Breakpoint instruction	BRK instruction	0002	80xl	Unspecified	Next location
3 Scientific floating point operation when SIP hardware not in system	Scientific instruction whose address expression generates a reference to a register	Instruction that caused trap	80xl	Unspecified	Unspecified
	Scientific instruction whose address expression generates a reference to memory	First word on in- struction that caused trap	00xy	Effective address generated by address expression <sup>b</sup>	Next instruction
4 Unrecognized op code	Instruction not recognized by CPU or SIP	Same as for trap 5	Same as for trap 5	Effective address of trap operand	Next instruction
5 Scientific Branch instruction when SIP hardware not in system, or any other operation not supported	Undefined instruction whose address expression generates a reference to a register	Instruction that caused trap	80xl	Unspecified	Next instruction
	Undefined instruction whose address expression generates a reference to memory	First word of in- struction that caused trap	00xy	Effective address generated by ad- dress expression	Next instruction
6 Integer arithmetic overflow (with appropriate overflow trap enable bit of M1 register set to 1)	Overflow of target R-regis- ter during execution of instruction whose address expression generates a reference to a register	Instruction that caused trap	80xl	Unspecified	Next instruction
	Overflow of target R-regis- ter during execution of instruction whose address expression generates a reference to memory	First word of in- struction that caused trap	00xy	Effective address generated by address	Next instruction
7 Scientific divide by zero	A scientific divide (SDV) instruction has a divisor of zero	Unspecified	Unspecified	Pointer to scien- tific instruction that caused trap	Next CPU instruction
8 Exponential overflow	A scientific operation produces an exponent greater than +63	Unspecified	Unspecified	Pointer to scien- tific instruction that caused trap	Next CPU instruction

Table 7-1 (cont). Contents of Selected Words of Trap Save Area When Trap Occurs

Trap Number and Condition	Specific Event	Saved Instruction	Saved Z-Word	Saved A-Word	Saved Program Counter
13 Unprivileged use of privileged operation	HLT, RTCN, RTCF, WDTN, or WDTF <sup>c</sup> instruction	Instruction that caused trap	8001	Unspecified	Next location
	LEV instruction whose address expression generates reference to a register	Instruction that caused trap	8001	Unspecified	Next instruction
	LEV instruction whose address expression generates a reference to memory	First word of instruction that caused trap	000y	Effective address generated by address expression	Next instruction
	Input/output instruction whose first-word address expression generates a reference to a register	First word of instruction that caused trap	8002	Unspecified	First word of instruction that caused trap, plus 2
	Input/output instruction whose first-word address expression generates a reference to memory	First word of instruction that caused trap	000y	Effective address generated by address expression	First word of instruction that caused trap, plus y
15 Reference to unavailable resource	Instruction whose address expression generates a reference to (1) a memory address higher than the highest memory address available but less than 64K or (2) through indexing, a "wrap-around" memory address higher than 64K or less than 0	First word of instruction that caused trap	00xy	Effective address generated by address expression	Next instruction
	Input/output instruction that specifies an improper channel number; address expression generates a reference to a register	First word of instruction that caused trap	808y	Unspecified	First word of instruction that caused trap, plus y
	Input/output instruction that specifies an improper channel number; address expression generates a reference to memory	First word of instruction that caused trap	008y	Effective address generated by address expression	First word of instruction that caused trap, plus y
	WDTN or WDTF instruction and watchdog timer not installed	0006 (WDTN); 0007 (WDTF)	80x1	Unspecified	Next location

Table 7-1 (cont). Contents of Selected Words of Trap  
Save Area When Trap Occurs

Trap Number and Condition	Specific Event	Saved Instruction	Saved Z-Word	Saved A-Word	Saved Program Counter
16 Program logic error	RTT instruction while TSAP contains a null pointer	Instruction that caused trap	80x1	Unspecified	Next instruction
	Instruction whose address expression illegally generates reference to a register (i.e., this instruction is not permitted to use a register address syllable)	Instruction that caused trap	80x1	Unspecified	Next instruction
17 Bus parity or memory error	Bus parity error or unrecoverable memory data error	Unspecified	Unspecified	Unspecified	Unspecified
19 Scientific underflow	An operation produces an exponent value of less than -64 while the associated enable bit in register M5 is set.	Unspecified	Unspecified	Pointer to scientific instruction that caused trap	Next CPU instruction
20 Program error (SIP)	A program error is detected by the SIP	Unspecified	Unspecified	Pointer to scientific instruction that caused trap	Next CPU instruction
21 Scientific significance error	An integer is truncated during floating point to integer conversion while the associated enable bit in register M5 is set.	Unspecified	Unspecified	Pointer to scientific instruction that caused trap	Next CPU instruction
22 Scientific precision	The nonzero portion of a fraction is truncated while the associated enable bit of register M5 is set.	Unspecified	Unspecified	Pointer to scientific instruction that caused trap	Next CPU instruction
23 Nonexistent resource error	The SIP or CIP attempts a write or read request bus cycle and receives a NAK	Unspecified	Unspecified	Pointer to scientific instruction that caused trap	Next CPU instruction
24 Noncorrectable memory error or Megabus error	A read error occurs which the EDAC cannot correct, or the SIP or CIP detects a parity error.	Unspecified	Unspecified	Unspecified	Unspecified
25 CIP divide by zero	The divisor of a decimal divide instruction (DDV) is equal to zero.	Unspecified	Unspecified	Pointer to CIP instruction that caused trap	Next CPU instruction

Table 7-1 (cont). Contents of Selected Words of Trap  
Save Area When Trap Occurs

Trap Number and Condition	Specific Event	Saved Instruction	Saved Z-word	Saved A-word	Saved Program Counter
<p>26<sup>d</sup> CIP illegal specification</p>	<p>Any of the following:</p> <ul style="list-style-type: none"> <li>o Undefined CIP op code detected.</li> <li>o One or both descriptors of an alphanumeric instruction is packed decimal.</li> <li>o Decimal operand has a zero length.</li> <li>o Operand in an Edit, VRF, or SRH instruction has a zero length.</li> <li>o A separate signed decimal operand consists of only a sign (i.e., no digits).</li> <li>o In a Move and Edit instruction, the length of the receiving field was not exhausted, but either there are no micro-ops or the sending field length is exhausted.</li> <li>o A second data descriptor specifies an IMO, except for DCM and ACM instructions.</li> <li>o The first data descriptor in a DSH specifies an IMO.</li> <li>o A third data descriptor specifies an IMO.</li> <li>o In an SRH instruction, the search length list is less than the search argument list, or operand length less than operand element length.</li> <li>o In a VRF instruction, verify list length is less than verify argument length, or operand length is less than operand element length.</li> </ul>	<p>Unspecified</p>	<p>Unspecified</p>	<p>Pointer to CIP instruction that caused trap</p>	<p>Next CPU instruction</p>

Table 7-1 (cont). Contents of Selected Words of Trap  
Save Area When Trap Occurs

Trap Number and Condition	Specific Event	Saved Instruction	Saved Z-Word	Saved A-Word	Saved Program Counter
27 <sup>d</sup> CIP illegal character	Any of the following: <ul style="list-style-type: none"> <li>o Illegal decimal digit detected (low-order four bits are not 0 through 9).</li> <li>o Illegal sign digit detected (not a recognized sign value).</li> <li>o Illegal overpunch digit detected.</li> </ul>	Unspecified	Unspecified	Pointer to CIP instruction that caused trap	Next CPU instruction
28 <sup>d</sup> CIP truncation error	Receiving field of an alphanumeric instruction cannot contain all characters of the result. Whether or not a trap occurs, the receiving field is altered to contain the leftmost part of the result and the CI (TR) is set.	Unspecified	Unspecified	Pointer to CIP instruction that caused the trap	Next CPU instruction
29 <sup>d</sup> CIP overflow	Any of the following: <ul style="list-style-type: none"> <li>o Receiving field of a decimal instruction cannot contain all significant digits of the result.</li> <li>o During a Shift Lift instruction, a nonzero digit is shifted out.</li> </ul>	Unspecified	Unspecified	Pointer to CIP instruction that caused the trap	Next CPU instruction
49 Software trap	UW command	Unspecified	Unspecified	Unspecified	Unspecified

<sup>a</sup>The Z-word format is described earlier in this section.

<sup>b</sup>This is the address of the high-order (leftmost) end of a two-word operand.

<sup>c</sup>If the watchdog timer is not present, this instruction causes a trap to vector 15 regardless of the privilege mode of the central processor.

<sup>d</sup>The Assembly Language Reference manual describes CIP trap handling in detail.

## HONEYWELL-SUPPLIED TRAP HANDLERS

The following software components provide trap handling facilities: Debug program, Commercial Simulator, and two versions of the Floating-Point Simulator and the Scientific Branch Simulator - one for single-precision and one for double-precision operations. The double-precision versions represent full hardware simulation. The simulators are used when the hardware feature for scientific instruction processing is not installed. The Assembly Language Reference manual describes commercial and scientific instructions.

### Trap Handling by the Debug Program

The Debug program operates as a unique task group identified by \$D. It is loaded by use of the spawn group (SG) command which also specifies the terminal from which the debug directives are issued. (For a detailed description of the Debug program, see the Program Execution and Checkout manual.) Once the Debug program is loaded, you may set, clear, or print breakpoints in the task code by use of Debug directives.

When the application program is executed, the Debug program is activated by trap number 2 which occurs each time a breakpoint is encountered. The action specified by the Debug directive (for that breakpoint) will then be executed. For example, designated memory locations may be printed out and execution of the application program continued without operator intervention. Alternatively, the Debug program may be placed in the interactive mode. In which case, you may use the Debug directives to display, change, and dump memory or registers. Information may be printed on a console or a line printer.

### Commercial Simulator

The Commercial Simulator is the software component that simulates the capabilities of the Honeywell Commercial Central Processor Model on other models that do not have native commercial instructions.

The Commercial Simulator reacts to trap vector 5 (uninstalled instructions). It permits Intermediate COBOL, RPG, and assembly language programs to simulate the use of commercial instructions.

The following programming considerations concern the Commercial Simulator:

- o The SIP/CIP arguments of the system building SYS directive indicate existence of the Commercial Simulator.

- o All other operation codes not handled by the Commercial Simulator are passed to the next trap handler for trap number 5.

### Floating-Point Simulator

The Floating-Point Simulator reacts to trap number 3 (scientific operation not in hardware). Trap number 3 (i.e., a trap to trap vector 3) occurs each time the central processor encounters a scientific instruction during the processing of a task.

While processing scientific instructions, the simulator provides automatic alignment of the operand's hexadecimal mantissas. It achieves maximum available precision by requiring that mantissas have no leading zeros (i.e., all mantissas must be normalized).

Note the following programming considerations relative to the simulator:

- o The choice of the single-precision version (SSIP) or the double-precision version (DSIP) of the simulator is indicated in the SIP argument of the SYS directive for system building.

For SSIP only:

- o The simulator uses the R4 register, the R5 register, and the R7 register to simulate a scientific register. A task that includes scientific instructions should dedicate these three registers to the simulator's use.

For SSIP and DSIP:

- o During its processing, the simulator may encounter an error condition specific to a scientific instruction: the following actions can occur:
  - The simulator will consult trap vector 5 if it encounters (1) a scientific instruction it does not support or (2) if a supported scientific instruction generates a reference to a register other than the simulated scientific register or the R6 register.
  - The simulator will consult trap vector 7 if an SDV (Scientific Divide) instruction has a divisor of 0. The instruction will not be executed.
  - The simulator will consult trap vector 8 if execution of a scientific instruction produces exponential overflow. The instruction will have been executed.

- To use a software routine to react to any of these trap conditions, you must provide your own trap handler. The simulator will be invoked to handle traps caused by execution of scientific instructions only if the trap numbers have been enabled for the task executing those instructions.
- o No "overflow trap enable" bit of the M register should set to 1 as the simulator begins operation.

### Scientific Branch Simulator

The Scientific Branch Simulator reacts to traps to trap vector 5. It provides FORTRAN and assembly language programs with the means to simulate the use of the scientific branch instructions.

Note the following programming considerations relative to the simulator:

- o The choice of the single-precision version (SSIP), or the double-precision (DSIP) of the simulator is indicated in the SIP argument of the system building SYS directive.

For SSIP only:

- o The simulator uses registers R4, R5, and R7 as scientific accumulator (S1) for comparisons; it uses R1, R2, and R3 as work registers.
- o The simulator uses the G, L, and U bits of the I register to determine if the branch condition is true or false. When a normal return is made to the user program, the branch will be executed if the branch condition is true; otherwise, the next sequential instruction following the one that was trapped will be executed.

For both SSIP and DSIP:

- o All other operation codes not handled by the Floating-Point Simulator or the Scientific Branch Simulator are passed to the trap handler for trap 5.

### Software Generated Traps

The following trap numbers (see Table 7-1) supports the task interrupt (break) function on KSR and TTY type devices.

- 1 - User typed in a PI response following the **\*\*BREAK\*\*** prompter message.



- 48 - The user pressed the break or interrupt key on a KSR or TTY device while trap 48 was enabled. The user will receive a software trap as well as the opportunity to respond to the **\*\*BREAK\*\*** message.
- 49 - User typed in a UW response following a **\*\*BREAK\*\*** prompter message.

For a user program to receive one of these traps, it must have been enabled for the trap with the \$TRPHD and \$ENTRP macro calls. The A- and Z-words in the TSA are meaningless; all other words are the same as for a hardware trap.

#### USER-WRITTEN TRAP HANDLERS

The system allows traps to be serviced in two ways:

1. Trap handler connect (\$TRPHD) and enable user trap (\$ENTRP) macro calls.
2. Trap handler directly attached to trap vector.

The macro call approach requires that each task explicitly use the macro calls, and either include a user-supplied trap handler in its bound unit (trap handler is part of task group dynamic memory pool) or reference an entry (linked with an EDEF directive) of a Monitor extension trap handler (trap handler is part of resident system). The user-supplied handler receives the TSA and register context exactly as if it was directly connected to the trap vector, but the Monitor has intercepted and analyzed the trap.

The direct attachment approach requires a Monitor extension trap handler. This approach bypasses the Monitor overhead to analyze the trap, but requires the trap handler to be permanently memory resident and to be attached to the trap vector by user code.

In either approach, the trap handlers must handle situations described below under "Trap Handlers Designed as Program Extensions" and "Programming Considerations for User-Written Trap Handlers."

#### Trap Handlers Designed as Monitor Extensions

It is assumed that all Honeywell and possibly some user-written trap handlers attached to the vector can encounter situations which should be passed to the system default trap handler. Also, several handlers may process the same trap. To pass a trap in GCOS 6 from one handler attached to a trap vector to the next handler:

1. Load the trap handler, with an LDBU directive. The trap handler becomes a permanent part of the system and will allow privileged mode operation. The system, at system building, does implicit LDBU's for SIP/Commercial Simulator handlers if the hardware is not present, and SSIP and DSIP arguments were specified in a SYS directive.
2. Write the handler to include initialization subroutine table (IST) code that will execute when the LDBU load operation occurs and save the current address contents of the trap vector(s) to be simulated, inserting its own pointer(s) instead.
3. Code the user-written simulator to save the contents of all registers upon entry so that if the trap should be passed to the next trap handler, this handler can:
  - a. Restore all saved registers.
  - b. Execute a jump-indirect through the location containing the pointer of the next handler saved in step 2 above. The J-bit in the \$M1 register must be off when the jump-indirect is executed.

The rule is that each trap handler must get exactly the same information in registers and TSA that it would have received if it was the first trap handler accessed.

Directly attached handlers are assumed to run in privileged mode. All handlers must contain IST code to insure that the trap vector pointer is to an odd location so that privileged mode is established. This is especially important when handlers are "chained" together on the same trap vector, since privilege must be set for all trap handlers regardless of the order in which they are loaded.

\*

#### PROGRAMMING CONSIDERATIONS FOR USER-WRITTEN TRAP HANDLERS

- o Any trap described in Table 7-1 can occur if the associated software- or hardware-related condition exists. As you are creating your application, you must consider which traps you wish to service with trap handlers.
- o A trap handler operates at the same priority level as the task whose execution caused the trap. The trap handler always executes in privileged mode.

When the trap handler has finished its work, the privilege mode for "normal" task processing is restored as follows: A logical AND operation is performed using (1) the current setting of the P-bit of the S-register and (2) the saved value of the privilege bit in the Z-word of the trap save area.

- o When a trap occurs, the hardware/firmware saves the task related contents of the I-register, the R3 register, and the B3 register in the trap save area. The trap handler is free to use these registers.
- o See Table 7-1 for a description of the contents of selected words in the trap save area when various traps occur.
- o Upon entry to the user trap handler, the J-bit in the M1 register is arbitrarily turned off. Other bits in the M1 register remain as they were when the trap occurred. Register B3 contains a pointer to the A word in the TSA. Register R3 contains the vector number of the trap.

\*

- o Traps that occur within the user trap handler abort the task if they are the same type as the trap currently being processed. This is done to prevent all TSA's from being tied up due to recursive traps, and to prevent traps within the MCL interface from going to the user trap handler.
- o Every trap handler should be reentrant; i.e., it should not use an internal work area to store interim information, since this information could be lost if an interrupt occurs and, later, the same trap handler is called upon to execute at a different priority level.
- o If you choose to define instructions of your own and have them interpreted by a trap handler connected to trap vector 5, you should limit the instructions to the user-reserved subset of the generic instructions. The following diagram illustrates the memory format of generic instructions.



## APPENDIX A

### DATA STRUCTURE FORMATS

This appendix describes the data structures referred to in Sections 2, 3, and 4. The following structures are described:

- o Clock request block (CRB)
- o File information block (FIB)
- o Input/output request block (IORB)
- o Task request block (TRB)
- o Parameter block
- o Wait list
- o Semaphore request block (SRB)
- o Message group request blocks (MGCRB, MGIRB, MGRRB)

Any of the structures can be hand coded. The CRB, FIB, SRB, and TRB can also be generated by macro calls or defined by macro call templates. The parameter block, input/output request block, wait list and message group request blocks, can be generated by macro calls.

The first four items of the request blocks have an identical format (but slightly different contents, depending on the block type) as shown in Figure A-1. Later diagrams show the format of each block type; tables show the contents of the block entries.

The data structures described here are as they appear for short address format (SAF) central processors, namely, one word items; the same structures for long address format (LAF) equipment would be 2-word entries for all pointers.

The first field (-SAF or -1) of a request block need be present only when the request block pointer/semaphore name is needed.

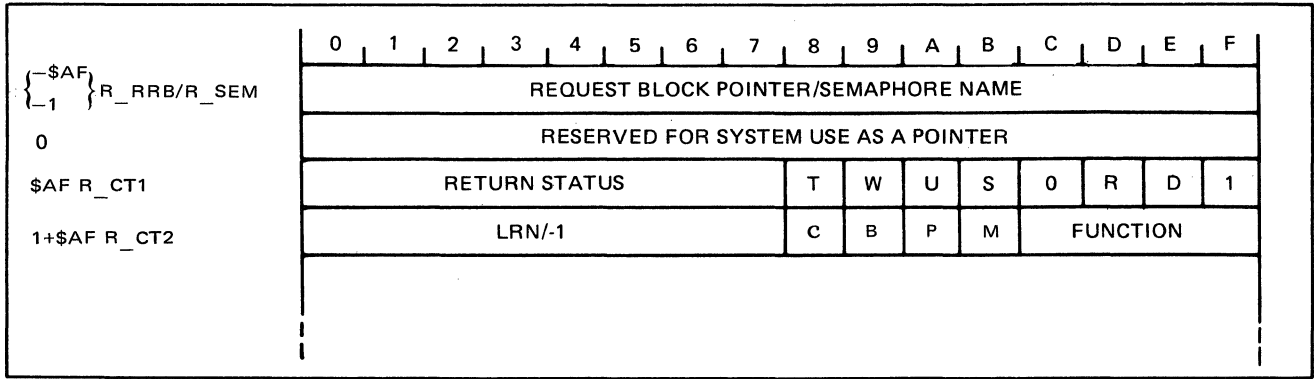


Figure A-1. First Four Items of Request Blocks

CLOCK REQUEST BLOCK FORMAT

Figure A-2 shows the format of the clock request block; Table A-1 shows its contents.

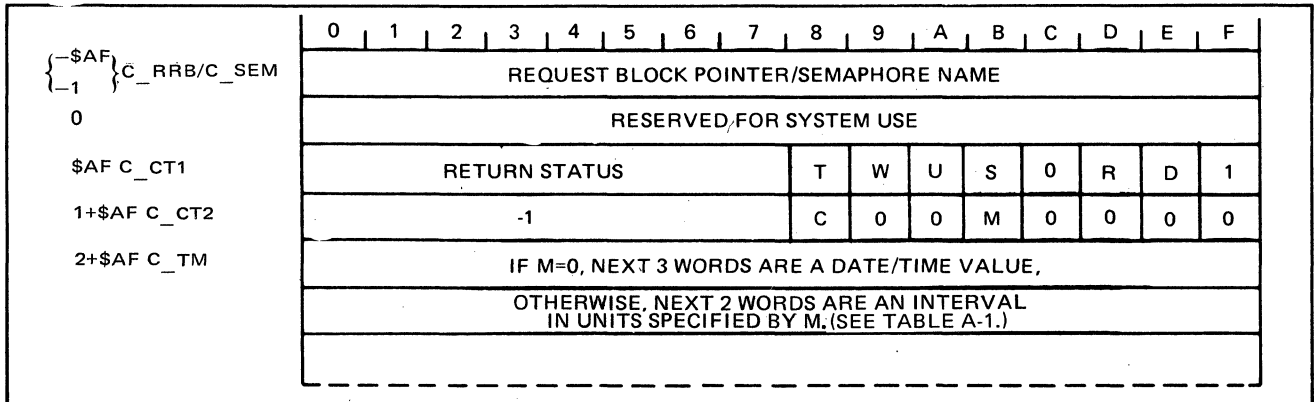


Figure A-2. Format of Clock Request Block

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 request block pointer (R-bit on), or a semaphore name (S-bit on).
0		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.
		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 timeout, of item named in C_RRB.
		C	Must be zero.
		D (R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in C_RRB, after timeout of this request.
		E (D)	Delete clock request block. Values: 0=No delete, 1=Return memory to the pool where CRB is the first entry of its memory block.
		F	I/O bit. Must be set in absence of start address.
1+\$AF	C_CT2	0-7	Value is -1.
		8 (C)	When set, indicates this block is associated with a cyclic clock function.

Table A-1 (cont). Contents of Clock Request Block

Item	Label	Bit(s)	Contents
1+\$AF (cont)	C_CT2 (cont)	9-B(M)	When set, last two words contain an interval in units specified by M. Each interval value is as follows: 001 - in milliseconds; 010 - in tenths of a second; 011 - in seconds; 100 - in minutes; 101 - in units of clock resolution.  When reset (off), the last <u>three</u> words contain a date/time interval.
2+\$AF	C_TM		Contents depend on M bit of C_CT2.

FILE INFORMATION BLOCK FORMAT

Figure A-3 shows the format of the file information block; Table A-2 shows its contents.

		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	F_LFN	LOGICAL FILE NUMBER															
1	F_PROV	PROGRAM VIEW															
2	F_URP/F_UBP	- USER RECORD/BUFFER POINTER _____															
3																	
4	F_IRL/F_BFSZ	INPUT RECORD LENGTH/BUFFER SIZE															
5	F_ORL/F_BKSZ	OUTPUT RECORD LENGTH/BLOCK SIZE															
6	F_LIRT/F_BKN01	RECORD TYPE RANGE/BLOCK NUMBER															
7	F_HIRT/F_BKN02	RECORD TYPE RANGE/BLOCK NUMBER															
8	F_ORT	RESERVED															
9	F_IKP	- INPUT KEY POINTER _____															
10																	
11	F_IKF/F_IKL	INPUT KEY FORMAT/INPUT KEY LENGTH															
12	F_ORA1	(LEFT) OUTPUT RECORD ADDRESS															
13	F_ORA2	(RIGHT) OUTPUT RECORD ADDRESS															
14	F_RFU	- RESERVED _____															
15																	

Figure A-3. Format of File Information Block (FIB)



Table A-2. Contents of File Information Block (FIB)

Item	Label	Bit(s)	Contents
0	F_LFN	0-15	Logical file number (LFN)
1	F_PROV	0	Access level - set on for storage management, off for data management.
		1-4	Process rules - bit 1 for \$RDREC/\$RDBLK, bit 2 for \$WRREC/\$WRBLK, bit 3 for \$RWREC, bit 4 for \$DLREC.
		5-9	Key type - bit 5 for primary keys, bit 8 for relative keys, bit 9 for simple keys (bits 6 and 7 must be 00).
		10	Record class - set on for fixed-length records only, off for fixed- and variable-length records.
		11	Record visibility - set on if deleted records are to be visible, off if invisible.
		12	Key storage alignment - set on if storage area begins at odd-byte boundary, off if even-byte boundary.
		13	Record storage area/buffer alignment - set on if record storage area (or buffer) begins on odd-byte boundary, off if even-byte boundary.
		14	Transcription mode - set on if data transferred in binary transcription mode, off if ASCII mode.
15	Synchronous/asynchronous indicator - set on if \$RDBLK/\$WRBLK calls executed asynchronously, off if executed synchronously.		
2	F_URP/ F_UBP	0-31	Start address of user record area (data management) or start address of buffer area (storage management).
4	F_IRL/ F_BPSZ	0-15	Input record length (data management) or transfer size (storage management).
5	F_ORL/ F_BKSZ	0-15	Output record length (data management) or block size (storage management).

Table A-2 (cont). Contents of File Information Block (FIB)

Item	Label	Bit(s)	Contents
6	F_LIRT/ F_BKNO	0-15	Must be 0000 for data management; is the left half of the block number (F_BKN01) for storage management.
7	F_HIRT/ F_BKN02	0-15	Must be FFFF for data management; is right half of the block number for storage management.
8	F_OR0	0-15	Must be 0000.
9	F_IKP	0-31	Start address of user key area.
11	F_IKF/ F_IKL	0-7	Input key format (0 for none specified, 1 for primary key, 2 for simple key)
		8-15	Input key length.
12	F_ORA1	0-15	Output record address (left half).
13	F_ORA2	0-15	Output record address (right half).
14	F_RFU	0-31	Reserved for future use.

INPUT/OUTPUT REQUEST BLOCK FORMAT

Figure A-4 shows the format of the input/output request block; Table A-3 shows its contents; Table A-4 summarizes the IORB fields for the operator interface functions.

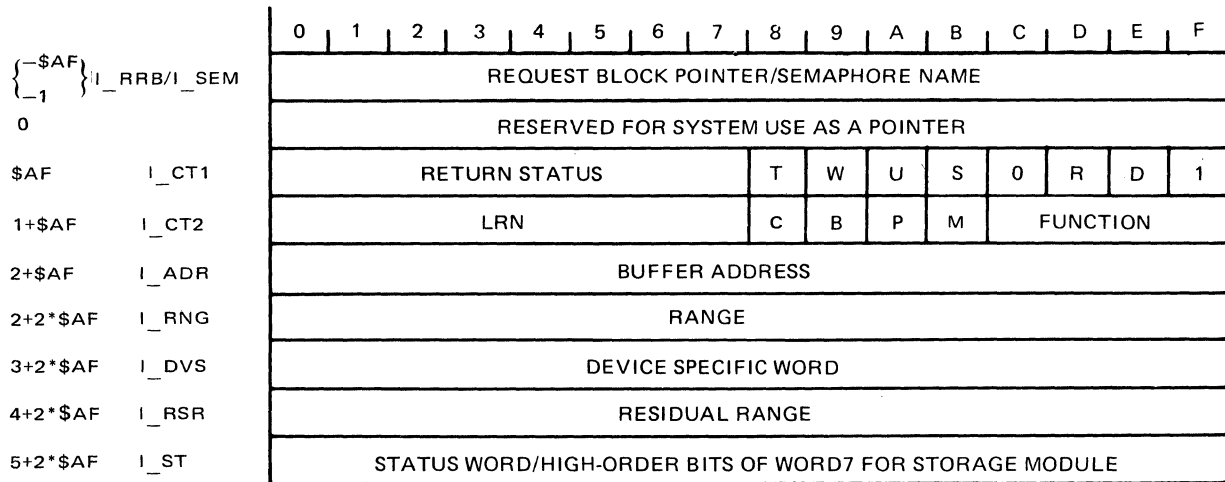


Figure A-4. Format of I/O Request Block

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

Item	Label	Bit(s)	Contents
$-\$AF$	I_RRB/	0-15	Depending on the condition or the S or R bits of I_CT1, this word contains a request block pointer (R-bit on), or a semaphore name (S-bit on). Set by user; used by system at termination of request.
-1	I_SEM	0-31 for LAF	
0		0-15; 0-31	Reserved for system use. 1- or 2-word pointer to indirect request block.
$\$AF$	I_CT1	0-7  8(T)	Return status  This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.

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

Item	Label	Bit(s)	Contents
\$AF (cont)	I_CT1 (cont)	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 timeout, of item named in I_RRB.
		C	Must be zero.
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in I_RRB, after timeout of this request. (System executes \$RQTSK, using I_RRB, upon task termination).
		E(D)	Delete I/O request block. Values: 0=No delete, 1=Return memory to the pool where IORB is the first entry of its memory block.
		F	I/O bit. Must be set.
1+\$AF	I_CT2	0-7	Logical resource number (LRN0; identifies device to be used.
		8(C)	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.
		A(P)	Private space; reserved for system use.
		B(M)	0=Standard IORB; 1=IORB is extended to at least 6+2*\$AF items.
		C-F	Function code. Driver or LPH function, see Table 6-1.

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

Item	Label	Bit(s)	Contents
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 disk storage unit.
3+2*\$AF	I_DVS	0-15	Device-specific information.
4+2*\$AF	I_RSR	0-15	Residual range. Indicates the number of bytes <u>not</u> transferred. Filled in by the system on completion of the order. Used by the cartridge disk and mass storage unit drivers as a data offset value.
5+2*\$AF	I_ST	0-15	Status word. Reflects the mapping of the hardware status into software status format; used as input field high-order bits of sector number for mass storage unit.

Table A-4. 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 <u>input</u> 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 9 (B)	LRN=0  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.

Table A-4 (cont). Summary of IORB Fields for Operator Interface

Item	Label	Bit(s)	Contents
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-5 shows the format of the semaphore request block; Table A-5 shows its contents.

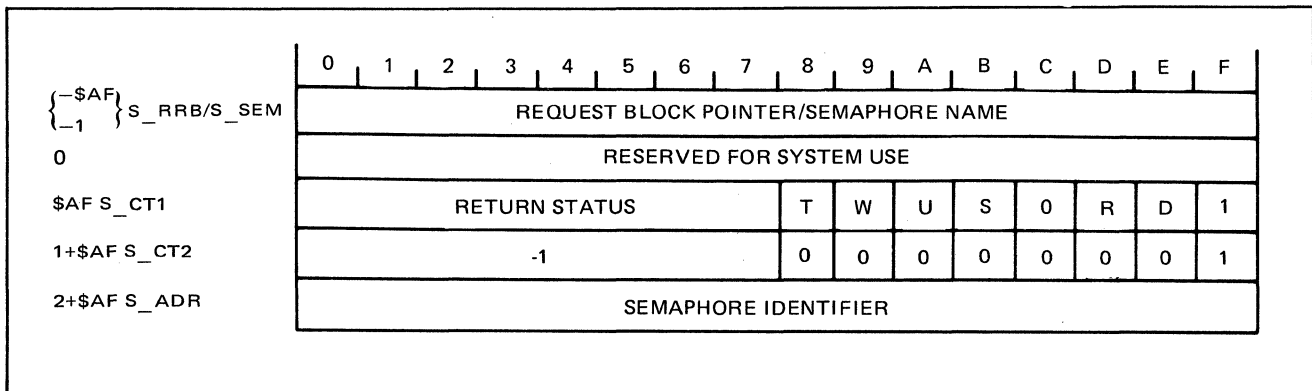


Figure A-5. Format of Semaphore Request Block

Table A-5. Contents of Semaphore Request Block

Item	Label	Bit(s)	Contents
-\$AF	S_RRB		Depending on the condition of the S or R bits of S_CT1, this word contains a request block pointer (R-bit on), or a semaphore name (S-bit on).
-1	S_SEM		
0		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.

Table A-5 (cont). Contents of Semaphore Request Block

Item	Label	Bit(s)	Contents
		9(W)	Wait bit - set if the requesting task is <u>not</u> to be suspended pending the completion of the request that uses this block.
		A(U)	User bit. User may or may not use this bit; the system does not change it.
		B(S)	Release semaphore indicator. Values: 0=No release, 1=Release, on timeout, of item named in S_RRB.
		C	Must be zero.
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in S_RRB, after timeout of this request.
		E(D)	Delete semaphore request block. Values: 0=No delete; 1=Return memory to the pool where SRB is the first entry of its memory block.
		F	I/O bit. Must be set in absence of start address.
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-6 shows the format of the task request block; Table A-6 shows its contents.

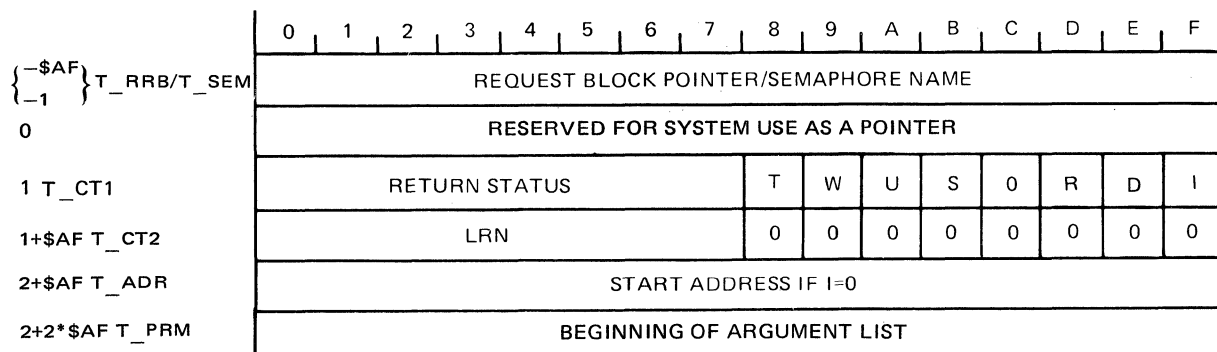


Figure A-6. Format of Task Request Block

Table A-6. 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 request block pointer (R-bit on), or a semaphore name (S-bit on).
0		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.
		9 (W)	Wait bit - set if the requesting task is <u>not</u> to be suspended pending the completion of the request that uses this block.
		A (U)	User bit. User may or may not use this bit; the system does not change it.
		B (S)	Release semaphore indicator. Values: 0=No release, 1=Release on timeout of item named in T_RRB.
		C	Must be zero.

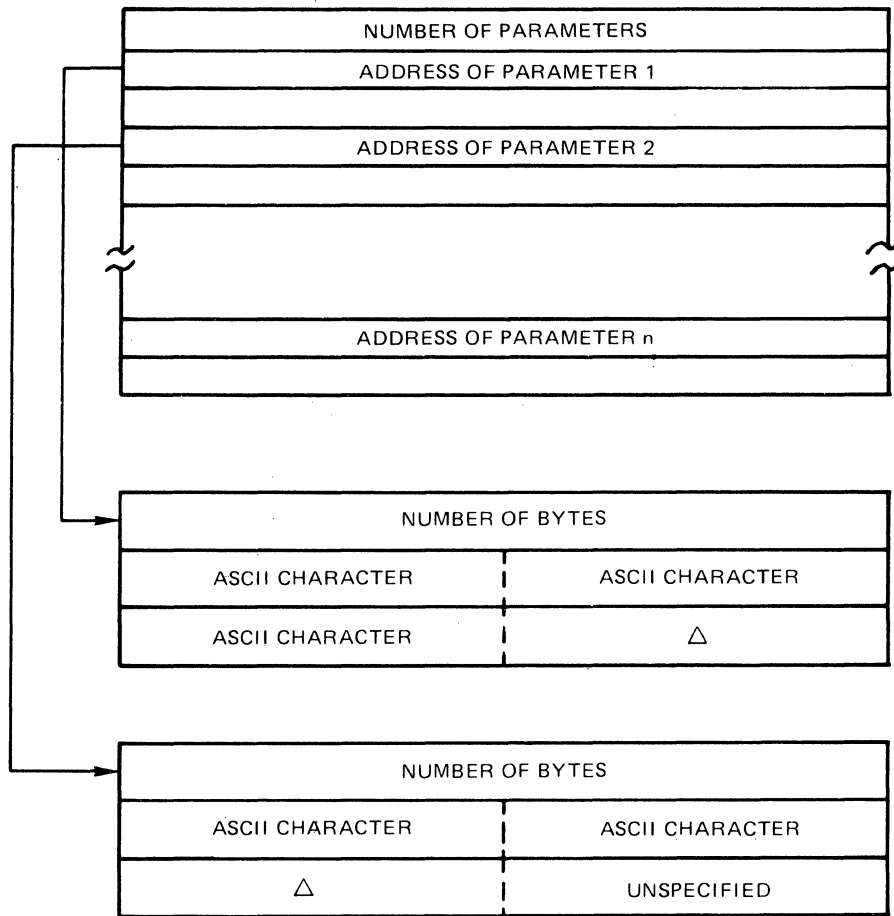


Table A-6 (cont). Contents of Task Request Block

Item	Label	Bit(s)	Contents
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in T_RRB, after timeout of this request.
		E(D)	Delete task request block. Values: 0=No delete; 1=Return memory to the pool where TRB is the first entry of its memory block.
		F(I)	I bit. Must be set in absence of start address.
1+\$AF	T_CT2	0-7	Logical resource number (LRN).
		8-15	Must be zero.
2+\$AF	T_ADR	0-15	Start address if the I-bit of T_CT1 is reset (zero).
2+2*\$AF	T_PRM		Beginning of argument list.

PARAMETER BLOCK FORMAT

Figure A-7 shows the format of the parameter block.



NOTE: The parameter value strings need not be contiguous with the address portion of the parameter block; if the block is system generated, each parameter will have a trailing blank that is not included in the byte count.

Figure A-7. Format of Parameter Block

WAIT LIST FORMAT

Figure A-8 shows the format of the wait list.

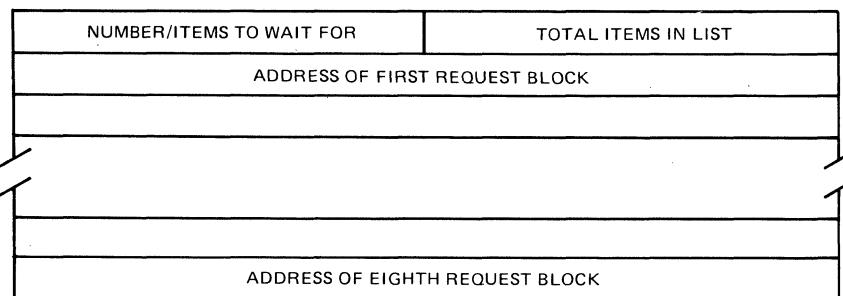


Figure A-8. Format of Wait List

## MESSAGE FACILITY MESSAGE GROUP REQUEST BLOCKS

Tables A-7, A-8, and A-9 respectively show the content of the following message facility message group request blocks:

- o Message group control request block (MGCRB)
- o Message group initialization request block (MGIRB)
- o Message group recovery request block (MGRRB)

Templates for these request blocks are generated by the \$MGCRT, \$MGIRT, and \$MGRRT macro calls respectively.

The request blocks can be generated by the \$MGCRB, \$MGIRB, and \$MGRRB macro calls respectively.

Table A-7. Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
0	MC_OS	Address 0-15 (SAF) 0-31 (LAF)	Pointer Reserved for system use. Reserved for system use.
\$SAF	MC_MAJ	0-7 8 (T) 9 (W) A (U) B (S)	Major status. Left byte: reserved for system use. Right byte: This byte 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. Wait bit - set if the requesting task is not to be suspended pending the completion of the request that uses this block. User bit. User may or may not use this bit; the system does not change it. Release semaphore indicator. Values: 0=No release, 1=Release, on timeout, of item named in I_RRB.

Table A-7 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
\$AF (cont)	MC_MAJ (cont)	C	Must be zero.
		D(R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in I_RRB, after timeout of this request. System executes \$RQTSK, using I_RRB upon task termination.
		E(D)	Delete I/O request block. Values: 0=No delete, 1=Return memory to the pool where IORB is the first entry of its memory block.
1+\$AF	MC_OPT	F	I/O bit. Must be set.
		0-7	General options: 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 IORB).
2+\$AF	MC_BIF	C-F	Must be 0.
		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.

Table A-7 (cont). Message Group Control Request Block (MGCRB)

Item	Label	Bit(s)	Contents
	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
6+2*\$AF	MC_EXT	0-7	Extension mechanism. Left byte: binary value of 13+2*\$AF, i.e., number of words in IORB 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	0-F	Message group i.d. Returned in the \$MINIT and \$MACPT macro calls.
16+2*\$AF	MC_LVL	0-7	Enclosure level. Left byte: enclosure level requested.

Table A-7 (cont). Message Group Control Request Block (MGRB)

Item	Label	Bit(s)	Contents
16+2*\$AF (cont)	MC_LVL (cont)	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 0-15 (SAF) 0-31 (LAF)	Name-list pointer.  Must be 0. Must be 0.
18+3*\$AF	MC_TGI	0-F	Reserved for system use.
19+3*\$AF	MC_TSK	Address 0-15 (SAF) 0-31 (LAF)	Pointer.  Reserved for system use. Reserved for system use.
19+4*\$AF	MC_NPI	0-F	Must be 0.

Table A-8. Message Group Initialization Request Block (MGIRB)

Item	Label	Bit(s)	Contents
0	MI_OS	Address 0-15 (SAF) 0-31 (LAF)	Pointer: reserved for system use.
\$AF	MI_MAJ	0-7  8 (T)	Major status.  Left byte: reserved for system use.  Right byte: This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it.

Table A-8 (cont). Message Group Initialization  
Request Block (MGIRB)

Item	Label	Bit(s)	Contents
\$AF (cont)	MI_MAJ (cont)	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 timeout, of item named in I_RRB.
		C	Must be zero.
		D (R)	Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block named in I_RRB, after timeout of this request. (System executes \$RQTSK, using I_RRB, on task termination.)
		E (D)	Delete I/O request block. Values: 0=No delete, 1=Return memory to the pool where IORB is the first entry of its memory block.
1+\$AF	MI_OPT	F	I/O bit. Must be set.
		0-7	Reserved for system use.
		8-A	Must be 0.
		B C-F	Must be 1 (extended IORB). 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	Must be hexadecimal 1.

Table A-8 (cont). Message Group Initialization  
Request Block (MGIRB)

Item	Label	Bit(s)	Contents
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	0-7  8-F	Extension mechanism.  Left byte: binary value of 31+2*\$AF, i.e., number of words in IORB 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	MI_FNC  MI_REV	0-7  8-F	Function.  Left byte: reserved for system use.  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.



Table A-8 (cont). Message Group Initialization  
Request Block (MGIRB)

Item	Label	Bit(s)	Contents
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.
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  (Six words)	0-F 0-F 0-F 0-F 0-F 0-F	Acceptor mailbox name.  Must be from 1 to 12 ASCII characters, blank-filled, left justified.
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-9. Message Group Recovery Request Block (MGRRB)

Item	Label	Bit(s)	Contents
0	MR_OS	Address 0-15 (SAF) 0-31 (LAF)	Pointer. Reserved for system use. Reserved for system use.
\$SAF	MR_MAJ	0-7 8 (T) 9 (W) A (U) B (S) C D (R) E (D) F	Major status. Left byte: reserved for system use. Right byte: 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. Wait bit - set if the requesting task is not to be suspended pending the completion of the request that uses this block. User bit. User may or may not use this bit; the system does not change it. Release semaphore indicator. Values: 0=No release, 1=Release, on timeout, of item named in I_RRB. Must be zero. Return request block indicator. Values: 0=No dispatch, 1=Dispatch of request block request. (System executes \$RQTSK, using I_RRB, upon task termination.) Delete I/O request block. Values: 0=No delete, 1=Return memory to the pool where IORB is the first entry of its memory block. I/O bit. Must be set.

Table A-9 (cont). Message Group Recovery  
Request Block (MGRRB)

Item	Label	Bit(s)	Contents
1+\$AF	MR_OPT	0-7  8-A B C-F	General options.  Left byte: reserved for system use.  Right byte:  Must be 0.  Must be 1 (extended IORB).  Must be 0.
2+\$AF	MR_BUF	Address 0-15 (SAF) 0-31 (LAF)	Pointer. Must be 0. Must be 0.
2+2*\$AF	MR_BSZ	0-F	Buffer range. Must be 0.
3+2*\$AF	MR_ITP	0-F	Must be 0.
4+2*\$AF	MR_RES	0-F	Residual range. Reserved for system use.
5+2*\$AF	MR_RSN	0-7   8-F	Reason-for-terminate code:  0 = Normal message group termination.  34-44 = User-defined abnormal termination of message group.  Reserved for system use.
6+2*\$AF	MR_EXT	0-7  8-F	Extension mechanism.  Left byte: binary value of 14+2*\$AF, i.e., number of words in IORB following the extension word.  Right byte: must be hexadecimal 7.
7+2*\$AF	Next seven words. Reserved for system physical I/O use.		

Table A-9 (cont). Message Group Recovery  
Request Block (MGRRB)

Item	Label	Bit(s)	Contents
14+2*\$AF	MR_FNC MR_REV	0-7 8-F	Left byte: function. Reserved for system use. Right byte: revision. Must be hexadecimal 01.
15+2*\$AF	MR_MGI	0-F	Message group i.d. Returned in the \$MINIT and \$MACPT macro calls.
16+2*\$AF	MR_CNC	0-F	Reserved for system use.
17+2*\$AF	MR_FMT	Address 0-15 (SAF) 0-31 (LAF)	Pointer. Must be 0. Must be 0.
18+3*\$AF	MR_MRU (Two words)	0-F 0-F	Reserved for system use. Reserved for system use.
19+3*\$AF	MR_AMU (Two words)	0-F 0-F	Reserved for system use. Reserved for system use.

## APPENDIX B

### WRITING A PERIPHERAL I/O DRIVER

To add a new function to a Honeywell-supplied driver, the user must modify its existing source code, then relink the system with the new driver (see Section 6).

To operate with a device that is not supported, the user must write his own driver. This appendix describes what the user must be aware of in writing a driver.

#### SYSTEM BUILDING CONSIDERATIONS IN WRITING A DRIVER

The system building process defines the driver and those data structures necessary for the driver to interface with the user and with the system. The driver can reference only two data structures, input/output request blocks (IORBs), and resource control tables (RCTs).

An RCT is generated by the DRIVER directive in system building (see "Driver Directive" in the GCOS 6 MOD 400 System Building manual). The RCT must be at least three words, in both short address form (SAF) and long address form (LAF). Requirements for stack space are 22 words in SAF, and 40 words in LAF. RCT and stack sizes are specified by the left and right bytes, respectively, in the RCT\_size argument of the system building DRIVER directive. The flags word R\_FLGS in the RCT must be set up during driver initialization.

The user must link the driver as a separate bound unit. Any references to any system function routine (described under "Driver Usable System Functions" below), must be specified in an EDEF directive to the Linker.

Example:

The following system building DRIVER directive generates a five-word RCT with 22 words of stack space:

```
DRIVER ^VOL1>OWNDRIVER,4,9,X'1380',X'1605'
```

The user-written driver OWNDRIVER will be loaded from the volume major directory on volume VOL1. A task control block (TCB) is generated and will be fixed to level 9. A pointer in the LRN 4 position of the logical resource table (LRT) will be set up to point to the generated RCT. The device channel X'1380' will be set up in the first word of RCT.

#### DRIVER INTERFACE IN WRITING A DRIVER

The user interfaces with the driver by placing task requests against the driver via the \$RQIO macro. The system uses an LRN-to-RCT-to-TCB (priority level) association to associate a request with a specified driver. When the driver is turned on to service a request, \$B4 will point to the IORB to be serviced. Since devices may interrupt to a specified level when an attention occurs, the user-written driver should first check \$B4 for null to ascertain if a request or attention is being processed.

Drivers should not alter the first six entries of an IORB. IORB's should be generated as described in Section 6.

Drivers must terminate with an internal terminate as described under "Driver Usable System Functions."

#### USER-WRITTEN DRIVER INITIALIZATION

On entry to the driver for the first request, the driver must locate the resource control table for the device, using the LRN in the input/output request block (IORB). The driver must set register \$B7 to point to a stack area, and must store in the stack a pointer to the RCT. Thus, when an attention occurs, the driver may locate the RCT by retrieving the RCT pointer from the stack. The following instructions sequence locates the RCT and sets the stack pointer:

```
LLH  $R2,$B4,I CT2  GET LRN FROM IORB
LNJ  $B5,<ZXS RCT  LOCATE RCT
LAB  $B7,$B2.-$AF   SET STACK POINTER
STB  $B2,-$B7      SAVE POINTER TO RCT
```

Driver initialization must also set up the flags word R\_FLGS in the RCT to reflect the characteristics of the device (see Section 6),

Finally, the driver must set the device interrupt level, and read the device status. These functions may be accomplished by calling one of the subroutines described under "Driver Usable System Functions" below.

The driver may also set up the device type (R\_TYP) during initialization, with the I/O instruction with a function code "input device ID" (see the Honeywell Level 6 Minicomputer Handbook).

#### DRIVER USABLE SYSTEM FUNCTIONS

To provide compatibility with the system, user-written drivers may call only the following system functions, using the format and register contents as shown for each function:

- o I/O subroutines (ZIOSUB)
- o Locate RCT for a device (ZXSRCT)
- o Terminate driver (ZXD TR)
- o Output address and range (ZIOD)

#### I/O Subroutines (ZIOSUB) For User-Written Drivers

The common driver subroutines are called by executing the instruction: LNJ \$B5,<ZIOSUB with standard register contents as follows:

Function code in \$R1  
RCT address in \$B6  
Current stack position in \$B7

#### INITIALIZE FUNCTION (Code 0)

The initialize subroutine initializes the device interrupt level, removes the level from the first word of the RCT, and exits with the device status in R\_STTS of the RCT.

Input registers: Standard registers for I/O subroutines  
Return registers:

- o \$R1 = Return status  
0 = Normal  
A = Controller unavailable
- o \$R4 = Channel number
- o \$R2, \$R6 altered

#### WAIT ON LINE FUNCTION (Code 1)

If the attention flag in the RCT is 0, wait for the next online interrupt, or until five minutes have elapsed.

If the attention flag is 1, return with successful completion status if the device is ready; otherwise wait for the next online interrupt or until five minutes have elapsed.

Input registers:

- o Standard registers for I/O subroutines
- o \$R4 = Channel number

Return registers:

- o \$R1 = Return status
  - 0 = Normal
  - 6 = Five minutes elapsed; not yet online
  - 1 = Status of device could not be read following interrupt.
  - A = Controller not available.
- o \$R5, \$R6, \$R7 altered

STOP I/O FUNCTION (Code 2)

Issue "Stop I/O" order to the device whose channel number is in the \$R4 register; call the Wait for Interrupt function.

Input registers: Same as Wait on Line function above.

Return registers:

- o \$R1 = Return status
  - 0 = Normal
  - 6 = Time-out
  - 1 = Negative acknowledgement to status request
- o \$R5 = Modified status. Bits 0, 1, and 12 cleared to 0; bits 13 through 15 cleared and logical OR performed with the result placed into bit 15 (fatal error bit).
- o \$R6 = Status (same as R\_STTS in the RCT)
- o \$R7 = Residual range

WAIT FOR INTERRUPT (Code 3)

Activate timer to time the number of seconds specified in \$R6; call the Read/Modify Status function.



Input registers:

- o Standard registers for I/O subroutines
- o \$R4 = Channel number
- o \$R6 = Timer value, in seconds

Return registers:

- o \$R1 = Return status
- o 6 = Timeout (same as Read/Modify Status if no time-out)
- o \$R5 = Modified status. Bits 0, 1, and 12 cleared to 0; bits 13 through 15 are cleared and logical OR performed with result placed into bit 15 (fatal error bit).
- o \$R6 = Status (same as R\_STTS in the RCT)
- o \$R7 = Residual range

READ/MODIFY STATUS FUNCTION (Code 4)

Input registers:

- o Standard registers for I/O subroutines
- o \$R4 = Channel number

Return registers:

- o \$R1 = Return status
  - 1 = Status request received negative acknowledgement
- o \$R5 = Modified status. Bits 12 through 15 cleared to 0 and logical OR performed with result placed into bit 15 (fatal error bit).

Locate RCT for Device (ZXRCT)

This system function, to locate an RCT for a specific device is called with the instruction: LNJ \$B5,<ZXRCT, with \$R2 containing the LRN for the device.

Input registers:

- o \$R2 = LRN value

Return registers:

- o \$R1 = Return status
  - 0 = RCT found for LRN
  - 0802 = Illegal LRN
- o \$B2 = Pointer to RCT
- o \$R2 = altered

Driver Terminate (ZXD\_TR)

The system function to terminate a driver is called with the instruction: LNJ \$B5<ZXD\_TR.

Input registers:

- o \$R2 = Return status for current request
- o \$B4 = Start address for next execution of driver
- o \$B7 = Current stack position.

Output Address and Range (ZIOLD)

The system function to output the address and range for a data transfer is called with the instruction: LNJ \$B5<ZIOLD. The ZIOLD subroutine adjusts any necessary change of address space and also performs the IOLD order.

Input registers:

- o \$B3 = Buffer address
- o \$R2 = Buffer byte offset
- o \$R3 = Range
- o \$R4 = Channel number.

No registers are altered by ZIOLD.

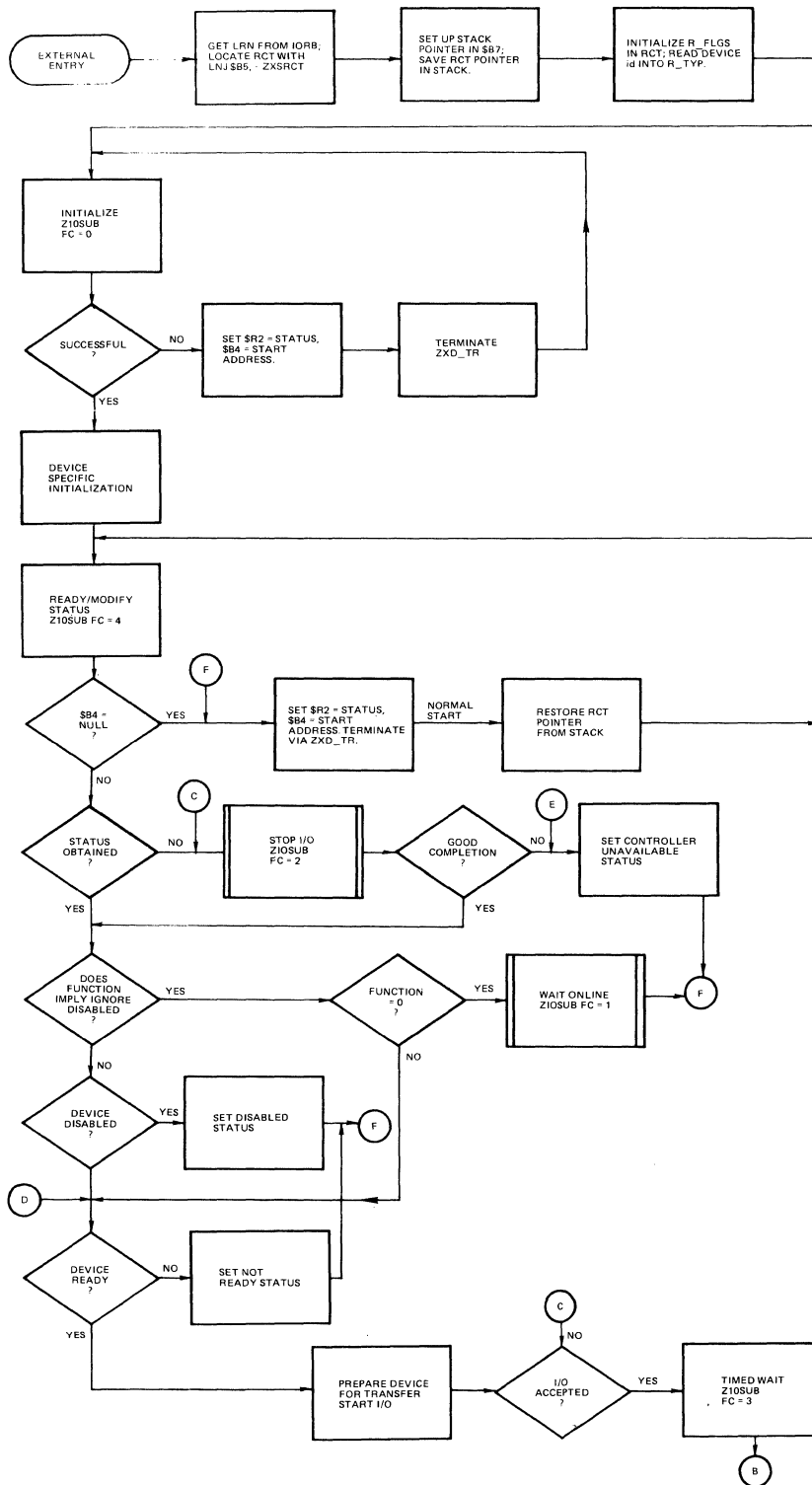


Figure B-1. Typical Device Driver

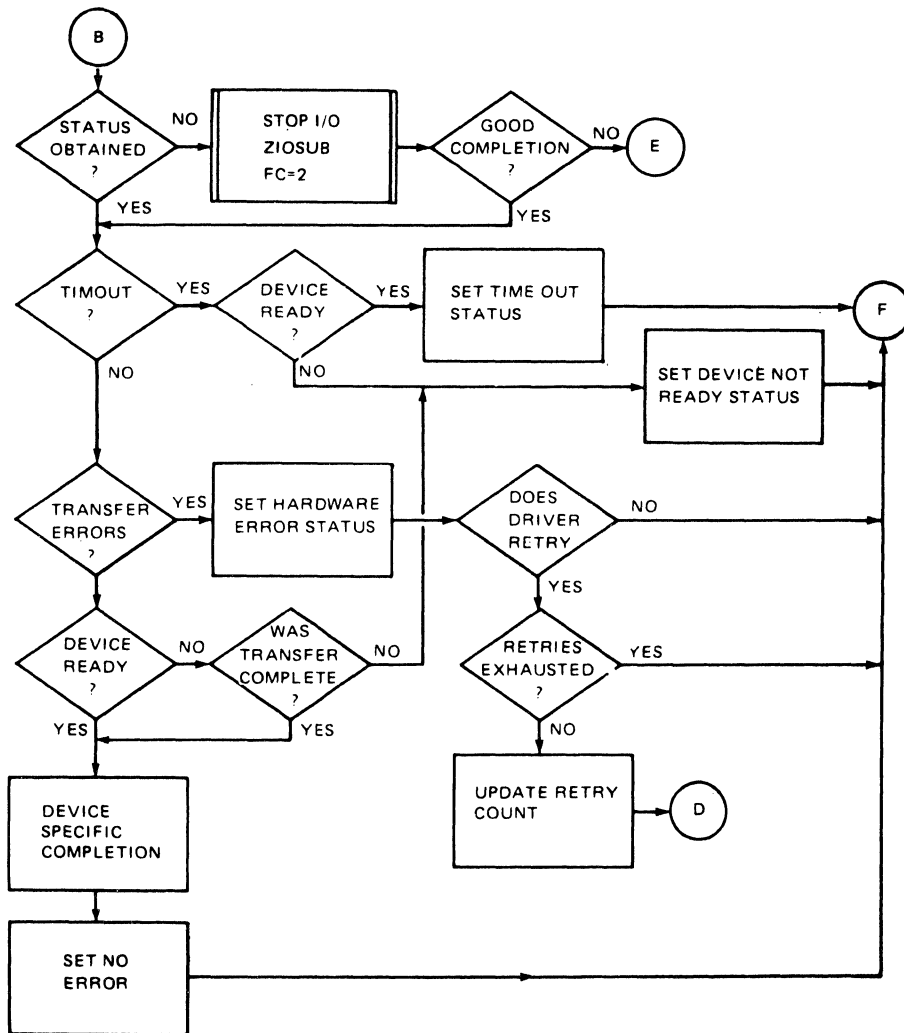


Figure B-1 (cont). Typical Device Driver

GENERAL I/O REQUIREMENTS FOR USER DEVICE DRIVER

This subsection describes in general terms how to initiate an I/O operation. The central processor instructions that are designed to initialize the data for an I/O operation and to actually start the operation are:

- o IO
- o IOLD
- o IOH

For GCOS 6 system compatibility, the user should perform IOLD instructions by using the ZIOLD subroutine instead of directly executing IOLD.

These instructions, described in the Assembly Language manual and in the Honeywell Level 6 Minicomputer Handbook, can define all required information about an I/O operation. After I/O completion they can be used for input status to verify the results of the I/O operation. Some output and input functions that can be performed with these instructions are shown below.

<u>Instruction</u>	<u>Function</u>	<u>Description</u>
IO	Output-interrupt control	Set interrupt level for the device.
IO	Input device id	Read the device id for the device.
IO	Input status	Test the state of the device.
IO	Output configuration	Define operation explicitly (for disk, this defines <u>track</u> and <u>sector</u> number of the I/O operation).
IO	Output task	Task defines the operation (i.e., seek, read, write ...).
IO	Input range	Get the residual range (i.e., number of bytes <u>not</u> transferred but requested in the IOLD instruction in ZIOLD).

Note that every device type has different requirements as to which functions are necessary to start an I/O operation. For example, the card reader has only one operation (i.e., read); so the IOLD (output address and range) actually initiates the I/O, and there is no I/O output task command.

To start a disk I/O operation, the sequence would be:

Output configuration	(Define sector and track)
Output address and range	(Use ZIOLD subroutine)
Output task	(Define operation and start I/O)
Input status	(Check results)

## APPENDIX C

### SUMMARY OF REGISTER CONTENTS FOR SYSTEM SERVICE MACRO CALLS

Table C-1 lists the register contents before and after execution of the system service macro calls. Since data structure macro calls do not affect registers, these are not listed.

The table is arranged in function code sequence.

Table C-1. Macro Calls, Function Codes, and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Request and Return Functions														
\$WAIT	01/00					Address RB	Status					Address RB		
\$WAITL	01/01					Wait list address	Status				Wait list address	Address RB		
\$TEST	01/02					Address RB	Status					Address RB		
\$RETRN	none	Code					Status						Terminate routine address	
\$TRMRO	01/03	Code					Status					Address RB	Terminate routine address	RB parameter list address
\$TRMRO	01/04	Code				Start address	Status							
\$RBADD	01/07						Status					Address RB	Terminate routine address	RB parameter list address
Physical I/O Functions														
\$RQIO	02/00					Address IORB	Status					Address IORB		
\$SDSV	02/02	LRN					Status	LRN						
\$RDVAT	02/03	LRN					Status	LRN						
\$ENDV	02/04	LRN					Status	LRN						
\$ELST	02/05					Device name address	User log table address	Status						

C-2

CB08



Table C-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
Physical I/O Functions (cont)														
\$ELEX	02/07				Device name address	User log table address	Status							
\$ELGT	02/08				Device name address	User log table address	Status							
\$ELEN	02/09				Device name address	User log table address	Status							
Memory Allocation Functions														
\$GMEM	04/02	Return condition	Size	Size			Status		Size	Size		Address memory		
\$GMEM	04/03		Size	Size			Status		Size	Size		Address memory		
\$RMEM	04/04					Address memory	Status							
\$RMEM	04/05		Size	Size		Address memory	Status		Size	Size		Address memory		
\$STMP	04/06	Pool id					Status	Memory (percent)	Memory (words)	Memory (words)				
Clock Functions														
\$RQCL	05/00					Address CRB	Status					Address CRB		
\$CNCRQ	05/01					Address CRB	Status					Address CRB		
\$SUSPN	05/02	Code	Internal value	Internal value			Status							
\$SUSPN	05/03	Internal value						Status						

C-3

CB08

Table C-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
Date/Time Functions														
\$EXTDT	05/04	Internal date/time				Receiving field address	Status	Internal date/time				Receiving field address		
\$EXTIM	05/05	Internal date/time				Receiving field address	Status	Internal date/time				Receiving field address		
\$GDTM	05/06	Internal date/time					Status	Internal date/time						
\$INDTM	05/07	Internal date/time R5=size				External date/time	Status	Internal date/time				External date/time address		
Semaphore Functions														
\$RQSM	06/00					Address SRB	Status					Address SRB		
\$CNSRQ	06/01					Address SRB	Status					Address SRB		
\$RSVSM	06/02	Code	Identifier				Status		Identifier					
\$DFSM	06/04	Value	Identifier				Status		Identifier					
\$RLSM	06/03		Identifier				Status		Identifier					
Overlay Handling Functions														
\$OVEXC	07/00	Overlay id	Offset			Base address	Status							
\$OVLD	07/01	Overlay id				Base address	Status	Overlay id	Offset			Base address		
\$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		

C-4

CB08

Table C-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	
\$OVRLS	07/06	\$B5 = Return point address					Code 0006								
\$OVRLC	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								
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			
\$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			

Table C-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
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						
External Switch Functions														
\$RLSW	0B/00	Mask						Value switch word						
\$SETSW	0B/01	Mask						Value switch word						
\$CLRSW	0B/02	Mask						Value switch word						
Task Control Functions														
\$RQTSK	0C/00					Address TRB	Status					Address TRB		
\$CANRQ	0C/01					Request block address	Status		Posted request block code			Request block address		

C-6

CB08

Table C-1 (cont). Macro Calls, Function Codes,  
and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
Task Control Functions (cont.)														
\$CRISK	0C/02	LRN	Level		Address start		Status	LRN						
\$CRISK	0C/03	LRN	Level		Address root name		Status	LRN						
\$DLTSK	0C/04	LRN					Status	LRN						
\$SPTSK	0C/05		Level		Address root name	Address TRB	Status							
\$SPTSK	0C/06		Level		Address start	Address TRB	Status							
\$CMDLN	0C/08		Size			Address command line	Status					Address command line		
\$CLPNT	0C/13					FIB address	Status							
Task Group Control Functions														
\$RQGRP	0D/00	Group id	B5 = Address fixed parameter block		Address argument list		Status							
\$CRGRP	0D/03	Group id	LRN	LFN	Address root name		Status	Group id						
			R4 = Memory pool id R5 = Priority level											
\$DLGRP	0D/04	Group id					Status	Group id						
\$SFGRP	0D/05	Group id	LRN	LFN	Address root name	Address argument list	Status	Group id						
			B5 = Address fixed parameter block R4 = Memory pool id R5 = Priority level											

C-7

CB08

Table C-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	R7
Task Group Control Functions (cont.)														
SABGRQ	0D/07	Group id	Abort				Status	Group id						
SSUSFG	0D/08	Group id					Status	Group id						
SACTVG	0D/09	Group id					Status	Group id						
SNPROC	0D/08						Status							
Batch Functions														
SRQBAT	0E/00					Address argument list	Status							
		B5 = Address fixed parameter block												
Error Handling Function														
SRPTER	0E/00		Size	Code			Status			Code				
		R2 = Component error code B3 = Expansion text address												
File Management Functions														
SASFIL	10/10					Address argument structure	Status							
SDSFIL	10/15					Address argument structure	Status							
SCTFIL	10/20					Address argument structure	Status							
SRMFIL	10/25					Address argument structure	Status							

C-8

CB08

Table C-1 (cont). Macro Calls, Function Codes,  
and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
File Management Functions (cont)														
\$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							
\$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							
\$WIFIL	10/64					Address argument structure	Status							
\$WOFIL	10/65					Address argument structure	Status							
\$CRDIR	10/A0					Address argument structure	Status							

Table C-1 (cont). Macro Calls, Function Codes,  
and Register Contents

Contents Before Execution							Contents Returned							
Macro Call	R1	R2	R6	R7	B2	B4	R1	R2	R6	R7	B2	B4	B5	B7
File Management Functions (cont)														
\$RLDIR	10/A5					Address argument structure	Status							
\$QWDIR	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							
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							

C-10

CB08



Table C-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
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							
\$BUID	14/06					Address receiving field	Status					Address receiving field		
\$HDIR	14/0B					Address receiving field	Status					Address receiving field		
\$TGIN	14/0C					Address receiving field	Status					Address receiving task group		
Intergroup Message Facility Functions														
\$MACPT	15/01					Request block address	Status					Request block address		
\$MINIT	15/02					Request block address	Status					Request block address		

C-11

CB08

Table C-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
Intergroup Message Facility Functions (cont)														
SMRECV	15/03					Request block address	Status					Request block address		
\$MIMG	15/04					Request block request	Status					Request block address		
\$MSEND	15/05					Request block address	Status					Request block address		
\$MOMG	15/07					Request block address	Status					Request block address		
User Terminal Functions														
\$RQIML	17/03					Request block address	Status					Request block address		
\$RLIML	17/04		LRN	Release status code			Status							
Communications Function														
SSDL	1B/00		Channel number or 0		Address device pathname	Address telephone number	Status							

C-12

CB08

## APPENDIX D

### ASCII AND EBCDIC CHARACTER SETS

Tables D-1 and D-2 illustrate the ASCII and EBCDIC character sets, respectively. In addition to the ASCII characters, Table D-1 shows the hexadecimal equivalents; Table D-2 shows the binary and hexadecimal equivalents of the EBCDIC character set.

Following are lists of the control characters and special graphic characters that appear in the two tables:

#### CONTROL CHARACTERS

ACK	Acknowledge	GE	Graphic Escape
BEL	Bell	GS	Group Separator
BS	Backspace	HT	Horizontal Tab
BYP	Bypass	IFS	Interchange File Separator
CAN	Cancel	IGS	Interchange Group Separator
CC	Cursor Control	IL	Idle
CR	Carriage Return	IRS	Interchange Record Separator
CU1	Customer Use 1	IUS	Interchange Unit Separator
CU2	Customer Use 2	LC	Lowercase
CU3	Customer Use 3	LF	Line Feed
DC1	Device Control 1	NAK	Negative Acknowledgment
DC2	Device Control 2	NL	New Line
DC3	Device Control 3	NUL	Null
DC4	Device Control 4	PF	Punch Off
DEL	Delete	PN	Punch On
DLE	Data Link Escape	RES	Restore
DS	Digit Select	RLF	Reverse Line Feed
EM	End of Medium	RS	Reader Stop
ENQ	Enquiry	SI	Shift In
EO	Eight Ones	SM	Set Mode
EOT	End of Transmission	SMM	Start of Manual Message
ESC	Escape	SO	Shift Out
ETB	End of Transmission Block	SOH	Start of Heading
ETX	End of Text	SOS	Start of Significance
FF	Form Feed	SP	Space
FS	Field Separator	STX	Start of Text

CONTROL CHARACTERS (cont)

SUB	Substitute	UC	Uppercase
SYN	Synchronous Idle	US	Unit Separator
TM	Tape Mark	VT	Vertical Tab

SPECIAL GRAPHIC CHARACTERS

¢	Cent Sign	>	Greater-than Sign
.	Period, Decimal Point	?	Question Mark
<	Less-than Sign	'	Grave Accent
(	Left Parenthesis	:	Colon
+	Plus Sign	#	Number Sign
	Logical OR	@	At Sign
&	Ampersand	'	Prime, Apostrophe
!	Exclamation Point	=	Equal Sign
\$	Dollar Sign	"	Quotation Mark
*	Asterisk	~	Tilde
)	Right Parenthesis	{	Opening Brace
;	Semicolon	⌋	Hook
⌋	Logical NOT	⌋	Fork
-	Minus Sign	}	Closing Brace
/	Slash	\	Reverse Slant
	Vertical Line	⌋	Chair
,	Comma		Long Vertical Mark
%	Percent	[	Opening Bracket
_	Underscore	]	Closing Bracket
ˆ	Circumflex		

Table D-1. ASCII/Hexadecimal Equivalents

H2	H1							
	0	1	2	3	4	5	6	7
0	NUL	DLE	SP	0	@	P	'	p
1	SOH	DC1	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3	#	3	C	S	c	s
4	EOT	DC4	\$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	:	K	[	k	⌋
C	FF	FS	,	<	L	\	l	⌋
D	CR	GS	-	=	M	]	m	⌋
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	DEL

Table D-2. EBCDIC/Hexadecimal/Binary Equivalents

Bit Positions 4, 5, 6, 7 Second Hexadecimal Digit	00				01				10				11				Bit Positions 0,1 Bit Positions 2,3 First Hexadecimal Digit
	00	01	10	11	00	01	10	11	00	01	10	11	00	01	10	11	
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0000	0	NUL	DLE	DS		SP	&	-					{ <sup>a</sup>	} <sup>a</sup>	\ <sup>a</sup>	0	
0001	1	SOH	DC1	SOS			/		a	i	~ <sup>a</sup>		A	J		1	
0010	2	STX	DC2	FS	SYN				b	k	s		B	K	S	2	
0011	3	ETX	TM						c	l	t		C	L	T	3	
0100	4	PF	RES	BYP	PN				d	m	u		D	M	U	4	
0101	5	HT	NL	LF	RS				e	n	v		E	N	V	5	
0110	6	LC	BS	ETB	UC				f	o	w		F	O	W	6	
0111	7	DEL	IL	ESC	EOT				g	p	x		G	P	X	7	
1000	8	GE <sup>a</sup>	CAN						h	q	y		H	Q	Y	8	
1001	9	RLF <sup>a</sup>	EM					\ <sup>a</sup>	i	r	z		I	R	Z	9	
1010	A	SMM	CC	SM		¢	!	: <sup>a</sup>	:							<sup>a</sup>	
1011	B	VT	CU1 <sup>a</sup>	CU2 <sup>a</sup>	CU3 <sup>a</sup>	.	S	-	#								
1100	C	FF	IFS		DC4	<	*	%	@				J <sup>a</sup>		^ <sup>a</sup>		
1101	D	CR	IGS	ENQ	NAK	(	)		'								
1110	E	SO	IRS	ACK		+	:	>	=				Y <sup>a</sup>				
1111	F	SI	IUS	BEL	SUB	'	~	?	"							EO <sup>a</sup>	

<sup>a</sup>This character is not supported in the 2780 character set.



INDEX

ABORT  
 ABORT GROUP REQUEST (\$ABGRQ)  
 MACRO CALL, 5-4  
 ABORT GROUP (\$ABGRP)  
 MACRO CALL, 5-2

ACCEPT  
 MESSAGE GROUP ACCEPT (\$MACPT)  
 MACRO CALL, 5-172

ACCEPTOR  
 MESSAGE GROUP ACCEPTOR GROUP,  
 5-172, 5-189

ACCOUNT  
 ACCOUNT IDENTIFICATION (\$ACTID)  
 MACRO CALL, 5-6

ACTID  
 ACCOUNT IDENTIFICATION (\$ACTID)  
 MACRO CALL 5-6

ACTIVATE  
 ACTIVATE GROUP (\$ACTVG)  
 MACRO CALL, 5-8

ADDRESS  
 RETURN REQUEST BLOCK ADDRESS  
 (\$RBADD) MACRO CALL, 5-316  
 USER-DRIVER OUTPUT ADDRESS AND  
 RANGE (ZIOLD) SUBROUTINE, B-6

ADDRESSING  
 ADDRESSING CONVENTIONS, 1-3

ALLOCATE  
 INITIALIZE, ALLOCATE GROUP DATA  
 STRUCTURES, 5-57  
 INITIALIZE, ALLOCATE TASK DATA  
 STRUCTURES, 5-64

ALLOCATION, MEMORY  
 MONITOR SERVICE FUNCTIONS, MEMORY  
 ALLOCATION, 2-4

APPEND CHARACTERS  
 APPEND ASCII CHARACTERS TO  
 PATHNAME, 5-133

AREA  
 CONTENTS OF TRAP-RELATED MEMORY  
 AREAS, 7-2  
 POINTER TO NEXT TRAP SAVE AREA  
 (NATSAP), 7-5  
 TRAP INTERRUPT SAVE AREA  
 (ISA), 7-6  
 TRAP SAVE AREAS, 7-6

ARGUMENT VALUES FOR MESSAGE GROUP  
 MACRO CALLS  
 ARGUMENT VALUES FOR \$MGCRB  
 MACRO CALL (TBL), 5-176  
 ARGUMENT VALUES FOR \$MGIRB  
 MACRO CALL (TBL), 5-185

ARGUMENT VALUES FOR MESSAGE GROUP  
 MACRO CALLS (CONT)  
 ARGUMENT VALUES FOR \$MGRRB  
 MACRO CALL (TBL), 5-197  
 MGCRB ARGUMENT VALUES FOR \$MRFCV  
 MACRO CALL (TBL), 5-193  
 MGCRB ARGUMENT VALUES FOR \$MSEND  
 MACRO CALL (TBL), 5-202  
 MGIRB ARGUMENT VALUES FOR \$MACPT  
 MACRO CALL (TBL), 5-173  
 MGIRB ARGUMENT VALUES FOR \$MCMG  
 MACRO CALL (TBL), 5-182  
 MGIRB ARGUMENT VALUES FOR \$MINIT  
 MACRO CALL (TBL), 5-190  
 MGRRB ARGUMENT VALUES FOR \$MTMG  
 MACRO CALL (TBL), 5-207

ASCII  
 APPEND ASCII CHARACTERS TO PATHNAME,  
 5-133  
 ASCII AND EBCDIC CHARACTER SETS, D-1  
 CARD ASCII MADE, 6-17

ASCII/HEXADECIMAL  
 ASCII/HEXADECIMAL EQUIVALENTS  
 (TBL), D-2

ASR/KSR  
 ASR/KSR DRIVERS, 6-34  
 ASR/KSR IORB FIELDS, 6-36  
 ASR/KSR KEYBOARD INPUT, 6-35  
 ASR/KSR PRINTER OUTPUT, 6-36  
 ASR/KSR RCT FIELDS, 6-37  
 ASR/KSR RCT/IORB HARDWARE/SOFTWARE  
 STATUS MAPPING, 6-38

ATTENTION  
 DISABLE DEVICE ON ATTENTION (\$DSDV)  
 MACRO CALL, 5-78  
 RCT ATTENTION STATUS  
 INDICATOR, 5-309  
 RESET DEVICE ATTENTION (\$RDVAT)  
 MACRO CALL, 5-309

AUTODIAL (AUTO CALL UNIT)  
 AUTODIAL TELEPHONE LIST, 5-324

BATCH  
 MONITOR SERVICE FUNCTIONS, BATCH  
 FUNCTIONS, 2-1  
 REQUEST BATCH (\$RQBAT)  
 MACRO CALL, 5-285

BLOCK  
 CLOCK REQUEST BLOCK FORMAT, A-2  
 CLOCK REQUEST BLOCK OFFSETS (\$CRBD)  
 MACRO CALL, 5-31  
 CLOCK REQUEST BLOCK (CRB), 5-288  
 CLOCK REQUEST BLOCK (\$CRB)  
 MACRO CALL, 5-28  
 DEVICE-DRIVER POSITION BLOCK  
 FUNCTION (FC=4), 6-6  
 FILE INFORMATION BLOCK FORMAT, A-4

INDEX

BLOCK (CONT)

FILE INFORMATION BLOCK OFFSETS  
(\$TFIB) MACRO CALL, 5-119  
FILE INFORMATION BLOCK (FIB),  
3-2, 4-5  
FILE INFORMATION BLOCK (\$FIB)  
MACRO CALL, 5-111  
FIRST FOUR ITEMS OF REQUEST BLOCK  
(FIG), A-2  
INPUT/OUTPUT REQUEST BLOCK, 6-7  
INPUT/OUTPUT REQUEST BLOCK  
FORMAT, 6-8, A-6  
INPUT/OUTPUT REQUEST BLOCK OFFSETS  
(\$IORBD) MACRO CALL, 5-168  
INPUT/OUTPUT REQUEST BLOCK (\$IORB)  
MACRO CALL, 5-165  
MESSAGE GROUP CONTROL REQUEST BLOCK  
(MGCRB), 5-179  
MESSAGE GROUP CONTROL REQUEST BLOCK  
(MGCRB) (TBL), A-15  
MESSAGE GROUP CONTROL REQUEST BLOCK  
(\$MGCRB) MACRO CALL, 5-175  
MESSAGE GROUP INITIALIZATION  
REQUEST BLOCK (MGIRB), 5-187  
MESSAGE GROUP INITIALIZATION  
REQUEST BLOCK (MGIRB) (TBL), A-18  
MESSAGE GROUP RECOVERY REQUEST  
BLOCK (MGRRB), 5-199  
MESSAGE GROUP RECOVERY REQUEST  
BLOCK (MGRRB) (TBL), A-22  
MESSAGE GROUP RECOVERY REQUEST  
BLOCK (\$MGRRB) MACRO CALL, 5-196  
PARAMETER BLOCK AND WAIT LIST, 4-3  
PARAMETER BLOCK FORMAT, A-13  
PARAMETER BLOCK (\$PRBLK)  
MACRO CALL, 5-250  
READ BLOCK (\$RDBLK)  
MACRO CALL, 5-254  
REQUEST BLOCK OFFSETS, 4-4  
REQUEST BLOCK TERMINATION STATUS,  
5-365, 5-390  
REQUEST BLOCK (RB), 1-17  
RETURN REQUEST BLOCK ADDRESS  
(\$RBADD) MACRO CALL, 5-316  
SEMAPHORE REQUEST BLOCK  
FORMAT A-10  
SEMAPHORE REQUEST BLOCK OFFSETS  
(\$SRBD) MACRO CALL, 5-323  
SEMAPHORE REQUEST BLOCK (\$SRB)  
MACRO CALL, 5-321  
SEMAPHORE REQUEST BLOCK  
(SRB), 5-322  
TASK CONTROL BLOCK - DEFINITION,  
1-17  
TASK REQUEST BLOCK FORMAT, A-11  
TASK REQUEST BLOCK OFFSETS (\$TRBD)  
MACRO CALL, 5-361  
TASK REQUEST BLOCK (\$TRB)  
MACRO CALL, 5-357  
TASK REQUEST BLOCK (TRB)  
FUNCTIONS, 5-300, 5-360  
WAIT BLOCK (\$WTBLK)  
MACRO CALL, 5-383

BLOCK (CONT)

WRITE BLOCK (\$WRBLK)  
MACRO CALL, 5-393  
BOUND UNIT IDENTIFICATION  
BOUND UNIT IDENTIFICATION (\$BUID)  
MACRO CALL, 5-13  
BREAK  
DEVICE BREAK NOTIFICATION FUNCTION  
(FC 9), 6-6  
BYTE, PRINT CONTROL  
PRINT CONTROL BYTE, 6-22  
CALLS, MACRO  
MACRO CALLS FUNCTION CODES AND  
REGISTER CONTENTS (TBL), C-2  
REGISTER CONTENTS FOR SYSTEM  
SERVICE MACRO CALLS, C-1  
CANCEL  
CANCEL CLOCK REQUEST (\$CNCRQ)  
MACRO CALL, 5-15  
CANCEL REQUEST (\$SCANRQ)  
MACRO CALL, 5-17  
CANCEL SEMAPHORE REQUEST (\$CNSRQ)  
MACRO CALL, 5-19  
CARD  
CARD ASCII MODE, 6-17  
CARD HOLLERITH-ASCII CODE TABLE  
(TBL), 6-18  
CARD READER IORB HARDWARE/SOFTWARE  
STATUS MAPPING (TBL), 6-21  
CARD READER/CARD READER-PUNCH  
DRIVER, 6-16  
CARD READER/CARD READER-PUNCH IORB  
FIELDS, 6-19  
CARD READER/CARD READER-PUNCH RCT  
FIELDS, 6-20  
CARD READER/CARD READER-PUNCH RCT/  
IORB STATUS MAPPING, 6-20  
CARD VERBATIM MODE, 6-19  
CARD-TO-MEMORY CODE  
ASCII CARD-TO-MEMORY CODE FORMAT  
(FIG), 6-17  
CARTRIDGE DISK  
CARTRIDGE DISK IORB FIELDS, 6-29  
CARTRIDGE DISK RCT FIELDS, 6-30  
CARTRIDGE DISK RCT/IORB  
HARDWARE/SOFTWARE STATUS MAPPING,  
6-31  
DISK DRIVER PROCESSING FOR  
CARTRIDGE DISK, 6-29  
CHARACTERS  
APPEND ASCII CHARACTERS TO PATHNAME,  
5-133  
CONTROL CHARACTERS, D-1  
SPECIAL GRAPHIC CHARACTERS, D-2



INDEX

CLEAN POINT  
 CLEAN POINT (\$CLPNT)  
 MACRO CALL, 5-24

CLEAR  
 CLEAR EXTERNAL SWITCHES (\$CLRSW)  
 MACRO CALL, 5-26

CLOCK  
 CANCEL CLOCK REQUEST (\$CNCRQ)  
 MACRO CALL, 5-15  
 CLOCK REQUEST BLOCK FORMAT, A-2  
 CLOCK REQUEST BLOCK OFFSETS (\$CRBD)  
 MACRO CALL, 5-31  
 CLOCK REQUEST BLOCK (CRB), 5-29  
 CLOCK REQUEST BLOCK (\$CRB)  
 MACRO CALL, 5-28  
 MONITORS SERVICE FUNCTIONS, CLOCK  
 FUNCTIONS, 2-2  
 REQUEST CLOCK (RQCL)  
 MACRO CALL, 5-288

CLOSE FILE  
 CLOSE FILE (\$CLFIL)  
 MACRO CALL, 5-32

CODE  
 ASCII CARD-TO-MEMORY CODE FORMAT  
 (FIG), 6-17  
 CARD HOLLERITH-ASCII CODE TABLE  
 (TBL), 6-13

CODE, FUNCTION  
 INPUT/OUTPUT FUNCTION CODE  
 (TBL) 6-3  
 USER DRIVER INITIALIZE FUNCTION  
 (CODE 0), B-3  
 USER DRIVER READ/MODIFY STATUS  
 FUNCTION (CODE 4), B-5  
 USER DRIVER STOP I/O FUNCTION (CODE  
 2), B-4  
 USER DRIVER WAIT FOR INTERRUPT  
 (CODE 3), B-4  
 USER DRIVER WAIT ON LINE FUNCTION  
 (CODE 1), B-3

CODES  
 COMMUNICATIONS FUNCTION CODES, 6-6  
 DRIVER FUNCTIONS AND FUNCTION  
 CODES, 6-2  
 MACRO CALLS FUNCTION CODES AND  
 REGISTER CONTENTS (TBL), C-2  
 RETURN STATUS CODES IN \$R1  
 REGISTER, 1-6  
 RETURN STATUS CODES (TBL), 6-5  
 SYSTEM SERVICE MACRO CALLS AND  
 FUNCTION CODES, 1-6

COMMAND  
 COMMAND IN (\$CIN) MACRO CALL, 5-36  
 COMMAND LINE PROCESS (\$CMDLN)  
 MACRO CALL, 5-39

COMMAND-IN FILE  
 RETURN COMMAND-IN FILE  
 PATHNAME, 5-355

COMMERCIAL SIMULATOR  
 COMMERCIAL SIMULATOR TRAP, 7-12

COMMUNICATION, TASK  
 TASK COMMUNICATION, 5-360

COMMUNICATIONS  
 COMMUNICATIONS FUNCTION CODES, 6-6  
 MONITOR SERVICE FUNCTIONS  
 COMMUNICATIONS, 2-2

CONCURRENCY CONTROL  
 ALTER CONCURRENCY CONTROL, 5-133  
 CONCURRENCY CONTROL, 5-125

CONDITION, ERROR  
 REPORT ERROR CONDITION (\$RPTER)  
 MACRO CALL, 5-282

CONNECT  
 CONNECT FUNCTION (FC A), 6-7  
 TRAP HANDLER CONNECT (\$TRPHD)  
 MACRO CALL, 5-370  
 USER WRITTEN TRAP HANDLER  
 CONNECT, 5-370

CONNECTION, MESSAGE  
 INITIATE MESSAGE CONNECTION, 5-189

CONSOLE MESSAGE  
 CONSOLE MESSAGE SUPPRESSION  
 (\$CMSUP) MACRO CALL, 5-42

CONTROL  
 ALTER CONCURRENCY CONTROL, 5-133  
 CONCURRENCY CONTROL, 5-125  
 CONTROL CHARACTERS, D-1  
 MONITOR SERVICE FUNCTION, TASK  
 CONTROL, 2-11  
 MONITOR SERVICE FUNCTION, TASK  
 GROUP CONTROL, 2-12  
 PRINT CONTROL BYTE, 6-22  
 PRINTER FORMS CONTROL, 6-22  
 RECORD LOCKS CONTROL, 5-125  
 RESOURCE CONTROL TABLE DEFINITION,  
 1-17  
 RESOURCE CONTROL TABLE (RCT), 6-12  
 TASK CONTROL BLOCK DEFINITION,  
 1-17

CONVERT  
 EXTERNAL DATE/TIME CONVERT TO  
 (\$EXTDT) MACRO CALL, 5-105  
 EXTERNAL TIME CONVERT TO (\$EXTIM)  
 MACRO CALL, 5-108  
 INTERNAL DATE/TIME CONVERT TO  
 (\$INDTM) MACRO CALL, 5-169

COUNT MESSAGE GROUP  
 MESSAGE GROUP COUNT (\$MCMG)  
 MACRO CALL, 5-181

CREATE MACRO CALLS  
 CREATE DIRECTORY (\$CRDIR)  
 MACRO CALL, 5-44  
 CREATE FILE (\$CRFIL)  
 MACRO CALL, 5-47  
 CREATE GROUP (\$CRGRP)  
 MACRO CALL, 5-56  
 CREATE OVERLAY AREA TABLE (\$CROAT)  
 MACRO CALL, 5-60  
 CREATE TASK (\$CRTSK)  
 MACRO CALL, 5-63

DATA STRUCTURES  
 DATA STRUCTURE FORMAT, A-1  
 DATA STRUCTURE GENERATION, 4-1  
 DATA STRUCTURES, 6-7  
 FILE SYSTEM DATA STRUCTURES, 4-5  
 INITIALIZE, ALLOCATE GROUP DATA  
 STRUCTURES, 5-57  
 INITIALIZE, ALLOCATE TASK DATA  
 STRUCTURES, 5-64  
 MONITOR SERVICES DATA STRUCTURES,  
 4-1  
 REMOVE GROUP DATA STRUCTURES, 5-71  
 REMOVE TASK DATA STRUCTURES, 5-76

DATA MANAGEMENT  
 DATA MANAGEMENT FUNCTIONS, 3-16

DATE/TIME  
 EXTERNAL DATE/TIME CONVERT TO  
 (\$EXTDT) MACRO CALL, 5-105  
 GET DATE/TIME (\$GDTM)  
 MACRO CALL, 5-121  
 INTERNAL DATE/TIME CONVERT TO  
 (\$INDTM) MACRO CALL, 5-169  
 MONITOR SERVICE FUNCTIONS, DATE/TIME  
 FUNCTIONS, 2-3

DEFINE SEMAPHORE  
 DEFINE SEMAPHORE (\$DFSM)  
 MACRO CALL, 5-67

DEFINITIONS  
 LOGICAL RESOURCE NUMBER (LRN),  
 1-16  
 LOGICAL RESOURCE TABLE (LRT),  
 1-16  
 OFFSETS DEFINITION MACRO CALLS  
 (TBL), 3-12  
 RESOURCE CONTROL TABLE (RCT)  
 1-17  
 TASK CONTROL BLOCK, 1-17

DEFINITION, OFFSETS  
 FIB OFFSETS DEFINITIONS, 3-11  
 FILE SYSTEM OFFSETS  
 DEFINITIONS, 4-5

DELETE  
 DELETE A FILE, 5-269  
 DELETE GROUP (\$DLGRP) MACRO CALL,  
 5-71  
 DELETE RECORD (\$DLREC), MACRO CALL,  
 5-73  
 DELETE TASK (\$DLTSK) MACRO CALL, 5-76

DEVICE  
 DEVICE BREAK NOTIFICATION FUNCTION  
 (FC 9), 6-6  
 DEVICE ERROR LOGGING, 5-94, 5-96  
 DEVICE DRIVER READ DISABLED DEVICE  
 FUNCTION (FC E), 6-5  
 DISABLE DEVICE ON ATTENTION (\$DSDV)  
 MACRO CALL, 5-78  
 ENABLE DEVICE (\$ENDV) MACRO CALL,  
 5-84  
 RESET DEVICE ATTENTION (\$RDVAT)  
 MACRO CALL, 5-309  
 USER DRIVER, LOCATE RCT FOR DEVICE  
 (ZXSRICT) SUBROUTINE, B-5

DEVICE DRIVER  
 CALLER INTERFACE WITH DEVICE DRIVER,  
 6-14  
 DEVICE DRIVER CONVENTIONS, 6-2  
 DEVICE DRIVERS, 6-16  
 DEVICE DRIVER, POSITION BLOCK  
 FUNCTION (FC 4), 6-6  
 DEVICE DRIVER, POSITION TAPE MARK  
 FUNCTION (FC 6), 6-6  
 DEVICE DRIVER, READ DISABLED DEVICE  
 FUNCTION (FC E), 6-5  
 DEVICE DRIVER, READ FUNCTION  
 (FC 2), 6-5  
 DEVICE DRIVER, WAIT ONLINE FUNCTION  
 (FC 0), 6-4  
 DEVICE DRIVER, WRITE FUNCTION  
 (FC 1), 6-4  
 DEVICE DRIVER, WRITE TAPE MARK  
 FUNCTION (FC 3), 6-6  
 INPUT/OUTPUT DEVICE DRIVERS, 6-1  
 USER-WRITTEN DEVICE DRIVER, B-1

DIAL, SET  
 SET DIAL (\$SDL) MACRO CALL, 5-324

DISABLE  
 DISABLE DEVICE ON ATTENTION (\$DSDV)  
 MACRO CALL, 5-78  
 DISABLE USER TRAP (\$DSTRP)  
 MACRO CALL, 5-80

DISCONNECT FUNCTION  
 DISCONNECT FUNCTION (FC B), 6-7

DISK  
 CARTRIDGE DISK IOFB FIELDS, 6-29  
 CARTRIDGE DISK RCT FIELDS, 6-30  
 CARTRIDGE DISK RCT/IOFB HARDWARE/  
 SOFTWARE STATUS MAPPING, 6-31  
 DISK DRIVER, 6-26

INDEX

DISK (CONT)

DISK DRIVER PROCESSING FOR  
CARTRIDGE DISK, 6-29  
DISK DRIVER PROCESSING FOR  
DISKETTE, 6-26  
DISK DRIVER PROCESSING FOR MASS  
STORAGE UNIT, 6-32

DISKETTE

DISK DRIVER PROCESSING FOR  
DISKETTE, 6-26  
DISKETTE IORB FIELDS, 6-27  
DISKETTE RCT FIELDS, 6-28  
DISKETTE RCT/IORB HARDWARE/SOFTWARE  
STATUS MAPPING, 6-28

DISSOCIATE

DISSOCIATE FILE (\$DSFIL)  
MACRO CALL, 5-82

DRIVER

ASR/KSR DRIVER, 6-34  
CALLER INTERFACE WITH DEVICE  
DRIVER, 6-14  
CARD READER/CARD READER-PUNCH  
DRIVER, 6-16  
DEVICE DRIVER CONVENTIONS, 6-2  
DISK DRIVER, 6-26  
DISK DRIVER PROCESSING FOR  
CARTRIDGE DISK, 6-29  
DISK DRIVER PROCESSING FOR  
DISKETTE, 6-26  
DISK DRIVER PROCESSING FOR MASS  
STORAGE UNIT, 6-32  
DRIVER FUNCTIONS AND FUNCTION  
CODES, 6-2  
DRIVER INTERFACE IN WRITING A  
DRIVER, B-2  
DRIVER USABLE SYSTEM  
FUNCTIONS, B-3  
INPUT/OUTPUT DRIVER, 6-1  
I/O REQUIREMENTS FOR USER DEVICE  
DRIVER, B-3  
MAGNETIC TAPE DRIVER, 6-39  
PRINTER DRIVER, 6-22  
REQUEST DEVICE DRIVER, 6-2  
SYSTEM BUILDING IN WRITING A  
DRIVER, B-1  
USER-WRITTEN DRIVER, B-1

DRIVES, TAPE

CHARACTERISTICS OF SUPPORTED TAPE  
DRIVES (TBL), 6-40

EBCDIC

ASCII AND EBCDIC CHARACTER  
SETS, D-1

EBCDIC/HEXADECIMAL/BINARY

EBCDIC/HEXADECIMAL/BINARY  
EQUIVALENTS (TBL), D-3

ENABLE

ENABLE DEVICE (\$ENDV)  
MACRO CALL, 5-84  
ENABLE USER TRAP (\$ENTRP)  
MACRO CALL, 5-86

ENABLED, TRAP

TRAP ENABLED, 7-2  
TRAP NOT ENABLED, 7-2

ENCLOSURE, MESSAGE GROUP

MESSAGE GROUP ENCLOSURE LEVEL,  
5-192, 5-202

END

ERROR LOGGING END (\$LEND)  
MACRO CALL, 5-88

EQUIVALENTS, CHARACTER

ASCII/HEXADECIMAL EQUIVALENTS  
(TBL), D-2  
EBCDIC/HEXADECIMAL/BINARY  
EQUIVALENTS (TBL), D-3

ERROR

DEVICE ERROR LOGGING, 5-94, 5-96  
ERROR LOGGING END (\$LEND)  
MACRO CALL, 5-88  
ERROR LOGGING INFORMATION EXCHANGE  
(\$ELEX) MACRO CALL, 5-90  
ERROR LOGGING INFORMATION, GET  
(\$ELGT) MACRO CALL, 5-94  
ERROR LOGGING START (\$ELST)  
MACRO CALL, 5-96  
ERROR OUT (\$EROUT)  
MACRO CALL, 5-99  
MONITOR SERVICE FUNCTIONS, ERROR  
HANDLING, 2-3  
REPORT ERROR CONDITION (\$RPTER)  
MACRO CALL, 5-282  
RESET ERROR LOGGING TABLE, 5-97  
SAVE ERROR LOG VALUES, 5-90  
SUMMARY ERROR LOG  
INFORMATION, 5-88  
USER-GENERATED TABLE FOR ERROR  
LOGGING MACRO CALLS (TBL), 5-91

ERROR-OUT FILE

ERROR-OUT FILE, 5-99

EXAMPLES FOR FILE SYSTEM MACRO CALLS

ASSUMPTIONS FOR FILE SYSTEM  
EXAMPLES, 3-13

EXCHANGE ERROR LOGGING

ERROR LOGGING INFORMATION EXCHANGE  
(\$ELEX) MACRO CALL, 5-90

EXECUTE

EXECUTE LEAD TASK, 5-292  
OVERLAY AREA RESERVE, AND EXECUTE  
OVERLAY (\$OVRSV)  
MACRO CALL, 5-230

INDEX

EXECUTE (CONT)

OVERLAY EXECUTE (\$OVEXC)  
MACRO CALL, 5-234

EXPAND PATHNAME

EXPAND PATHNAME (\$XPATH)  
MACRO CALL, 5-102

EXTERNAL

CLEAR EXTERNAL SWITCHES (\$CLRSW)  
MACRO CALL, 5-26  
EXTERNAL DATE/TIME, CONVERT TO  
(\$EXTDT) MACRO CALL, 5-105  
EXTERNAL TIME, CONVERT TO (\$EXTIM)  
MACRO CALL, 5-108  
MONITOR SERVICE FUNCTIONS EXTERNAL  
SWITCH FUNCTIONS, 2-3  
READ EXTERNAL SWITCHES (\$RDSW)  
MACRO CALL, 5-258  
SET EXTERNAL SWITCHES (\$SETSW)  
MACRO CALL, 5-328

FACILITY, MESSAGE

MESSAGE FACILITY MESSAGE GROUP  
REQUEST BLOCKS, A-15  
MONITOR SERVICE FUNCTIONS MESSAGE  
FACILITY, 2-5

FIB (FILE INFORMATION BLOCK)

CONTENTS OF FILE INFORMATION BLOCK  
(FIB) (TBL), 3-2, A-5  
FIB OFFSETS DEFINITIONS, 3-11  
FILE INFORMATION BLOCK (FIB),  
3-2, 4-5  
FILE INFORMATION BLOCK (\$FIB)  
MACRO CALL, 5-111  
FORMAT OF FILE INFORMATION BLOCK  
(FIB) (FIG), A-4  
PROGRAM VIEW ENTRY IN FIB, 3-6

FILE

ASSOCIATE FILE (\$ASFIL)  
MACRO CALL, 5-10  
ASSUMPTIONS FOR FILE SYSTEM  
EXAMPLES, 3-13  
CLOSE FILE (\$CLFIL)  
MACRO CALL, 5-32  
CREATE FILE (\$CRFIL)  
MACRO CALL, 5-47  
DELETE A FILE, 5-269  
DISSOCIATE FILE (\$DSFIL)  
MACRO CALL, 5-82  
ERROR-OUT FILE, 5-99  
FILE INFORMATION BLOCK OFFSETS  
(\$TFIB) MACRO CALL, 5-119  
FILE INFORMATION BLOCK (FIB)  
3-2, 4-5, A-4  
FILE INFORMATION BLOCK (\$FIB)  
MACRO CALL, 5-111  
GET FILE INFORMATION (\$GIFIL)  
MACRO CALL, 5-143  
GET FILE (\$GTFIL)  
MACRO CALL, 5-124

FILE (CONT)

LIFE CYCLE OF A FILE (FIG), 3-17  
LOCATE RESERVE FILE, 5-124  
OPEN FILE (\$OPFIL)  
MACRO CALL, 5-215  
READ USER-IN FILE, 5-376  
RELEASE FILE (\$RLFIL)  
MACRO CALL, 5-269  
REMOVE FILE (\$RMFIL)  
MACRO CALL, 5-276  
RETURN COMMAND-IN FILE  
PATHNAME, 5-355  
TERMINATE FILE PROCESSING, 5-32  
TEST FILE (\$TOFIL)  
MACRO CALL, 5-367  
TEST FILE (\$TIFIL)  
MACRO CALL, 5-367  
USER-IN FILE, 5-211  
USER-OUT FILE, 5-213  
WAIT FILE (\$WIFIL)  
MACRO CALL, 5-385  
WAIT FILE (\$WOFIL)  
MACRO CALL, 5-385  
WRITE TO USER-OUT FILE, 5-378

FILE MANAGEMENT

FILE MANAGEMENT FUNCTIONS, 3-14

FILE SYSTEM

FILE SYSTEM DATA STRUCTURES, 4-5  
FILE SYSTEM FUNCTIONS, 3-1  
FILE SYSTEM OFFSETS  
DEFINITIONS, 4-5

FILE /RENAME

RENAME FILE/RENAME DIRECTORY  
(\$RNFIL) MACRO CALL, 5-279

FUNCTION AND FUNCTION CODE

COMMUNICATIONS FUNCTION CODES, 6-6  
CONNECT FUNCTION (FC A), 6-7  
DEVICE BREAK NOTIFICATION FUNCTION  
(FC 9), 6-6  
DEVICE DRIVER, POSITION BLOCK  
FUNCTION (FC 4), 6-6  
DEVICE DRIVER, POSITION TAPE MARK  
FUNCTION (FC 6), 6-6  
DEVICE DRIVER, READ DISABLED DEVICE  
FUNCTION (FC E), 6-5  
DEVICE DRIVER, READ FUNCTION  
(FC 2), 6-5  
DEVICE DRIVER, WAIT ONLINE FUNCTION  
(FC 0), 6-4  
DEVICE DRIVER, WRITE FUNCTION  
(FC 1), 6-4  
DEVICE DRIVER, WRITE TAPE MARK  
FUNCTION (FC 3), 6-6  
DISCONNECT FUNCTION (FC B), 6-7  
DRIVER FUNCTIONS AND FUNCTION  
CODES, 6-2  
INPUT/OUTPUT FUNCTION CODE  
(TBL), 6-3

INDEX

FUNCTION AND FUNCTION CODE (CONT)

MACRO CALLS, FUNCTION CODES, AND REGISTER CONTENTS (TBL), C-2  
 SYSTEM SERVICE MACRO CALLS AND FUNCTION CODES, 1-6  
 USER DRIVER, INITIALIZE FUNCTION (CODE 0), 3-3  
 USER DRIVER, READ, MODIFY STATUS FUNCTION (CODE 4), B-5  
 USER DRIVER, STOP I/O FUNCTION (CODE 2), B-4  
 USER DRIVER, WAIT ON LINE FUNCTION (CODE 1), B-3

FUNCTIONS

DATA MANAGEMENT FUNCTIONS, 3-16  
 DRIVER FUNCTIONS AND FUNCTION CODES, 6-2  
 DRIVER USABLE SYSTEM FUNCTIONS, B-3  
 FILE MANAGEMENT FUNCTIONS, 3-14  
 MONITOR SERVICE FUNCTIONS, BATCH FUNCTIONS, 2-1  
 MONITOR SERVICE FUNCTIONS CLOCK FUNCTIONS, 2-2  
 MONITOR SERVICE FUNCTIONS, DATE/TIME FUNCTIONS, 2-3  
 MONITOR SERVICE FUNCTIONS, EXTERNAL SWITCH FUNCTIONS, 2-3  
 MONITOR SERVICE FUNCTIONS, SECONDARY USER TERMINAL FUNCTIONS, 2-8  
 MONITOR SERVICE FUNCTIONS, SEMAPHORE FUNCTIONS, 2-9  
 MONITOR SERVICE FUNCTIONS, STANDARD SYSTEM FILE I/O FUNCTIONS, 2-10  
 \$RQIO MACRO CALL FOR I/O FUNCTIONS, 6-14  
 STORAGE MANAGEMENT FUNCTIONS, 3-18

GENERATE, WAIT LIST

WAIT LIST, GENERATE (\$WLIST)  
 MACRO CALL, 5-388

GET

ERROR LOGGING INFORMATION GET (\$ELGT) MACRO CALL, 5-94  
 GET DATE/TIME (\$GD TM) MACRO CALL, 5-121  
 GET FILE INFORMATION (\$GIFIL) MACRO CALL, 5-143  
 GET FILE (\$GT FIL) MACRO CALL, 5-124  
 GET MEMORY/GET AVAILABLE MEMORY (\$GMEM) MACRO CALL, 5-156  
 GET WORKING DIRECTORY (\$GWDIR) MACRO CALL, 5-161

GROUP

ABORT GROUP REQUEST (\$ABGRQ) MACRO CALL, 5-4  
 ABORT GROUP (\$ABGRP) MACRO CALL, 5-2

GROUP (CONT)

ACTIVATE GROUP (\$ACTVG) MACRO CALL, 5-8  
 CREATE GROUP (\$CRGRP) MACRO CALL, 5-56  
 DELETE GROUP (\$DLGRP) MACRO CALL, 5-71  
 INITIALIZE, ALLOCATE GROUP DATA STRUCTURES, 5-57  
 MESSAGE GROUP ACCEPTOR GROUP, 5-172, 5-189  
 MESSAGE GROUP INITIATOR GROUP, 5-172, 5-189  
 MONITOR SERVICE FUNCTIONS, TASK GROUP CONTROL, 2-12  
 REACTIVATE SUSPENDED TASK GROUP, 5-8  
 REMOVE GROUP DATA STRUCTURES, 5-71  
 REQUEST GROUP (\$RQGRP) MACRO CALL, 5-290  
 SPAWN GROUP (\$SPGRP) MACRO CALL, 5-333  
 SUSPEND GROUP (\$SUSPG) MACRO CALL, 5-345  
 TASK GROUP INPUT (\$TGIN) MACRO CALL, 5-355  
 TASK GROUP USER IDENTIFICATION, 5-373

HOLLERITH-ASCII CODE

CARD HOLLERITH-ASCII CODE TABLE (TBL), 6-18

HOME DIRECTORY

HOME DIRECTORY (\$HDIR) MACRO CALL, 5-163

IDENTIFICATION

ACCOUNT IDENTIFICATION (\$ACTID) MACRO CALL, 5-6  
 BOUND UNIT IDENTIFICATION (\$BUID) MACRO CALL, 5-13  
 MODE IDENTIFICATION (\$MODID) MACRO CALL, 5-208  
 MONITOR SERVICE FUNCTIONS IDENTIFICATION AND INFORMATION, 2-4  
 PERSON IDENTIFICATION (\$PERID) MACRO CALL, 5-252  
 SYSTEM IDENTIFICATION (\$SYSID) MACRO CALL, 5-353  
 TASK GROUP USER IDENTIFICATION, 5-373  
 USER IDENTIFICATION (\$USRID) MACRO CALL, 5-373

IDENTIFIER

MESSAGE GROUP IDENTIFIER, 5-192  
 SEMAPHORE IDENTIFIER, 5-68

I\_ST IN IORB

IORB SOFTWARE STATUS WORD (I\_ST) (TBL), 6-11

INDEX

INFORMATION

MONITOR SERVICE FUNCTIONS  
IDENTIFICATION AND INFORMATION  
2-4  
OPERATOR INFORMATION MESSAGE  
(\$OPMSG) MACRO CALL, 5-222  
SUMMARY ERROR LOG INFORMATION,  
5-88

INITIALIZATION

MESSAGE GROUP INITIALIZATION  
REQUEST BLOCK (MGIRB), 5-187  
MESSAGE GROUP INITIALIZATION  
REQUEST BLOCK (MGIRB) (TBL), A-18  
USER-WRITTEN DRIVER  
INITIALIZATION, B-2

INITIALIZE

INITIALIZE, ALLOCATE, GROUP DATA  
STRUCTURES, 5-57  
INITIALIZE, ALLOCATE, TASK DATA  
STRUCTURES, 5-64  
USER-DRIVER INITIALIZE FUNCTION  
(CODE 0), B-3

INITIATE

INITIATE MESSAGE CONNECTION, 5-189  
MESSAGE GROUP INITIATE (\$MINIT)  
MACRO CALL, 5-189

INITIATOR, MESSAGE GROUP

MESSAGE GROUP INITIATOR GROUP,  
5-172, 5-189

INPUT

ASR/KSR KDYBOARD INPUT, 6-35  
NEW USER INPUT (\$NUIN)  
MACRO CALL, 5-211  
TASK GROUP INPUT (\$TGIN)  
MACRO CALL, 5-355  
USER INPUT (\$USIN)  
MACRO CALL, 5-375

INPUT/OUTPUT

INPUT/OUTPUT DEVICE DRIVERS, 6-1  
INPUT/OUTPUT DRIVERS, 6-1  
INPUT/OUTPUT FUNCTION CODE  
(TBL), 6-3  
INPUT/OUTPUT REQUEST BLOCK, 6-7  
INPUT/OUTPUT REQUEST BLOCK  
FORMAT, A-6  
INPUT/OUTPUT REQUEST BLOCK OFFSETS  
(\$IORBD) MACRO CALL, 5-168  
INPUT/OUTPUT REQUEST BLOCK (\$IORB)  
MACRO CALL, 5-165

INTERNAL

INTERNAL DATE/TIME, CONVERT TO  
(\$INDTM) MACRO CALL, 5-169

INTERRUPT

TRAP INTERRUPT SAVE AREA  
(ISA), 7-6

INTERRUPT (CONT)

TRAP INTERRUPT VECTOR, 7-6  
USER-DRIVER WAIT FOR INTERRUPT  
(CODE 3), B-4

INTERVAL, SUSPEND

SUSPEND FOR INTERVAL (\$SUSPN)  
MACRO CALL, 5-347

IORB

ASR/KSR IORB FIELDS, 6-36  
CARD READER CARD/READER-PUNCH IORB  
FIELDS (TBL), 6-19  
CARD READER/CARD READER-PUNCH IORB  
FIELDS, 6-19  
CARTRIDGE DISK IORB FIELDS, 6-29  
DISKETTE IORB FIELDS, 6-27  
INPUT/OUTPUT REQUEST BLOCK (\$IORB)  
MACRO CALL, 5-165  
IORB IN I/O TRANSFER, 5-294  
IORB SOFTWARE STATUS WORD (I\_ST)  
(TBL), 6-11  
MAGNETIC TAPE IORB FIELDS, 6-42  
MASS STORAGE UNIT IORB  
FIELDS, 6-33  
PRINTER IORB FIELDS, 6-24  
SUMMARY OF IORB FIELDS FOR OPERATOR  
INTERFACE (TBL), A-9

I/O

FORMAT I/O REQUEST BLOCK  
(FIG), 6-8, A-7  
IORB IN I/O TRANSFER, 5-294  
I/O OPERATION SEQUENCE, 6-14  
I/O REQUIREMENTS FOR USER DEVICE  
DRIVER, B-8  
I/O SUBROUTINES (ZIOSUB) FOR  
USER-WRITTEN DRIVERS, B-3  
MONITOR SERVICE FUNCTIONS, PHYSICAL  
I/O, 2-7  
MONITOR SERVICE FUNCTIONS, STANDARD  
SYSTEM FILE I/O FUNCTIONS, 2-10  
REQUEST I/O (\$RQIO)  
MACRO CALL, 5-294  
\$RQIO MACRO CALL FOR I/O  
FUNCTIONS, 6-14  
USER-DRIVER, STOP I/O FUNCTION  
(CODE 2), B-4  
WAIT FOR I/O COMPLETION, 5-385  
WRITING PERIPHERAL I/O DRIVER, B-1

ISA

TRAP INTERRUPT SAVE AREA (ISA), 7-6

KEYBOARD INPUT

ASR/KSR KEYBOARD INPUT, 6-35

LEAD TASK, EXECUTE

EXECUTE LEAD TASK, 5-292

LEVEL, ENCLOSURE

MESSAGE GROUP ENCLOSURE LEVEL,  
5-192, 5-202

## LIST

AUTODIAL TELEPHONE LIST, 5-324  
FORMAT OF WAIT LIST (FIG), A-14  
PARAMETER BLOCK AND WAIT LIST, 4-3  
WAIT LIST FORMAT, A-14  
WAIT LIST, GENERATE (\$WLIST)  
MACRO CALL, 5-388  
WAIT ON REQUEST LIST (\$WAITL)  
MACRO CALL, 5-390

## LOCATE

LOCATE RESERVE FILE, 5-124  
USER-DRIVER LOCATE RCT FOR DEVICE  
(ZXSRTC) SUBROUTINE, B-5

## LOCKS

RECORD LOCKS, 5-24  
RECORD LOCKS CONFLICT, 5-136  
RECORD LOCKS CONTROL, 5-125  
RECORD LOCKS OPERATION, 5-134  
UNLOCK RECORD LOCKS, 5-24

## LOG

SAVE ERROR LOG VALUES, 5-90  
SUMMARY ERROR LOG  
INFORMATION, 5-88

## LOGGING

DEVICE ERROR LOGGING, 5-94, 5-96  
ERROR LOGGING END (\$ELEND)  
MACRO CALL, 5-88  
ERROR LOGGING INFORMATION EXCHANGE  
(\$ELEX) MACRO CALL, 5-90  
ERROR LOGGING INFORMATION, GET  
(\$ELGT) MACRO CALL, 5-94  
ERROR LOGGING, START (\$ELST)  
MACRO CALL, 5-96  
RESET ERROR LOGGING TABLE, 5-97  
USER-GENERATED TABLE FOR ERROR  
LOGGING MACRO CALLS (TBL), 5-91

## LRN

LOGICAL RESOURCE NUMBER (LRN)  
DEFINITION, 1-16  
LRN AS POINTER TO RCT (FIG), 6-15

## MACRO

LOCATION OF MACRO ROUTINES, 1-6  
MACRO CALLS FUNCTION CODES AND  
REGISTER CONTENTS (TBL), C-2  
MACRO ROUTINE/CALL, DESCRIPTIONS OF  
MACRO CALLS, 5-1  
REGISTER CONTENTS FOR SYSTEM  
SERVICE MACRO CALLS, C-1

## MACRO CALL LIST, MACRO NAMES

ABORT GROUP (\$ABGRP), 5-2  
ABORT GROUP REQUEST (\$ABGRQ), 5-4  
ACCOUNT IDENTIFICATION (\$ACTID), 5-6  
ACTIVATE GROUP (\$ACTVG), 5-8  
ASSOCIATE FILE (\$ASFIL), 5-10  
BOUND UNIT IDENTIFICATION  
(BUID), 5-13  
CANCEL CLOCK REQUEST (\$CNCRQ), 5-15

## MACRO CALL LIST, MACRO NAMES (CONT)

CANCEL REQUEST (\$CANRQ), 5-17  
CANCEL SEMAPHORE REQUEST (\$CNSRQ), 5-19  
CHANGE WORKING DIRECTORY (\$CWDIR), 5-21  
CLEAN POINT (\$CLPNT), 5-24  
CLEAR EXTERNAL SWITCHES (\$CLRSW), 5-26  
CLOCK REQUEST BLOCK (\$CRB), 5-28  
CLOCK REQUEST BLOCK OFFSETS  
(\$CRBD), 5-31  
CLOSE FILE (\$CLFIL), 5-32  
COMMAND IN (\$CIN), 5-36  
COMMAND LINE PROCESS (\$CMDLN), 5-39  
CONSOLE MESSAGE SUPPRESSION  
(\$CMSUP), 5-42  
CREATE DIRECTORY (\$CRDIR), 5-44  
CREATE FILE (\$CRFIL), 5-47  
CREATE FILE PARAMETER STRUCTURE  
BLOCK - OFFSETS (\$CRPSB), 5-55  
CREATE GROUP (\$CRGRP), 5-56  
CREATE OVERLAY AREA TABLE  
(\$CROAT), 5-60  
CREATE TASK (\$CRTSK), 5-63  
DEFINE SEMAPHORE (\$DFSM), 5-67  
DELETE GROUP (\$DLGRP), 5-71  
DELETE RECORD (\$DLREC), 5-73  
DELETE TASK (\$DLTSK), 5-76  
DISABLE DEVICE ON ATTENTION  
(\$DSDV), 5-78  
DISABLE USER TRAP (\$DSTRP), 5-80  
DISSOCIATE FILE (\$DSFIL), 5-82  
ENABLE DEVICE (\$ENDV), 5-84  
ENABLE USER TRAP (\$ENTRP), 5-86  
ERROR LOGGING, END (\$ELEND), 5-88  
ERROR LOGGING INFORMATION, EXCHANGE  
(\$ELEX), 5-90  
ERROR LOGGING INFORMATION, GET  
(\$ELGT), 5-94  
ERROR LOGGING, START (\$ELST), 5-96  
ERROR OUT (\$EROUT), 5-99  
EXPAND PATHNAME (\$XPATH), 5-102  
EXTERNAL DATE/TIME, CONVERT TO  
(\$EXTDT), 5-105  
EXTERNAL TIME, CONVERT TO  
(\$EXTIM), 5-108  
FILE INFORMATION BLOCK (\$FIB), 5-111  
FILE INFORMATION BLOCK OFFSETS  
(\$TFIB), 5-119  
GET DATE/TIME (\$GDTM), 5-121  
GET FILE (\$GTFIL), 5-124  
GET FILE PARAMETER STRUCTURE BLOCK  
OFFSET (\$GTPSB), 5-142  
GET FILE INFORMATION (\$GIFIL), 5-143  
GET FILE INFORMATION, FILE  
ATTRIBUTE BLOCK OFFSETS  
(\$GIFAB), 5-153  
GET FILE INFORMATION, KEY  
DESCRIPTORS BLOCK OFFSETS  
(\$GIKDB), 5-154  
GET FILE INFORMATION, PARAMETER  
STRUCTURE BLOCK OFFSETS  
(\$GIPSB), 5-155  
GET MEMORY/GET AVAILABLE MEMORY  
(\$GMEM), 5-156

INDEX

MACRO CALL LIST, MACRO NAMES (CONT)  
 GET WORKING DIRECTORY (\$GWDIR),  
 5-161  
 HOME DIRECTORY (\$HDIR), 5-163  
 INPUT/OUTPUT REQUEST BLOCK (\$IORB),  
 5-165  
 INPUT/OUTPUT REQUEST BLOCK OFFSETS  
 (\$IORBD), 5-168  
 INTERNAL DATE/TIME, CONVERT TO  
 (\$INDTM), 5-169  
 MESSAGE GROUP, ACCEPT (\$MACPT),  
 5-172  
 MESSAGE GROUP CONTROL REQUEST BLOCK  
 (\$MGCRB), 5-175  
 MESSAGE GROUP CONTROL REQUEST BLOCK  
 OFFSETS (\$MGCRT), 5-180  
 MESSAGE GROUP, COUNT (\$MCMG), 5-181  
 MESSAGE GROUP INITIALIZATION  
 REQUEST BLOCK (\$MGIRB), 5-184  
 MESSAGE GROUP INITIALIZATION  
 REQUEST BLOCK OFFSETS (\$MGIRT),  
 5-188  
 MESSAGE GROUP, INITIATE (\$MINIT),  
 5-189  
 MESSAGE GROUP, RECEIVE (\$MRECV),  
 5-192  
 MESSAGE GROUP RECOVERY REQUEST  
 BLOCK (\$MGRRB), 5-196  
 MESSAGE GROUP RECOVERY REQUEST  
 BLOCK OFFSETS (\$MGRRT), 5-200  
 MESSAGE GROUP, SEND (\$MSEND), 5-201  
 MESSAGE GROUP, TERMINATE (\$MTMG),  
 5-205  
 MODE IDENTIFICATION (\$MODID), 5-208  
 NEW PROCESS (\$NPROC), 5-210  
 NEW USER INPUT (\$NUIN), 5-211  
 NEW USER OUTPUT (\$NUOUT), 5-213  
 OPEN FILE (\$OPFIL), 5-215  
 OPERATOR INFORMATION MESSAGE  
 (\$OPMSG), 5-222  
 OPERATOR RESPONSE MESSAGE (\$OPRSP),  
 5-225  
 OVERLAY AREA, RELEASE (\$OVRLS),  
 5-228  
 OVERLAY AREA RESERVE, AND EXECUTE  
 OVERLAY (\$OVRSV), 5-230  
 OVERLAY, EXECUTE (\$OVEXC), 5-234  
 OVERLAY, LOAD (\$OVLD), 5-237  
 OVERLAY RELEASE, WAIT, AND RECALL  
 (\$OVRL), 5-240  
 OVERLAY STATUS (\$OVST), 5-244  
 OVERLAY, UNLOAD (\$OVUN), 5-247  
 PARAMETER BLOCK (\$PRBLK), 5-250  
 PERSON IDENTIFICATION (\$PERID),  
 5-252  
 READ BLOCK (\$RDBLK), 5-254  
 READ EXTERNAL SWITCHES (\$RDWS),  
 5-258  
 READ RECORD (\$RDREC), 5-260  
 RELEASE DIRECTORY (\$RLDIR), 5-266  
 RELEASE FILE (\$RLFIL), 5-269  
 RELEASE SEMAPHORE (\$RLSM), 5-272  
 RELEASE TERMINAL (\$RLTML), 5-274  
 REMOVE FILE (\$RMFIL), 5-276

MACRO CALL LIST, MACRO NAMES (CONT)  
 RENAME FILE/RENAME DIRECTORY  
 (\$RNFIL), 5-279  
 REPORT ERROR CONDITION (\$RPTER),  
 5-282  
 REQUEST BATCH (\$RQBAT), 5-285  
 REQUEST CLOCK (\$RQCL), 5-288  
 REQUEST GROUP (\$RQGRP), 5-290  
 REQUEST I/O (\$RQIO), 5-294  
 REQUEST SEMAPHORE (\$RQSM), 5-297  
 REQUEST TASK (\$RQTSK), 5-300  
 REQUEST TERMINAL (\$RQTML), 5-303  
 RESERVE SEMAPHORE (\$RSVSM), 5-306  
 RESET DEVICE ATTENTION (\$RDVAT),  
 5-309  
 RETURN (\$RETRN), 5-311  
 RETURN MEMORY/RETURN PARTIAL BLOCK  
 OF MEMORY (\$RMEM), 5-313  
 RETURN REQUEST BLOCK ADDRESS  
 (\$RBADD), 5-316  
 REWRITE RECORD (\$RWREC), 5-318  
 SEMAPHORE REQUEST BLOCK (\$SRB),  
 5-321  
 SEMAPHORE REQUEST BLOCK OFFSETS  
 (\$SRBD), 5-323  
 SET DIAL (\$SDL), 5-324  
 SET EXTERNAL SWITCHES (\$SETSW),  
 5-328  
 SET TERMINAL CHARACTERISTICS (\$STTY),  
 5-330  
 SPAWN GROUP (\$SPGRP), 5-333  
 SPAWN TASK (\$SPTSK), 5-339  
 STATUS MEMORY POOL (\$STMP), 5-343  
 SUSPEND GROUP (\$SUSPG), 5-345  
 SUSPEND FOR INTERVAL (\$SUSPN), 5-347  
 SUSPEND UNTIL TIME (\$SUSPN), 5-350  
 SYSTEM IDENTIFICATION (\$SYSID),  
 5-353  
 TASK GROUP INPUT (\$TGIN), 5-355  
 TASK REQUEST BLOCK (\$TRB), 5-357  
 TASK REQUEST BLOCK OFFSETS (\$TRBD),  
 5-361  
 TERMINATE REQUEST (\$TRMRQ), 5-362  
 TEST COMPLETION STATUS (\$TEST),  
 5-365  
 TEST FILE (\$TIFIL), 5-367  
 TEST FILE (\$TOFIL), 5-367  
 TRAP HANDLER CONNECT (\$TRPHD), 5-370  
 USER IDENTIFICATION (\$USRID), 5-373  
 USER INPUT (\$USIN), 5-375  
 USER OUTPUT (\$USOUT), 5-378  
 WAIT (\$WAIT), 5-381  
 WAIT BLOCK (\$WTBLK), 5-383  
 WAIT FILE (WIFIL), 5-385  
 WAIT FILE (\$WOFIL), 5-385  
 WAIT LIST, GENERATE (\$WLIST), 5-388  
 WAIT ON REQUEST LIST (\$WAITL), 5-390  
 WAIT BLOCK (\$WRBLK), 5-393  
 WAIT RECORD (\$WRREC), 5-397



INDEX

MACRO CALLS

OFFSETS DEFINITION MACRO CALLS (TBL), 3-12  
 SYSTEM SERVICE MACRO CALLS AND FUNCTION CODES, 1-6  
 SYSTEM SERVICE MACRO CALLS (TBL), 1-7

MAGNETIC TAPE

MAGNETIC TAPE DRIVER, 6-39  
 MAGNETIC TAPE FILE SEARCH RULES, 5-217  
 MAGNETIC TAPE IORB FIELDS, 6-42  
 MAGNETIC TAPE RCT FIELDS, 6-43  
 MAGNETIC TAPE RCT/IORB HARDWARE/SOFTWARE STATUS MAPPING, 6-43

MAILBOX

MESSAGE QUEUE MAILBOX, 5-172

MARK, TAPE

DEVICE-DRIVER, POSITION TAPE MARK FUNCTION (FC 6), 6-6  
 DEVICE-DRIVER, WRITE TAPE MARK FUNCTION (FC 3), 6-6

MASS STORAGE UNIT

DISK DRIVER PROCESSING FOR MASS STORAGE UNIT, 6-32  
 MASS STORAGE UNIT IORB FIELDS, 6-33  
 MASS STORAGE UNIT RCT FIELDS, 6-33  
 MASS STORAGE UNIT RCT/IORB HARDWARE/SOFTWARE STATUS MAPPING, 6-34

MEMORY

CONTENTS OF TRAP-RELATED MEMORY AREAS, 7-2  
 GET MEMORY/GET AVAILABLE MEMORY (\$GMEM) MACRO CALL, 5-156  
 MONITOR SERVICE FUNCTIONS, MEMORY ALLOCATION, 2-4  
 STATUS MEMORY POOL (\$STMP) MACRO CALL, 5-343

MESSAGE

CONSOLE MESSAGE SUPPRESSION (\$CMSUP) MACRO CALL, 5-42  
 INITIATE MESSAGE CONNECTION, 5-189  
 MESSAGE FACILITY MESSAGE GROUP REQUEST BLOCKS, A-15  
 MESSAGE QUEUE MAILBOX, 5-172  
 MONITOR SERVICE FUNCTIONS, MESSAGE FACILITY, 2-5  
 OPERATOR INFORMATION MESSAGE (\$OPMSG) MACRO CALL, 5-222  
 OPERATOR RESPONSE MESSAGE (\$OPRSP) MACRO CALL, 5-225

MESSAGE GROUP

MESSAGE FACILITY MESSAGE GROUP REQUEST BLOCKS, A-15  
 MESSAGE GROUP, ACCEPT (\$MACPT) MACRO CALL, 5-172  
 MESSAGE GROUP ACCEPTOR GROUP, 5-172, 5-189  
 MESSAGE GROUP CONTROL REQUEST BLOCK (MGCRB), 5-179  
 MESSAGE GROUP CONTROL REQUEST BLOCK (MGCRB) (TBL), A-15  
 MESSAGE GROUP CONTROL REQUEST BLOCK (\$MGCRB) MACRO CALL, 5-175  
 MESSAGE GROUP, COUNT (\$MCMG) MACRO CALL, 5-181  
 MESSAGE GROUP ENCLOSURE LEVEL, 5-192, 5-202  
 MESSAGE GROUP IDENTIFIER, 5-192  
 MESSAGE GROUP INITIALIZATION REQUEST BLOCK (MGIRB), 5-187  
 MESSAGE GROUP INITIALIZATION REQUEST BLOCK (MGIRB) (TBL), A-18  
 MESSAGE GROUP INITIATE (\$MINIT) MACRO CALL, 5-189  
 MESSAGE GROUP INITIATOR GROUP, 5-172, 5-189  
 MESSAGE GROUP QUARANTINE UNIT, 5-192, 5-201  
 MESSAGE GROUP RECEIVE (\$MRECV) MACRO CALL, 5-192  
 MESSAGE GROUP RECOVERY REQUEST BLOCK (MGRRB), 5-199  
 MESSAGE GROUP RECOVERY REQUEST BLOCK (MGRRB) (TBL), A-22  
 MESSAGE GROUP RECOVERY REQUEST BLOCK (\$MGRRB) MACRO CALL, 5-196  
 MESSAGE GROUP SEND (\$MSEND) MACRO CALL, 5-201  
 MESSAGE GROUP TERMINATE (\$MTMG) MACRO CALL, 5-205  
 TERMINATE MESSAGE GROUP, 5-203

MGCRB

ARGUMENT VALUES FOR \$MGCRB MACRO CALL (TBL), 5-176  
 MESSAGE GROUP CONTROL REQUEST BLOCK (MGCRB), 5-179  
 MESSAGE GROUP CONTROL REQUEST BLOCK (MGCRB) (TBL), A-15  
 MGCRB ARGUMENT VALUES FOR \$MRECV MACRO CALL (TBL), 5-193  
 MGCRB ARGUMENT VALUES FOR \$MSEND MACRO CALL (TBL), 5-202

MGIRB

ARGUMENT VALUES FOR \$MGIRB MACRO CALL (TBL), 5-185  
 MESSAGE GROUP INITIALIZATION REQUEST BLOCK (MGIRB), 5-187  
 MESSAGE GROUP INITIALIZATION REQUEST BLOCK (MGIRB) (TBL), A-18

INDEX

MGIRB (CONT)

MGIRB ARGUMENT VALUES FOR \$MACPT  
MACRO CALL (TBL), 5-173  
MGIRB ARGUMENT VALUES FOR \$MCMG  
MACRO CALL (TBL), 5-182  
MGIRB ARGUMENT VALUES FOR \$MINIT  
MACRO CALL (TBL), 5-190

MGRRB

ARGUMENT VALUES FOR \$MGRRB  
MACRO CALL (TBL), 5-197  
MESSAGE GROUP RECOVERY REQUEST  
BLOCK (MGRRB), 5-199  
MESSAGE GROUP RECOVERY REQUEST  
BLOCK (MGRRB) (TBL), A-22  
MGRRB ARGUMENT VALUES FOR \$MTMG  
MACRO CALL (TBL), 5-207

MODE

CARD ASCII MODE, 6-17  
CARD VERBATIM MODE, 6-19  
CARD VERBATIM MODE FORMAT (FIG),  
6-19  
MODE IDENTIFICATION (\$MODID)  
MACRO CALL, 5-208

MONITOR SERVICE FUNCTIONS

MONITOR SERVICE FUNCTIONS, 2-1  
MONITOR SERVICE FUNCTIONS, BATCH  
2-1  
MONITOR SERVICE FUNCTIONS, CLOCK  
2-2  
MONITOR SERVICE FUNCTIONS,  
COMMUNICATIONS, 2-2  
MONITOR SERVICE FUNCTIONS,  
DATE/TIME, 2-3  
MONITOR SERVICE FUNCTIONS, ERROR  
HANDLING, 2-3  
MONITOR SERVICE FUNCTIONS, EXTERNAL  
SWITCH, 2-3  
MONITOR SERVICE FUNCTIONS,  
IDENTIFICATION AND INFORMATION,  
2-4  
MONITOR SERVICE FUNCTIONS, MESSAGE  
FACILITY, 2-5  
MONITOR SERVICE FUNCTIONS, OPERATOR  
INTERFACE, 2-6  
MONITOR SERVICE FUNCTIONS, OVERLAY  
HANDLING, 2-6  
MONITOR SERVICE FUNCTIONS, PHYSICAL  
I/O, 2-7  
MONITOR SERVICE FUNCTIONS, REQUEST  
AND RETURN, 2-8  
MONITOR SERVICE FUNCTIONS,  
SECONDARY USER TERMINAL, 2-8  
MONITOR SERVICE FUNCTIONS, STANDARD  
SYSTEM FILE I/O, 2-10  
MONITOR SERVICE FUNCTIONS, TASK  
CONTROL, 2-11  
MONITOR SERVICE FUNCTIONS, TASK  
GROUP CONTROL, 2-12  
MONITOR SERVICE FUNCTIONS, TRAP  
HANDLING, 2-13

MONITOR SERVICES DATA STRUCTURES  
MONITOR SERVICES, DATA  
STRUCTURES, 4-1

NUMBER, LOGICAL RESOURCE

LOGICAL RESOURCE NUMBER (LRN),  
DEFINITION, 1-16

OFFSETS

CLOCK REQUEST BLOCK OFFSETS (\$CRBD)  
MACRO CALL, 5-31  
FIB OFFSETS DEFINITIONS, 3-11  
FILE INFORMATION BLOCK OFFSETS  
(\$TFIB) MACRO CALL, 5-119  
FILE SYSTEM OFFSETS DEFINITIONS,  
4-5  
INPUT/OUTPUT REQUEST BLOCK OFFSETS  
(\$IORBD) MACRO CALL, 5-168  
OFFSETS DEFINITION MACRO CALLS  
(TBL), 3-12  
REQUEST BLOCK OFFSETS, 4-4  
SEMAPHORE REQUEST BLOCK OFFSETS  
(\$SRBD) MACRO CALL, 5-323  
TASK REQUEST BLOCK OFFSETS (\$TRBD)  
MACRO CALL, 5-361

OPEN FILE

OPEN FILE (\$OPFIL)  
MACRO CALL, 5-215

OPERATOR

MONITOR SERVICE FUNCTIONS, OPERATOR  
INTERFACE, 2-6  
OPERATOR INFORMATION MESSAGE  
(\$OPMSG) MACRO CALL, 5-222  
OPERATOR RESPONSE MESSAGE (\$OPRSP)  
MACRO CALL, 5-225  
SUMMARY OF IOFB FIELDS FOR OPERATOR  
INTERFACE (TBL), A-9

OUTPUT

ASR/KSR PRINTER OUTPUT, 6-36  
NEW USER OUTPUT (\$NUOUT)  
MACRO CALL, 5-213  
USER OUTPUT (\$USOUT)  
MACRO CALL, 5-378  
USER-DRIVER, OUTPUT ADDRESS AND  
RANGE (ZIOD) SUBROUTINE, B-6

OVERLAY

CREATE OVERLAY AREA TABLE (\$CROAT)  
MACRO CALL, 5-60  
MONITOR SERVICE FUNCTIONS, OVERLAY  
HANDLING, 2-6

INDEX

OVERLAY (CONT)  
 OVERLAY AREA, RELEASE (\$OVRLS)  
 MACRO CALL, 5-228  
 OVERLAY AREA, RESERVE AND EXECUTE  
 OVERLAY (\$OVRSV)  
 MACRO CALL, 5-230  
 OVERLAY, EXECUTE (\$OVEXC)  
 MACRO CALL, 5-234  
 OVERLAY, LOAD (\$OVLD)  
 MACRO CALL, 5-237  
 OVERLAY, RELEASE, WAIT AND RECALL  
 (\$OVRCL) MACRO CALL, 5-240  
 OVERLAY STATUS (\$OVST)  
 MACRO CALL, 5-244  
 OVERLAY, UNLOAD (\$OVUN)  
 MACRO CALL, 5-247

PACKED TAPE  
 PACKED AND 6-BIT MODES ON 7-TRACK  
 TAPE (FIG), 6-39

PARAMETER  
 FORMAT OF PARAMETER BLOCK  
 (FIG), A-14  
 PARAMETER BLOCK AND WAIT LIST, 4-3  
 PARAMETER BLOCK FORMAT, A-13  
 PARAMETER BLOCK (\$PRBLK)  
 MACRO CALL, 5-250

PATHNAME  
 APPEND ASCII CHARACTERS TO  
 PATHNAME, 5-133  
 EXPAND PATHNAME (\$XPAT)  
 MACRO CALL, 5-102  
 RETURN COMMAND-IN FILE  
 PATHNAME, 5-355  
 RETURN WORKING DIRECTORY  
 PATHNAME, 5-161

PERSON  
 PERSON IDENTIFICATION (\$PERID)  
 MACRO CALL, 5-252

PHYSICAL I/O FUNCTIONS  
 PHYSICAL I/O, 2-7

P-OP  
 P-OP OPERATION, 5-306

P-TEST  
 P-TEST OPERATION, 5-306

POINT  
 CLEAN POINT (\$CLPNT)  
 MACRO CALL, 5-24

POINTER  
 LRN AS POINTER TO RCT (FIG), 6-15  
 POINTER TO NEXT TRAP SAVE AREA  
 (NATSAT), 7-5  
 READ POINTER, 5-263

POOL, MEMORY  
 STATUS MEMORY POOL (\$STMP)  
 MACRO CALL, 5-343

POSITION  
 DEVICE DRIVER, POSITION BLOCK  
 FUNCTION (FC 4), 6-6  
 DEVICE DRIVER POSITION TAPE MARK  
 FUNCTION (FC 6), 6-6

PRINT CONTROL  
 PRINT CONTROL BYTE, 6-22

PRINTER  
 ASR/KSR PRINTER OUTPUT, 6-36  
 PRINTER DRIVER, 6-22  
 PRINTER FORMS CONTROL, 6-22  
 PRINTER IORB FIELDS, 6-24  
 PRINTER RCT FIELDS, 6-25  
 PRINTER RCT/IORB HARDWARE/SOFTWARE  
 STATUS MAPPING, 6-25

PROCESS  
 COMMAND LINE PROCESS (\$CMDLN)  
 MACRO CALL, 5-39  
 NEW PROCESS (\$NPROC)  
 MACRO CALL, 5-210

PROCESSING  
 DISK DRIVER PROCESSING FOR  
 CARTRIDGE DISK, 6-29  
 DISK DRIVER PROCESSING FOR  
 DISKETTE, 6-26  
 DISK DRIVER PROCESSING FOR MASS  
 STORAGE UNIT, 6-32  
 TERMINATE FILE PROCESSING, 5-32

PROGRAM VIEW  
 CONTENTS OF PROGRAM VIEW ENTRY IN  
 FIB (TBL), 3-7  
 PROGRAM VIEW ENTRY IN FIB, 3-6

QUARANTINE UNIT  
 MESSAGE GROUP QUARANTINE UNIT,  
 5-192, 5-201

QUEUE  
 MESSAGE QUEUE MAILBOX, 5-172  
 REQUEST QUEUE, 1-17  
 TASK REQUEST QUEUE, 1-16, 1-17

\$R1 REGISTER  
 RETURN STATUS CODES IN \$R1  
 REGISTER, 1-6

RANGE  
 RESIDUAL RANGE, 6-13  
 USER-DRIVER OUTPUT ADDRESS AND  
 RANGE (ZOILD) SUBROUTINE, B-6

INDEX

RCT

ASR/KSR RCT FIELDS, 6-37  
 CARD READER/CARD READER-PUNCH RCT  
 FIELDS, 6-20  
 CARTRIDGE DISK RCT FIELDS, 6-30  
 DISKETTE RCT FIELDS, 6-28  
 LRN AS POINTER TO RCT (FIG), 6-15  
 MAGNETIC TAPE RCT FIELDS, 6-43  
 MASS STORAGE UNIT RCT FIELDS, 6-33  
 PRINTER RCT FIELDS, 6-25  
 RCT ATTENTION STATUS INDICATOR,  
 5-309  
 RESOURCE CONTROL TABLE (RCT), 6-12  
 USER-DRIVER LOCATE RCT FOR DEVICE  
 (ZXSRTC) SUBROUTINE, B-5

REACTIVATE TASK GROUP

REACTIVATE SUSPENDED TASK  
 GROUP, 5-8

READ

DEVICE DRIVER, READ DISABLED DEVICE  
 FUNCTION (FC E), 6-5  
 DEVICE DRIVER, READ FUNCTION  
 (FC 2), 6-5  
 READ BLOCK (\$RDBLK) MACRO CALL,  
 5-254  
 READ EXTERNAL SWITCHES (\$RDSW)  
 MACRO CALL, 5-258  
 READ POINTER, 5-263  
 READ RECORD (\$RDREC)  
 MACRO CALL, 5-260  
 READ USER-IN FILE, 5-376

READER-PUNCH

CARD READER/CARD READER-PUNCH  
 DRIVER, 6-16  
 CARD READER/CARD READER-PUNCH IORB  
 FIELDS, 6-19  
 CARD READER/CARD READER-PUNCH RCT  
 FIELDS, 6-20  
 CARD READER/CARD READER-PUNCH  
 RCT/IORB STATUS MAPPING, 6-20

READ/MODIFY

USER-DRIVER, READ/MODIFY STATUS  
 FUNCTION (CODE 4), B-5

RECALL OVERLAY

OVERLAY RELEASE, WAIT, AND RECALL  
 (\$OVRCL) MACRO CALL, 5-240

RECEIVE MESSAGE GROUP

MESSAGE GROUP RECEIVE (\$MRECV)  
 MACRO CALL, 5-192

RECORD

DELETE RECORD (\$DLREC)  
 MACRO CALL, 5-73  
 READ RECORD (\$RDREC)  
 MACRO CALL, 5-260  
 RECORD LOCKS, 5-24  
 RECORD LOCKS CONFLICT, 5-136

RECORD (CONT)

RECORD LOCKS CONTROL, 5-125  
 RECORD LOCKS OPERATION, 5-134  
 REWRITE RECORD (\$RWREC)  
 MACRO CALL, 5-318  
 UNLOCK RECORD LOCKS, 5-24, 5-136  
 WRITE RECORD (\$WRREC)  
 MACRO CALL, 5-397  
 WRITE UPDATED RECORDS, 5-24

REGISTER

MACRO CALLS FUNCTION CODES AND  
 REGISTER CONTENTS (TBL), C-2  
 REGISTER CONTENTS AT TASK  
 ACTIVATION, 1-5  
 REGISTER CONTENTS FOR SYSTEM  
 SERVICE MACRO CALLS, C-1  
 REGISTER CONVENTIONS AND  
 CONTENTS, 1-2  
 RETURN STATUS CODES IN \$R1  
 REGISTER, 1-6

RELEASE

OVERLAY AREA, RELEASE (\$OVRSL)  
 MACRO CALL, 5-228  
 OVERLAY, RELEASE, WAIT, AND RECALL  
 (\$OVRCL) MACRO CALL, 5-240  
 RELEASE DIRECTORY (\$RLDIR)  
 MACRO CALL, 5-266  
 RELEASE FILE (\$RLFIL)  
 MACRO CALL, 5-269  
 RELEASE SEMAPHORE (\$RLSM)  
 MACRO CALL, 5-272  
 RELEASE TERMINAL (\$RLTML)  
 MACRO CALL, 5-274

RENAME FILE/DIRECTORY

RENAME FILE/RENAME DIRECTORY  
 (\$RNFIL) MACRO CALL, 5-279

REPORT ERROR

REPORT ERROR CONDITION (\$RPTER)  
 MACRO CALL, 5-282

REQUEST

ABORT GROUP REQUEST (\$ABGRQ)  
 MACRO CALL, 5-4  
 CANCEL CLOCK REQUEST (\$CNCRQ)  
 MACRO CALL, 5-15  
 CANCEL REQUEST (\$CANRQ)  
 MACRO CALL, 5-17  
 CANCEL SEMAPHORE REQUEST (\$CNSRQ)  
 MACRO CALL, 5-19  
 CLOCK REQUEST BLOCK OFFSETS (\$CRBD)  
 MACRO CALL, 5-31  
 CLOCK REQUEST BLOCK (CRB), 5-288,  
 A-2  
 CLOCK REQUEST BLOCK (\$CRB)  
 MACRO CALL, 5-28  
 FIRST FOUR ITEMS OF REQUEST BLOCK  
 (FIG), A-2  
 INPUT/OUTPUT REQUEST BLOCK, (IORB)  
 6-7, 6-8, A-6, A-7

## REQUEST (CONT)

INPUT/OUTPUT REQUEST BLOCK OFFSETS  
 (\$IORBD) MACRO CALL, 5-168  
 INPUT/OUTPUT REQUEST BLOCK (\$IORB)  
 MACRO CALL, 5-165  
 MESSAGE GROUP REQUEST BLOCKS,  
 A-15  
 MESSAGE GROUP CONTROL REQUEST  
 BLOCK (MGCRB), 5-179, A-15  
 MESSAGE GROUP CONTROL REQUEST  
 BLOCK (\$MGRB) MACRO CALL,  
 5-175  
 MESSAGE GROUP INITIALIZATION  
 REQUEST BLOCK (MGIRB),  
 5-187, A-18  
 MESSAGE GROUP RECOVERY REQUEST  
 BLOCK (MGRRB), 5-199, A-22  
 MESSAGE GROUP RECOVERY REQUEST  
 BLOCK (\$MGRRB) MACRO CALL,  
 5-196  
 MONITOR SERVICE FUNCTIONS, REQUEST  
 AND RETURN, 2-8  
 REQUEST BATCH (\$RQBAT)  
 MACRO CALL, 5-285  
 REQUEST BLOCK OFFSETS, 4-4  
 REQUEST BLOCK TERMINATION STATUS,  
 5-365, 5-390  
 REQUEST BLOCK (RB), 1-17, 4-1  
 REQUEST CLOCK (\$RQCL)  
 MACRO CALL, 5-288  
 REQUEST DEVICE DRIVER, 6-2  
 REQUEST GROUP (\$RQGRP)  
 MACRO CALL, 5-290  
 REQUEST I/O (\$RQIO)  
 MACRO CALL, 5-294  
 REQUEST QUEUE, 1-17  
 REQUEST SEMAPHORE (\$RQSM)  
 MACRO CALL, 5-297  
 REQUEST TASK (\$RQTSK)  
 MACRO CALL, 5-300  
 REQUEST TERMINAL (\$RQTML)  
 MACRO CALL, 5-303  
 RETURN REQUEST BLOCK ADDRESS  
 (\$RBADD) MACRO CALL, 5-316  
 SEMAPHORE REQUEST BLOCK OFFSETS  
 (\$SRBD) MACRO CALL, 5-323  
 SEMAPHORE REQUEST BLOCK (\$SRB)  
 MACRO CALL, 5-321  
 SEMAPHORE REQUEST BLOCK  
 (SRB), 5-322, A-10  
 TASK REQUEST BLOCK, A-11, A-12  
 TASK REQUEST BLOCK OFFSETS (\$TRBD)  
 MACRO CALL, 5-361  
 TASK REQUEST BLOCK (\$TRB)  
 MACRO CALL, 5-357  
 TASK REQUEST BLOCK (TRB)  
 FUNCTIONS, 5-300, 5-360  
 TASK REQUEST QUEUES, 1-16, 1-17  
 TERMINATE REQUEST (\$TRMRQ)  
 MACRO CALL, 5-362  
 WAIT ON REQUEST LIST (\$WAITL)  
 MACRO CALL, 5-390

## RESERVE

LOCATE AND RESERVE FILE, 5-124  
 OVERLAY AREA, RESERVE AND EXECUTE  
 OVERLAY (\$OVRSV) MACRO CALL,  
 5-230  
 RESERVE RESOURCE, 5-297  
 RESERVE SEMAPHORE (\$RSVSM)  
 MACRO CALL, 5-306

## RESET

RESET DEVICE ATTENTION (\$RDVAT)  
 MACRO CALL, 5-309  
 RESET ERROR LOGGING TABLE, 5-97

## RESIDUAL RANGE

RESIDUAL RANGE, 6-13

## RESOURCE

LOGICAL RESOURCE NUMBER (LRN)  
 DEFINITION, 1-16  
 LOGICAL RESOURCE TABLE  
 DEFINITION, 1-16  
 RESERVE RESOURCE, 5-297  
 RESOURCE CONTROL TABLE  
 DEFINITION, 1-17  
 RESOURCE CONTROL TABLE (RCT), 6-12

## RESPONSE, OPERATOR

OPERATOR RESPONSE MESSAGE (\$OPRSP)  
 MACRO CALL, 5-225

## RETURN

MONITOR SERVICE FUNCTIONS, REQUEST  
 AND RETURN, 2-8  
 RETURN COMMAND-IN FILE  
 PATHNAME, 5-355  
 RETURN REQUEST BLOCK ADDRESS  
 (\$RBADD) MACRO CALL, 5-316  
 RETURN STATUS CODES IN \$R1  
 REGISTER, 1-6  
 RETURN STATUS CODES (TBL), 6-5  
 RETURN WORKING DIRECTORY  
 PATHNAME, 5-161  
 RETURN (\$RETRN) MACRO CALL, 5-311  
 STANDARD RETURN SEQUENCE, 5-311

## REWRITE RECORD

REWRITE RECORD (\$RWREC)  
 MACRO CALL, 5-318

## ROUTINES, MACRO

LOCATION OF MACRO ROUTINES, 1-6

## \$RQIO

\$RQIO MACRO CALL FOR I/O  
 FUNCTIONS, 6-14

## RULES FOR TAPE FILE SEARCH

TAPE FILE SEARCH RULES FOR \$OPFIL  
 MACRO CALL (TBL), 5-217

INDEX

SAVE

CONTENTS TRAP SAVE AREA WHEN TRAP OCCURS (TBL), 7-7  
 POINTER TO NEXT TRAP SAVE AREA (NATSAP), 7-5  
 SAVE ERROR LOG VALUES, 5-90  
 TRAP SAVE AREAS, 7-6

SCIENTIFIC BRANCH SIMULATOR

SCIENTIFIC BRANCH SIMULATOR TRAP, 7-14

SEARCH RULES FOR TAPE FILES

TAPE FILE SEARCH RULES FOR \$OPFIL  
 MACRO CALL (TBL), 5-217

SEMAPHORE

CANCEL SEMAPHORE REQUEST (\$CNSRQ) MACRO CALL, 5-19  
 DEFINE SEMAPHORE (\$DFSM) MACRO CALL, 5-67  
 MONITOR SERVICE FUNCTIONS, SEMAPHORE FUNCTIONS, 2-9  
 RELEASE SEMAPHORE (\$RLSM) MACRO CALL, 5-272  
 REQUEST SEMAPHORE (\$RQSM) MACRO CALL, 5-297  
 RESERVE SEMAPHORE (\$RSVSM) MACRO CALL, 5-306  
 SEMAPHORE IDENTIFIER, 5-68  
 SEMAPHORE REQUEST BLOCK OFFSETS (\$SRBD) MACRO CALL, 5-323  
 SEMAPHORE REQUEST BLOCK (\$SRB) MACRO CALL, 5-321  
 SEMAPHORE REQUEST BLOCK (SRB), 5-322, A-10

SEND MESSAGE GROUP

MESSAGE GROUP, SEND (\$MSEND) MACRO CALL, 5-201

SEQUENCE

I/O OPERATION SEQUENCE, 6-14  
 STANDARD RETURN SEQUENCE, 5-311

SET

SET DIAL (\$SDL) MACRO CALL, 5-324  
 SET EXTERNAL SWITCHES (\$SETSW) MACRO CALL, 5-328  
 SET TERMINAL CHARACTERISTICS (\$STTY) MACRO CALL, 5-330

SIMULATOR TRAP

COMMERCIAL SIMULATOR TRAP, 7-12  
 FLOATING-POINT SIMULATOR TRAP, 7-13  
 SCIENTIFIC BRANCH SIMULATOR TRAP, 7-14

SPAWN

SPAWN GROUP (\$SPGRP) MACRO CALL, 5-333

SPAWN (CONT)

SPAWN TASK (\$SPTSK) MACRO CALL, 5-339

SPECIAL

SPECIAL GRAPHIC CHARACTERS, D-2

SPGRP

SPAWN GROUP (\$SPGRP) MACRO CALL, 5-333

SPTSK

SPAWN TASK (\$SPTSK) MACRO CALL, 5-339

START ERROR LOGGING

ERROR LOGGING START (\$ELST) MACRO CALL, 5-96

STATUS

IORB SOFTWARE STATUS WORD (I\_ST) (TBL), 6-11  
 OVERLAY STATUS (\$OVST) MACRO CALL, 5-224  
 RCT ATTENTION STATUS INDICATOR, 5-309  
 REQUEST BLOCK TERMINATION STATUS, 5-365, 5-390  
 RETURN STATUS CODES IN \$RI REGISTER, 1-6  
 RETURN STATUS CODES (TBL), 6-5  
 STATUS MEMORY POOL (\$STMP) MACRO CALL, 5-343  
 TEST COMPLETION STATUS (\$TEST) MACRO CALL, 5-365  
 USER-DRIVER READ/MODIFY STATUS FUNCTION (CODE 4), B-5

STOP

USER DRIVER, STOP I/O FUNCTION (CODE 2), B-4

STORAGE MANAGEMENT

STORAGE MANAGEMENT FUNCTIONS, 3-18

STRUCTURE

DATA STRUCTURE FORMAT, A-1  
 DATA STRUCTURE GENERATION, 4-1  
 DATA STRUCTURES, 6-7  
 FILE SYSTEM DATA STRUCTURES, 4-5  
 INITIALIZE, ALLOCATE GROUP DATA STRUCTURES, 5-64  
 MONITOR SERVICES DATA STRUCTURES, 4-1  
 REMOVE GROUP DATA STRUCTURES, 5-71  
 REMOVE TASK DATA STRUCTURES, 5-76

SUPPRESSION, MESSAGE

CONSOLE MESSAGE SUPPRESSION (\$CMSUP) MACRO CALL, 5-42

INDEX

SUSPEND  
 SUSPEND A TASK, 5-347, 5-350  
 SUSPEND FOR INTERVAL (\$SUSPN)  
 MACRO CALL, 5-347  
 SUSPEND GROUP (\$SUSPG)  
 MACRO CALL, 5-345  
 SUSPEND UNTIL TIME (\$SUSPN)  
 MACRO CALL, 5-350

SUSPENDED GROUP, REACTIVATE  
 REACTIVATE SUSPENDED TASK  
 GROUP, 5-8

SWITCHES  
 CLEAR EXTERNAL SWITCHES (\$CLRSW)  
 MACRO CALL, 5-26  
 READ EXTERNAL SWITCHES (\$RDSW)  
 MACRO CALL, 5-258  
 SET EXTERNAL SWITCHES (\$SETSW)  
 MACRO CALL, 5-328

SYNTAX, MACRO CALL  
 MACRO CALL SYNTAX, 1-1

SYSTEM  
 DRIVER USABLE SYSTEM  
 FUNCTIONS, B-3  
 MONITOR SERVICE FUNCTIONS, STANDARD  
 SYSTEM FILE I/O FUNCTIONS, 2-10  
 REGISTER CONTENTS, SYSTEM SERVICE  
 MACRO CALLS, C-1  
 SYSTEM BUILDING IN WRITING A DRIVER,  
 B-1  
 SYSTEM IDENTIFICATION (\$SYSID)  
 MACRO CALL, 5-353  
 SYSTEM SERVICE MACRO CALLS AND  
 FUNCTION CODES, 1-6, 1-7

TAPE  
 CHARACTERISTICS OF SUPPORTED TAPE  
 DRIVES (TBL), 6-40  
 DEVICE DRIVER, POSITION TAPE MARK  
 FUNCTION (FC 6), 6-6  
 DEVICE DRIVER, WRITE TAPE MARK  
 FUNCTION (FC 3), 6-6  
 MAGNETIC TAPE DRIVER, 6-39  
 MAGNETIC TAPE IORB FIELDS, 6-42  
 MAGNETIC TAPE RCT FIELDS, 6-43  
 MAGNETIC TAPE RCT/IORB  
 HARDWARE/SOFTWARE STATUS  
 MAPPING, 6-43  
 PACKED AND 6-BIT MODES ON 7-TRACK  
 TAPE (FIG), 6-39  
 TAPE FILE SEARCH RULES FOR \$OPFIL  
 MACRO CALL (TBL), 5-217

TASK  
 CREATE TASK (\$CRTSK)  
 MACRO CALL, 5-63  
 DELETE TASK (\$DLTSK)  
 MACRO CALL, 5-76  
 EXECUTE LEAD TASK, 5-292

TASK (CONT)

INITIALIZE, ALLOCATE, TASK DATA  
 STRUCTURES, 5-64  
 MONITOR SERVICE FUNCTIONS, TASK  
 CONTROL, 2-11  
 MONITOR SERVICE FUNCTIONS, TASK  
 GROUP CONTROL, 2-12  
 REACTIVATE SUSPENDED TASK GROUP,  
 5-8  
 REGISTER CONTENTS AT TASK  
 ACTIVATION, 1-5  
 REMOVE TASK DATA STRUCTURES, 5-76  
 REQUEST TASK (\$RQTSK)  
 MACRO CALL, 5-300  
 SPAWN TASK (\$SPTSK)  
 MACRO CALL, 5-339  
 SUSPEND A TASK, 5-347, 5-350  
 TASK COMMUNICATION, 5-360  
 TASK CONTROL BLOCK DEFINITION,  
 1-17  
 TASK GROUP INPUT (\$TGIN)  
 MACRO CALL, 5-355  
 TASK GROUP USER IDENTIFICATION,  
 5-373  
 TASK REQUEST BLOCK (TRB), A-11,  
 A-12  
 TASK REQUEST BLOCK OFFSETS (\$TRBD)  
 MACRO CALL, 5-361  
 TASK REQUEST BLOCK (\$TRB)  
 MACRO CALL, 5-357  
 TASK REQUEST BLOCK (TRB)  
 FUNCTIONS, 5-300, 5-360  
 TASK REQUEST QUEUES, 1-16, 1-17  
 TRAP CONDITIONS DURING TASK  
 EXECUTION, 7-2

TELEPHONE LIST, AUTODIAL  
 AUTODIAL TELEPHONE LIST, 5-324

TERMINAL  
 MONITOR SERVICE FUNCTIONS,  
 SECONDARY USER TERMINAL FUNCTIONS,  
 2-8  
 RELEASE TERMINAL (\$RLTML)  
 MACRO CALL, 5-274  
 REQUEST TERMINAL (\$RQTML)  
 MACRO CALL, 5-303  
 SECONDARY USER TERMINAL,  
 5-274, 5-303  
 SET TERMINAL CHARACTERISTICS  
 (\$STTY) MACRO CALL, 5-330

TERMINATE  
 MESSAGE GROUP, TERMINATE (\$MTMG)  
 MACRO CALL, 5-205  
 TERMINATE FILE PROCESSING, 5-32  
 TERMINATE MESSAGE GROUP, 5-203  
 TERMINATE REQUEST (\$TRMRQ)  
 MACRO CALL, 5-362  
 USER DRIVER TERMINATE (ZXD\_TR)  
 SUBROUTINE, B-6

INDEX

TERMINATION STATUS  
 REQUEST BLOCK TERMINATION STATUS,  
 5-365, 5-390

TEST  
 TEST COMPLETION STATUS (\$TEST)  
 MACRO CALL, 5-365  
 TEST FILE (\$TOFIL)  
 MACRO CALL, 5-367  
 TEST FILE (\$TIFIL)  
 MACRO CALL, 5-367

TIME  
 EXTERNAL TIME, CONVERT TO (\$EXTIM)  
 MACRO CALL, 5-108  
 SUSPEND UNTIL TIME (\$SUSPN)  
 MACRO CALL, 5-350

TRAP  
 COMMERCIAL SIMULATOR TRAP, 7-12  
 CONTENTS TRAP SAVE AREA WHEN TRAP  
 OCCURS (TBL), 7-7  
 DISABLE USER TRAP (\$DSTRP)  
 MACRO CALL, 5-80  
 ENABLE USER TRAP (\$ENTRP)  
 MACRO CALL, 5-86  
 FLOATING POINT SIMULATOR TRAP,  
 7-13  
 HONEYWELL SUPPLIED TRAP HANDLERS,  
 7-12  
 MONITOR SERVICE FUNCTIONS, TRAP  
 HANDLING, 2-13  
 POINTER TO NEXT TRAP SAVE AREA  
 (NATSAP), 7-5  
 PROGRAMMING USER WRITTEN TRAP  
 HANDLERS, 7-16  
 SCIENTIFIC BRANCH SIMULATOR TRAP,  
 7-14  
 SOFTWARE GENERATED TRAP, 7-14  
 TRAP CONDITIONS DURING TASK  
 EXECUTION, 7-2  
 TRAP ENABLED, 7-2  
 TRAP HANDLERS AS MONITOR  
 EXTENSIONS, 7-15  
 TRAP HANDLER CONNECT (\$TRPHD)  
 MACRO CALL, 5-370  
 TRAP HANDLING BY DEBUG PROGRAM,  
 7-12  
 TRAP HANDLING, 7-1  
 TRAP INTERRUPT VECTOR, 7-6  
 TRAP NOT ENABLED, 7-2  
 TRAP SAVE AREAS, 7-6  
 TRAP VECTOR, 7-6  
 USER WRITTEN TRAP HANDLER CONNECT,  
 5-370  
 USER WRITTEN TRAP HANDERS, 7-15

TRB  
 TASK REQUEST BLOCK (TRB)  
 FUNCTIONS, 5-300, 5-360

UNIT  
 BOUND UNIT IDENTIFICATION (\$BUID)  
 MACRO CALL, 5-13

UNLOAD  
 OVERLAY, UNLOAD (\$OVUN)  
 MACRO CALL, 5-247

UNLOCK RECORD  
 UNLOCK RECORD LOCKS, 5-24  
 UNLOCK RECORDS, 5-136

UPDATED RECORDS  
 WRITE UPDATED RECORDS, 5-24

USER DRIVER  
 USER DRIVER, INITIALIZE FUNCTION  
 (CODE 0), B-3  
 USER DRIVER, LOCATE RCT FOR DEVICE  
 (ZXSRTC) SUBROUTINE, B-5  
 USER DRIVER, OUTPUT ADDRESS AND  
 RANGE (ZIOD) SUBROUTINE, B-6  
 USER DRIVER, READ/MODIFY STATUS  
 FUNCTION (CODE 4), B-5  
 USER DRIVER, STOP I/O FUNCTION  
 (CODE 2), B-4  
 USER DRIVER, TERMINATE (ZXD\_TR)  
 SUBROUTINE, B-6  
 USER DRIVER, WAIT FOR INTERRUPT  
 (CODE 3), B-4  
 USER DRIVER, WAIT ON LINE FUNCTION  
 (CODE 1), B-3

USER-IN FILE  
 READ USER-IN FILE, 5-376  
 USER-IN FILE, 5-211

USER-OUT FILE  
 USER-OUT FILE, 5-213  
 WRITE TO USER-OUT FILE, 5-378

VALUES FOR MACRO CALL ARGUMENTS  
 ARGUMENT VALUES FOR \$MGCRB  
 MACRO CALL (TBL), 5-176  
 ARGUMENT VALUES FOR \$MGIRB  
 MACRO CALL (TBL), 5-185  
 ARGUMENT VALUES FOR \$MGRRB  
 MACRO CALL (TBL), 5-197  
 MGCRB ARGUMENT VALUES FOR \$MRECV  
 MACRO CALL (TBL), 5-193  
 MGCRB ARGUMENT VALUES FOR \$MSEND  
 MACRO CALL (TBL), 5-202  
 MGIRB ARGUMENT VALUES FOR \$MACPT  
 MACRO CALL (TBL), 5-173  
 MGIRB ARGUMENT VALUES FOR \$MCMG  
 MACRO CALL (TBL), 5-182  
 MGIRB ARGUMENT VALUES FOR \$MINIT  
 MACRO CALL (TBL), 5-190  
 MGRRB ARGUMENT VALUES FOR \$MTMG  
 MACRO CALL (TBL), 5-207

VECTOR, TRAP  
 TRAP INTERRUPT VECTOR, 7-6  
 TRAP VECTOR, 7-6

VERBATIM CARD MODE  
 CARD VERBATIM MODE, 6-19



INDEX

VIEW, PROGRAM  
PROGRAM VIEW ENTRY IN FIB, 3-6

V-OP  
V-OP OPERATION, 5-272

WAIT  
DEVICE DRIVER, WAIT ONLINE FUNCTION  
(FC 0), 6-4  
FORMAT OF WAIT LIST (FIG), A-14  
OVERLAY RELEASE, WAIT, AND RECALL  
(\$OVRCL) MACRO CALL, 5-240  
PARAMETER BLOCK AND WAIT LIST, 4-3  
USER DRIVER, WAIT FOR INTERRUPT  
(CODE 3), B-4  
USER DRIVER, WAIT ON LINE FUNCTION  
(CODE 1), 3-3  
WAIT BLOCK (\$WTBLK)  
MACRO CALL, 5-383  
WAIT FILE (\$WIFIL)  
MACRO CALL, 5-385  
WAIT FILE (\$WOFIL)  
MACRO CALL, 5-385  
WAIT FOR I/O COMPLETION, 5-385  
WAIT LIST FORMAT, A-14  
WAIT LIST GENERATE (\$WLIST)  
MACRO CALL, 5-388  
WAIT ON REQUEST LIST (\$WAITL)  
MACRO CALL, 5-390  
WAIT (\$WAIT) MACRO CALL, 5-381

WORD, IOFB SOFTWARE STATUS  
IOFB SOFTWARE STATUS WORD (I\_ST)  
(TBL), 6-11

WORKING DIRECTORY  
CHANGE WORKING DIRECTORY (\$CWDIR)  
MACRO CALL, 5-21  
GET WORKING DIRECTORY (\$GWDIR)  
MACRO CALL, 5-161  
RETURN WORKING DIRECTORY  
PATHNAME, 5-161

WRITE  
DEVICE DRIVER, WRITE FUNCTION  
(FC 1), 6-4  
DEVICE DRIVER, WRITE TAPE MARK  
FUNCTION (FC 3), 6-6  
WRITE BLOCK (\$WRBLK)  
MACRO CALL, 5-393  
WRITE RECORD (\$WRREC)  
MACRO CALL, 5-397  
WRITE TO USER-OUT FILE, 5-378  
WRITE UPDATED RECORDS, 5-24

WRITING  
DRIVER INTERFACE, WRITING A  
DRIVER, B-2  
SYSTEM BUILDING, WRITING A  
DRIVER, B-1  
WRITING PERIPHERAL I/O DRIVER,  
B-1

ZIOLD SUBROUTINE  
USER DRIVER OUTPUT ADDRESS AND  
RANGE (ZIOLD) SUBROUTINE, B-6

ZIOSUB SUBROUTINES  
I/O SUBROUTINES (ZIOSUB) FOR  
USER-WRITTEN DRIVERS, B-3

ZXD TR SUBROUTINE  
USER DRIVER, TERMINATE (ZXD\_TR)  
SUBROUTINE, B-6

ZXSRCT SUBROUTINE  
USER DRIVER, LOCATE RCT FOR  
DEVICE (ZXSRCT) SUBROUTINE, B-5



**HONEYWELL INFORMATION SYSTEMS**

Technical Publications Remarks Form

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

**ORDER NO.** CB08, REV. 1


**DATED** JULY 1978

**ERRORS IN PUBLICATION**

[Empty box for reporting errors in publication]

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

[Empty box for providing 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** \_\_\_\_\_  
**TITLE** \_\_\_\_\_  
**COMPANY** \_\_\_\_\_  
**ADDRESS** \_\_\_\_\_  
\_\_\_\_\_

**DATE** \_\_\_\_\_

CUT ALONG LIF.

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

CUT ALONG LINE

FOLD ALONG LINE

FIRST CLASS  
PERMIT NO. 39531  
WALTHAM, MA  
02154

Business Reply Mail  
Postage Stamp Not Necessary if Mailed in the United States

Postage Will Be Paid By:

HONEYWELL INFORMATION SYSTEMS  
200 SMITH STREET  
WALTHAM, MA 02154

ATTENTION: PUBLICATIONS, MS 486

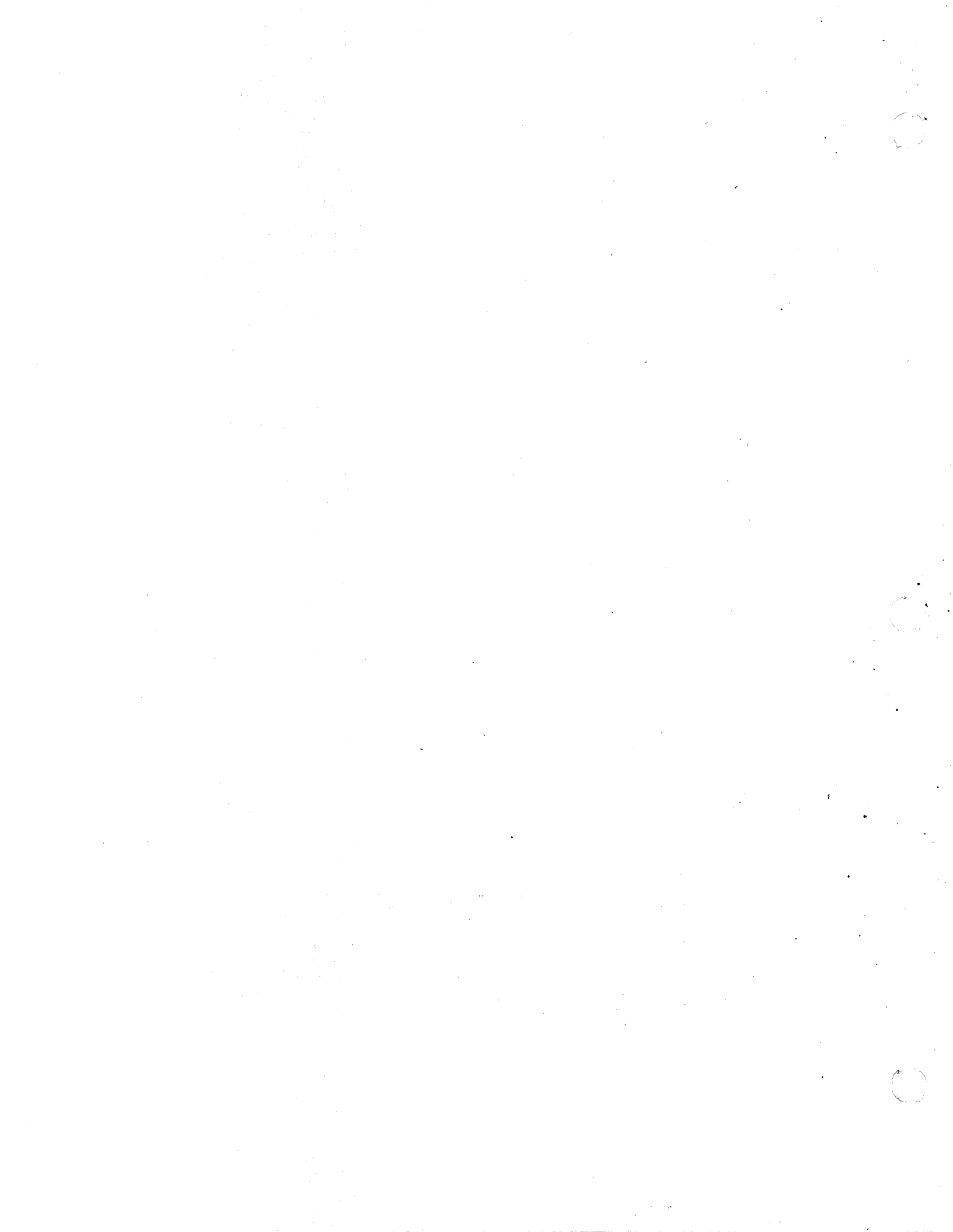
FOLD ALONG LINE

**Honeywell**



Small, faint, illegible text or mark located in the middle left margin of the page.







# Honeywell

## Honeywell Information Systems

In the U.S.A.: 200 Smith Street, MS 486, Waltham, Massachusetts 02154  
In Canada: 2025 Sheppard Avenue East, Willowdale, Ontario M2J 1W5  
In Mexico: Avenida Nuevo Leon 250, Mexico 11, D.F.

21415, 1.2778, Printed in U.S.A.

CB08, Rev. 1