

November 1978

RSX-11M System Logic Manual

Order No. AA-5579A-TC

VOLUME 2

RSX-11M V3.1

To order additional copies of this document, contact the Accessories and Supplies Group, Product Line 86, Digital Equipment Corporation, Cotton Road, Nashua, New Hampshire 03060.

digital equipment corporation · maynard. massachusetts

First Printing, November 1978

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

Copyright © 1978 by Digital Equipment Corporation

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10
VAX	VMS	SBI
DECnet	IAS	

CONTENTS

VOLUME II

		Page
APPENDIX A	RSX-11M SUPPORTED DEVICES	A-1
A.1	RSX-11M DEVICE SUPPORT	A-1
A.1.1	Processors And Options	A-1
A.1.2	Card Readers	A-2
A.1.3	Communications	A-2
A.1.4	Data Acquisition	A-2
A.1.5	Disk Devices	A-3
A.1.6	Laboratory/industrial Control	A-3
A.1.7	Printers	A-4
A.1.8	Tape Devices, Magnetic	A-4
A.1.9	Tape Devices, Paper	A-5
A.1.10	Terminals	A-5
APPENDIX B	CODING STANDARDS AND CONVENTIONS	B-1
B.1	CODING STANDARD INTRODUCTION	B-1
B.1.1	Line Format	B-1
B.1.2	Comments	B-1
B.1.3	Naming Standards	B-2
B.1.4	Symbols	B-3
B.1.5	Program Modules	B-5
B.1.6	Formatting Standards	B-9
B.1.7	Program Source Files	B-11
B.1.8	Forbidden Instruction Usage	B-11
B.1.9	Recommended Coding Practice	B-12
B.1.10	PDP-11 Version Number Standard	B-12
B.1.11	Co-routines	B-14
APPENDIX C	MACRO EXPANSIONS	C-1
C.1	COMMAND STRING INTERPRETER MACRO EXPANSIONS	C-1
C.1.1	CSI\$1 Macro	C-1
C.1.2	CSI\$2 Macro	C-1
C.1.3	CSI\$SW Macro	C-2
C.1.4	CSI\$ND Macro	C-3
C.1.5	CSI\$SV Macro	C-3
C.1.6	LDR0\$ Macro	C-4
C.1.7	CSI\$ Macro	C-4
C.2	DIRECTIVE MACRO EXPANSIONS	C-5
C.2.1	ABRT\$C Macro	C-5
C.2.2	ABRT\$\$ Macro	C-6
C.2.3	ABRT\$ Macro	C-6
C.2.4	ALTP\$C Macro	C-6
C.2.5	ALTP\$\$ Macro	C-7
C.2.6	ALTP\$ Macro	C-7
C.2.7	ALUN\$C Macro	C-8

CONTENTS (Cont.)

		Page
C.2.8	ALUN\$\$ Macro	C-8
C.2.9	ALUN\$ Macro	C-9
C.2.10	ASTX\$C Macro	C-10
C.2.11	ASTX\$\$ Macro	C-10
C.2.12	ASTX\$ Macro	C-10
C.2.13	CINT\$C Macro	C-11
C.2.14	CINT\$\$ Macro	C-11
C.2.15	CINT\$ Macro	C-12
C.2.16	CLEF\$C Macro	C-13
C.2.17	CLEF\$\$ Macro	C-13
C.2.18	CLEF\$ Macro	C-14
C.2.19	CMKT\$C Macro	C-14
C.2.20	CMKT\$\$ Macro	C-15
C.2.21	CMKT\$ Macro	C-15
C.2.22	CSRQ\$C Macro	C-15
C.2.23	CSRQ\$\$ Macro	C-16
C.2.24	CSRQ\$ Macro	C-16
C.2.25	DECL\$C Macro	C-17
C.2.26	DECL\$\$ Macro	C-17
C.2.27	DECL\$ Macro	C-18
C.2.28	DIR\$ Macro	C-18
C.2.29	DSAR\$C Macro	C-19
C.2.30	DSAR\$\$ Macro	C-19
C.2.31	DSAR\$ Macro	C-19
C.2.32	DSCP\$C Macro	C-20
C.2.33	DSCP\$\$ Macro	C-20
C.2.34	DSCP\$ Macro	C-20
C.2.35	ENAR\$C Macro	C-21
C.2.36	ENAR\$\$ Macro	C-21
C.2.37	ENAR\$ Macro	C-21
C.2.38	ENCP\$C Macro	C-22
C.2.39	ENCP\$\$ Macro	C-22
C.2.40	ENCP\$ Macro	C-22
C.2.41	ERR\$ Macro	C-23
C.2.42	EXIF\$C Macro	C-23
C.2.43	EXIF\$\$ Macro	C-24
C.2.44	EXIF\$ Macro	C-24
C.2.45	EXIT\$C Macro	C-24
C.2.46	EXIT\$\$ Macro	C-25
C.2.47	EXIT\$ Macro	C-25
C.2.48	EXTK\$C Macro	C-25
C.2.49	EXTK\$\$ Macro	C-26
C.2.50	EXTK\$ Macro	C-26
C.2.51	GLUN\$C Macro	C-27
C.2.52	GLUN\$\$ Macro	C-27
C.2.53	GLUN\$ Macro	C-28
C.2.54	GMCR\$C Macro	C-29
C.2.55	GMCR\$ Macro	C-29
C.2.56	GPRT\$C Macro	C-30
C.2.57	GPRT\$\$ Macro	C-30
C.2.58	GPRT\$ Macro	C-31
C.2.59	GSSW\$C Macro	C-32
C.2.60	GSSW\$\$ Macro	C-32
C.2.61	GSSW\$ Macro	C-32
C.2.62	GTIM\$C Macro	C-33

CONTENTS (Cont.)

	Page	
C.5.63	OFNB\$W Macro	C-97
C.5.64	PUT\$ Macro	C-97
C.5.65	PUT\$R Macro	C-98
C.5.66	PUT\$\$ Macro	C-98
C.5.67	RAD50\$ Macro	C-98
C.5.68	READ\$ Macro	C-99
C.5.69	TRUNC\$ Macro	C-99
C.5.70	WAIT\$ Macro	C-99
C.5.71	WRITE\$ Macro	C-99
C.6	NETWORK SYMBOL DEFINITION MACRO	C-100
C.6.1	COMDF\$ Macro	C-100
C.7	PROGRAM LOGICAL ADDRESS SPACE EXTENSION MACRO EXPANSIONS	C-101
C.7.1	ATRG\$, ATRG\$C, and Atrg\$\$ Macros	C-101
C.7.2	.BLK., .BLKB., and .BLKW. Macros	C-102
C.7.3	CRAW\$, CRAW\$C, CRAW\$\$ Macro	C-102
C.7.4	CRRG\$, CRRG\$C, and CRRG\$\$ Macros	C-103
C.7.5	DTRG\$, DTRG\$C, and DTRG\$\$ Macros	C-104
C.7.6	ELAW\$, ELAW\$C, and ELAW\$\$ Macros	C-104
C.7.7	GMCX\$, GMCX\$C, and GMCX\$\$ Macros	C-105
C.7.8	GREG\$, GREG\$C, and GREG\$\$ Macros	C-105
C.7.9	MAP\$, MAP\$C, MAP\$\$ Macros	C-106
C.7.10	RDBBK\$ Macro	C-107
C.7.11	RREF\$, RREF\$C, and RREF\$\$ Macros	C-107
C.7.12	SREF\$, SREF\$C, and SREF\$\$ Macros	C-108
C.7.13	SRRA\$, SRRA\$C, and SRRA\$\$ Macros	C-109
C.7.14	UMAP\$, UMAP\$C, and UMAP\$\$ Macros	C-109
C.7.15	WDBBK\$ Macro	C-110
C.8	RELATIVE FILES MACROS - EXPANSIONS	C-110
C.8.1	RCLOS\$ Macro	C-110
C.8.2	RFDBT\$ Macro	C-110
C.8.3	RFIND\$ Macro	C-111
C.8.4	RFOF\$L Macro	C-111
C.8.5	RFOFF\$ Macro	C-111
C.8.6	RGET\$ Macro	C-112
C.8.7	ROPN\$ Macro	C-112
C.8.8	ROPN\$A Macro	C-112
C.8.9	ROPN\$M Macro	C-112
C.8.10	ROPN\$R Macro	C-113
C.8.11	ROPN\$U Macro	C-113
C.8.12	ROPN\$W Macro	C-113
C.8.13	ROPS\$A Macro	C-113
C.8.14	ROPS\$M Macro	C-113
C.8.15	ROPS\$R Macro	C-113
C.8.16	ROPS\$U Macro	C-114
C.8.17	ROPS\$W Macro	C-114
C.8.18	RPORT\$ Macro	C-114
C.8.19	RPRTC\$ Macro	C-114
C.8.20	RPUT\$ Macro	C-114
C.9	QIOMAC - QIOSYM MACRO DEFINITIONS	C-115
C.9.1	DRERR\$ Macro	C-115
C.9.2	FILIO\$ Macro	C-116
C.9.3	.IOER. Macro	C-117
C.9.4	IOERR\$ Macro	C-117
C.9.5	.QIOE. Macro	C-120

CONTENTS (Cont.)

	Page	
C.2.118	SPRA\$C Macro	C-61
C.2.119	SPRA\$\$ Macro	C-61
C.2.120	SPRA\$ Macro	C-62
C.2.121	SRDA\$C Macro	C-62
C.2.122	SRDA\$\$ Macro	C-63
C.2.123	SRDA\$ Macro	C-63
C.2.124	SVDB\$C Macro	C-64
C.2.125	SVDB\$\$ Macro	C-64
C.2.126	SVDB\$ Macro	C-65
C.2.127	SVTK\$C Macro	C-65
C.2.128	SVTK\$\$ Macro	C-66
C.2.129	SVTK\$ Macro	C-66
C.2.130	WSIG\$C Macro	C-67
C.2.131	WSIG\$\$ Macro	C-67
C.2.132	WSIG\$ Macro	C-67
C.2.133	WTLO\$C Macro	C-68
C.2.134	WTLO\$\$ Macro	C-68
C.2.135	WTLO\$ Macro	C-69
C.2.136	WTSE\$C Macro	C-69
C.2.137	WTSE\$\$ Macro	C-70
C.2.138	WTSE\$ Macro	C-70
C.3	EXECUTIVE MACRO EXPANSIONS	C-71
C.3.1	CALL Macro	C-71
C.3.2	CALLR Macro	C-71
C.3.3	CRASH Macro	C-72
C.3.4	DIRSV\$ Macro	C-72
C.3.5	DRSTS Macro	C-72
C.3.6	GTUCB\$ Macro	C-72
C.3.7	INTLB Macro	C-72
C.3.8	INTSE\$ Macro	C-73
C.3.9	INTSV\$ Macro	C-73
C.3.10	MFPS/MTPS Macros	C-74
C.3.11	RETURN Macro	C-74
C.3.12	SAVNR Macro	C-74
C.3.13	SCBLB Macro	C-74
C.3.14	SETD Macro	C-75
C.3.15	SOB Macro	C-75
C.3.16	STD Macro	C-75
C.3.17	STFPS Macro	C-75
C.3.18	STST Macro	C-75
C.3.19	SWSTK\$ Macro	C-76
C.3.20	LDD Macro	C-76
C.3.21	LDFPS Macro	C-76
C.4	FILES-11 HEADER OFFSETS MACRO DEFINITIONS	C-76
C.4.1	FHDO1\$ Macro	C-76
C.4.2	FHDOF\$ Macro	C-77
C.4.3	HMBOF\$ And HMBO1\$ Macros	C-78
C.5	FILE CONTROL SERVICES MACRO EXPANSIONS	C-79
C.5.1	BDOFF\$ Macro	C-79
C.5.2	CBYTE\$ Macro	C-80
C.5.3	CLOSE\$ Macro	C-80
C.5.4	CMOV\$2 Macro	C-80
C.5.5	CMOV\$B Macro	C-80
C.5.6	CMOV\$W Macro	C-81
C.5.7	CWORD\$ Macro	C-82

CONTENTS (Cont.)

		Page
C.9.6	QIOSY\$ Macro	C-120
C.9.7	SPCIO\$ Macro	C-121
C.9.8	UMDIO\$ Macro	C-123
C.9.9	.WORD. Macro	C-124
C.10	SNAP CONTROL BLOCK AND SNAPSHOT DUMP MACROS	C-124
C.10.1	SNAP\$ Macro	C-124
C.10.2	SNPBK\$ Macro	C-124
C.10.3	SNPDF\$ Macro	C-125
C.11	STATE AND KEYWORD TABLE GENERATION MACROS	C-125
C.11.1	ISTAT\$ Macro	C-125
C.11.2	MTRAN\$ Macro	C-126
C.11.3	STATE\$ Macro	C-126
C.11.4	TRAN\$ Macro	C-127
C.12	SUBMIT FILE TO PRINT SPOOLER (PRT...) MACRO (PRINT\$)	C-128
C.12.1	PRINT\$ Macro	C-128
C.13	SET/GET SYMBOL (TTSYM\$) MACRO EXPANSIONS	C-129
C.13.1	TTSYM\$ Macro	C-129
 APPENDIX D	 LISTING OF CONDITIONAL ASSEMBLY PARAMETERS	 D-1
D.1	LISTING OF CONDITIONAL ASSEMBLY PARAMETERS	D-1
 APPENDIX E	 GENERAL FAULT ISOLATION	 E-1
E.1	INTRODUCTION	E-1
E.2	FAULT CLASSIFICATIONS	E-1
E.3	SERVICING FAULTS	E-1
E.3.1	Gathering Pertinent Fault Isolation Data	E-3
E.3.2	Tracing Faults	E-4
 APPENDIX F	 SYSTEM TUNING	 F-1
F.1	HARDWARE CONSIDERATIONS	F-1
F.2	MEMORY LAYOUT	F-1
F.3	EXECUTIVE SOFTWARE OPTIONS	F-2
F.4	FILE SYSTEM OPTIONS	F-3
F.5	HELPFUL HINTS	F-4
F.6	SOME USEFUL COMMANDS	F-5
F.7	A USEFUL TOOL	F-8
 INDEX		 Index-1

FIGURES

FIGURE	1-1	Sample Unmapped 16K System Memory Layout	1-8
	1-2	Example of a Mapped 124K RSX-11M System	1-10
	2-1	Memory Management - Virtual to Logical Address Space Relationship	2-5
	2-2	Routines That Call \$NXTSK	2-19
	2-3	\$ALCLK Logical Flow Diagram	2-25
	2-4	\$ALOCB Logical Flow Diagram	2-26

CONTENTS (Cont.)

		Page
C.2.63	GTIM\$\$ Macro	C-33
C.2.64	GTIM\$ Macro	C-34
C.2.65	GTSK\$C Macro	C-34
C.2.66	GTSK\$\$ Macro	C-35
C.2.67	GTSK\$ Macro	C-36
C.2.68	IHAR\$C Macro	C-37
C.2.69	IHAR\$\$ Macro	C-37
C.2.70	IHAR\$ Macro	C-37
C.2.71	MOV\$ Macro	C-38
C.2.72	MRKT\$C Macro	C-38
C.2.73	MRKT\$\$ Macro	C-39
C.2.74	MRKT\$ Macro	C-39
C.2.75	MVB\$ Macro	C-40
C.2.76	OFF\$ Macro	C-40
C.2.77	QDPB\$\$ Macro	C-41
C.2.78	QDPB\$ Macro	C-41
C.2.79	QIO\$C Macro	C-42
C.2.80	QIO\$\$ Macro	C-42
C.2.81	QIO\$ Macro	C-43
C.2.82	QIOW\$C Macro	C-43
C.2.83	QIOW\$\$ Macro	C-44
C.2.84	QIOW\$ Macro	C-44
C.2.85	R50\$ Macro	C-45
C.2.86	RCVD\$C Macro	C-45
C.2.87	RCVD\$\$ Macro	C-46
C.2.88	RCVD\$ Macro	C-46
C.2.89	RCVX\$C Macro	C-47
C.2.90	RCVX\$\$ Macro	C-47
C.2.91	RCVX\$ Macro	C-48
C.2.92	RDAF\$C Macro	C-48
C.2.93	RDAF\$\$ Macro	C-49
C.2.94	RDAF\$ Macro	C-49
C.2.95	RFA\$ Macro	C-50
C.2.96	RQST\$C Macro	C-50
C.2.97	RQST\$\$ Macro	C-51
C.2.98	RQST\$ Macro	C-51
C.2.99	RSUM\$C Macro	C-52
C.2.100	RSUM\$\$ Macro	C-52
C.2.101	RSUM\$ Macro	C-53
C.2.102	RUN\$C Macro	C-53
C.2.103	RUN\$\$ Macro	C-54
C.2.104	RUN\$ Macro	C-54
C.2.105	RVP\$ Macro	C-55
C.2.106	SDAT\$C Macro	C-56
C.2.107	SDAT\$\$ Macro	C-56
C.2.108	SDAT\$ Macro	C-57
C.2.109	SETF\$C Macro	C-57
C.2.110	SETF\$\$ Macro	C-58
C.2.111	SETF\$ Macro	C-58
C.2.112	SFPA\$C Macro	C-59
C.2.113	SFPA\$\$ Macro	C-59
C.2.114	SFPA\$ Macro	C-59
C.2.115	SPND\$C Macro	C-60
C.2.116	SPND\$\$ Macro	C-60
C.2.117	SPND\$ Macro	C-61

CONTENTS (Cont.)

		Page
C.5.8	DEF\$G Macro	C-81
C.5.9	DEF\$I Macro	C-81
C.5.10	DEF\$L Macro	C-81
C.5.11	DEF\$N Macro	C-82
C.5.12	DEFIN\$ Macro	C-82
C.5.13	DELET\$ Macro	C-82
C.5.14	FCSBT\$ Macro	C-82
C.5.15	FCSMC\$ Macro	C-84
C.5.16	FDAT\$A Macro	C-85
C.5.17	FDAT\$R Macro	C-85
C.5.18	FDBDF\$ Macro	C-85
C.5.19	FDBF\$A Macro	C-85
C.5.20	FDBF\$R Macro	C-86
C.5.21	FDBK\$A Macro	C-86
C.5.22	FDBK\$R Macro	C-68
C.5.23	FDBSZ\$ Macro	C-86
C.5.24	FDOP\$A Macro	C-87
C.5.25	FDOP\$R Macro	C-87
C.5.26	FDRC\$A Macro	C-87
C.5.27	FDRC\$R Macro	C-87
C.5.28	FDOF\$L Macro	C-88
C.5.29	FDOFF\$ Macro	C-88
C.5.30	FDSOF\$ Macro	C-89
C.5.31	FINIT\$ and FSRSZ\$ Macros	C-90
C.5.32	FSROF\$ Macro	C-90
C.5.33	GET\$ Macro	C-91
C.5.34	GET\$R Macro	C-91
C.5.35	GET\$\$ Macro	C-91
C.5.36	NBOF\$L Macro	C-91
C.5.37	NBOFF\$ Macro	C-92
C.5.38	NMBLK\$ Macro	C-92
C.5.39	OPEN\$ Macro	C-93
C.5.40	OPEN\$A Macro	C-93
C.5.41	OPEN\$M Macro	C-93
C.5.42	OPEN\$R Macro	C-93
C.5.43	OPEN\$U Macro	C-94
C.5.44	OPEN\$W Macro	C-94
C.5.45	OPNS\$A Macro	C-94
C.5.46	OPNS\$M Macro	C-94
C.5.47	OPNS\$R Macro	C-94
C.5.48	OPNS\$U Macro	C-94
C.5.49	OPNS\$W Macro	C-95
C.5.50	OPNT\$D Macro	C-95
C.5.51	OPNT\$W Macro	C-95
C.5.52	OFID\$ Macro	C-95
C.5.53	OFID\$A Macro	C-95
C.5.54	OFID\$M Macro	C-96
C.5.55	OFID\$R Macro	C-96
C.5.56	OFID\$U Macro	C-96
C.5.57	OFID\$W Macro	C-96
C.5.58	OFNB\$ Macro	C-96
C.5.59	OFNB\$A Macro	C-97
C.5.60	OFNB\$M Macro	C-97
C.5.61	OFNB\$R Macro	C-97
C.5.62	OFNB\$U Macro	C-97

CONTENTS (Cont.)

	Page	
2-5	\$CHKPT Logical Flow Diagram	2-28
2-6	\$DECLK-\$DEPKT-\$DEACB Logical Flow Diagram	2-30
2-7	\$FN DSP Logical Flow Diagram	2-35
2-8	\$ICHKP Logical Flow Diagram	2-36
2-9	\$NXTSK Logical Flow Diagram	2-37
2-10	\$TSTCP Logical Flow Diagram	2-42
2-11	Loader Logical Flow Diagram	2-43
2-12	Shuffler Logical Flow Diagram	2-51
2-13	Partition Control Block	2-63
2-14	Task Control Block	2-66
3-1	INTSV\$ Macro Expansion	3-6
3-2	Example of a Driver Using \$INTSV	3-6
3-3	Example of Use of \$DIRSV by the \$EMTRP Routine	3-8
3-4	Stack State Upon Entry into Directive Processing	3-10
3-5	Example Driver Interrupt Routine	3-14
3-6	Interrupt Flow of Control	3-18
4-1	User Task in Unmapped System	4-11
4-2	4K Nonprivileged User Task Mapping in a PDP-11/70	4-12
4-3	8K Nonprivileged User Task Mapping in a PDP-11/70	4-13
4-4	8K Nonprivileged Task Mapping in a PDP-11/70 Using PLAS Directives	4-14
4-5	Privileged Task Mapping	4-15
5-1	MCR Tree Structure	5-2
5-2	Input Buffer	5-6
5-3	Function Table Entry	5-7
5-4	Parser Table Entry	5-8
6-1	Queue Directive Parameter Block	6-3
6-2	QIO Directive Processing	6-6
8-1	Linked Lists on RSX-11M	8-2
8-2	Overview of RSX-11M System Control Blocks	8-3
8-3	Example of PCB Listings	8-8
8-4	Example of a Partition Wait Queue	8-9
8-5	Example of a PCB List for Checkpoint Files	8-10
8-6	Example of a System Task Directory (STD) and Active Task List	8-11
8-7	Simplified User-Controlled Partition TCB, Task Header, and PCB Relationship	8-12
8-8	TCB, Task Header, and PCB Relationships in a System-Controlled Partition	8-13
8-9	Example of an AST Queue	8-14
8-10	The Loader Queue	8-15
8-11	Send/Receive Data Queue	8-15
8-12	Send/Receive by Reference Queue	8-16
8-13	The Clock Queue	8-16
8-14	The Fork Queue	8-17
8-15	Example of DCB, SCB, UCB, LCB Relationship	8-18
8-16	Logical Assignment Control Block (LCB) List	8-19
8-17	MCR Queues	8-20
8-18	Pre-allocated I/O Packet Queue	8-21
8-19	Task Termination Notification (TKTN) Queues	8-22

CONTENTS (Cont.)

		Page
8-20	Dynamic Storage Region Free Block Queue	8-23
8-21	DH11 Terminal I/O Data Structure	8-24
8-22	RK11 Disk I/O Data Structure	8-25
8-23	I/O Data Structure for Two RK11 Disk Controllers	8-25 8-27
8-24	I/O Data Structure	
B-1	Difference Among Global and Local Symbols	B-3
E-1	Task Header on an Unmapped System	E-5
E-2	Task Header on a Mapped System	E-5
E-3	Stack Structure: Internal SST Fault	E-6
E-4	Stack Structure: Abnormal SST Fault	E-7
E-5	Stack Structure: Data Items on Stack	E-8

APPENDIX A

RSX-11M SUPPORTED DEVICES

A.1 RSX-11M DEVICE SUPPORT

RSX-11M is a disk based real-time operating system that runs on any UNIBUS PDP-11 processor. During system generation, the user can configure RSX-11M for systems ranging in size from 16K- to 1920K-words.

The user can generate RSX-11M as either a mapped or unmapped system. A mapped system must include the KT11 Memory Management Unit. Without the KT11 Memory Management Unit, RSX-11M supports between 16K- and 28K-words of memory. With the KT11, RSX-11M supports between 24K- and 124K-words of memory on processors other than the PDP-11/70. RSX-11M provides the same primary services for both mapped and unmapped systems; however, some supplied optional features and separately orderable options require hardware configurations larger than the minimum supported system.

For complete and detailed information about minimum hardware required for the different kinds of RSX-11M distribution kits, various options, and combinations of peripherals, see the DIGITAL Software Product Description for RSX11-M, Version 3.1, Real Time Operating System and the RSX-11M System Generation Manual. The RSX-11M Operator's Procedures Manual contains a table of RSX-11M peripheral, pseudo, and null devices along with their respective identifiers (device-unit names recognized by RSX-11M).

RSX-11M supports the following:

A.1.1 Processors And Options

- Any PDP-11 processor except the LSI-11 and PDP-11/03
 - A minimum of 16K-words of memory (unmapped system)
 - Between 24K and 124K-words of memory on a system other than the PDP-11/70 (mapped system with the KT11 Memory Management Unit); and between 64K and 1920K-words on the PDP-11/70.
 - KT11 Memory Management Unit (requires a minimum of 24K-words of memory)
 - Kell-A,B Extended Arithmetic Element (on systems without a memory management unit)
 - Kell-E Extended Instruction Set
 - Kell-F Floating Instruction Set

RSX-11M SUPPORTED DEVICES

- FP11 Floating Point Processor
- KW11-Y Watch-dog Timer Clock

A.1.2 Card Readers

- CR11 card reader
- CM11 card reader

A.1.3 Communications

- DL11-E single line interface
- DP11 synchronous line interface
- DU11 synchronous line interface
- DUP11 synchronous line interface
- DQ11 DMA synchronous line interface
- DA11-B DMA UNIBUS link
- DMC11 interprocessor link

A.1.4 Data Acquisition

One or more of the following subsystems:

- LPS11 Laboratory Peripheral System (requires LPS11-S, LPSAD-12, and LPSKW)
- AR11 Analog Real-time System
- LPA11-K Laboratory Peripheral Accelerator
- Laboratory I/O Subsystem configured using the following options:
 1. ADK11-KT 12-bit A/D converter with 16-channel multiplexer and dual clock; one per subsystem)
 2. AD11-K 12-bit A/D converter with 16-channel multiplexer; 16 per subsystem (15 if ADK11-KT is part of same subsystem)
 3. KW11-K Dual real-time clock with Schmitt triggers; 1 per subsystem (clock already included in ADK11-KT, not KW11-K required if one is present)
 4. AM11-K 48-channel A/D multiplexer with gain ranging; one per AD11-K or ADK11-KT
 5. DR11-K 16-bit digital I/O option; 16 per subsystem
 6. AA11-K 4-channel 12-bit D/A converter with scope control, 16 per subsystem

RSX-11M SUPPORTED DEVICES

A.1.5 Disk Devices

- RX11 floppy disk system
- RF11 fixed-head disk system
- RK11 disk cartridge controller with RK05J or RK05F disk drives
- RK611 disk cartridge controller with RK06 or RK07 disk drives
- RPR02 disk pack drives (with appropriate controller)
- RP03 disk pack drives (with appropriate controller)
- RP04, RP05, or RP06 disk pack drives (with appropriate controller)
- RS03 or RS04 fixed-head disks (with appropriate controller)
- RM03 drives (with appropriate controller)
- RL11 disk cartridge controller with RL01 disk drives
- RM02 drives (with appropriate controller)

A.1.6 Laboratory/industrial Control

- AD01-D A/D converter
- AFC11 A/D converter
- DRS/DSS11 industrial control system modules
- UDC11 Universal Digital Controller
- IDA11-AA Contact sense module
- IDA11-AB Contact sense module
- IDA11-BA Contact interrupt module
- IDA11-BB Contact interrupt module
- IDA11-CA I/O converter
- IDA11-DA Solid state AC/DC driver
- IDA11-EA Flip-flop DC driver
- IDA11-FA Single shot driver
- IDA11-GA Latching output relay
- IDA11-HA Flip-flop output relay
- IDA11-JA Single shot output relay
- IAA11-AA Multi-range A/D converter
- IAA11-BA D/A converter
- IAA11-BB D/A converter

RSX-11M SUPPORTED DEVICES

- IAAll-BC D/A converter
- IAAll-BD D/A converter
- ICS11/ICR11 Industrial Control subsystem
- IDC-IA Isolated DC sense
- IDC-IB Isolated DC interrupt
- IDC-IC I/O counter
- IDC-ID Non-isolated DC sense
- IDC-IE Non-isolated DC interrupt
- IAC-IA Isolated AC sense
- IAC-IB Isolated AC interrupt
- IDC-OA DC flip-flop driver
- IDC-OB DC single shot driver
- IAC-OA AC flip-flop driver
- IAC-OB AC single shot driver
- IRL-OA Latching output relay
- IRL-OB Flip-flop output relay
- IDA-OB 4-channel D/A converter
- IAD-IA 8-channel A/D converter
- IMX-IA 16-channel multiplexer for IAD-IA

A.1.7 Printers

- LA35 line printer
- LA180 line printer
- LS11 line printer
- LP11 line printer
- LV11 line printer (no plotter support)

A.1.8 Tape Devices, Magnetic

- TA11 Dual drive cassette system
- TC11 DEctape controller and dual transport
- TS03 magnetic tape transport (with appropriate controller)
- TU10 magnetic tape transport (with appropriate controller)

RSX-11M SUPPORTED DEVICES

- TE10 magnetic tape transport (with appropriate controller)
- TU16 magnetic tape transport (with appropriate controller)
- TE16 magnetic tape transport (with appropriate controller)
- TU45 magnetic tape transport (with appropriate controller)

A.1.9 Tape Devices, Paper

- PC11 paper tape reader/punch
- PR11 paper tape reader

A.1.10 Terminals

- LA30 terminal
- LA36 terminal
- LA180S terminal on the following line devices only:
 - DH11 serial line terminal multiplexer (optional with DM11-BB modem control)
 - DZ11 serial line terminal multiplexer
 - DJ11 serial line terminal multiplexer
 - DL11-A,B,C,D,W single line terminal interface
- LT33 terminal
- LT35 terminal
- VT05 terminal
- VT50 terminal
- VT52 terminal
- VT55 terminal

APPENDIX B
CODING STANDARDS AND CONVENTIONS

B.1 CODING STANDARD INTRODUCTION

This Appendix contains DIGITAL's PDP-11 Program Coding Standard. We suggest that you use this standard for your own installation when writing code, such as I/O drivers, for use with the PDP-11.

B.1.1 Line Format

All source lines consist of from one to a maximum of eighty characters (not including the audit trail added by the SLIPR (SLP in RSX-11M) editor). This program is described in the RSX-11M utilities manual.

Assembly language code lines have the following format:

1. Label field - if present, the label starts at tab stop 0 (column 1).
2. Operation field - the operation field starts at tab stop 1 (column 9).
3. Operand field - the operand field starts at tab stop 2 (column 17).
4. Comments field - the comments field starts at tab stop 4 (column 33) and may continue to column 80.

Include comment lines in the code body by delimiting them with a line that contains only a semicolon. The comment lines contain a leading semicolon with the comment itself starting in column 3.

If the operand field extends beyond tab stop 4 (column 33), leave a space and start the comment. Comments that apply to an instruction but require continuation should always line up with the character that started the comment.

B.1.2 Comments

Comment all coding to convey the global role of an instruction, rather than a simple literal translation of the instruction into English. In general, this consists of a comment for each line of code. If a particularly elegant instruction sequence is used, precede that section of code with a paragraph of comments.

Delimit preface text, which describes formats, algorithms, program-local variables, etc., by the character sequence ;+ at the

CODING STANDARDS AND CONVENTIONS

start of the text and ; - at the end. These delimiters ease the extraction of the text by a program which could be designed for the purpose. The comments themselves start in column 3.

For example:

```
    ;+
    ; THE INVERT ROUTINE ACCEPTS
    ; A LIST OF RANDOM NUMBERS AND
    ; APPLIES THE KOLMOGOROV ALGORITHM
    ; TO THEM
    ;-
```

B.1.3 Naming Standards

B.1.3.1 Register Standards -

B.1.3.1.1 General Purpose Registers - The following names are the only ones permitted for register names and may not be used for other purposes:

R0=0	;REG 0
R1=1	;REG 1
R2=2	;REG 2
R3=3	;REG 3
R4=4	;REG 4
R5=5	;REG 5
SP=6	;STACK POINTER (REG 6)
PC=7	;PROGRAM COUNTER (REG 7)

B.1.3.1.2 Hardware Registers - Name hardware registers identically to the hardware definition. For example, PS and SWR.

B.1.3.1.3 Device Registers - Device registers are symbolically and identically named to the hardware notation. For example, the control status register for the RK disk is RKCS. You may use only this symbolic name to refer to this register.

B.1.3.2 Processor Priority - Use the following bits to test or alter processor priority:

PR0, PR1, PR2,PR7

The system equates these bits to their corresponding bit pattern.

B.1.3.3 Other Symbols - Make frequently-used bit patterns, such as CR and LF, conventional symbols on an as-needed basis.

CODING STANDARDS AND CONVENTIONS

B.1.3.4 Using the Standard Symbols - Register standards are defined within the assembler. All other standard symbols appear in a file and are linked prior to program execution.

B.1.4 Symbols

B.1.4.1 Global Symbols - You can easily recognize global symbols by their format. The following standards apply and completely define symbol standards for PDP-11 Medium/Large software products.

Figure B-1 describes the difference among global and local symbols.

Symbol	Character Positions						Length
	1	2	3	4	5	6	
Non global symbol	L, AN	AN or null	AN or null	AN or null	AN or null	AN or null	>=1
Global symbol	\$ or .	AN or null	AN or null	AN or null	AN or null	AN or null	>=1
Global offset	L	\$ or .	AN	AN or null	AN or null	AN or null	>=3
Global bit pattern	L	AN	\$ or .	AN	AN or null	AN or null	>=4
Local symbol	N *	\$					>=2

where:

- L is a letter
- AN is an alphanumeric character
- null is no character
- \$ is a dollar sign (reserved for DEC-supplied software)
- . is a period (reserved for DEC-supplied software)
- N is a number in the range of 0 through 65535
- * means that branch targets in the form of x\$, where x is a number, are also called labels.

Figure B-1 Difference Among Global and Local Symbols

CODING STANDARDS AND CONVENTIONS

B.1.4.2 Symbol Examples - Non-global Symbols

ALB

ZXCJ1

INSRT

Global Address Symbols

\$JIM

.VECTR

\$SEC

Global Absolute Offset Symbols

A\$JIM

A\$XT

A.ENT

Global Bit Pattern Symbols

A1\$20

B3.6

JI.M

Local Symbols

37\$

271\$

6\$

B.1.4.3 Program-Local Symbols - Self-relative address arithmetic (.*n*) is absolutely forbidden in branch instructions. It must not be used in other contexts if at all possible.

Target labels (symbols) for branches that exist only for positional reference use local symbols of the form:

<num>\$:

Restrict use of global symbols to cases where reference to the symbol occurs external to the code. The assembler formats local symbols to proceed sequentially down the page and from page to page.

B.1.4.4 Macro Names - The last two characters in a macro name have special significance. The next to last character is a \$ and the last character specifies the mode of the macro. (The last character may also be a null character.)

CODING STANDARDS AND CONVENTIONS

For example, three macro forms exist: in-line, stack, and p-section. The in-line form has no suffix (the \$ is last). The stack has an S as the last character. The p-section has a C as the last character. Thus, the QUEUE I/O macro can be written in any one of the three following ways:

QIO\$

QIO\$\$

QIO\$C

The letters S and C are not reserved but the form of the macro is standard.

B.1.5 Program Modules

B.1.5.1 General Comments on Programs - In DEC software, a program provides a single distinct function. No limits exist on size, but the single function limitation makes large modules a rarity. Because any software may exploit the memory capacity of the 11/40, 11/45, or 11/70, programs should maintain a dense reference locus (branching should occur over short address distances).

All code is read only. Code and data areas are distinct and each contains explanatory text. Read-only data should be segregated from read-write data.

B.1.5.2 Module Preface - Each program module in the system exists as a separate file. The filename reflects the name of the module and the extension is of the form 'NNN'. NNN signifies the edit number or the version number. The version number is changed only when a new base level is created. Furthermore, if no corrections are made to a file from one base level to the next, the version number is not changed. The availability of File Control Services and File Control Primitives greatly simplify version number maintenance. Program modules adhere to a strict format. Reading and understanding the code is easier because of this format. The following sections are in each module:

For the code section:

1. A .TITLE statement that specifies the name of the module. If a module contains more than one routine, you may use subtitles.
2. A .IDENT statement that specifies the version number. The PDP-11 version number standard appears in section B.10.
3. A .PSECT statement that defines the program section in which the module resides.
4. A copyright statement and the disclaimer.
5. The version number of the file.

The PDP-11 version number standard is described in section B.10.

CODING STANDARDS AND CONVENTIONS

6. The name of the principal author and the date on which the module was first created.
7. The name of each modifying author and the date of the modification. Names and modification dates appear one per line and in chronological order.
8. A brief statement of the function of the module.
Note: Items 1-8 should appear on the same page.
9. A list of the definitions of all equated local symbols used in the module. These definitions appear one per line and in alphabetical order.
10. All local macro definitions, preferably in alphabetical order by name.
11. All local data. The data should indicate:
 - a. Description of each element (type, size, etc.)
 - b. Organization (functional, alpha, adjacent, etc.)
 - c. Adjacency requirements.
12. A more detailed definition of the function of the module.
13. A list of the inputs expected by the module. This includes the calling sequence if non-standard, condition code settings, and global data settings.
14. A list of the outputs produced as a result of entering this module. These include delivered results, condition code settings, but not side effects. (All these outputs are visible to the caller.)
15. A list of all effects (including side effects) produced as a result of entering this module. Effects include alterations in the state of the system not explicitly expected in the calling sequence, or those not visible to the caller.
16. The module code.

B.1.5.3 Formatting the Module Preface - Rules:

1. The first eight items appear on the same page and should not have explicit headings. Item 3 may be omitted if the blank P-section is being used.
2. Headings start at the left margin. The left margin consists of a semi-colon followed by a space. Therefore, the heading starts in column 3. Indent descriptive text one tab position.
3. Items 7-14 have headings that start at the left margin and are preceded and followed by lines that contain only a leading semi-colon. Omit items that do not apply.

CODING STANDARDS AND CONVENTIONS

An example of a module preface follows:

```
FILE-EXAMPL.S01
.TITLE   EXAMPLE
.IDENT   /01/
.PSECT   KERNEL

;
; COPYRIGHT (C) 1978
; DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.
;
; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY ON A
; SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH THE
; INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE, OR
; ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE
; MADE AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH
; SYSTEM AND TO ONE WHO AGREES TO THESE LICENSE TERMS. TITLE
; TO AND OWNERSHIP OF THE SOFTWARE SHALL AT ALL TIMES REMAIN
; IN DEC.
;
; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT
; NOTICE AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL
; EQUIPMENT CORPORATION.
;
; DEC ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF
; ITS SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DEC.
;
; VERSION 01
;
; JOHN HANCOCK 1-JAN-75
;
; MODIFIED BY:
;
; QUINCY ADAMS 21-JAN-76
;
; JAMES MONROE 12-JUN-76
;
; Brief statement of the module's function.
;
; EQUATED SYMBOLS
;
; List equated symbols.
;
; LOCAL MACROS
;
; Local macros
;
; LOCAL DATA
;
; Local data
;+
; Module function details
;
; INPUTS:
;
; Description of inputs
;
; OUTPUTS:
;
; Description of outputs
;
; EFFECTS:
;
; Descriptions of effects
;-
Begin module code
```

CODING STANDARDS AND CONVENTIONS

B.1.5.4 Modularity - No other characteristic has more impact on the ultimate engineering success of a system than does modularity. Modularity for PDP-11 Software Engineering's products consists of the application of the single-function philosophy described in section B.5.1 and the adherence to a set of calling and return conventions.

B.1.5.5 Inter-Module Calling Conventions - The following calling conventions must be observed:

Transfer of Control

Macros exist for call and return. The actual transfer is by a JSR PC instruction. For register save routines, a JSR Rn,SAVE instruction is permitted.

The CALL macro is:

CALL subroutine-name

The RETURN macro is:

RETURN

Register Conventions

Upon entry, a subroutine minimally saves all registers it intends to alter except result registers. Upon exit it restores these registers. (State preservation is assumed across calls.)

Argument Passing

Any registers may be used to pass arguments, but their use should follow a coherent pattern. For example, if passing three arguments, pass them in R0, R1, and R2 rather than R0, R2, and R5. Saving and restoring occurs in one place.

B.1.5.6 Exiting - All subroutine exits occur through a single RETURN macro.

B.1.5.7 Intra-Module Calling Conventions - These calling conventions are designer optional, but consistency favors a calling sequence identical to that of the inter-module sequence.

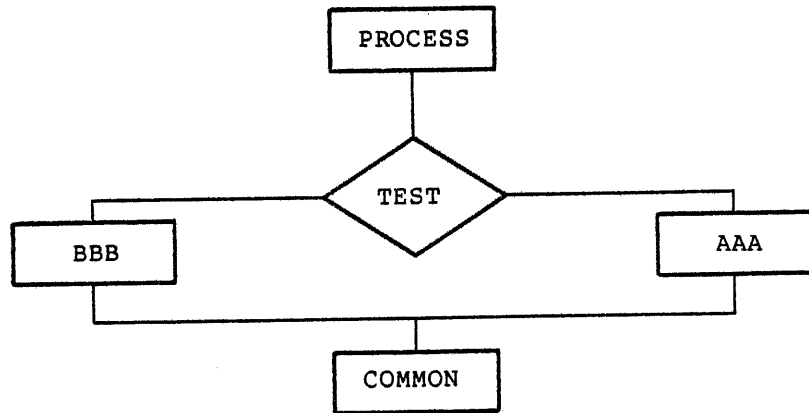
B.1.5.8 Success or Failure Indication - The C-bit indicates success or failure where success equals 0 and failure equals 1. You can use the argument registers to return values or additional success or failure data.

B.1.5.9 Module Checking Routines - Modules must verify the validity of arguments passed to them. The design of a module's calling sequence should aim at minimizing the validity checks by minimizing invalid combinations. You may add test code to perform additional checks during checkout. All code should aim at discovering an error as close (in terms of instruction executions) to its occurrence as possible.

CODING STANDARDS AND CONVENTIONS

B.1.6 Formatting Standards

B.1.6.1 Program Flow - Programs should be organized on the listing such that they flow down the page, even at the cost of an extra branch or jump. For example:



shall appear on the listing as:

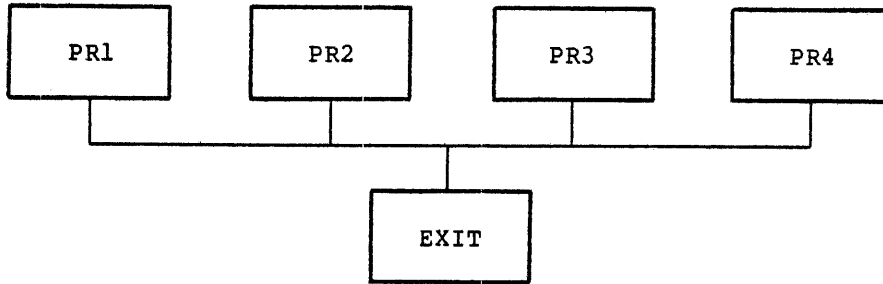
```
TST
BNE   BBB
AAA:  ....
      ....
BR    CMN
BBB:  ....
      ....
CMN:  ....
      ....
```

Rather than:

```
TST
BNE   BBB
AAA:  ....
      ....
CMN:  ....
      ....
BBB:  ....
      ....
BR    CMN
```

CODING STANDARDS AND CONVENTIONS

B.1.6.2 Common Exits - A common exit appears as the last code sequence on the listing, illustrated by the following flow chart:



should appear on the listing as:

```
PR1:      .....
          .....
          BR   EXIT
PR2:      .....
          .....
          BR   EXIT
PR3:      .....
          .....
          BR   EXIT
PR4:      .....
          .....
EXIT:     .....
```

and not as:

```
PR1:      .....
          .....
EXIT:     .....
          .....
PR2:      .....
          .....
          BR   EXIT
PR3:      .....
          .....
          BR   EXIT
PR4:      .....
          .....
          BR   EXIT
```

CODING STANDARDS AND CONVENTIONS

B.1.6.3 Code with Interrupts Inhibited - Code that executes instructions with interrupts inhibited, should be flagged by a comment delimiter of three semicolons. For example:

```
..ERTZ:                                ;ENABLE BY RETURNING
                                        ;BY SYSTEM SUBROUTINES
BIS      PR7,PS                          ;;; INHIBIT INTERRUPTS
BIT      PR7,+2(SP)                       ;;; C
BEQ      10$                              ;;; O
RTT      .                               ;;; M
                                        ;;; M
10$:     ....                             ;;; E
        ....                             ;;; N
        ....                             ;;; T
        ....                             ;;; S
```

B.1.7 Program Source Files

Creation of and maintenance of source code is done in base levels. A base level is defined as a point at which changing the program source files is no longer allowed. From this "frozen" point to the next base level, corrections are not made to the base level itself; rather, a file of corrections is accumulated for each file in the base level. Whenever a updated source file is desired, the correction file is applied to the base file.

The accumulation of corrections proceeds until a logical breaking point has occurred (for example, a milestone or significant implementation point is reached). At this time, all accumulated corrections are applied to the previous base level to create a new base level. Correction files are then started for the new base level.

B.1.8 Forbidden Instruction Usage

The following instruction usage is not allowed:

1. The use of intructions or index words as literals of the previous instruction. For example, the sequence:

```
MOV      @PC,REGISTER
BIC      SRC,DST
```

uses the bit clear instruction as a literal. This may seem to be a very good way to save a word but this compounds the problem of program maintenance. To make matters worse, this sequence does not execute correctly if I/D space is enabled on the PDP-11/45. In this case, @PC is a D bank reference.

2. The use of the MOV instruction instead of a JMP instruction to transfer program control to another location. For example:

```
MOV      ALPHA,PC
```

transfers control to location ALPHA. Besides taking longer to execute (2.3 microseconds for the MOV vs. 1.2 for the JMP) the use of MOV instead of JMP makes it nearly impossible to pick up someone else's program and tell where transfers of control take place. A jump trace of a program's execution is

CODING STANDARDS AND CONVENTIONS

impossible in this case (a move trace is unheard of). As a more general issue, other operations such as ADD and SUB from the PC should be discouraged. Possibly one or two words can be saved by using these operations, but occurrences where these operations can be used are rare.

3. The seemingly clever use of all single word instructions where one double-word instruction could be used, would execute faster, and would not consume additional memory. Consider the following instruction sequence:

```
CMP      -(R1),(-R1)
```

```
CMP      -(R1),-(R1)
```

The intent of this instruction sequence is to subtract 8 from register R1 (not to set condition codes). This can be accomplished in approximately 1/3 the time by using a SUB instruction (9.4 vs. 3.8 microseconds) at no additional cost in memory space. Also, if R1 is odd this instruction does not give the correct result. The SUB instruction is always faster and always executes correctly.

B.1.9 Recommended Coding Practice

B.1.9.1 Conditional Branches - When using the PDP-11 conditional branch instructions, the correct choice must be made between the signed and the unsigned branches.

Signed	Unsigned
BGE	BHIS (BCC)
BLT	BLO
BGT	BHI
BLE	BLOS (BCS)

A common pitfall is to use a signed branch (for example, BGT) when comparing two memory addresses. This compare works until the two addresses have opposite signs; that is, one of them goes across the 16K (100000(8)) boundary. This type of coding error usually occurs when a program is re-linked at different addresses or the program's size is changed.

B.1.10 PDP-11 Version Number Standard

The PDP-11 Version Number Standard applies to all modules, parameter files, complete programs, and libraries that are written as part of the PDP-11 Software Development effort. It provides unique identification of all released, pre-released, and in-house software.

It is limited because, as currently specified, only six characters of identification are used. Future implementations of the Macro Assembler, Task Builder, and Librarian should provide for at least

CODING STANDARDS AND CONVENTIONS

nine characters, and possibly twelve. This standard will change as the need arises. The version identifier takes the following form:

Version Identifier = <form> <version> <edit> <patch>

<form>	Identifies a particular form of a module or program, where applicable, as in the case of LINK-11. One alphabetic character, if used, and null (for example, a binary 0) if not used.
<version>	Identifies the release or generation of a program. The version number is two decimal digits that start at 00 and is incremented at the discretion of the project to reflect a major change.
<edit>	Identifies the level to which a particular release or generation of a program or module has been edited. An edit is an alteration to the source form. The edit number consists of two decimal digits that begin at 01. The number is incremented with each edit and remains null if no edit occurs.
<patch>	Identifies the level to which a particular release or generation of a program or module has been patched. A patch is an alteration to a binary form. The patch identification consists of one alphabetic character that starts at B. The character is changed toward Z each time a set of patches is released and remains null if no patches are made.

These fields are interrelated. When version is changed, patch and edit must be reset to nulls. When edit is incremented, patch is re-set to null because various bugs have been fixed.

B.1.10.1 Displaying the Version Identifier - The visible output of the version identifier should appear as:

Key <letter> <form> <version> - <edit> <patch>

where the following Key letters have been identified:

V	released or frozen version
X	in-house experimental version
Y	field test, pre-release, or in-house release version

and 'V' to company support.

Use the dash that separates version from edit only if edit or patch is not null. When a version identifier is displayed as part of a program identification, the format is:

Program <space><key-letter><form><version>-<edit><patch>
Name

Examples:

PIP X03
LINK VB04-C
MACRO Y05-01

CODING STANDARDS AND CONVENTIONS

B.1.10.2 Use of the Version Number in the Program - All sources must contain the version number in an `.IDENT` directive. For programs (or libraries) that consist of more than one module, each individual module follows this version number standard. The version number of the program or library is not necessarily related to the version numbers of the constituent modules; it is perfectly reasonable, for example, that the first version of a new FORTRAN library, V00, contain an existing SIN routine, say V05-01.

Parameter files are also required to contain the version number in an `.IDENT` directive. Because the assembler records the last `.IDENT` seen, parameter files must precede the program.

Entities that consist of a collection of modules or programs (for example, the FORTRAN library) have an identification module in the first position. An identification module exists solely to provide identification and normally consists of something like:

```

;OTS IDENTIFICATION
.TITLE FTNLIB
.IDENT /003010/
.END
    
```

B.1.11 Co-routines

In some cases, two program routines are highly interactive. They then use a special case of the JSR instruction (`JSR PC,@(SP)+`). This form of the JSR instruction exchanges the top element of the stack and the contents of the program counter (PC). In this way, two routines may swap program control and resume operation where they stopped, when recalled. Such routines are called co-routines.

This swapping of control is illustrated below with an actual sequence of instructions from RSX-11M Executive code (Version 3.1).

CODE	ACTION	STACK
<pre> \$DRECP:: . . CALLR \$NXTSK (1) (JMP \$NXTSK) </pre>	<pre> Address of caller of \$DRECP > (1) PC=Address of \$NXTSK </pre>	<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> CALLER ADR </div>
<pre> \$NXTSK:: SAVNR (1) (JSR R5,\$SAVNR) (2) MOV P.MAIN(R0),R5 </pre>	<pre> (1) Push R5 contents onto stack (save R5). Put address of next instruction (2) into R5 (return address). Put address of \$SAVNR into PC. </pre>	<div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> STACK CALLER ADR <hr style="border: none; border-top: 1px solid black;"/> R5 </div>

CODING STANDARDS AND CONVENTIONS

\$SAVNR::

- | | |
|------------------------------------|---|
| (1) MOV R4, (SP) | (1) Save R4 on stack |
| (2) MOV R5, (SP) | (2) Save R5 on stack |
| (3) MOV 4(SP),R5 | (3) Restore saved R5 from the stack into R5 |
| (4) CALL @(SP)+
(JSR PC,@(SP)+) | (4) Pop "call to" address (address of step (2) in \$NXTSK) into PC. |
| (5) MOV (SP)+,R4 | Push address of next instruction (5) onto stack. |
| (6) MOV (SP)+,R5 | |
| (7) RETURN
(RTS PC) | |

STACK

CALLER ADR
R5
R4
(5)

\$NXTSK::

- | | |
|------------------------|--|
| SAVNR | |
| (1) MOV P.MAIN(R0),R5 | (1) Start \$NXTSK processing here. |
| . | (2) Pop address of (5), in \$SAVNR, into PC. |
| . | |
| (2) RETURN
(RTS PC) | |

STACK

CALLER ADR
R5
R4

\$SAVNR::

- | | |
|------------------------|----------------------------------|
| . | (5) Pop saved R4 into R4. |
| (5) MOV (SP)+,R4 | (6) Pop saved R5 into R5. |
| (6) MOV (SP)+,R5 | (7) Return to caller of \$DRECP. |
| (7) RETURN
(RTS PC) | |

STACK

CALLER ADR

APPENDIX C
MACRO EXPANSIONS

C.1 COMMAND STRING INTERPRETER MACRO EXPANSIONS

C.1.1 CSI\$1 Macro

This macro calls the command string syntax analyzer.

```
.MACRO CSI$1 CSBLK,BUFF,LEN
.GLOBL .CS11
.MCALL CSI$,LDR0$
.IF NDF C.SIZE
CSI$
.ENDC
LDR0$ CSBLK
.IF NB <BUFF>
MOV BUFF,C.CMLD+2(R0)
.ENDC
.IF NB <LEN>
MOV LEN,C.CMLD(R0)
.ENDC
JSR PC,.CS11
.ENDM
```

C.1.2 CSI\$2 Macro

This macro calls the command string semantic parser.

```
.MACRO CSI$2 CSBLK,IO,SWTAB
.GLOBL .CS12
.MCALL CSI$,LDR0$
.IF NDF C.SIZE
CSI$
.ENDC
LDR0$ CSBLK
.IF NB <IO>
.IF IDN <INPUT>,<IO>
MOVB #CS.INP,(R0)
.IFF
.IF IDN <OUTPUT>,<IO>
MOVB #CS.OUT,(R0)
.IFF
.ERROR ;Incorrect request to .CS12
.ENDC
.ENDC
.ENDC
```

MACRO EXPANSIONS

```
.IF NB <SWTAB>
MOV SWTAB,C.SWAD(R0)
.ENDC
JSR PC,.CSI2
.ENDM
```

C.1.3 CSI\$SW Macro

This macro defines the switch table entry.

```
.MACRO CSI$SW SW,MK,MKW,CLR,NEGS,VALTAB,LNG
.IF B,SW
.ERROR ;Missing switch name
.IFF
.EVEN ;Force to word alignment
.NCHR SIZ$$,SW ;Set SIZ$$ to no. of chars in sw
.IF GT,SIZ$$-2 ;If string is longer than 2 chars
.IF B,LNG ;and not "long" or "exact"
SIZ$$=2 ;trim it back to 2 characters
.ENDC
.ENDC
CNT$$=0
.IRPC CHR$$,SW ;For each character in SW
CHR$$='CHR$$
CNT$$=CNT$$+1
.IF GE,<CHR$$-<'A+^O40>> ;If .GE. lower case A
.IF LE,<CHR$$-<'Z+^O40>> ; and .LE. lower case Z
CHR$$=CHR$$-^O40 ; convert to upper case alpha
.ENDC
.ENDC
.IF LE,CNT$$-SIZ$$ ;If more characters to store
$,=0
.IF NB,LNG ;If "long" or "exact"
.IF IDN,<LNG>,<EXACT> ;If "exact"
.IF GT,CNT$$&1 ; and if first byte of word
.IF GE,CNT$$+1-SIZ$$ ; and if last word in switch name
$$=^O200 ; then set the exact match flag
.ENDC
.IFF ;If 2nd byte of word
.IF LT,CNT$$-SIZ$$ ; and if not the last character
$$=^O200 ; set the "more ASCII words
; coming" bit
.ENDC
.ENDC ;End of "if first byte of word"
.IFF ;If not "exact"
.IF IDN,<LNG>,<LONG> ;Must be "long", otherwise error
.IF EQ,CNT$$&1 ;If 2nd byte of word
.IF LT,CNT$$-SIZ$$ ; and not the last character
$$=^O20 ; set the "more ASCII words
; coming" bit
.ENDC
.ENDC
.IFF ;If not "long"
.ERROR ;Illegal "long" or "exact"
; specifier
.ENDC ;End of if "long"
.ENDC ;End of if "exact"
.ENDC ;End of IF NB,LNG
.BYTE CHR$$!$$ ;Generate a character
.ENDC ;End of "if more characters"
.ENDM ;End of the IRPC loop
.EVEN ;Round up to next word boundary
```

MACRO EXPANSIONS

```

.ENDC                                ;End of .IF B,SW
.IF NB MK
.WORD MK
.IFF
.WORD 0
.ENDC
$$=0
.IF NB CLR
.IF IDN <CLEAR>,<CLR>
$$=1
.IFF
.IF IDN <SET>,<CLR>
.IFF
.ERROR                                ;Invalid set/clear spec
.ENDC
.ENDC
.ENDC
.WORD MKW+$$
$$=0
.IF NB NEGS
.IF IDN <NEG>,<NEGS>
$$=1
.IFF
.ERROR                                ;Invalid negate spec
.ENDC
.ENDC
.WORD VALTAB+$$
.ENDM

```

C.1.4 CSI\$ND Macro

This macro defines the end of the switch value table end.

```

.MACRO CSI$ND
.WORD 0
.ENDM

```

C.1.5 CSI\$SV Macro

This macro defines the switch value table entry.

```

.MACRO CSI$SV TYPE,ADDR,LEN,VALTAB
.IF NB VALTAB
VALTAB:
.ENDC
.IF NB TYPE
.IF IDN <ASCII>,<TYPE>
.BYTE 1
.IFF
.IF IDN <NUMERIC>,<TYPE>
.BYTE 2
.IFF
.IF IDN <OCTAL>,<TYPE>
.BYTE 2
.IFF
.IF IDN <DECIMAL>,<TYPE>
.BYTE 3
.IFF
.ERROR                                ;Invalid conversion type
.ENDC

```

MACRO EXPANSIONS

```

.ENDC
.ENDC
.ENDC
.IFF
.BYTE 1 ;ASCII conversion assumed
.ENDC
.IF NB LEN
.BYTE LEN
.IFF
.ERROR ;Length missing
.BYTE 0
.ENDC
.IF NB ADDR
.WORD ADDR
.IFF
.ERROR ;Value address missing
.WORD 0
.ENDC
.ENDM

```

C.1.6 LDR0\$ Macro

This macro conditionally loads R0.

```

.MACRO LDR0$ ARG
.IIF B,ARG,.MEXIT
.NTYPE PAR$$$ ,ARG
.IIF EQ,PAR$$$ ,.MEXIT
MOV ARG,R0
.ENDM

```

C.1.7 CSI\$ Macro

This macro defines command string interpreter symbols.

```

.MACRO CSI$ GBL
.IF IDN <GBL>,<DEF$G>
.GLOBL C.TYPR,C.STAT,C.CMLD,C.DEVD,C.DIRD,C.FILD,C.SWAD
.GLOBL C.MKW1,C.MKW2
.GLOBL C.SIZE,C.DSDS,CS.INP,CS.OUT
.GLOBL CS.NMF,CS.DIF,CS.DVF,CS.WLD,CS.MOR,CS.EQU
.ENDC

```

```

C.TYPR=0 ;Request type (for CSI2)
C.STAT=1 ;Status (from CSI2)
C.CMLD=2 ;Length of command string buffer
;Address of command string buffer
C.DEVD=6 ;Length of device name string
;Address of most recent device
; name string
C.DIRD=^O<12> ;Length of most recent DIR info
;Address of DIR info
C.FILD=^O<16> ;Length appropriate to C.FILD+2
;CSI1 - address of string seg
; where error occurred
;CSI2 - address of file name for
; current request
C.SWAD=^O<22> ;Address of current switch table
C.MKW1=^O<24> ;OR of masks for all switches
; found this call

```

MACRO EXPANSIONS

```
C.MKW2=^O<26> ;On/off settings of masks for
; switches found
C.SIZE=^O<54> ;Size of CSI control block (in
; bytes)
C.DSDS=C.DEVD ;Displacement to 6 word block for
; FCS
;C.TYPR VALUES
CS.INP=1 ;Request input string
CS.OUT=2 ;Request output string
;C.STAT VALUES
CS.NMF=1 ;l = File name specified this seg
; of string
CS.DIF=2 ;l = Dir info specified this seg
; of string
CS.DVF=4 ;l = Device name specified this
; seg of string
CS.WLD=^O<10> ;File name has wild card switch
CS.MOR=^O<20> ;More string segs follow
CS.EQU=^O<40> ;Equal sign seen by CS11
```

C.2 DIRECTIVE MACRO EXPANSIONS

```
.MACRO CALL ADR
JSR PC,ADR
.ENDM CALL
.MACRO RETURN
RTS PC
.ENDM RETURN

.MACRO CALLR ADR
JMP ADR
.ENDM CALLR
```

C.2.1 ABRT\$C Macro

ABRT\$C generates a DPB for the ABORT TASK directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: ABRT\$C TSK,PSCT,ERR

Description: ABRT\$C generates a DPB for the ABORT TASK directive in the separate \$DPB\$\$ program section and generates an EMT 377 in the program section specified by PSCT. TSK, the task name, is the only argument required for the DPB definition. The optional argument, ERR, must be a valid assembler destination operand specifying an error routine address. Because this routine invokes ABRT\$, the same factors govern the expansion of code and symbolic offsets.

```
.MACRO ABRT$C TSK,PSCT,ERR
.MCALL ABRT$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ABRT$ TSK
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM ABRT$C
```

MACRO EXPANSIONS

C.2.2 ABRT\$\$ Macro

ABRT\$\$ pushes a DPB for the ABORT TASK directive on the stack and generates an EMT 377.

Macro call: ABRT\$\$ NAMADR,ERR

Description: ABRT\$\$ pushes a DPB for the ABORT TASK directive on the stack. The argument, NAMADR, must be a valid assembler operand that yields the address of a double-word with the task name in RADIX-50. ABRT\$\$ then generates an EMT 377 and considers the optional argument, ERR, the error routine address.

```
.MACRO ABRT$$ NADR,ERR
.MCALL RFA$,DIR$
RFA$ NADR
MOV (PC)+,-(SP)
.BYTE 83.,3
DIR$,ERR
.ENDM ABRT$$
```

C.2.3 ABRT\$ Macro

The ABRT\$ macro generates a DPB for the ABORT TASK directive.

Macro call: ABRT\$ TSK

Description: This macro creates a DPB for the ABORT TASK directive. The argument, TSK, is the name of the task to be aborted.

This macro defines the following symbolic offset:

A.BTTN-(length is 4 bytes) Task name.

The ABRT\$ macro generates the A.BTTN symbolic offset only when the \$\$\$GLB is defined. The symbol, A.BTTN, is a global symbol in this case.

```
.MACRO ABRT$ TSK
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 83.,3
R50$ TSK
.ENDC
.IF NDF A.BTTN
.NLIST
OFF$
OFF$ A.BTTN,4
.LIST
.ENDC
.ENDM ABRT$
```

C.2.4 ALTP\$C Macro

The ALTP\$C macro generates a DPB for the ALTER TASK PRIORITY directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: ALTP\$C TTASK,NPRIO,PSCT,ERR

MACRO EXPANSIONS

Description: This macro generates a DPB for the ALTER TASK PRIORITY directive in the program section named \$DPB\$\$\$. The DPB parameters are described in the ALTP\$ macro. Then the macro generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The ALTP\$ macro controls symbolic offset generation.

```
.MACRO ALTP$C TTASK,NPRIO,PSCT,ERR
.MCALL ALTP$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$$
.IFTF
ALTP$ TTASK,NPRIO
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM ALTP$C
```

C.2.5 ALTP\$\$ Macro

The ALTP\$\$ macro generates the code to push a DPB for the ALTER TASK PRIORITY directive on the stack followed by EMT 377.

Macro call: ALTP\$\$ NAMADR,NPRIO,ERR

Description: This macro generates the code to push a DPB for ALTER TASK PRIORITY on the stack. The argument, TNAMADR, must be a valid assembler source operand that yields the address of a double-word containing the RADIX-50 name specified in the ALTP\$ macro. The macro then generates an EMT 377. The DIR\$ macro describes the ERR error service address.

```
.MACRO ALTP$$ NAME,PRI,ERR
.MCALL MOV$,RFA$,DIR$
MOV$ PRI
RFA$ NAME
MOV (PC)+,-(SP)
.BYTE 9.,4
DIR$ ,ERR
.ENDM ALTP$$
```

C.2.6 ALTP\$ Macro

The ALTP\$ macro generates a DPB for the ALTER TASK PRIORITY directive.

Macro call: ALTP\$ TTASK,NPRIO

Description: This macro generates a DPB for the ALTER TASK PRIORITY directive. The arguments are assumed to have the following meanings:

TTASK=Name of target task for new priority
NPRIO=New priority for task

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

A.LTTN-(length 4 bytes) target task name,
T.LTPR-(length 2 bytes) new priority

MACRO EXPANSIONS

If the macro is invoked with the \$\$\$GLB symbol defined, the ALTP\$ macro does not generate the DPB and ALTP\$ defines the offsets as global symbols.

```
.MACRO ALTP$ NAME,PRI
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 9.,4
R50$ NAME
.WORD PRI
.ENDC
.IF NDF A.LTTN
.NLIST
OFF$
OFF$ A.LTTN,4
OFF$ A.LTPR,2
.LIST
.ENDC
.ENDM ALTP$
```

C.2.7 ALUN\$C Macro

The ALUN\$C macro generates a DPB for the ASSIGN LUN directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro Call: ALUN\$C LUN,DEVNAM,DEVNUM,PSCT,ERR

Description: This macro generates a DPB for the ASSIGN LUN directive in the program section named \$DPB\$\$\$. The arguments through DEVNUM are the DPB parameters as described for the ALUN\$ macro. The macro then generates an EMT 377 in the program section as specified by PSCT. The argument, ERR is as described for the DIR\$ macro. For an explanation of symbolic offsets, see the ALUN\$ macro.

```
.MACRO ALUN$C LUN,DA,DU,CS,ERR
.MCALL ALUN$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ALUN$ LUN,DA,DU
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM ALUN$C
```

C.2.8 ALUN\$\$S Macro

The ALUN\$\$S macro generates the code to push a DPB for the ASSIGN LUN directive on the stack followed by an EMT 377.

Macro call: ALUN\$\$S LUN,DEVNAM,DEVNUM,ERR

Description: This macro generates the code to push a DPB for the ASSIGN LUN directive on the stack. The arguments through DEVNUM must be valid assembler source operands, and they must specify the DPB parameters listed in the ALUN\$ macro. The DIR\$ macro describes the ERR argument.

MACRO EXPANSIONS

```
.MACRO ALUN$S LUN,DA,DU,ERR
.MCALL MOV$,DIR$
MOV$ DU
MOV$ DA
MOV$ LUN
MOV (PC)+,-(SP)
.BYTE 7,4
DIR$,ERR
.ENDM ALUN$S
```

C.2.9 ALUN\$ Macro

The ALUN\$ macro generates a DPB for the ASSIGN LUN directive.

Macro call: ALUN\$ LUN,DEVNAM,DEVNUM

Description: This macro creates a DPB for the ASSIGN LUN directive. The arguments are assumed to have the following meanings:

```
LUN=Logical unit number,
DEVNAM=Physical device name (two characters),
DEVNUM=Physical device unit number.
```

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

```
A.LULU-(length 2 bytes) Logical unit number,
A.LUNA-(2) Physical device name,
A.LUNU-(2) Physical device unit number.
```

If the macro is invoked with the symbol, \$\$\$GLB, defined, ALUN\$ does not generate the DPB and ALUN\$ defines the offsets as global symbols.

```
.MACRO ALUN$ LUN,DA,DU
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 7,4
.WORD LUN
.IF B <DA>
.WORD 0
.IFF
.NCHR $$$T1,<DA>
$$$T2=.
.ASCII /DA/
.=$$$T2+2
.ENDC
.WORD DU
.ENDC
.IF NDF A.LULU
.NLIST
.IRP X,<,<A.LULU,2>,<A.LUNA,2>,<A.LUNU,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM ALUN$
```

MACRO EXPANSIONS

C.2.10 ASTX\$C Macro

The ASTX\$C macro generates a DPB for the AST SERVICE EXIT directive in a separate PSECT followed by an EMT 377 in the user specified program PSECT.

Macro call: ASTX\$C PSCT,ERR

Description: This macro generates a DPB for the AST SERVICE EXIT directive in the program section named \$DPB\$\$\$. The argument, PSCT, is the program section in which to generate an EMT 377. The DIR\$ macro describes the ERR argument.

```
.MACRO ASTX$C PSCT,ERR
.MCALL ASTX$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ASTX$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM ASTX$C
```

C.2.11 ASTX\$\$ Macro

The ASTX\$\$ macro generates the code to push a DPB for the AST SERVICE EXIT directive on the stack followed by an EMT 377.

Macro call: ASTX\$\$ ERR

Description: This macro generates the code to push a DPB for the AST SERVICE EXIT directive on the stack. Then a MONITOR TRAP is generated. The error routine address argument, ERR, results in an unconditional call following the EMT because control should never be returned on the AST exit.

```
.MACRO ASTX$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 115.,1
DIR$
.IIF NB <ERR>, JSR PC,ERR
.ENDM ASTX$$
```

C.2.12 ASTX\$ Macro

The ASTX\$ macro generates a DPB for the AST SERVICE EXIT directive.

Macro Call: ASTX\$

Description: This macro creates a DPB for the AST SERVICE EXIT directive. It takes no arguments.

MACRO EXPANSIONS

Note: If the symbol, \$\$\$GLB, is defined, this macro does not generate any code.

```
.MACRO ASTX$
  .IF NDF $$$GLB
  .BYTE 115.,1
  .ENDC
.ENDM ASTX$
```

C.2.13 CINT\$C Macro

The CINT\$C macro generates a DPB in a separate PSECT for the CONNECT TO INTERRUPT VECTOR directive followed by an EMT 377 in the user specified PSECT.

Macro call: CINT\$C VEC,BASE,ISR,DSI,PSW,AST,PSCT,ERR

Description: CINT\$C generates a DPB in the program section named \$DPB\$\$ for the CONNECT TO INTERRUPT VECTOR directive. CINT\$ describes the DPB parameters. CINT\$C then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO CINT$C VEC,BASE,ISR,DSI,PSW,AST,PSCT,ERR
.MCALL CINT$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CINT$ VEC,BASE,ISR,DSI,PSW,AST
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM CINT$C
```

C.2.14 CINT\$\$ Macro

The CINT\$\$ macro generates the code to push a DPB on the stack for the CONNECT TO INTERRUPT VECTOR directive followed by an EMT 377.

Macro call: CINT\$\$ VEC,BASE,ISR,DSI,PSW,AST,ERR

Description: CINT\$\$ generates the code to push a DPB for the CONNECT TO INTERRUPT VECTOR directive on the stack. The DPB arguments must be valid assembler source operands and then must specify the information described in the CINT\$ macro. The macro, RVP\$, pushes the arguments on the stack in reverse order. CINT\$\$ also generates an EMT 377 and uses the ERR error service address as described in the DIR\$ macro.

```
.MACRO CINT$$ VEC,BASE,ISR,DSI,PSW,AST,ERR
.MCALL RVP$,DIR$
RVP$ VEC,BASE,ISR,DSI,PSW,AST
MOV (PC)+,-(SP)
.BYTE 129.,7
DIR$ ,ERR
.ENDM CINT$$
```

MACRO EXPANSIONS

C.2.15 CINT\$ Macro

CINT\$ generates a DPB for the CONNECT TO INTERRUPT VECTOR directive.

Macro call: CINT\$ VEC,BASE,ISR,DSI,PSW,AST

Description: CINT\$ generates a DPB for the CONNECT TO INTERRUPT VECTOR directive. The arguments are assumed to have the following meanings:

- VEC = Must be in the range 60(8) through the highest vector specified during SYSGEN, and must be a multiple of 4.
- BASE = Virtual base address for kernel APR 5 mapping of the ISR, and enable/disable interrupt routines -- This address is automatically truncated to a 32(10)-word boundary. The "base" argument is ignored in an unmapped system.
- ISR = Virtual address of the ISR, or 0 to disconnect from the interrupt vector.
- DSI = Virtual address of the enable/disable interrupt routine.
PSW = Initial priority at which the ISR is to execute -- This is normally equal to the hard-wired interrupt priority, and is expressed in the form n*40, where n is a number in the range 0-7. This form puts the value in bits 5-7 of pri. It is recommended that the programmer make use of the symbols PR4, PR5, PR6, and PR7 for this purpose. These are implemented via the macro HWDDF\$ found in [1,1]EXEMC.MLB. AST = Virtual address of an AST routine to be entered after the fork level routine queues an AST.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

- C.INVE -- Vector address (2)
- C.INBA -- Base address (2)
- C.INIS -- ISR address (2)
- C.INDI -- Enable/disable interrupt routine address (2)
- C.INPS -- Priority (2)
- C.INAS -- AST address (2)

If the macro is invoked with the \$\$\$GLB symbol defined, the CINT\$ macro does not generate the DPB and CINT\$ defines the offsets as global symbols.

```
.MACRO CINT$ VEC,BASE,ISR,DSI,PSW,AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 129.,7
.WORD VEC
.WORD BASE
.WORD ISR
.WORD DSI
.WORD PSW
.WORD AST
.ENDC
.IF NDF C.INVE
OFF$
OFF$ C.INVE,2
OFF$ C.INBA,2
OFF$ C.INIS,2
```

MACRO EXPANSIONS

```
OFF$ C.INDI,2
OFF$ C.INPS,2
OFF$ C.INAS,2
.ENDC
.ENDM CINT$
```

C.2.16 CLEF\$C Macro

The CLEF\$C macro generates a DPB for the CLEAR EVENT FLAG directive in a separate PSECT followed by an EMT 377 in the user specified program PSECT.

Macro call: CLEF\$C EFN,PSCT,ERR

Description: This macro generates a DPB for the CLEAR EVENT FLAG directive in the \$DPB\$\$ program section. The CLEF\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The CLEF\$ macro controls symbolic address generation.

```
.MACRO CLEF$C EFN,CS,ERR
.MCALL CLEF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CLEF$ EFN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM CLEF$C
```

C.2.17 CLEF\$\$ Macro

The CLEF\$\$ macro generates the code to push a DPB for the CLEAR EVENT FLAG directive on the stack followed by an EMT 377.

Macro call: CLEF\$\$ EFN,ERR

Description: This macro generates the code to push a DPB for the CLEAR EVENT FLAG directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the CLEF\$ macro. This macro also generates a monitor trap, using the error service address, ERR, as described in the DIR\$ macro.

```
.MACRO CLEF$$ EFN,ERR
.MCALL MOV$,DIR$
MOV$ EFN
MOV (PC)+,-(SP)
.BYTE 31.,2
DIR$ ,ERR
.ENDM CLEF$$
```

MACRO EXPANSIONS

C.2.18 CLEF\$ Macro

The CLEF\$ macro generates a DPB for the CLEAR EVENT FLAG directive.

Macro call: CLEF\$ EFN

Description: This macro generates a DPB for the CLEAR EVENT FLAG directive. The argument is assumed to have the following meaning:

EFN=EVENT FLAG NUMBER.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

C.LEEF-(length 2 bytes) Event flag number.

If the macro is invoked with the symbol, \$\$\$GLB, defined, CLEF\$ does not generate the DPB and CLEF\$ defines the offset as a global symbol.

```
.MACRO CLEF$ EFN
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 31.,2
.WORD EFN
.ENDC
.IF NDF C.LEEF
.NLIST
OFF$
OFF$ C.LEEF,2
.LIST
.ENDC
.ENDM CLEF$
```

C.2.19 CMKT\$C Macro

The CMKT\$C macro generates a DPB for the CANCEL MARK-TIME REQUESTS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro Call: CMKT\$C EFN,AST,PSCT,ERR

Description: This macro generates a DPB for the CANCEL MARK-TIME REQUESTS directive in the \$DPB\$\$ program section. The CMKT\$\$ macro describes the DPB parameters. Then the macro generates an EMT 377 in the program section named in the PSCT argument. The DIR\$ macro describes the ERR error service address.

```
.MACRO CMKT$C EFN,AST,PSCT,ERR
.MCALL CMKT$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CMKT$ EFN,AST
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM CMKT$C
```


MACRO EXPANSIONS

C.2.20 CMKT\$\$ Macro

The CMKT\$\$ macro generates the code to push a DPB for the CANCEL MARK-TIME REQUESTS directive on the stack followed by an EMT 377.

Macro call: CMKT\$\$ EFN,AST,ERR

Description: This macro generates the code to push a DPB for the CANCEL MARK-TIME REQUESTS directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the CMKT\$ macro. The macro also generates an EMT 377. The DIR\$ macro describes the ERR error service address.

Note: The EFN and AST arguments are required to maintain compatibility with RSX-11D. Despite the fact that they must be specified, the CMKT\$\$ macro ignores them.

```
.MACRO CMKT$$ EFN,AST,ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 27.,1
DIR$ ,ERR
.ENDM CMKT$$
```

C.2.21 CMKT\$ Macro

The CMKT\$ macro generates a DPB for the CANCEL MARK-TIME REQUESTS directive.

Macro call: CMKT\$ EFN,AST

Description: This macro generates a DPB for the CANCEL MARK-TIME REQUESTS directive.

Note: The EFN and AST arguments are required to maintain compatibility with RSX-11D. Despite the fact that they must be specified, the CMKT\$ macro ignores them. If the \$\$\$GLB symbol is defined, this macro does not generate any code.

```
.MACRO CMKT$ EFN,AST
.IF NDF $$$GLB
.BYTE 27.,1
.ENDC
.ENDM CMKT$
```

C.2.22 CSRQ\$C Macro

The CSRQ\$C macro generates a DPB for the CANCEL SCHEDULED REQUESTS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: CSRQ\$C TTASK,RTASK,PSCT,ERR

Description: This macro generates a DPB for the CANCEL SCHEDULED REQUESTS directive in the \$DPB\$\$ program section. The DPB parameters are described in the CSRQ\$ macro. Then the macro generates an EMT 377 in the original program section named in PSCT. The DIR\$ macro describes the ERR error service address. The CSRQ\$ macro controls symbolic address generation.

MACRO EXPANSIONS

Note: Read the discussion notice concerning ignored arguments in the CSRQ\$ macro below.

```
.MACRO CSRQ$C TT,RT,CS,ERR
.MCALL CSRQ$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CSRQ$ TT,RT
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM CSRQ$C
```

C.2.23 CSRQ\$\$ Macro

The CSRQ\$\$ macro generates the code to push a DPB for the CANCEL SCHEDULED REQUESTS directive on the stack followed by an EMT 377.

Macro call: CSRQ\$\$ TNAMADR,RNAMADR,ERR

Description: This macro generates the code to push a DPB for CANCEL SCHEDULED REQUESTS on the stack. The TNAMADR and RNAMADR arguments must be valid assembler source operands that yield the addresses of double-words containing the RADIX-50 names specified in the CSRQ\$ macro. The macro then generates an EMT 377. The DIR\$ macro describes the ERR error service address.

Note: Read the discussion notice concerning ignored arguments in the CSRQ\$ macro below.

```
.MACRO CSRQ$$ TN,RN,ERR
.MCALL RFA$,DIR$
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 25.,3
DIR$ ,ERR
.ENDM CSRQ$$
```

C.2.24 CSRQ\$ Macro

The CSRQ\$ macro generates a DPB for the CANCEL SCHEDULED REQUESTS directive.

Macro call: CSRQ\$ TTASK,RTASK

Description: This macro generates a DPB for the CANCEL SCHEDULED REQUESTS directive. The arguments are assumed to have the following meanings:

TTASK=Scheduled (target) task name,
RTASK=Scheduler (requester) task name.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

C.SRTN-(length 4 bytes) target task name,
C.SRRN-(4) requester task name.

MACRO EXPANSIONS

If the macro is invoked with the \$\$\$GLB symbol defined, CSRQ\$ does not generate the DPB and it globally defines the symbolic offsets.

Note: The RTASK argument is required to maintain compatibility with RSX-11D. Despite the fact that it must be specified, the CSRQ\$ macro ignores it.

```
.MACRO CSRQ$ TT,RT
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 25.,3
R50$ TT
.ENDC
.IF NDF C.SRTN
.NLIST
OFF$
OFF$ C.SRTN,4
OFF$ C.SRRN,4
.LIST
.ENDC
.ENDM CSRQ$
```

C.2.25 DECL\$C Macro

The DECL\$C macro generates a DPB for the DECLARE SIGNIFICANT EVENT directive in a separate PSECT followed by an EMT 377 in the user specified program section.

Macro call: DECL\$C EFN,PSCT,ERR

Description: This macro generates a DPB for the DECLARE SIGNIFICANT EVENT directive in the \$DPB\$\$ program section. This macro also generates an EMT 377 in the user specified program section, PSCT, with the ERR error service address as described in the DIR\$ macro.

Note: The EFN argument is required to maintain compatibility with RSX-11D. Despite the fact that it must be specified, the DECL\$C macro ignores it.

```
.MACRO DECL$C EFN,PSCT,ERR
.MCALL DECL$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
DECL$ EFN
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM DECL$C
```

C.2.26 DECL\$\$ Macro

The DECL\$\$ macro generates the code to push a DPB for the DECLARE SIGNIFICANT EVENT directive on the stack followed by an EMT 377.

Macro call: DECL\$\$ EFN,ERR

MACRO EXPANSIONS

Description: This macro generates the code to push a DPB for the DECLARE SIGNIFICANT EVENT directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the DECL\$ macro. This macro also generates an EMT 377, using the ERR error service address as described in the DIR\$ macro.

Note: The EFN argument is required to maintain compatibility with RSX-11D. Despite the fact that it must be specified, the DECL\$ macro ignores it.

```
.MACRO DECL$$ EFN,ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 35.,1
DIR$ ,ERR
.ENDM DECL$$
```

C.2.27 DECL\$ Macro

The DECL\$ macro generates a DPB for the DECLARE SIGNIFICANT EVENT directive.

Macro call: DECL\$ EFN

Description: This macro generates a DPB for the DECLARE SIGNIFICANT EVENT directive. DECL\$ ignores the EFN argument but EFN must be present to maintain compatibility with RSX-11D.

```
.MACRO DECL$ EFN
.IF NDF $$$GLB
.BYTE 35.,1
.ENDC
.ENDM DECL$
```

C.2.28 DIR\$ Macro

DIR\$ generates an RSX-11M directive call (EMT 377) with a pre-defined DPB.

Macro call: DIR\$ ADR,ERR

Description: The argument, ADR, must be a valid assembler source operand. It pushes the DPB address on the stack. DIR\$ then generates an EMT 377 to trap to the Executive. The argument, ERR, is optional. If ERR is defined, it must be a valid assembler destination operand to permit the execution of a JUMP TO SUBROUTINE instruction to an error handler if the directive CALL fails.

```
.MACRO DIR$ ADR,ERR
.IF NB <ADR>
MOV ADR,-(SP)
.ENDC
EMT ^O<377>
.IIF NB <ERR>, .MCALL ERR$
.IIF NB <ERR>, ERR$ ERR
.ENDM DIR$
```

MACRO EXPANSIONS

C.2.29 DSAR\$C Macro

The DSAR\$C macro generates the DPB for the DISABLE AST RECOGNITION directive in a separate program section followed by an EMT 377 in the user specified PSECT.

Macro call: DSAR\$C PSCT,ERR

Description: This macro generates a DPB for the DISABLE AST RECOGNITION directive in the PSECT \$DPB\$\$ followed by an EMT 377 in the program section specified in PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO DSAR$C PSCT,ERR
.MCALL DSAR$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
DSAR$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM DSAR$C
```

C.2.30 DSAR\$\$ Macro

The DSAR\$\$ macro generates the code to push a DPB for the DISABLE AST RECOGNITION directive on the stack followed by an EMT 377.

Macro call: DSAR\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the DISABLE AST RECOGNITION directive on the stack. Then the macro generates an EMT 377, using the ERR error service address, described in the DIR\$ macro.

```
.MACRO DSAR$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 99.,1
DIR$ ,ERR
.ENDM DSAR$$
```

C.2.31 DSAR\$ Macro

The DSAR\$ macro generates a DPB for the DISABLE AST RECOGNITION directive.

Macro call: DSAR\$

Description: This macro generates a DPB for the DISABLE AST RECOGNITION directive.

```
.MACRO DSAR$
.IF NDF $$$GLB
.BYTE 99.,1
.ENDC
.ENDM DSAR$
```

MACRO EXPANSIONS

C.2.32 DSCP\$C Macro

The DSCP\$C macro generates a DPB for the DISABLE CHECKPOINTING directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: DSCP\$C PSCT,ERR

Description: This macro generates a DPB for the DISABLE CHECKPOINTING directive in the \$DPB\$\$ program section and an EMT 377 in the user specified PSCT program section. The DIR\$ macro describes the ERR error service address.

```
.MACRO DSCP$C PSCT,ERR
.MCALL DSCP$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
DSCP$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM DSCP$C
```

C.2.33 DSCP\$\$ Macro

The DSCP\$\$ macro generates the code to push a DPB for the DISABLE CHECKPOINTING directive on the stack followed by an EMT 377.

Macro call: DSCP\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the DISABLE CHECKPOINTING directive on the stack. Then an EMT 377 is generated, using the ERR error service address, as described in the DIR\$ macro.

```
.MACRO DSCP$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 95.,1
DIR$ ,ERR
.ENDM DSCP$$
```

C.2.34 DSCP\$ Macro

The DSCP\$ macro generates a DPB for the DISABLE CHECKPOINTING directive.

Macro call: DSCP\$

Description: This macro generates a DPB for the DISABLE CHECKPOINTING directive. There are no arguments.

```
.MACRO DSCP$
.IF NDF $$$GLB
.BYTE 95.,1
.ENDC
.ENDM DSCP$
```

MACRO EXPANSIONS

C.2.35 ENAR\$C Macro

The ENAR\$C macro generates a DPB for the ENABLE AST RECOGNITION directive in a separate PSECT, followed by an EMT 377 in the user specified PSECT.

Macro call: ENAR\$C PSCT,ERR

Description: This macro generates a DPB for the ENABLE AST RECOGNITION directive in the \$DPB\$\$ program section followed by an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO ENAR$C PSCT,ERR
.MCALL ENAR$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ENAR$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM ENAR$C
```

C.2.36 ENAR\$\$ Macro

The ENAR\$\$ macro generates the code to push a DPB for the ENABLE AST RECOGNITION directive on the stack followed by an EMT 377.

Macro call: ENAR\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the ENABLE AST RECOGNITION directive on the stack. Then an EMT 377 is generated using the ERR error service address as described in the DIR\$ macro.

```
.MACRO ENAR$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 101.,1
DIR$ ,ERR
.ENDM ENAR$$
```

C.2.37 ENAR\$ Macro

The ENAR\$ macro generates a DPB for the ENABLE AST RECOGNITION directive.

Macro call: ENAR\$

Description: This macro generates a DPB for the ENABLE AST RECOGNITION directive. There are no parameters.

```
MACRO ENAR$
.IF NDF $$$GLB
.BYTE 101.,1
.ENDC
.ENDM ENAR$
```

MACRO EXPANSIONS

C.2.38 ENCP\$C Macro

The ENCP\$C macro generates a DPB for the ENABLE CHECKPOINTING directive in a separate PSECT followed by an EMT 377 in the user specified program PSECT.

Macro call: ENCP\$C PSCT,ERR

Description: This macro generates a DPB for the ENABLE CHECKPOINTING directive in the \$DPB\$\$ program section. Then the macro generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO ENCP$C PSCT,ERR
.MCALL ENCP$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ENCP$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM ENCP$C
```

C.2.39 ENCP\$\$ Macro

The ENCP\$\$ macro generates the code to push a DPB for the ENABLE CHECKPOINTING directive on the stack followed by an EMT 377.

Macro call: ENCP\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the ENABLE CHECKPOINTING directive on the stack. Then an EMT 377 is generated using the ERR error service address as described in the DIR\$ macro.

```
.MACRO ENCP$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 97.,1
DIR$ ,ERR
.ENDM ENCP$$
```

C.2.40 ENCP\$ Macro

The ENCP\$ macro generates a DPB for the ENABLE CHECKPOINTING directive.

Macro call: ENCP\$

Description: This macro generates a DPB for the ENABLE CHECKPOINTING directive. There are no parameters

```
.MACRO ENCP$
.IF NDF $$$GLB
.BYTE 97.,1
.ENDC
.ENDM ENCP$
```


MACRO EXPANSIONS

C.2.41 ERR\$ Macro

ERR\$ is an internal macro that generates a test for directive failure accompanied by a JSR instruction to an error handler.

Macro call: ERR\$ ERR

Description: A set C-bit indicates directive failure. If the ERR argument is defined, ERR\$ generates code to test the C-bit and skip if not set followed by a JSR to an error handler specified by ERR.

```
.MACRO ERR$ ERR
.IF NB <ERR>
.NLIST
.NTYPE $$$T1,ERR
.LIST
.IIF EQ ^O<$$$T1-27>, BCC .+6
.IIF EQ ^O<$$$T1-37>, BCC .+6
.IF GE ^O<$$$T1-60>
BCC .+6
.IFF
.IF NE ^O<$$$T1-27>
.IIF NE ^O<$$$T1-37>, BCC .+4
.ENDC
.ENDC
CALL ERR
.ENDC
.ENDM ERR$
```

C.2.42 EXIF\$C Macro

THE EXIF\$C macro generates a DPB for the EXIT IF directive in a separate PSECT followed by an EMT 377 in the user specified program PSECT.

Macro call: EXIF\$C EFN,PSCT,ERR

Description: This macro generates a DPB for the EXIT IF directive in the \$DPB\$\$ program section. The DPB parameters are described in the EXIF\$ macro. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The EXIF\$ macro controls symbolic address generation.

```
.MACRO EXIF$C EFN,CS,ERR
.MCALL EXIF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
EXIF$ EFN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM EXIF$C
```

MACRO EXPANSIONS

C.2.43 EXIF\$\$ Macro

The EXIF\$\$ macro generates the code to push a DPB for the EXIT IF directive on the stack followed by an EMT 377.

Macro call: EXIF\$\$ EFN,ERR

Description: This macro generates the code to push a DPB for the EXIT IF directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the EXIF\$ macro. This macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO EXIF$$ EFN,ERR
.MCALL MOV$,DIR$
MOV$ EFN
MOV (PC)+,-(SP)
.BYTE 53.,2
DIR$ ,ERR
.ENDM EXIF$$
```

C.2.44 EXIF\$ Macro

The EXIF\$ macro generates a DPB for the EXIT IF directive.

Macro call: EXIF\$ EFN

Description: This macro generates a DPB for the EXIT IF directive. The argument is assumed to have the following meaning:

EFN=Event flag number.

The following symbol is locally defined with its assigned values equal to the byte offset from the start of the DPB to the DPB element:

E.XFEE-(length 2 bytes) Event flag number.

If the macro is invoked with the symbol \$\$\$GLB defined, EXIF\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO EXIF$ EFN
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 53.,2
.WORD EFN
.ENDC
.IF NDF E.XFEE
.NLIST
OFF$
OFF$ E.XFEE,2
.LIST
.ENDC
.ENDM EXIF$
```

C.2.45 EXIT\$C Macro

The EXIT\$C macro generates a DPB for the TASK EXIT directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: EXIT\$C PSCT,ERR

MACRO EXPANSIONS

Description: This macro generates a DPB in the program SECTION \$DPB\$\$ followed by an EMT 377 in the PSCT program section. The DIR\$ macro describes the ERR error service address.

```
.MACRO EXIT$C PSCT,ERR
.MCALL EXIT$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
EXIT$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM EXIT$C
```

C.2.46 EXIT\$\$ Macro

The EXIT\$\$ macro generates the code to push a DPB for the TASK EXIT directive on the stack followed by an EMT 377.

Macro call: EXIT\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the TASK EXIT directive on the stack. Then, EXIT\$\$ generates an EMT 377 using the ERR error service address, as described in the EXIT\$ macro.

```
.MACRO EXIT$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 51.,1
DIR$
.IIF NB <ERR>, CALL ERR
.ENDM EXIT$$
```

C.2.47 EXIT\$ Macro

The EXIT\$ macro generates a DPB for the TASK EXIT directive.

Macro call: EXIT\$

Description: This macro generates a DPB for the TASK EXIT directive. There are no parameters.

```
.MACRO EXIT$
.IF NDF $$$GLB
.BYTE 51.,1
.ENDC
.ENDM EXIT$
```

C.2.48 EXTRK\$C Macro

The EXTRK\$C macro generates a DPB for the EXTEND TASK directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: EXTRK\$C INC,PSCT,ERR

MACRO EXPANSIONS

Description: This macro generates a DPB for the EXTEND TASK directive in the \$DPB\$\$ program section. The DPB parameters are described in the EXTRK\$ macro. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The EXTRK\$ macro controls the generation of symbolic offsets.

```
.MACRO EXTRK$C INC,CS,ERR
.MCALL EXTRK$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
EXTRK$ INC
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM EXTRK$C
```

C.2.49 EXTRK\$\$ Macro

EXTRK\$\$ pushes a DPB for the EXTEND TASK directive on the stack and generates an EMT 377.

Macro call: EXTRK\$\$ INC,ERR

Description: EXTRK\$\$ pushes a DPB for the EXTEND TASK directive on the stack. The DPB arguments must be valid assembler operands and they must specify the information that the EXTRK\$ macro describes. EXTRK\$\$ also generates an EMT 377 using the ERR error service address that the DIR\$ macro describes.

```
.MACRO EXTRK$$ INC,ERR
.MCALL MOV$,DIR$
CLR -(SP)
MOV$ INC
MOV (PC)+,-(SP)
.BYTE 89.,3
DIR$ ,ERR
.ENDM EXTRK$$
```

C.2.50 EXTRK\$ Macro

The EXTRK\$ macro generates a DPB for the EXTEND TASK directive.

Macro call: EXTRK\$ INC

Description: EXTRK\$ generates a DPB for the EXTEND TASK directive. The argument must have the following meaning:

INC=Task size increment in 32 word blocks.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

E.XTIN-(length 2 bytes) Task size increment.

MACRO EXPANSIONS

If EXTK\$ is invoked with the \$\$\$GLB symbol defined, EXTK\$ does not generate the DPB and it globally defines symbolic offsets.

```
.MACRO EXTK$ INC
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 89.,3
.WORD INC
.WORD 0
.ENDC
.IF NDF E.XTIN
.NLIST
OFF$
OFF$ E.XTIN,2
.LIST
.ENDC
.ENDM EXTK$
```

C.2.51 GLUN\$C Macro

GLUN\$C generates a DPB for the GET LUN INFORMATION directive in a separate PSECT followed by an EMT 377 in the user specified program PSECT.

Macro call: GLUN\$C LUN,BUFADR,PSCT,ERR

Description: GLUN\$C generates a DPB for the GET LUN INFORMATION directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSCT. GLUN\$ describes the DPB parameters and controls symbolic address generation. The DIR\$ macro describes the ERR error service address.

```
.MACRO GLUN$C LUN,BUFA,CS,ERR
.MCALL GLUN$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GLUN$ LUN,BUFA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM GLUN$C
```

C.2.52 GLUN\$\$ Macro

GLUN\$\$ generates the code to push a DPB for the GET LUN INFORMATION directive on the stack and generates an EMT 377.

Macro call: GLUN\$\$ LUN,BUFADR,ERR

Description: GLUN\$\$ generates the code to push a DPB for the GET LUN INFORMATION directive on the stack and generates an EMT 377 by using the ERR error service address as described in the DIR\$ macro. GLUN\$\$ also generates symbolic offsets relative to the information buffer as described in the GLUN\$ macro. DPB arguments must be valid assembler source operands and must specify the information described in the GLUN\$ macro.

MACRO EXPANSIONS

```
.MACRO GLUN$$ LUN,BUFA,ERR
.MCALL MOV$,DIR$,OFF$
MOV$ BUFA
MOV$ LUN
MOV (PC)+,-(SP)
.BYTE 5,3
DIR$ ,ERR
.IF NDF G.LUNA
.NLIST
$$$OST=0
.IRP X,<<G.LUNA,2>,<G.LUNU,1>,<G.LUFB,1>,<G.LUCW,8.>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM GLUN$$
```

C.2.53 GLUN\$ Macro

The GLUN\$ macro generates a DPB for the GET LUN INFORMATION directive.

Macro call: GLUN\$ LUN,BUFADR

Description: This macro generates a DPB for the GET LUN INFORMATION directive. The arguments are assumed to have the following meaning:

LUN=Logical unit number,
BUFADR=Address of six-word buffer.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

G.LULU-(length 2 bytes) Logical unit number,
G.LUBA-(2) Buffer address,
G.LUBL-(2) Buffer length.

The following symbols are assigned relative to the start of the LUN information buffer:

G.LUNA-(2) Device name,
G.LUNU-(1) Device unit number,
G.LUFB-(1) Flags byte,
G.LUCW-(8) Four device characteristic words.

If the macro is invoked with the \$\$\$GLB symbol defined, GLUN\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO GLUN$ LUN,BUFA
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 5,3
.WORD LUN
.WORD BUFA
.ENDC
.IF NDF G.LULU
.NLIST
.IRP X,<,<G.LULU,2>,<G.LUBA,2>,<G.LUBL,2>>
OFF$ X
.ENDM
.IF NDF G.LUNA
$$$OST=0
```

MACRO EXPANSIONS

```
.IRP X,<<G.LUNA,2>,<G.LUNU,1>,<G.LUFB,1>,<G.LUCW,8.>>
OFF$ X
.ENDM
.ENDC
.LIST
.ENDC
.ENDM GLUN$
```

C.2.54 GMCR\$C Macro

GMCR\$C generates a DPB for the GET MCR COMMAND LINE directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: GMCR\$C PSCT,ERR

Description: GMCR\$C generates a DPB for the GET MCR COMMAND LINE directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSCT. GMCR\$ describes the DPB parameters and controls symbolic address generation. DIR\$ describes the ERR error service address.

```
.MACRO GMCR$C CS,ERR
.MCALL GMCR$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GMCR$
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM GMCR$C
```

C.2.55 GMCR\$ Macro

The GMCR\$ macro generates a DPB for the GET MCR COMMAND LINE directive.

Macro call: GMCR\$

Description: This macro generates a DPB for the GET MCR COMMAND LINE directive. GMCR\$ locally defines the following symbol with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

G.MCRB-(length 80 bytes) MCR line buffer

If the macro is invoked with the \$\$\$GLB symbol defined, GMCR\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO GMCR$
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 127.,41.
.BLKW 40.
.ENDC
.IF NDF G.MCRB
.NLIST
OFF$
```

MACRO EXPANSIONS

```
OFF$ G.MCRB,80.  
.LIST  
.ENDC  
.ENDM GPCR$
```

C.2.56 GPRT\$C Macro

GPRT\$C generates a DPB for the GET PARTITION PARAMETERS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: GPRT\$C PRTNAM,BUFADR,PSCT,ERR

Description: GPRT\$C generates a DPB for the GET PARTITION PARAMETERS directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSECT. GPRT\$ describes the DPB parameters and controls symbolic offset generation. DIR\$ describes the ERR error service address.

```
.MACRO GPRT$C PRT,BUF,CS,ERR  
.MCALL GPRT$,DIR$  
.IF NDF $$$GLB  
.PSECT $DPB$$  
$$$=  
.IFTF  
GPRT$ PRT,BUF  
.IFT  
.PSECT CS  
DIR$ #$$$ ,ERR  
.ENDC  
.ENDM GPRT$C
```

C.2.57 GPRT\$\$ Macro

GPRT\$\$ generates the code to push a DPB for the GET PARTITION PARAMETERS directive on the stack and generates an EMT 377.

Macro call: GPRT\$\$ PRTNAMADR,BUFADR,ERR

Description: GPRT\$\$ generates the code to push a DPB for the GET PARTITION PARAMETERS directive on the stack and generates an EMT 377 using the ERR error service address described in the DIR\$ macro. The DPB arguments must be valid assembler source operands and they must specify the information described in the GPRT\$ macro. GPRT\$\$ also generates symbolic offsets relative to the start of the partition parameters buffer described in the GPRT\$ macro.

```
.MACRO GPRT$$ PRT,BUF,ERR  
.MCALL MOV$,RFA$,DIR$,OFF$  
MOV$ BUF  
RFA$ PRT  
MOV (PC)+,-(SP)  
.BYTE 65.,4  
DIR$ ,ERR  
.IF NDF G.PRPB  
.NLIST  
$$$OST=0  
.IRP X,<<G.PRPB,2>,<G.PRPS,2>,<G.PRFW,2>>  
OFF$ X  
.ENDM  
.LIST
```


MACRO EXPANSIONS

```
.ENDC  
.ENDM GPRT$
```

C.2.58 GPRT\$ Macro

The GPRT\$ macro generates a DPB for the GET PARTITION PARAMETERS directive.

Macro call: GPRT\$ PRTNAM, BUF

Description: This macro generates a DPB for the GET PARTITION PARAMETERS directive. The arguments are assumed to have the following meanings:

PRTNAM=Partition name,
BUFADR=Address of three word buffer.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

G.PRPN-(length 4 bytes) Partition name,
G.PRBA-(2) Buffer address.

The following offsets are assigned relative to the start of the partition parameters buffer:

G.PRPB-(2) 1/64 Partition base address,
G.PRPS-(2) 1/64 Partition size,
G.PRFW-(2) Partition flags word.

If GPRT\$ is invoked with the \$\$\$GLB symbol defined, GPRT\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO GPRT$ PRT, BUF  
.MCALL OFF$, R50$  
.IF NDF $$$GLB  
.BYTE 65., 4  
R50$ PRT  
.WORD BUF  
.ENDC  
.IF NDF G.PRPN  
.NLIST  
OFF$  
OFF$ G.PRPN, 4  
OFF$ G.PRBA, 2  
.IF NDF G.PRPB  
$$$OST=0  
.IRP X, <<G.PRPB, 2>, <G.PRPS, 2>, <G.PRFW, 2>>  
OFF$ X  
.ENDM  
.ENDC  
.LIST  
.ENDC  
.ENDM GPRT$
```

MACRO EXPANSIONS

C.2.59 GSSW\$C Macro

The GSSW\$C macro generates a DPB for the GET SENSE SWITCHES directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: GSSW\$C PSCT,ERR

Description: This macro generates a DPB for the GET SENSE SWITCHES directive in the \$DPB\$\$ program section followed by an EMT 377 in the user specified program section PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO GSSW$C PSCT,ERR
.MCALL GSSW$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GSSW$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM GSSW$C
```

C.2.60 GSSW\$\$ Macro

The GSSW\$\$ macro generates the code to push a DPB for the GET SENSE SWITCHES directive on the stack followed by an EMT 377.

Macro call: GSSW\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the GET SENSE SWITCHES directive on the stack. Then it generates an EMT 377 using the error service address as described in the DIR\$ macro.

```
.MACRO GSSW$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 125.,1
DIR$ ,ERR
.ENDM GSSW$$
```

C.2.61 GSSW\$ Macro

The GSSW\$ macro generates a DPB for the GET SENSE SWITCHES directive.

Macro call: GSSW\$

Description: This macro generates a DPB for the GET SENSE SWITCHES directive. There are no arguments.

```
.MACRO GSSW$
.IF NDF $$$GLB
.BYTE 125.,1
.ENDC
.ENDM GSSW$
```

MACRO EXPANSIONS

C.2.62 GTIM\$C Macro

The GTIM\$C macro generates a DPB for the GET TIME PARAMETERS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: GTIM\$C BUFADR,PSCT,ERR

Description: This macro generates a DPB for the GET TIME PARAMETERS directive in the \$DPB\$\$ program section. The GTIM\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The GTIM\$ macro controls symbolic address generation.

```
.MACRO GTIM$C BUFA,CS ERR
.MCALL GTIM$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GTIM$ BUFA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM GTIM$C
```

C.2.63 GTIM\$\$ Macro

The GTIM\$\$ macro generates the code to push a DPB for the GET TIME PARAMETERS directive on the stack followed by an EMT 377.

Macro call: GTIM\$\$ BUFADR,ERR

Description: This macro generates the code to push a DPB for the GET TIME PARAMETERS directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the GTIM\$ macro. This macro also generates an EMT 377 using the ERR error service address, as described in the DIR\$ macro. This macro also generates symbolic offsets relative to the time parameters buffer as described in the GTIM\$ macro.

```
.MACRO GTIM$$ BUFA,ERR
.MCALL MOV$,DIR$,OFF$
MOV$ BUFA
MOV (PC)+,-(SP)
.BYTE 61.,2
DIR$ ,ERR
.IF NDF G.TIYR
.NLIST
$$$OST=0
.IRP X,<<G.TIYR,2>>,<G.TIMO,2>,<G.TIDA,2>,<G.TIHR,2>>
OFF$ X
.ENDM
.IRP X,<<G.TIMI,2>>,<G.TISC,2>,<G.TICT,2>,<G.TICP,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM GTIM$$
```

MACRO EXPANSIONS

C.2.64 GTIM\$ Macro

The GTIM\$ macro generates a DPB for the GET TIME PARAMETERS directive.

Macro call: GTIM\$ BUFADR

Description: This macro generates a DPB for the GET TIME PARAMETERS directive. The argument is assumed to have the following meaning:

BUFADR=Address of eight word buffer.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the element:

G.TIBA-(length 2 bytes) Buffer address.

The following offsets are assigned relative to the start of the time parameters buffer:

G.TIYR-(2) Year,
G.TIMO-(2) Month,
G.TIDA-(2) Day,
G.TIHR-(2) Hour,
G.TIMI-(2) Minute,
G.TISC-(2) Second,
G.TICT-(2) Clock tick,
G.TICP-(2) Clock ticks per second.

If GTIM\$ is invoked with the \$\$\$GLB symbol defined, GTIM\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO GTIM$ BUFA
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 61.,2
.WORD BUFA
.ENDC
.IF NDF G.TIBA
.NLIST
OFF$
OFF$ G.TIBA,2
.IF NDF G.TIYR
$$$OST=0
.IRP X,<<G.TIYR,2>>,<G.TIMO,2>>,<G.TIDA,2>>,<G.TIHR,2>>
OFF$ X
.ENDM
.IRP X,<<G.TIMI,2>>,<G.TISC,2>>,<G.TICT,2>>,<G.TICP,2>>
OFF$ X
.ENDM
.ENDC
.LIST
.ENDC
.ENDM GTIM$
```

C.2.65 GTSK\$C Macro

The GTSK\$C macro generates a DPB for the GET TASK PARAMETERS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: GTSK\$C BUFADR,PSCT,ERR

MACRO EXPANSIONS

Description: This macro generates a DPB for the GET TASK PARAMETERS directive in the \$DPB\$\$ program section. The GTSK\$\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the original program control section named in PSCT. The DIR\$ macro describes the ERR error service address. The GTSK\$ macro controls symbolic offset generation.

```
.MACRO GTSK$C BUFA,CS,ERR
.MCALL GTSK$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFT
GTSK$ BUFA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM GTSK$C
```

C.2.66 GTSK\$\$ Macro

The GTSK\$\$ macro generates the code to push a DPB for the GET TASK PARAMETERS directive on the stack followed by an EMT 377.

Macro call: GTSK\$\$ BUFADR,ERR

Description: GTSK\$\$ generates the code to push a DPB for the GET TASK PARAMETERS directive on the stack. The DPB arguments must be valid assembler source operands and they must specify the information described in the GTSK\$ macro. GTSK\$\$ generates an EMT 377 using the ERR error service address, as described in the DIR\$ macro. GTSK\$\$ also generates symbolic offsets relative to the task parameters buffer as described in the GTSK\$ macro.

```
.MACRO GTSK$$ BUFA,ERR
.MCALL MOV$,DIR$,OFF$
MOV$ BUFA
MOV (PC)+,-(SP)
.BYTE 63.,2
DIR$ ,ERR
.IF NDF G.TSTN
.NLIST
$$$OST=0
.IRP X,<<G.TSTN,4>,<G.TSPN,4>,<G.TSRN,4>,<G.TSPR,2>,<G.TSPC,1>>
OFF$ X
.ENDM
.IRP X,<<G.TSGC,1>,<G.TSNL,2>,<G.TSMT,2>,<G.TSFW,2>,<G.TSVA,2>>
VL,2
OFF$ G.TSTS,2
OFF$ G.TSSY,2
OFF$ G.TSDU,2
$$$OST=0
.IRP X,<<RX$11D,1>,<RX$11M,1>,<RX$11S,1>,<RX$IAS,1>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM GTSK$$
```

MACRO EXPANSIONS

C.2.67 GTSK\$ Macro

The GTSK\$ macro generates a DPB for the GET TASK PARAMETERS directive.

Macro call: GTSK\$ BUFADR

Description: GTSK\$ generates a DPB for the GET TASK PARAMETERS directive. The argument is assumed to have the following meaning:

BUFADR=Address of a 16 word buffer.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

G.TSBA-(length 2 bytes) Buffer address.

The following offsets are assigned relative to the start of the task parameters buffer:

G.TSTN-(4) Task name
G.TSPN-(4) Partition name
G.TSRN-(4) Name of task's requester
G.TSPR-(2) Priority
G.TSGC-(1) UIC group code
G.TSPC-(1) UIC programmer code
G.TSNL-(2) Number of logical units
G.TSMT-(2) Machine type
G.TSFW-(2) Std flags word
G.TSVA-(2) Address of task SST vector table
G.TSVL-(2) Length (number of words) of task SST vector table
G.TSTS-(2) Size (in bytes) of task address space
G.TSSY-(2) System in which task is running:
 RX\$11D - System is RSX-11D
 RX\$11M - System is RSX-11M
 RX\$11S - System is RSX-11S
 RX\$IAS - System is IAS
G.TSDU-(2) Task's (default) protection UIC

If GTSK\$ is invoked with the \$\$\$GLB symbol defined, GTSK\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO GTSK$ BUFA
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 63.,2
.WORD BUFA
.ENDC
.IF NDF G.TSBA
.NLIST
OFF$
OFF$ G.TSBA,2
.IF NDF G.TSTN
$$$OST=0
.IRP X,<<G.TSTN,4>,<G.TSPN,4>,<G.TSRN,4>,<G.TSPR,2>,<G.TSPC,1>>
OFF$ X
.ENDM
.IRP X,<<G.TSGC,1>,<G.TSNL,2>,<G.TSMT,2>,<G.TSFW,2>,<G.TSVA,2>>
OFF$ X
.ENDM
OFF$ G.TSVL,2
OFF$ G.TSTS,2
OFF$ G.TSSY,2
OFF$ G.TSDU,2
$$$OST=0
```

MACRO EXPANSIONS

```
.IRP X,<<RX$11D,1>,<RX$11M,1>,<RX$11S,1>,<RX$IAS,1>>  
OFF$ X  
.ENDM  
.ENDC  
.LIST  
.ENDC  
.ENDM GTSK$
```

C.2.68 IHAR\$C Macro

The IHAR\$C macro generates a DPB for the INHIBIT DISABLE AST RECOGNITION directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: IHAR\$C PSCT,ERR

Description: This macro generates a DPB for the INHIBIT DISABLE AST RECOGNITION directive in the \$DPB\$\$ program section followed by an EMT 377 in the user specified program section named PSCT. The DIR\$ macro describes the ERR error service address.

Note: This macro is the same as the DSAR\$C macro, and is included only for compatibility with RSX-11D.

```
.MACRO IHAR$C PSCT,ERR  
.MCALL DSAR$C  
DSAR$C PSCT,ERR  
.ENDM IHAR$C
```

C.2.69 IHAR\$\$ Macro

The IHAR\$\$ macro generates the code to push a DPB for the INHIBIT AST RECOGNITION directive on the stack and an EMT 377.

Macro call: IHAR\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the DISABLE AST RECOGNITION directive on the stack. Then the macro generates an EMT 377 using the ERR error service address described in the DIR\$ macro.

Note: This macro is included only for compatibility with RSX-11D.

```
.MACRO IHAR$$ ERR  
.MCALL DSAR$$  
DSAR$$ ERR  
.ENDM IHAR$$
```

C.2.70 IHAR\$ Macro

The IHAR\$ macro generates a DPB for the INHIBIT DISABLE AST RECOGNITION directive.

Macro call: IHAR\$

Description: This macro generates a DPB for the INHIBIT DISABLE AST RECOGNITION directive. There are no parameters.

MACRO EXPANSIONS

Note: This macro is included only for compatibility with RSX-11D.

```
.MACRO IHAR$
.MCALL DSAR$
DSAR$
.ENDM IHAR$
```

C.2.71 MOV\$ Macro

This is an internal macro that pushes an argument on the stack.

Macro call: MOV\$ <ARG>

Description: MOV\$ pushes an argument on the stack. MOV\$ examines the argument first to determine which code is valid to perform the push. If the argument is left blank, MOV\$ uses a CLR instruction. Otherwise, MOV\$ generates a "MOV ARG,-(SP)" instruction.

```
.MACRO MOV$ ARG
.IF NB <ARG>
.IF DIF <ARG>,<#0>
MOV ARG,-(SP)
.MEXIT
.ENDC
.ENDC
CLR -(SP)
.ENDM MOV$
```

C.2.72 MRKT\$C Macro

The MRKT\$C macro generates a DPB for the MARK TIME directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: MRKT\$C EFN,TIMMAG,TIMUNIT,AST,PSCT,ERR

Description: This macro generates a DPB for the MARK TIME directive in the \$DPB\$\$ program section. The MRKT\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The MRKT\$ macro controls the generation of symbolic offsets.

```
.MACRO MRKT$C EF, TM, TU, AST, CS, ERR
.MCALL MRKT$, DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
MRKT$ EF, TM, TU, AST
.IFT
.PSECT CS
DIR$ $$$$, ERR
.ENDC
.ENDM MRKT$C
```


MACRO EXPANSIONS

C.2.73 MRKT\$\$ Macro

The MRKT\$\$ macro generates the code to push a DPB for the MARK TIME directive on the stack followed by an EMT 377.

Macro call: MRKT\$\$ EFN,TIMMAG,TIMUNIT,AST,ERR

Description: This macro generates the code to push a DPB for the MARK TIME directive on the stack. The DPB arguments must be valid assembler source operands and they must specify the information described in the MRKT\$ macro. This macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO MRKT$$ EF, TM, TU, AST, ERR
.MCALL RVP$, DIR$
RVP$ EF, TM, TU, AST
MOV (PC)+, -(SP)
.BYTE 23., 5
DIR$, ERR
.ENDM MRKT$$
```

C.2.74 MRKT\$ Macro

The MRKT\$ macro generates a DPB for the MARK TIME DIRECTIVE.

Macro call: MRKT\$ EFN,TIMMAG,TIMUNIT,AST

Description: This macro generates a DPB for the MARK TIME directive. The arguments are assumed to have the following meanings:

```
EFN=Event flag number,
TIMMAG=Time interval magnitude,
TIMUNIT=Time interval unit,
AST=AST entry address.
```

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

```
M.KTEF-(length 2 bytes) Event flag,
M.KTMG-(2) Time magnitude,
M.KTUN-(2) Time unit,
M.KTAE-(2) AST entry address.
```

If MRKT\$ is invoked with the \$\$\$GLB symbol defined, MRKT\$ does not generate the DPB and globally defines the symbolic offsets.

```
.MACRO MRKT$ EFN, TM, TU, AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 23., 5
.WORD EFN
.WORD TM
.WORD TU
.WORD AST
.ENDC
.IF NDF M.KTEF
.NLIST
.IRP X, <, <M.KTEF, 2>, <M.KTMG, 2>, <M.KTUN, 2>, <M.KTAE, 2>>
OFF$ X
.ENDM
.LIST
```

MACRO EXPANSIONS

```
.ENDC
.ENDM MRKT$
```

C.2.75 MVB\$ Macro

MVB\$ is an internal macro that pushes two bytes on the stack.

Macro call: MVB\$ LOWB,HIGHB

Description: MVB\$ examines the two arguments and based on that examination generates the code to push the bytes on the stack.

```
.MACRO MVB$ LOW,HIGH
  .IF B <LOW>
  .IF B <HIGH>
  CLR -(SP)
  .IFF
  CLRB -(SP)
  MOVB HIGH,1(SP)
  .ENDC
  .IFF
  .IF B <HIGH>
  CLR -(SP)
  MOVB LOW,(SP)
  .IFF
  MOVB LOW,-(SP)
  MOVB HIGH,1(SP)
  .ENDC
  .ENDC
.ENDM MVB$
```

C.2.76 OFF\$ Macro

OFF\$ is an internal macro that generates symbolic offsets.

Macro call: OFF\$ SYMB,LEN

Description: OFF\$ creates tables of symbolic offsets for the elements within DPBs. The first argument, SYMB, is the name of the symbol. The second argument, LEN, is the length of the symbol in bytes. OFF\$ adds LEN to the offset counter after OFF\$ makes the symbolic assignment. If the \$\$\$GLB symbol is defined, OFF\$ makes the symbolic assignments global. Otherwise, OFF\$ makes all assignments local.

If both SYMB and LEN are blank, OFF\$ initializes the offset counter to two (2).

```
.MACRO OFF$ SYMB,LEN
  .IF B <'SYMB'LEN>
  $$$OST=2
  .MEXIT
  .ENDC
  .IF NB <SYMB>
  .IF NDF $$$GLB
  SYMB=$$$OST
  .IFF
  SYMB==$$$OST
  .ENDC
  .ENDC
```

MACRO EXPANSIONS

C.2.77 QDPB\$\$ Macro

The QDPB\$\$ macro generates the code to push a QUEUE I/O or QUEUE I/O AND WAIT DPB on the stack and it generates an EMT 377.

Macro call: QDPB\$\$ DIC,FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR

Description: This macro generates the code to push the proper QUEUE I/O DPB on the stack. It is meant to be called only by QIO\$\$, or QIOW\$\$ to generate the required code to build a DPB and issue the EMT 377.

```
.MACRO QDPB$$ DIC,FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR
.MCALL RVP$,MVB$,DIR$
.NLIST
$$$ARG=0
.IRP X,<PRMLST>
$$$ARG=$$$ARG+1
.ENDM
.LIST
.IF GT 6-$$$ARG
.REPT <6-$$$ARG>
CLR -(SP)
.ENDR
.ENDC
.IIF NB <PRMLST>, RVP$ PRMLST
RVP$ IOST,AST
MVB$ EFN,
RVP$ FNC,LUN
MOV (PC)+,-(SP)
.BYTE DIC,12.
DIR$ ,ERR
.ENDM QDPB$$
```

C.2.78 QDPB\$ Macro

The QDPB\$ macro generates the QUEUE I/O or QUEUE I/O AND WAIT DPB.

Macro call: QDPB\$ DIC,FNC,LUN,EFN,PRI,IOST,AST,PRMLST

Description: This macro generates the required DPB for the specified QUEUE I/O directive. It is only meant to be called by the QIO\$, QIO\$, QIOW\$, OR QIOW\$C macros to generate the required DPB.

```
.MACRO QDPB$ DIC,FNC,LUN,EFN,PRI,IOST,AST,PRMLST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE DIC,12.
.WORD FNC
.WORD LUN
.BYTE EFN,0
.WORD IOST
.WORD AST
.NLIST
$$$ARG=0
.LIST
.IRP X,<PRMLST>
.WORD X
.NLIST
$$$ARG=$$$ARG+1
.LIST
.ENDM
```

MACRO EXPANSIONS

```
.IF GT 6-$$$ARG
.REPT <6-$$$ARG>
.WORD 0
.ENDR
.ENDC
.ENDC
.IF NDF Q.IOFN
.NLIST
.IRP X,<,<Q.IOFN,2>,<Q.IOLU,2>,<Q.IOEF,1>,<Q.IOPR,1>>
OFF$ X
.ENDM
.IRP X,<<Q.IOSB,2>,<Q.IOAE,2>,<Q.IOPL>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM QDPB$
```

C.2.79 QIO\$C Macro

The QIO\$C macro generates a QUEUE I/O DPB in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: QIO\$C FNC,LUN,EFN,PRI,IOST,AST,PRMLST,PSECT,ERR

Description: This macro generates the required QUEUE I/O DPB in the \$DPB\$\$ program section. All of the arguments through PRMLST are as described for the QIO\$ macro. The PSECT argument should be the name of the original program program section into which the standard monitor trap code is generated. The optional argument, ERR, is the error routine address. This macro invokes QIO\$ and, therefore, follows the same expansion guidelines.

Note: Read the note in the QIO\$ macro description concerning ignored arguments in the QIO\$ macro below.

```
.MACRO QIO$C FNC,LUN,EFN,PRI,IOST,AST,PRMLST,PSCT,ERR
.MCALL QDPB$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
QDPB$ 1,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM QIO$C
```

C.2.80 QIO\$\$ Macro

The QIO\$\$ macro generates the code to push a QUEUE I/O DPB on the stack and generate an EMT 377.

Macro call: QIO\$\$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR

Description: This macro generates the code to push the required queue I/O DPB on the stack. The arguments through PRMLST represent the information as described in the macro QIO\$. However, the arguments must all be valid assembler source operands.

MACRO EXPANSIONS

Note: Read the note in the QIO\$ macro description concerning ignored arguments in the QIO\$ macro below.

```
.MACRO QIO$$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR
.MCALL QDPB$$
QDPB$$ 1,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>,ERR
.ENDM QIO$$
```

C.2.81 QIO\$ Macro

The QIO\$ macro generates the QUEUE I/O DPB.

Macro call: QIO\$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST

Description: This macro generates the required DPB specified for the QUEUE I/O directive. The arguments are treated as follows:

```
FNC-I/O Function code,
LUN-Logical unit number,
EFN-Event flag number,
PRI-Priority,
IOST-Address of I/O status block,
AST-Address of I/O done AST entry point
PRMLST-Parameter list of the form <P1,...,P6>
```

All of the arguments must be valid expressions to be used in assembler data storage directives. The first call of this macro defines the following symbols and assigns, as a value, their byte offset from the beginning of the DPB:

```
Q.IOFN-(length 2 bytes) I/O function,
Q.IOLU-(2) Logical unit number,
Q.IOEF-(1) Event flag number,
Q.IOPR-(1) Priority,
Q.IOSB-(2) Address of I/O status block,
Q.IOAE-(2) Address of I/O done AST entry point,
Q.IOPL-(0) Parameter list (up to 6 words).
```

If the \$\$\$GLB symbol is defined, QIO\$ does not generate the DPB and it globally defines the symbolic offsets.

Note: The argument PRI is required to maintain compatibility with RSX-11D. Despite the fact that it must be specified, the QIO\$ macro ignores it.

```
.MACRO QIO$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST
.MCALL QDPB$
QDPB$ 1,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>
.ENDM QIO$
```

C.2.82 QIOW\$C Macro

The QIOW\$C macro generates a QUEUE I/O AND WAIT DPB in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: QIOW\$C FNC,LUN,EFN,PRI,IOST,AST,PRMLST,PSECT,ERR

Description: This macro generates the required QUEUE I/O AND WAIT DPB in the program section named \$DPB\$\$\$. All of the arguments through

MACRO EXPANSIONS

PRMLST are as described for the QIOW\$ macro. The argument PSECT should be the name of the original program program section into which the standard monitor trap code is generated. The optional argument, ERR, is the error routine address.

Note: Read the note concerning ignored arguments in the QIOW\$ macro below.

```
.MACRO QIOW$C FNC,LUN,EFN,PRI,IOST,AST,PRMLST,PSCT,ERR
.MCALL QDPB$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
QDPB$ 3,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM QIOW$C
```

C.2.83 QIOW\$\$ Macro

The QIOW\$\$ macro generates the code to push a queue I/O and wait DPB on the stack and generate an EMT 377.

Macro call: QIOW\$\$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR QIOW\$\$ Macro

Description: This macro generates the code to push the required QUEUE I/O AND WAIT DPB on the stack. The arguments through PRMLST represent the information as described in the QIOW\$ macro. However, the arguments must all be valid assembler source operands.

Note: Read the note concerning ignored arguments in the QIOW\$ macro below.

```
.MACRO QIOW$$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST,ERR
.MCALL QDPB$$
QDPB$$ 3,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>,ERR
.ENDM QIOW$$
```

C.2.84 QIOW\$ Macro

The QIOW\$ macro generates the QUEUE I/O AND WAIT DPB.

Macro call: QIOW\$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST

Description: This macro generates the required DPB specified for the QUEUE I/O AND WAIT directive. The arguments are treated as follows:

```
FNC-I/O function code,
LUN-Logical unit number,
EFN-Event flag number,
PRI-Priority,
IOST-Address of I/O status block,
AST-Address of I/O done AST entry point
PRMLST-Parameter list of the form <P1,...,P6> .
```

All of the arguments must be valid expressions to be used in assembler data storage directives. The first call of this macro defines the

MACRO EXPANSIONS

following symbols and assigns, as values, their byte offset from the beginning of the DPB:

```
Q.IOFN-(Length 2 bytes) I/O function,  
Q.IOLU-(2) Logical unit number,  
Q.IOEF-(1) Event flag number,  
Q.IOPR-(1) Priority,  
Q.IOSB-(2) Address of I/O status block,  
Q.IOAE-(2) Address of I/O done AST entry point,  
Q.IOPL-(0) Parameter list (up to 6 words).
```

If the \$\$\$GLB symbol is defined, QIOW\$ does not generate the DPB and it globally defines the symbolic offsets.

Note: This macro requires the PRI argument to maintain compatibility with RSX-11D. Despite the fact that it must be specified, QIOW\$ ignores it.

```
.MACRO QIOW$ FNC,LUN,EFN,PRI,IOST,AST,PRMLST  
.MCALL QDPB$  
QDPB$ 3,FNC,LUN,EFN,PRI,IOST,AST,<PRMLST>  
.ENDM QIOW$
```

C.2.85 R50\$ Macro

This internal macro generates a two-word RADIX-50 name.

Macro call: R50\$ NAME

Description: This macro converts the NAME argument into RADIX-50 using the .RAD50 assembler directive. Furthermore, R50\$ ensures that the data generated is at least two words long.

```
.MACRO R50$ NAME  
.NLIST  
.NCHR $$$T1,NAME  
.LIST  
.IF EQ $$$T1  
.WORD 0,0  
.IFF  
.IF GT $$$T1-6  
$$$T4=,+4  
.IFTF  
.RAD50 /NAME/  
.IFT  
.=$$$T4  
.ENDC  
.IF LT $$$T1-4, .WORD 0  
.ENDC  
.ENDM R50$
```

C.2.86 RCVD\$C Macro

RCVD\$C generates a DPB for the RECEIVE DATA directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RCVD\$C TASK,BUFADR,PSCT,ERR

Description: RCVD\$C generates a DPB for the RECEIVE DATA directive in the \$DPB\$\$ program section. RCVD\$ describes the DPB parameters.

MACRO EXPANSIONS

RCVD\$C then generates an EMT 377 in the program section named in PSCT. DIR\$ describes the ERR error service address. RCVD\$ controls the generation of symbolic offsets.

```
.MACRO RCVD$C TN,BA,CS,ERR
.MCALL RCVD$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
RCVD$ TN,BA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RCVD$C
```

C.2.87 RCVD\$\$ Macro

The RCVD\$\$ macro generates the code to push a DPB for the RECEIVE DATA directive on the stack followed by an EMT 377.

Macro call: RCVD\$\$ TSKNAMADR,BUFADR,ERR

Description: This macro generates the code to push a DPB for the RECEIVE DATA directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the RCVD\$ macro. This macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO RCVD$$ TN,BA,ERR
.MCALL MOV$,DIR$,RFA$
MOV$ BA
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 75.,4
DIR$ ,ERR
.ENDM RCVD$$
```

C.2.88 RCVD\$ Macro

The RCVD\$ macro generates a DPB for the RECEIVE DATA directive.

Macro call: RCVD\$ TASK,BUFADR

Description: This macro generates a DPB for the RECEIVE DATA directive. The arguments are assumed to have the following meanings:

TASK=Sender task name,
BUFADR=Address of fifteen word buffer.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

R.VDTN-(length 4 bytes) Task name,
R.VDBA-(2) Buffer address.

If RCVD\$ is invoked with the \$\$\$GLB symbol defined, RCVD\$ does not generate the DPB and it globally defines the symbolic offsets.

MACRO EXPANSIONS

```
.MACRO RCVDS TN,BA
.MCALL OFF$, R50$
.IF NDF $$$GLB
.BYTE 75.,4
R50$ TN
.WORD BA
.ENDC
.IF NDF R.VDTN
.NLIST
.IRP X,<,<R.VDTN,4>,<R.VDBA,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM RCVDS
```

C.2.89 RCVX\$C Macro

The RCVX\$C macro generates a DPB for the RECEIVE DATA OR EXIT directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RCVX\$C TASK,BUFADR,PSCT,ERR

Description: This macro generates a DPB for the RECEIVE DATA OR EXIT DIRECTIVE in the program section named \$DPB\$\$\$. The RCVX\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The ERR error service address is described in the DIR\$ macro. The RCVX\$ macro controls the generation of symbolic offsets.

```
.MACRO RCVX$C TN,BA,CS,ERR
.MCALL RCVX$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$$
$$$=.
.IFTF
RCVX$ TN,BA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RCVX$C
```

C.2.90 RCVX\$\$S Macro

The RCVX\$\$S macro generates the code to push a DPB for the RECEIVE DATA OR EXIT directive on the stack followed by an EMT 377.

Macro call: RCVX\$\$S TSKNAMADR,BUFADR,ERR

Description: This macro generates the code to push a DPB for the RECEIVE DATA OR EXIT directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the RCVX\$ macro. This macro also generates an EMT 377 using the ERR error service address as defined in the DIR\$ macro.

```
.MACRO RCVX$$S TN,BA,ERR
.MCALL MOV$,DIR$,RFA$
```

MACRO EXPANSIONS

```
MOV$ BA
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 77.,4
DIR$ ,ERR
.ENDM RCVX$$
```

C.2.91 RCVX\$ Macro

The RCVX\$ macro generates a DPB for the RECEIVE DATA OR EXIT directive.

Macro call: RCVX\$ TASK,BUFADR

Description: This macro generates a DPB for the RECEIVE DATA OR EXIT directive. The arguments are assumed to have the following meanings:

TASK=Sender task name,
BUFADR=Address of fifteen word buffer.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

R.VXTN-(length 4 bytes) Task name,
R.VXBA-(2) Buffer address.

If RCVX\$ is invoked with the \$\$\$GLB symbol defined, RCVX\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO RCVX$ TN,BA
.MCALL OFF$, R50$
.IF NDF $$$GLB
.BYTE 77.,4
R50$ TN
.WORD BA
.ENDC
.IF NDF R.VXTN
.NLIST
.IRP X,<,<R.VXTN,4>,<R.VXBA,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM RCVX$
```

C.2.92 RDAF\$C Macro

The RDAF\$C macro generates a DPB for the READ ALL FLAGS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RDAF\$C BUFADR,PSCT,ERR

Description: This macro generates a DPB for the READ ALL FLAGS directive in the program section named \$DPB\$\$\$. The DPB parameters are described in the RDAF\$ macro. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The RDAF\$ macro controls symbolic address generation.

MACRO EXPANSIONS

```
.MACRO RDAF$C BA,CS,ERR
.MCALL RDAF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
RDAF$ BA
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RDAF$C
```

C.2.93 RDAF\$\$ Macro

The RDAF\$\$ macro generates the code to push a DPB for the READ ALL FLAGS directive on the stack followed by an EMT 377.

Macro call: RDAF\$\$ BUFADR,ERR

Description: This macro generates the code to push a DPB for the READ ALL FLAGS directive on the stack. The DPB arguments must be valid assembler source operands and must specify the information described in the RDAF\$ macro. RDAF\$\$ also generates an EMT 377, using the ERR error service address, as described in the DIR\$ macro.

```
.MACRO RDAF$$ BA,ERR
.MCALL MOV$,DIR$
MOV$ BA
MOV (PC)+,-(SP)
.BYTE 39.,2
DIR$ ,ERR
.ENDM RDAF$$
```

C.2.94 RDAF\$ Macro

The RDAF\$ macro generates a DPB for the READ ALL FLAGS directive.

Macro call: RDAF\$ BUFADR

Description: This macro generates a DPB for the READ ALL FLAGS directive. The argument is assumed to have the following meaning:

BUFADR=Address of four word buffer.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

R.DABA-(length 2 bytes) Buffer address.

If the macro is invoked with the \$\$\$GLB symbol defined, RDAF\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO RDAF$ BA
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 39.,2
.WORD BA
.ENDC
.IF NDF R.DABA
```

MACRO EXPANSIONS

```
.NLIST
OFF$
OFF$ R.DABA,2.
.LIST
.ENDC
.ENDM RDAF$
```

C.2.95 RFA\$ Macro

RFA\$ is an internal macro that pushes two words on the stack in reverse order from the specified double-word.

Macro call: RFA\$ ADR

Description: RFA\$ generates the code to push two words on the stack in reverse order from the double word with its address specified by the source operand's ADR.

```
.MACRO RFA$ ADR
.IF NB <ADR>
.NLIST
.NTYPE $$$T2,ADR
.LIST
.IF LT $$$T2-6
MOV 2(ADR),-(SP)
MOV (ADR),-(SP)
.IFF
MOV ADR,-(SP)
MOV @(SP),-(SP)
ADD #2,2(SP)
MOV @2(SP),2(SP)
.ENDC
.IFF
CLR -(SP)
CLR -(SP)
.ENDC
.ENDM RFA$
```

C.2.96 RQST\$C Macro

The RQST\$C macro generates a DPB for the REQUEST directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RQST\$C TASK,PART,PRI,UICGC,UICPC,PSCT,ERR

Description: This macro generates a DPB for the REQUEST directive in the \$DPB\$\$ program section. The RQST\$ macro describes the DPB parameters. RQST\$C then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The RQST\$ macro controls symbolic address generation.

Note: Read the notice concerning ignored arguments in the RQST\$ macro below.

```
.MACRO RQST$C TN,PN,PR,GC,P,CS,ERR
.MCALL RQST$,DIR$
.IF NDF $$$GLB
PSECT $DPB$$
$$$=.
.IFTF
```

MACRO EXPANSIONS

```
RQST$ TN,PN,PR,GC,P
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM
```

C.2.97 RQST\$\$ Macro

The RQST\$\$ macro generates the code to push a DPB for the REQUEST directive on the stack followed by an EMT 377.

Macro call: RQST\$\$ TSKNAMADR,PRTNAMADR,PRI,UICGC,UICPC,ERR

Description: RQST\$\$ generates the code to push a DPB for the REQUEST directive on the stack. The DPB arguments must be valid assembler source operands and must specify the information described in the RQST\$ macro. RQST\$\$ also generates an EMT 377 using the ERR error service address described in DIR\$.

Note: Read the notice concerning ignored arguments in the RQST\$ macro below.

```
.MACRO RQST$$ TN,PN,PR,GC,P,ERR
.MCALL MVB$,RFA$,DIR$
MVB$ P,GC
CLR -(SP)
RFA$
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 11.,7
DIR$ ,ERR
.ENDM RQST$$
```

C.2.98 RQST\$ Macro

The RQST\$ macro generates a DPB for the REQUEST directive.

Macro call: RQST\$ TASK,PART,PRI,UICGC,UICPC

Description: This macro generates a DPB for the REQUEST directive. The arguments are assumed to have the following meanings:

```
TASK=Task name,
PART=Partition name,
PRI=Priority,
UICGC=UIC group code,
UICPC=UIC programmer code.
```

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

```
R.QSTN-(length 4 bytes) Task name,
R.QSPN-(4) Partition name,
R.QSPR-(2) Priority,
R.QSGC-(1) UIC Group,
R.QSPC-(1) UIC Programmer.
```

MACRO EXPANSIONS

If RQST\$ is invoked with the \$\$\$GLB symbol defined, RQST\$ does not generate the DPB and it globally defines the symbolic offsets.

Note: RQST\$ requires the arguments PART and PRI to maintain compatibility with RSX-11D. Despite the fact that they must be specified, RQST\$ ignores them.

```
.MACRO RQST$ TN,PN,PR,GC,P
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 11.,7
R50$ TN
.WORD 0,0
.WORD 0
.BYTE P,GC
.ENDC
.IF NDF R.QSTN
.NLIST
.IRP X,<,<R.QSTN,4>,<R.QSPN,4>,<R.QSPR,2>,<R.QSPC,1>,<R.QSGC,1>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM RQST$
```

C.2.99 RSUM\$C Macro

The RSUM\$C macro generates a DPB for the RESUME directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RSUM\$C TASK,PSCT,ERR

Description: This macro generates a DPB for the RESUME directive in the \$DPB\$\$ program section. The RSUM\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The ERR error service address is described in the DIR\$ macro. The RSUM\$ macro controls generation of symbolic offsets.

```
.MACRO RSUM$C TN,CS,ERR
.MCALL RSUM$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
RSUM$ TN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RSUM$C
```

C.2.100 RSUM\$\$ Macro

The RSUM\$\$ macro generates the code to push a DPB for the RESUME directive on the stack and the code for an EMT 377.

Macro call: RSUM\$\$ TSKNAMADR,ERR

MACRO EXPANSIONS

Description: This macro generates the code to push a DPB for the RESUME directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the RSUM\$ macro. This macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO RSUM$$ TN,ERR
.MCALL RFA$,DIR$
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 47.,3
DIR$ ,ERR
.ENDM RSUM$$
```

C.2.101 RSUM\$ Macro

The RSUM\$ macro generates a DPB for the RESUME directive.

Macro call: RSUM\$ TASK

Description: This macro generates a DPB for the RESUME directive. The argument is assumed to have the following meaning:

TASK=Task name.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

R.SUTN-(length 4 bytes) Task name.

If RSUM\$ is invoked with the \$\$\$GLB symbol defined, RSUM\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO RSUM$ TN
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 47.,3
R50$ TN
.ENDC
.IF NDF R.SUTN
.NLIST
OFF$
OFF$ R.SUTN,4
.LIST
.ENDC
.ENDM RSUM$
```

C.2.102 RUN\$C Macro

RUN\$C generates a DPB for the RUN directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: RUN\$C TSK,PRT,PRI,UGC,UPC,SMG,SNT,RMG,RNT,PSCT,ERR

Description: RUN\$C generates a DPB for the RUN directive in the \$DPB\$\$ program section. RUN\$ describes the DPB parameters. RUN\$C then generates an EMT 377 in the program section named in PSCT. DIR\$ describes the ERR error service address. The RUN\$ macro controls symbolic address generation.

MACRO EXPANSIONS

Note: Read the discussion notice concerning ignored arguments in the RUN\$ macro below.

```
.MACRO RUN$C TN,PN,PR,UG,UP,SM,SU,RM,RU,CS,ERR
.MCALL RUN$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
RUN$ TN,PN,PR,UG,UP,SM,SU,RM,RU
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RUN$C
```

C.2.103 RUN\$\$ Macro

RUN\$\$ generates the code to push a DPB for the RUN directive on the stack and generates an EMT 377.

Macro call: RUN\$\$ TNAMADR,PNAMADR,PRI,UGC,UPC,SMG,SNT,RMG,RNT,ERR

Description: RUN\$\$ generates the code to push a DPB for the RUN directive on the stack and generates an EMT 377 using the ERR error service address described in DIR\$. DPB arguments must be valid assembler source operands and must specify the information described in the RUN\$ macro.

Note: Read the discussion notice concerning ignored arguments in the RUN\$ macro below.

```
.MACRO RUN$$ TN,PN,PR,UG,UP,SM,SU,RM,RU,ERR
.MCALL RVP$,MVB$,RFA$,DIR$
RVP$ SM,SU,RM,RU
MVB$ UP,UG
CLR -(SP)
RFA$
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 17.,11.
DIR$ ,ERR
.ENDM RUN$$
```

C.2.104 RUN\$ Macro

The RUN\$ macro generates a DPB for the RUN directive.

Macro call: RUN\$ TSK,PRT,PRI,UGC,UPC,SMG,SNT,RMG,RNT

Description: The arguments must have the following meanings:

```
TSK=Task name,
PRT=Partition name,
PRI=Priority,
UGC=UIC group code,
UPC=UIC programmer code,
SMG=Schedule delta magnitude,
SNT=Schedule delta unit,
RMG=Re-schedule interval magnitude,
RNT=Re-schedule interval unit.
```


MACRO EXPANSIONS

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

```
R.UNTN-(length 4 bytes) Task name,  
R.UNPN-(4) Partition name,  
R.UNPR-(2) Priority,  
R.UNGC-(1) UIC group,  
R.UNPC-(1) UIC programmer,  
R.UNSM-(2) Schedule magnitude,  
R.UNSU-(2) Schedule unit,  
R.UNRM-(2) Re-schedule magnitude,  
R.UNRU-(2) Re-schedule unit.
```

If RUN\$ is invoked with the \$\$\$GLB symbol defined, RUN\$ does not generate the DPB and RUN\$ globally defines the symbolic offsets.

Note: RUN\$ requires the PRT and PRI arguments to maintain compatibility with RSX-11D. Despite the fact that they must be specified, RUN\$ ignores them.

```
.MACRO RUN$ TN,PN,PR,GC,P,SM,SU,RM,RU  
.MCALL R50$,OFF$  
.IF NDF $$$GLB  
.BYTE 17.,11.  
R50$ TN  
.WORD 0,0  
.WORD 0  
.BYTE P,GC  
.WORD SM  
.WORD SU  
.WORD RM  
.WORD RU  
.ENDC  
.IF NDF R.UNTN  
.NLIST  
.IRP X,<,<R.UNTN,4>,<R.UNPN,4>,<R.UNPR,2>,<R.UNPC,1>,<R.UNGC,1>>  
OFF$ X  
.ENDM  
.IRP X,<<R.UNSM,2>,<R.UNSU,2>,<R.UNRM,2>,<R.UNRU,2>>  
OFF$ X  
.ENDM  
.LIST  
.ENDC  
.ENDM RUN$
```

C.2.105 RVP\$ Macro

RVP\$ is an internal macro that places the parameters passed as an argument list into the stack in reverse order.

Macro call: RVP\$ A1,A2,...,A10

Description: RVP\$ pushes the elements of the argument list on the stack in reverse order. RVP\$ assumes all elements to be full-word. The list is limited to a maximum of ten elements.

```
.MACRO RVP$ P0,P1,P2,P3,P4,P5,P6,P7,P8,P9  
.MCALL MOV$  
.NLIST  
.NARG $$$ARG  
.LIST
```

MACRO EXPANSIONS

```
.IIF GT $$$ARG-9., MOV$ <P9>
.IIF GT $$$ARG-8., MOV$ <P8>
.IIF GT $$$ARG-7, MOV$ <P7>
.IIF GT $$$ARG-6, MOV$ <P6>
.IIF GT $$$ARG-5, MOV$ <P5>
.IIF GT $$$ARG-4, MOV$ <P4>
.IIF GT $$$ARG-3, MOV$ <P3>
.IIF GT $$$ARG-2, MOV$ <P2>
.IIF GT $$$ARG-1, MOV$ <P1>
.IIF GT $$$ARG, MOV$ <P0>
.ENDM RVP$
```

C.2.106 SDAT\$C Macro

The SDAT\$C macro generates a DPB for the SEND DATA directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SDAT\$C TASK,BUFADR,EFN,PSCT,ERR

Description: SDAT\$C generates a DPB for the SEND DATA directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSCT. SDAT\$ describes the DPB parameters and controls symbolic address generation. DIR\$ describes the ERR error service address.

```
.MACRO SDAT$C TN,BA,EFN,CS,ERR
.MCALL SDAT$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SDAT$ TN,BA,EFN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SDAT$C
```

C.2.107 SDAT\$\$ Macro

SDAT\$\$ generates the code to push a DPB for the SEND DATA directive on the stack and generates an EMT 377.

Macro call: SDAT\$\$ TSKNAMADR,BUFADR,EFN,ERR

Description: SDAT\$\$ generates the code to push a DPB for the SEND DATA directive on the stack and generates an EMT 377 using the ERR error service address described in DIR\$. The DPB arguments must be valid assembler source operands and they must specify the information described in the SDAT\$ macro.

```
.MACRO SDAT$$ TN,BA,EFN,ERR
.MCALL MOV$,RFA$,DIR$
MOV$ EFN
MOV$ BA
RFA$ TN
MOV (PC)+,-(SP)
.BYTE 71.,5
DIR$ ,ERR
.ENDM SDAT$$
```

MACRO EXPANSIONS

C.2.108 SDAT\$ Macro

The SDAT\$ macro generates a DPB for the SEND DATA directive.

Macro call: SDAT\$ TASK,BUFADR,EFN

Description: This macro generates a DPB for the SEND DATA directive. The arguments are assumed to have the following meanings:

TASK=Receiver task name,
BUFADR=Address of thirteen word data buffer,
EFN=Event flag number.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

S.DATN-(length 4 bytes) Task name,
S.DABA-(2) Buffer address,
S.DAEF-(2) Event flag number.

If SDAT\$ is invoked with the \$\$\$GLB symbol defined, SDAT\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO SDAT$ TN,BA,EFN
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 71.,5
R50$ TN
.WORD BA
.WORD EFN
.ENDC
.IF NDF S.DATN
.NLIST
.IRP X,<,<S.DATN,4>,<S.DABA,2>,<S.DAEF,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM SDAT$
```

C.2.109 SETF\$C Macro

The SETF\$C macro generates a DPB for the SET EVENT FLAG directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SETF\$C EFN,PSCT,ERR

Description: This macro generates a DPB for the SET EVENT FLAG directive in the program section named \$DPB\$\$\$. The SETF\$ macro describes the DPB parameters. The SETF\$C macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The SETF\$ macro controls symbolic offset generation.

```
.MACRO SETF$C EFN,CS,ERR
.MCALL SETF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SETF$ EFN
```

MACRO EXPANSIONS

```
.IFT  
.PSECT CS  
DIR$ $$$$,ERR  
.ENDC  
.ENDM SETF$C
```

C.2.110 SETF\$\$ Macro

SETF\$\$ pushes a DPB for the SET EVENT FLAG directive on the stack and generates an EMT 377.

Macro call: SETF\$\$ EFN,ERR

Description: SETF\$ pushes a DPB for the SET EVENT FLAG directive on the stack. DPB arguments must be valid assembler operands and specify the information described in the SETF\$ macro. SETF\$ also generates an EMT 377 using the ERR error service address that DIR\$ describes.

```
.MACRO SETF$$ EFN,ERR  
.MCALL MOV$,DIR$  
MOV$ EFN  
MOV (PC)+,-(SP)  
.BYTE 33.,2  
DIR$ ,ERR  
.ENDM SETF$$
```

C.2.111 SETF\$ Macro

The SETF\$ macro generates a DPB for the SET EVENT FLAG directive.

Macro call: SETF\$ EFN

Description: This macro generates a DPB for the SET EVENT FLAG directive. The argument is assumed to have the following meaning:

EFN=Event flag number.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

S.ETEF-(length 2 bytes) Event flag.

If SETF\$ is invoked with the \$\$\$GLB symbol defined, SETF\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO SETF$ EFN  
.MCALL OFF$  
.IF NDF $$$GLB  
.BYTE 33.,2  
.WORD EFN  
.ENDC  
.IF NDF S.ETEF  
.NLIST  
OFF$  
OFF$ S.ETEF,2  
.LIST  
.ENDC  
.ENDM SETF$
```

MACRO EXPANSIONS

C.2.112 SFPASC Macro

The SFPASC macro generates a DPB for the SPECIFY FLOATING POINT EXCEPTION AST directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SFPASC AST,PSCT,ERR

Description: SFPASC generates a DPB for the SPECIFY FLOATING POINT EXCEPTION AST directive in the \$DPB\$\$ program section. SFPAS describes the DPB parameters. SFPASC then generates an EMT 377 in the program section named in PSCT. DIR\$ describes the ERR error service address. The SFPAS macro controls symbolic offset generation.

```
.MACRO SFPASC AST,CS,ERR
.MCALL SFPAS,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SFPAS AST
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SFPASC
```

C.2.113 SFPASS Macro

SFPASS generates the code to push a DPB for the SPECIFY FLOATING POINT EXCEPTION AST directive on the stack and generates an EMT 377.

Macro call: SFPASS AST,ERR

Description: This macro generates the code to push a DPB for the SPECIFY FLOATING POINT EXCEPTION AST directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the SFPAS macro. The SFPASS macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO SFPASS AST,ERR
.MCALL MOV$,DIR$
MOV$ AST
MOV (PC)+,-(SP)
.BYTE 111.,2
DIR$ ,ERR
.ENDM SFPASS
```

C.2.114 SFPAS Macro

SFPAS generates a DPB for the SPECIFY FLOATING POINT EXCEPTION AST directive.

Macro call: SFPAS AST

Description: The argument must have the following meaning:

AST=AST Service entry address.

MACRO EXPANSIONS

The following symbol is locally defined with its assigned values equal to the byte offset from the start of the DPB to the DPB element:

S.FPAE-(length 2 bytes) AST Entry address.

If SFPAS\$ is invoked with the \$\$\$GLB symbol defined, SFPAS\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO SFPAS$ AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 111.,2
.WORD AST
.ENDC
.IF NDF S.FPAE
.NLIST
OFF$
OFF$ S.FPAE,2
.LIST
.ENDC
.ENDM SFPAS$
```

C.2.115 SPND\$C Macro

SPND\$C generates a DPB for the SUSPEND directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SPND\$C PSCT,ERR

Description: SPND\$C generates a DPB for the SUSPEND directive in the \$DPB\$\$ program section followed by an EMT 377 in the original program section named PSCT. DIR\$ describes the ERR error service address.

```
.MACRO SPND$C PSCT,ERR
.MCALL SPND$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SPND$
.IFT
.PSECT PSCT
DIR$ #$$$ ,ERR
.ENDC
.ENDM SPND$C
```

C.2.116 SPND\$\$S Macro

The SPND\$\$S macro generates the code to push a DPB for the SUSPEND directive on the stack and generates an EMT 377.

Macro call: SPND\$\$S ERR

Description: This macro generates the code to push a one-word DPB for the SUSPEND directive on the stack. Then it generates an EMT 377 using the error service address as described in the DIR\$ macro.

```
.MACRO SPND$$S ERR
.MCALL DIR$
MOV (PC)+,-(SP)
```

MACRO EXPANSIONS

```
.BYTE 45.,1
DIR$ ,ERR
.ENDM SPND$$
```

C.2.117 SPND\$ Macro

The SPND\$ macro generates a DPB for the SUSPEND directive.

Macro call: SPND\$

Description: This macro generates a DPB for the SUSPEND directive. There are no arguments.

```
.MACRO SPND$
.IF NDF $$$GLB
.BYTE 45.,1
.ENDC
.ENDM SPND$
```

C.2.118 SPRA\$C Macro

The SPRA\$C macro generates a DPB for the SPECIFY POWER RECOVERY AST directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SPRA\$C AST,PSCT,ERR

Description: This macro generates a DPB for the SPECIFY POWER RECOVERY AST directive in the \$DPB\$\$ program section. The SPRA\$ macro describes the DPB parameters. The SPRA\$C macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The SPRA\$ macro controls the generation of symbolic offsets.

```
.MACRO SPRA$C AST,CS,ERR
.MCALL SPRA$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SPRA$ AST
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SPRA$C
```

C.2.119 SPRA\$\$ Macro

The SPRA\$\$ macro generates the code to push a DPB for the SPECIFY POWER RECOVERY AST directive on the stack and generates an EMT 377.

Macro call: SPRA\$\$ AST,ERR

Description: This macro generates the code to push a DPB for the SPECIFY POWER RECOVERY AST directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the SPRA\$ macro. The SPRA\$\$ macro also

MACRO EXPANSIONS

generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO SPRA$S AST,ERR
.MCALL MOV$,DIR$
MOV$ AST
MOV (PC)+,-(SP)
.BYTE 109.,2
DIR$ ,ERR
.ENDM SPRA$S
```

C.2.120 SPRA\$ Macro

The SPRA\$ macro generates a DPB for the SPECIFY POWER RECOVERY AST directive.

Macro call: SPRA\$ AST

Description: This macro generates a DPB for the SPECIFY POWER RECOVERY AST directive. The argument is assumed to have the following meaning:

AST=AST Service entry address.

The following symbol is locally defined with is assigned a value equal to the byte offset from the start of the DPB to the DPB element:

S.PRAE-(length 2 bytes) AST entry address.

If SPRA\$ is invoked with the \$\$\$GLB symbol defined, SPRA\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO SPRA$ AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 109.,2
.WORD AST
.ENDC
.IF NDF S.PRAE
.NLIST
OFF$
OFF$ S.PRAE,2
.LIST
.ENDC
.ENDM SPRA$
```

C.2.121 SRDA\$C Macro

The SRDA\$C macro generates a DPB for the SPECIFY RECEIVE DATA AST directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SRDA\$C AST,PSCT,ERR

Description: This macro generates a DPB for the SPECIFY RECEIVE DATA AST directive in the \$DPB\$\$ program section. The SRDA\$ macro describes the DPB parameters. This macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The SRDA\$ macro controls symbolic offset generation.

MACRO EXPANSIONS

```
.MACRO SRDA$C AST,CS,ERR
.MCALL SRDA$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SRDA$ AST
.IFT
.PSECT CS
DIR$ $$$$,ERR
.ENDC
.ENDM SRDA$C
```

C.2.122 SRDA\$\$ Macro

SRDA\$\$ macro pushes a DPB for the SPECIFY RECEIVE DATA AST directive on the stack and generates an EMT 377.

Macro call: SRDA\$\$ AST,ERR

Description: This macro generates the code to push a DPB for the SPECIFY RECEIVE DATA AST directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the SRDA\$ macro. The SRDA\$\$ macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO SRDA$$ AST,ERR
.MCALL MOV$,DIR$
MOV$ AST
MOV (PC)+,-(SP)
.BYTE 107.,2
DIR$ ,ERR
.ENDM SRDA$$
```

C.2.123 SRDA\$ Macro

SRDA\$ generates a DPB for the SPECIFY RECEIVE DATA AST directive.

Macro call: SRDA\$ AST

Description: This macro generates a DPB for the SPECIFY RECEIVE DATA AST directive. The argument is assumed to have the following meaning:

AST=AST service entry address.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

S.RDAE-(length 2 bytes) AST entry address.

If SRDA\$ is invoked with the \$\$\$GLB symbol defined, SRDA\$ does not generate the DPB and it globally defines the symbolic offset.

```
.MACRO SRDA$ AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 107.,2
.WORD AST
.ENDC
```

MACRO EXPANSIONS

```
.IF NDF S.RDAE
.NLIST
OFF$
OFF$ S.RDAE,2
.LIST
.ENDC
.ENDM SRDA$
```

C.2.124 SVDB\$C Macro

SVDB\$C generates a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SVDB\$C ADR,LEN,PSCT,ERR

Description: SVDB\$C generates a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSCT. SVDB\$ describes the DPB parameters and controls symbolic address generation. DIR\$ describes the ERR error service address.

```
.MACRO SVDB$C AD,LN,CS,ERR
.MCALL SVDB$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SVDB$ AD,LN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SVDB$C
```

C.2.125 SVDB\$\$ Macro

The SVDB\$\$ macro generates the code to push a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive on the stack and generates an EMT 377.

Macro call: SVDB\$\$ ADR,LEN,ERR

Description: SRDA\$\$ macro pushes a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive on the stack. DPB arguments must be valid assembler operands and must specify the information described by SVDB\$. SVDB\$\$ also generates an EMT 377 using the ERR error service address described by DIR\$.

```
.MACRO SVDB$$ AD,LN,ERR
.MCALL MOV$,DIR$
MOV$ LN
MOV$ AD
MOV (PC)+,-(SP)
.BYTE 103.,3
DIR$ ,ERR
.ENDM SVDB$$
```

MACRO EXPANSIONS

C.2.126 SVDB\$ Macro

The SVDB\$ macro generates a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive.

Macro call: SVDB\$ ADR,LEN

Description: This macro generates a DPB for the SPECIFY SST VECTOR TABLE FOR DEBUGGING AID directive. The arguments are assumed to have the following meanings:

ADR=Address of SST vector table,
LEN=Length of (number of entries in) table.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

S.VDTA-(length 2 bytes) Table address,
S.VDTL-(2) Table length.

If SVDB\$ is invoked with the \$\$\$GLB symbol defined, SVDB\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO SVDB$ A,L
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 103.,3
.WORD A
.WORD L
.ENDC
.IF NDF S.VDTA
.NLIST
.IRP X,<,<S.VDTA,2>,<S.VDTL,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM SVDB$
```

C.2.127 SVTK\$C Macro

SVTK\$C generates a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: SVTK\$C ADR,LEN,PSCT,ERR

Description: SVTK\$C generates a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive in the \$DPB\$\$ program section and generates an EMT 377 in the program section named in PSCT. DIR\$ describes the ERR error service address. The SVTK\$ macro controls symbolic address generation and describes DPB parameters.

```
.MACRO SVTK$C ADR,LEN,CS,ERR
.MCALL SVTK$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SVTK$ ADR,LEN
.IFT
```

MACRO EXPANSIONS

```
.PSECT CS
DIR$ $$$$,ERR
.ENDC
.ENDM SVTK$C
```

C.2.128 SVTK\$\$ Macro

The SVTK\$\$ macro generates the code to push a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive on the stack and generates an EMT 377.

Macro call: SVTK\$\$ ADR,LEN,ERR

Description: This macro generates the code to push a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the SVTK\$ macro. This macro also generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO SVTK$$ ADR,LEN,ERR
.MCALL MOV$,DIR$
MOV$ LEN
MOV$ ADR
MOV (PC)+,-(SP)
.BYTE 105.,3
DIR$ ,ERR
.ENDM SVTK$$
```

C.2.129 SVTK\$ Macro

The SVTK\$ macro generates a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive.

Macro call: SVTK\$ ADR,LEN

Description: This macro generates a DPB for the SPECIFY SST VECTOR TABLE FOR TASK directive. The arguments are assumed to have the following meanings:

ADR=Address of SST vector table,
LEN=Length of (number of entries in) table.

The following symbols are locally defined with their assigned values equal to the byte offset from the start of the DPB to the respective DPB elements:

S.VTTA-(length 2 bytes) Table address,
S.VTTL-(2) Table length.

If SVTK\$ is invoked with the \$\$\$GLB symbol defined, SVTK\$ does not generate the DPB and it globally defines the symbolic offsets.

```
.MACRO SVTK$ TA,TL
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 105.,3
.WORD TA
.WORD TL
.ENDC
.IF NDF S.VTTA
```

MACRO EXPANSIONS

```
.NLIST
.IRP X,<,<S.VTTA,2>,<S.VTTL,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM SVTK$
```

C.2.130 WSIG\$C Macro

The WSIG\$C macro generates a DPB for the WAIT FOR SIGNIFICANT EVENT directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: WSIG\$C PSCT,ERR

Description: This macro generates a DPB for the WAIT FOR SIGNIFICANT EVENT directive in the \$DPB\$\$ program section followed by an EMT 377 in the user specified program section that is named PSCT. The DIR\$ macro describes the ERR error service address.

```
.MACRO WSIG$C PSCT,ERR
.MCALL WSIG$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
WSIG$
.IFT
.PSECT PSCT
DIR$ $$$,ERR
.ENDC
.ENDM WSIG$C
```

C.2.131 WSIG\$\$ Macro

The WSIG\$\$ macro generates the code to push a DPB for the WAIT FOR SIGNIFICANT event directive on the stack and generates an EMT 377.

Macro call: WSIG\$\$ ERR

Description: This macro generates the code to push a one-word DPB for the WAIT FOR SIGNIFICANT EVENT directive on the stack. Then it generates an EMT 377 using the ERR error service address as described in the DIR\$ macro.

```
.MACRO WSIG$$ ERR
.MCALL DIR$
MOV (PC)+,-(SP)
.BYTE 49.,1
DIR$ ,ERR
.ENDM WSIG$$
```

C.2.132 WSIG\$ Macro

The WSIG\$ macro generates a DPB for the WAIT FOR SIGNIFICANT EVENT directive.

MACRO EXPANSIONS

Macro call: WSIG\$

Description: This macro generates a DPB for the WAIT FOR SIGNIFICANT EVENT directive. There are no arguments.

```
.MACRO WSIG$  
.IF NDF $$$GLB  
.BYTE 49.,1  
.ENDC  
.ENDM WSIG$
```

C.2.133 WTLO\$C Macro

The WTLO\$C Macro generates a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: WTLO\$C SET, MASK, PSCT, ERR

Description: This macro generates a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive in the \$DPB\$\$ program section. The WTLO\$ macro describes the DPB parameters. The WTLO\$C macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The WTLO\$ macro controls symbolic offset generation.

Note: Read the discussion notice concerning ignored arguments in the WTLO\$ macro below.

```
.MACRO WTLO$C SET, MASK, CS, ERR  
.MCALL WTLO$, DIR$  
.IF NDF $$$GLB  
.PSECT $DPB$$  
$$$=  
.IFTF  
WTLO$ SET, <MASK>  
.IFT  
.PSECT CS  
DIR$ #$$$ , ERR  
.ENDC  
.ENDM WTLO$C
```

C.2.134 WTLO\$\$ Macro

The WTLO\$\$ macro generates the code to push a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive on the stack and generates an EMT 377.

Macro call: WTLO\$\$ SET, MASK, ERR

Description: This macro generates the code to push a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive on the stack. The arguments are assumed to have the following meanings:

```
SET=Desired set of event flags (0-4)  
MASK=Assembler source operand that yields the desired  
mask word
```

The macro also generates an EMT 377, using the ERR error service address as described in the DIR\$ macro.

MACRO EXPANSIONS

Note: Read the discussion notice concerning ignored arguments in the WTLO\$ macro below.

```
.MACRO WTLO$ SET,MSK,ERR
.MCALL MOV$,DIR$
.IF EQ SET-4
.ERROR SET           ;Unsupported macro option;
.IFF
MOV$ MSK
.IF NE SET
MOV #SET,-(SP)
.IFF
LR -(SP)
.ENDC
MOV (PC)+,-(SP)
.BYTE 43.,3
.ENDC
DIR$ ,ERR
.ENDM WTLO$
```

C.2.135 WTLO\$ Macro

The WTLO\$ macro generates a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive.

Macro call: WTLO\$ SET,MASK

Description: This macro generates a DPB for the WAIT FOR LOGICAL OR OF EVENT FLAGS directive. The arguments are assumed to have the following meanings:

```
SET=Desired set of event flags (0-4),
MASK=If set is 0, 1, 2, or 3, a 16-bit (16 flag) mask word;
      If set is 4, a list of four mask words (<M1,M2,M3,M4>).
```

WTLO\$ does not define any symbolic offsets. WTLO\$ does not generate any code if the \$\$\$GLB symbol is defined at the time WTLO\$ is invoked.

Note: RSX-11M does not support the specification of flags set number 4. If flag set number 4 is specified it causes an error.

```
.MACRO WTLO$ SET,MASK
.IF NDF $$$GLB
.IF EQ SET-4
.ERROR SET           ;Unsupported macro option
.IFF
.BYTE 43.,3
.WORD SET
.WORD MASK
.ENDC
.ENDC
.ENDM WTLO$
```

C.2.136 WTSE\$C Macro

The WTSE\$C macro generates a DPB for the WAIT FOR SINGLE EVENT FLAG directive in a separate PSECT followed by an EMT 377 in the user specified PSECT.

Macro call: WTSE\$C EFN,PSCT,ERR

MACRO EXPANSIONS

Description: This macro generates a DPB for the WAIT FOR SINGLE EVENT FLAG directive in the \$DPB\$\$ section. The WTSE\$ macro describes the DPB parameters. The WTSE\$C macro then generates an EMT 377 in the program section named in PSCT. The DIR\$ macro describes the ERR error service address. The WTSE\$ macro controls symbolic address generation.

```
.MACRO WTSE$C EF,CS,ERR
.MCALL WTSE$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
WTSE$ EF
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM WTSE$C
```

C.2.137 WTSE\$\$ Macro

The WTSE\$\$ macro generates the code to push a DPB for the WAIT FOR SINGLE EVENT FLAG directive on the stack and generates an EMT 377.

Macro call: WTSE\$\$ EFN,ERR

Description: This macro generates the code to push a DPB for the WAIT FOR SINGLE EVENT FLAG directive on the stack. The DPB arguments must be valid assembler source operands, and they must specify the information described in the WTSE\$\$ macro. This macro also generates an EMT 377, using the ERR error service address as described in the DIR\$ macro.

```
.MACRO WTSE$$ EF,ERR
.MCALL MOV$,DIR$
MOV$ EF
MOV (PC)+,-(SP)
.BYTE 41.,2
DIR$ ,ERR
.ENDM WTSE$$
```

C.2.138 WTSE\$ Macro

WTSE\$ generates a DPB for the WAIT FOR SINGLE EVENT FLAG directive.

Macro call: WTSE\$ EFN

Description: WTSE\$ generates a DPB for the WAIT FOR SINGLE EVENT FLAG directive. The argument must have the following meaning:

EFN=Event flag number.

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

W.TSEF-(length 2 bytes) Event flag number.

If WTSE\$ is invoked with the \$\$\$GLB symbol defined, WTSE\$ does not generate the DPB and it globally defines the symbolic offset.

MACRO EXPANSIONS

```
.MACRO WTSE$ EFN
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 41.,2
.WORD EFN
.ENDC
.IF NDF W.TSEF
.NLIST
OFF$
OFF$ W.TSEF,2
.LIST
.ENDC
.ENDM WTSE$
```

C.3 EXECUTIVE MACRO EXPANSIONS

The following coded instructions are expansions of macros that you may find in the Executive source listings of RSX-11M.

C.3.1 CALL Macro

```
.MACRO CALL SUBR ARG
.IF IDN <$INTSV>,<SUBR>
JSR R5,$INTSV
.IF DF L$$$I1
.WORD ARG
.IFF
.WORD ^C<ARG>&PR7
.ENDC
.IFF
.IF IDN <$$SAVNR>,<SUBR>
JSR R5,$SAVNR
.IFF
.IF IDN <$$SWSTK>,<SUBR>
EMT 376
.WORD ARG
.IFF
JSR PC,SUBR
.ENDC
.ENDC
.ENDC
.ENDM
```

C.3.2 CALLR Macro

CALLR is a call and return from subroutine macro.

```
.MACRO CALLR SUBR
JMP SUBR
.ENDM
```

MACRO EXPANSIONS

C.3.3 CRASH Macro

CRASH crashes the system.

```
.MACRO CRASH
IOT
.ENDM
```

C.3.4 DIRSV\$ Macro

DIRSV\$ is the directive register save and set priority macro.

```
.MACRO DIRSV$
JSR R5,$DIRSV
.ENDM
```

C.3.5 DRSTS Macro

DRSTS causes a trap to set directive status.

```
.MACRO DRSTS VALUE
TRAP VALUE
.ENDM
```

C.3.6 GTUCB\$ Macro

GTUCB\$ generates code to load the UCB address into R5. It is called by INTSE\$ and INTSV\$ only.

```
.MACRO GTUCB$ UCBSV,NCTRLR
.IF NB <UCBSV>
.IF GT NCTRLR-1
MOV UCBSV(R4),R5
.IFF
MOV UCBSV,R5
.ENDC
.IFF
.IF GT NCTRLR-1
MOV CNTBL(R4),R5
.IFF
MOV CNTBL,R5
.ENDC
.ENDC
.ENDM
```

C.3.7 INTLB Macro

Generates an interrupt routine entry point label.

```
.IF DF E$$DVC
.MACRO INTLB NUM,NAM
$'NAM'NUM'T::
.ENDM
.ENDC
```

MACRO EXPANSIONS

C.3.8 INTSE\$ Macro

This is the interrupt save generation macro for error logging devices.

```
.MACRO INTSE$ DEV,PRI,NCTRLR,PSWSV,UCBSV,?LAB
  .IF DF L$$DRV & LD$'DEV & M$$MGE
$'DEV'INT::
  .IF NDF E$$DVC
  .IF EQ NCTRLR-1
  CLR R4
  .ENDC
  .ENDC
  .IFF
  .IF NDF E$$DVC
$'DEV'INT::INTSV$ DEV,PRI,NCTRLR,PSWSV,UCBSV
  .IF EQ NCTRLR-1
  CLR R4
  .ENDC
  .MEXIT
  .IFF
$$$=0
  .REPT NCTRLR
  INTLB \$$$ ,DEV
  JSR R5,$INTSE
  SCBLB \$$$ ,DEV
  .IF DF L$$SII
  .WORD PRI
  .IFF
  .WORD ^C<PRI>&PR7
  .ENDC
  .IF GT NCTRLR-$$$-1
  BR LAB
  .ENDC
$$$=$$$+1
  .ENDR
LAB:
  .ENDC
  .ENDC
  GTUCB$ UCBSV,NCTRLR
  .ENDM
```

C.3.9 INTSV\$ Macro

INTSV\$ causes interrupt save generation for non-error logging devices.

```
.MACRO INTSV$ DEV,PRI,NCTRLR,PSWSV,UCBSV
  .IF NDF L$$DRV ! M$$MGE ! LD$'DEV
  .IF GT NCTRLR-1
  .IF B <PSWSV>
  MFPS TEMP
  .IFF
  MFPS PSWSV
  .ENDC
  .IFTF
  JSR R5,$INTSV
  .IF DF L$$SII
  .WORD PRI
  .IFF
  .WORD ^C<PRI>&PR7
  .ENDC
  .IFT
  .IF B <PSWSV>
```

MACRO EXPANSIONS

```
MOV TEMP,R4
.IFF
MOV PSWSV,R4
.ENDC
BIC #177760,R4
ASL R4
.ENDC
.ENDC
GTUCB$ UCBSV,NCTRLR
.ENDM
```

C.3.10 MFPS/MTPS Macros

These macros either read or write processor status.

```
.IF NDF L$$$I1
.MACRO MFPS DST
MOVB @#PS,DST
.ENDM
.MACRO MTPS SRC
.IF IDN <#0>,<SRC>
CLRB @#PS
.IFF
MOVB SRC,@#PS
.ENDC
.ENDM
.ENDC
```

C.3.11 RETURN Macro

RETURN generates an RTS PC instruction.

```
.MACRO RETURN
RTS PC
.ENDM
```

C.3.12 SAVNR Macro

SAVNR causes a jump to a subroutine to save nonvolatile registers.

```
.MACRO SAVNR
JSR R5,$SAVNR
.ENDM
```

C.3.13 SCBLB Macro

SCBLB generates a general status control block reference label.

```
.IF DF E$$DVC
.MACRO SCBLB NUM,NAM
.WORD $'NAM'NUM
.ENDM
.ENDC
```

MACRO EXPANSIONS

C.3.14 SETD Macro

SETD sets the floating-point processor to double precision mode.

```
.MACRO SETD
.WORD 170011
.ENDM
```

C.3.15 SOB Macro

SOB generates a decrement loop.

```
.IF NDF R$$EIS
.MACRO SOB A,B
DEC A
BNE B
.ENDM
.ENDC
```

C.3.16 STD Macro

STD stores a single or double precision word from the floating-point processor.

```
.MACRO STD A,B
.NTYPE N,A
.NTYPE M,B
.WORD 174000+<N*64.>+M
.ENDM
```

C.3.17 STFPS Macro

STFPS stores the floating-point processor's status.

```
.MACRO STFPS A
.NTYPE N,A
.WORD 170200+N
.ENDM
```

C.3.18 STST Macro

STST stores the floating-point exception code and exception address pointer.

```
.MACRO STST A
.WORD 170337
.WORD FLSTS
.ENDM
.ENDC
.IIF NDF S$$YDF, .LIST
```

MACRO EXPANSIONS

C.3.19 SWSTK\$ Macro

SWSTK\$ calls \$SWSTK to switch system states.

```
.MACRO SWSTK$ ARG
CALL $SWSTK,ARG
.ENDM
.IF DF F$$LPP
```

C.3.20 LDD Macro

LDD loads double precision floating-point word into the accumulator.

```
.MACRO LDD A,B
.NTYPE N,A
.NTYPE M,B
.WORD 172400+<M*64.>+N
.ENDM
```

C.3.21 LDFPS Macro

LDFPS loads floating-point status.

```
.MACRO LDFPS A
.IF IDN <A>,<@H.FPSA(R2)>
.WORD 170100+72
.WORD H.FPSA
.IFF
.NTYPE N,A
.WORD 170100+N
.ENDC
.ENDM
```

C.4 FILES-11 HEADER OFFSETS MACRO DEFINITIONS

C.4.1 FHD01\$ Macro

FHD01\$ defines Files-11 offsets.

```
.MACRO FHD01$ OFFSET
.MCALL DEF$I,OFFSET,DEFIN$
.IF DF,H.FPRO
.ERROR
.ENDC
```

Header Area Offsets

```
DEF$I 0
OFFSET H.IDOF,1 ;IDENT area offset in words
OFFSET H.MPOF,1 ;Map area offset in words
OFFSET H.FNUM,2 ;File number
OFFSET H.FSEQ,2 ;File sequence number
OFFSET H.FLEV,2 ;Structure level and system number
OFFSET H.FOWN ;Owner of file consisting of:
OFFSET H.PROG,1 ; Programmer number
```

MACRO EXPANSIONS

```
OFFSET H.PROJ,1      ; Project number
OFFSET H.FPRO,2     ; File protection code
```

File Protection Bits

```
DEFIN$ FP.RDV,1     ;Read access allowed if clear
DEFIN$ FP.WRV,2     ;Write access allowed if clear
DEFIN$ FP.EXT,4     ;Extend access allowed if clear
DEFIN$ FP.DEL,10    ;Delete allowed if clear
DEFIN$ FP.RAT,1     ;Read attributes allowed if clear
OFFSET H.FCHA       ;File characteristics code
                   ; consisting of:
OFFSET H.UCHA,1     ; User controlled characteristics byte
```

Bit Definitions for User Controlled Characteristics Byte

```
DEFIN$ UC.CON,200   ;File is logically contiguous if set
DEFIN$ UC.DLK,100   ;Deaccess lock set if bit is set
OFFSET H.SCHA,1     ;System controlled characteristics
                   ; byte
```

Bit Definitions for System Controlled Characteristics Byte

```
DEFIN$ SC.MDL,200   ;Marked for delete if set
DEFIN$ SC.BAD,100   ;Bad data block in file if set
OFFSET H.UFAT,32.   ;User file attributes
OFFSET S.HDHD       ;Size in bytes of header area
```

Ident Area Offsets

```
DEF$ I 0
OFFSET I.FNAM,6     ;File name in RAD50
OFFSET I.FTYP,2     ;File type in RAD50
OFFSET I.FVER,2     ;File version number in binary
OFFSET I.RVNO,2     ;Revision number
OFFSET I.RVDT,7     ;Revision date
OFFSET I.RVTI,6     ;Revision time
OFFSET I.CRDT,7     ;Creation date
OFFSET I.CRTI,6     ;Creation time
OFFSET I.EXDT,7     ;Expiration date
OFFSET ,1          ;Round up to word boundary
OFFSET S.IDHD       ;Size in bytes of IDENT area
DEFIN$ I.DASZ,7     ;Number of bytes in date string
DEFIN$ I.TISZ,6     ;Number of bytes in time string
```

C.4.2 FHDOF\$ Macro

FHDOF\$ defines Files-11 header offsets.

```
          FHDOF$           ;Define offsets locally
FHDOF$ DEF$L           ;Define offsets locally
FHDOF$ DEF$G           ;Define offsets globally
.MACRO FHDOF$ GLOBAL
.MCALL FHD01$
...GBL=0
.IF B,GLOBAL
FHD01$ DEF$L
.IFF
.IF IDN,<GLOBAL>,<DEF$G>
...GBL=1
.ENDC
FHD01$ GLOBAL
```

MACRO EXPANSIONS

```
.ENDC
.IF DIF,<GLOBAL>,<DEF$N>
.MACRO FHDOF$ ARG1 ;Redefine macro
.ENDM FHDOF$
.ENDC
.ENDM FHDOF$
```

Map Area Offset

```
DEF$I 0
OFFSET M.ESQN,1 ;Extension sequence number
OFFSET M.ERVN,1 ;Extension relative volume number
OFFSET M.EFNU,2 ;Extension file number
OFFSET M.EFSQ,2 ;Extension file sequence number
OFFSET M.CTSZ,1 ;Block count field size
OFFSET M.LBSZ,1 ;Logical block number field size
OFFSET M.USE,1 ;Words in use in the map
OFFSET M.MAX,1 ;Max number of words available in map
OFFSET M.RTRV ;Start of retrieval pointers
OFFSET S.MPHD ;Size in bytes of the map area
```

Checksum for File Header

```
DEFIN$ H.CKSM,510. ;Sum of words 0-255.
.IF DIF,<OFFSET>,<DEF$N>
.MACRO FHD01$ ARG1
.ENDM FHD01$
.ENDC
.ENDM FHD01$
```

C.4.3 HMBOF\$ And HMB01\$ Macros

HMBOF\$ and HMB01\$ define Files-11 home block offsets.

```
HMBOF$ ;Define offsets locally
HMBOF$ DEF$L ;Define offsets locally
HMBOF$ DEF$G ;Define offsets globally
.MACRO HMBOF$ GLOBAL
.MCALL HMB01$
...GBL=0
.IF B,GLOBAL
HMB01$ DEF$L
.IFF
.IF IDN,<GLOBAL>,<DEF$G>
...GBL=1
.ENDC
HMB01$ GLOBAL
.ENDC
.IF DIF,<GLOBAL>,<DEF$N>
.MACRO HMBOF$ ARG1
.ENDM HMBOF$
.ENDC
.ENDM HMBOF$
.MACRO HMB01$ OFFSET
.MCALL DEF$I,OFFSET,DEFIN$
DEF$I 0
OFFSET H.IBSZ,2 ;Index bit map size
OFFSET H.IBLB,4 ;Index bit map LBN
OFFSET H.FMAX,2 ;Maximum number of files on volume
OFFSET H.SBCL,2 ;Storage bit map cluster factor
OFFSET H.DVTY,2 ;Disk device type
OFFSET H.VLEV,2 ;Structure level
```


MACRO EXPANSIONS

```

OFFSET H.VNAM,12.    ;Volume name - ASCII
OFFSET ,4            ;RESERVED
OFFSET H.VOWN,2      ;Volume owner's UIC
OFFSET H.VPRO,2      ;Volume protection code
                   ;Volume characteristics
OFFSET H.VCHA,2

```

New Home Block Information Down to H.FIEX

```

OFFSET H.DFPR,2      ;Default file protection
.IF DF,H.FPRO
.ERROR               ;Tried to multiply define H.FPRO
.ENDC
.IF DF,R$$11M
OFFSET H.VFSQ,2      ;Volume file sequence number
                   ;Updated by dismount

.IFF
OFFSET ,2            ;Not used
.ENDC
OFFSET ,4            ;Not used
OFFSET H.WISZ,1      ;Window size for all files on volume
OFFSET H.FIEX,1      ;Default file extend in blocks
OFFSET H.LRUC,1      ;Number of entries in directory LRU
OFFSET ,11.          ;Available
OFFSET H.CHK1,2      ;Checksum of words 0-28.
OFFSET H.VDAT,14.    ;Creation date
OFFSET ,398.         ;Reserved for relative volume table
OFFSET H.INDN,12.    ;System independent volume name
OFFSET H.INDO,12.    ;System independent owner name
OFFSET H.INDF,12.    ;System independent format type
OFFSET ,2            ;Not used
OFFSET H.CHK2,2      ;Checksum of words 0-255.
.IF DIF,<OFFSET>,<DEF$N>
.MACRO HMBOL$ ARG1
.ENDM HMBOL$
.ENDC
.ENDM HMBOL$
.IIF NDF,S$$YDF,.LIST

```

C.5 FILE CONTROL SERVICES MACRO EXPANSIONS

C.5.1 BDOFF\$ Macro

BDOFF\$ generates the buffer descriptor offsets. The same type of call occurs as for file descriptor offsets.

```

.MACRO BDOFF$ OFFSET
.MCALL OFFSET,DEF$I
DEF$I 0
OFFSET ,4
OFFSET B.VBN,4      ;Virtual block number for this buffer
OFFSET B.BBFS,2     ;NUMBER OF bytes read or written
OFFSET B.NXBD,2     ;Pointer to next buffer descriptor
OFFSET ,1           ;Spare byte
OFFSET B.BFST,1     ;Buffer status byte
OFFSET ,2
OFFSET S.BFHD
.IIF IDN,<DEF$N>,<OFFSET>,.MEXIT
.MACRO BDOFF$ ARG
.ENDM BDOFF$
ENDM BDOFF$

```

MACRO EXPANSIONS

C.5.2 CBYTE\$ Macro

CBYTE\$ conditionally assembles a .BYTE directive for VAR at the specified offset from the beginning of the FDB.

```
.MACRO CBYTE$ VAR,OFFSET
  .IF NB,VAR           ;If non null
  ...PC2=             ;Save PC
  ...PC1+OFFSET       ;Set PC to specified offset
  .BYTE VAR           ;Assemble the byte
  ...PC2              ;Restore PC
  .ENDC
  .ENDM CBYTE$
```

C.5.3 CLOSE\$ Macro

CLOSE\$ closes a file.

```
.MACRO CLOSE$ FDB,ERR
  .MCALL LDFDB$,ERR$
  LDFDB$ FDB
  .GLOBL .CLOSE
  JSR PC,.CLOSE
  ERR$ ERR
  .ENDM CLOSE$
```

C.5.4 CMOV\$2 Macro

CMOV\$2 conditionally moves two words from VAR to the specified offset of R0.

```
.MACRO CMOV$2 VAR,OFFSET
  .IIF B,VAR,.MEXIT
  MOV VAR,-(SP)        ;Address of 2 words to state
  MOV @(SP),OFFSET(R0) ;First word to FDB
  ADD #2,(SP)          ;Calculate address of 2nd word
  MOV @(SP)+,OFFSET+2(R0) ;2nd word to FDB
  .ENDM CMOV$2
```

C.5.5 CMOV\$B Macro

CMOV\$B conditionally generates a MOVE BYTE instruction to move a byte to the specified offset of R0.

```
.MACRO CMOV$B VAR,OFFSET
  .IF IDN,<#0>,<VAR>
  CLRB OFFSET(R0)
  .MEXIT
  .ENDC
  .IIF NB,VAR, MOV B VAR,OFFSET(R0)
  .ENDM CMOV$B
```

MACRO EXPANSIONS

C.5.6 CMOV\$W Macro

CMOV\$W conditionally moves a word to the specified offset of R0.

```
.MACRO CMOV$W VAR,OFFSET
  .IF IDN,<#0>,<VAR>
  CLR OFFSET(R0)
  .MEXIT
  .ENDC
  .IIF NB,VAR, MOV VAR,OFFSET(R0)
  .ENDM CMOV$W
```

C.5.7 CWORD\$ Macro

CWORD\$ conditionally assembles a .WORD directive for VAR at the specified offset for the beginning of the FDB.

```
MACRO CWORD$ VAR,OFFSET
  .IF NB,VAR ;If VAR is not null
  ...PC2=. ;Save the PC
  ...PC1+OFFSET ;Set the PC to the specified offset
  .WORD VAR ;Assemble the word
  ...PC2 ;Restore the PC
  .ENDC
  .ENDM CWORD$
```

C.5.8 DEF\$G Macro

DEF\$G defines offsets globally.

```
.MACRO DEF$G VAR,SIZ ;Define global offset
  .MCALL DEF$L
  .IIF NB,VAR,.GLOBL VAR
  DEF$L VAR,SIZ
  .ENDM DEF$G
```

C.5.9 DEF\$I Macro

```
.MACRO DEF$I IVAL ;Initialize definition
  ...TPC=^O<IVAL> ;Macros program counter
  .ENDM DEF$I
```

C.5.10 DEF\$L Macro

```
.MACRO DEF$L SYM,SIZ
  .IF NB,SYM
  SYM=^O<...TPC>
  .ENDC
  .IF NB,SIZ
  ...TPC=^O<...TPC+SIZ>
  .ENDC
  .ENDM DEF$L
```

MACRO EXPANSIONS

C.5.11 DEF\$N Macro

DEF\$N updates ...TPC but does not define the symbol.

```
.MACRO DEF$N VAR,SIZ      ;Define no offset-calculate the size
.MCALL DEF$L
DEF$L ,SIZ
.ENDM DEF$N
```

C.5.12 DEFIN\$ Macro

DEFIN\$ equates the symbol with its specified value and defines it globally if ...GBL = 1. Otherwise, it locally defines the symbol.

```
.MACRO DEFIN$ SYM,VAL
.IIF EQ,...GBL-1,.GLOBL SYM
SYM=^O<VAL>
.ENDM DEFIN$
```

C.5.13 DELET\$ Macro

DELET\$ deletes a file.

```
.MACRO DELET$ FDB,ERR
.MCALL LDFDB$,ERR$
LDFDB$ FDB
.GLOBL .DELET
JSR PC,.DELET
ERR$ ERR
.ENDM DELET$
```

C.5.14 FCSBT\$ Macro

FCSBT\$ defines FCS bits and values locally or globally. Call with DEF\$G for global definitions; conventionally called with DEF\$L for local, but anything not equal to DEF\$G will do

```
.MACRO          FCSBT$ GLOBL
.MCALL DEFIN$
...GBL=0
.IIF IDN,<GLOBL>,<DEF$G>,...GBL=1
```

F.RATT bits - Record Attribute Byte

```
DEFIN$ FD.FTN,1          ;FORTRAN carriage control bit
DEFIN$ FD.CR,2           ;Insert carriage returns between
                        ;records
DEFIN$ FD.PRN,4         ;R.SEQ/VFC print file
DEFIN$ FD.BLK,10        ;1 if records cannot cross block
                        ;boundaries
```

F.RACC bits - Record Access Byte

```
DEFIN$ FD.RWM,1         ;On if read/write, off if GET/PUT
DEFIN$ FD.RAN,2         ;On if random, off if sequential
DEFIN$ FD.PLC,4         ;On if partial locate, off if
                        ;sequential
DEFIN$ FD.INS,10        ;On if put sequential insert
                        ;mode; off if truncate mode
```

MACRO EXPANSIONS

F.RCTL bits - Record Control Byte - Device Characteristics

```

DEFIN$ FD.REC,1           ;On if record oriented device,
                           ;off if block oriented
DEFIN$ .D.CCL,2          ;On if carriage control output
  device;                 ;off if not
DEFIN$ FD.TTY,4          ;On if this device is a TTY
DEFIN$ FD.DIR,10         ;On if directory device, off if
                           ;not
DEFIN$ FD.SDI,20         ;On if single directory device
DEFIN$ FD.SQD,40         ;On if sequential device
DEFIN$ FD.ISP,2000       ;Input spooling
DEFIN$ FD.OSP,4000       ;Output spooling
DEFIN$ FD.PSE,10000
DEFIN$ FD.COM,20000
DEFIN$ FD.F11,40000
DEFIN$ FD.MNT,100000

```

N.STAT Bits - File Name Block Status Word - Set by parse

```

DEFIN$ NB.VER,1          ;Set if file version was explicit
DEFIN$ NB.TYP,2          ;Set if file type was explicit
DEFIN$ NB.NAM,4          ;Set if file name was explicit
DEFIN$ NB.SVR,10        ;Set if * in version field
DEFIN$ NB.STP,20        ;Set if * in type field
DEFIN$ NB.SNM,40        ;Set if * in name field
DEFIN$ NB.DIR,100       ;Set if explicit directory
                           ;specified
DEFIN$ NB.DEV,200       ;Set if device name was explicit
DEFIN$ NB.SD1,400       ;Set if * in project number of
  PPN
DEFIN$ NB.SD2,1000      ;Set if * in programmer number of
  PPN

```

***** NOTE: MORE BITS IN N.STAT ARE LOCALLY USED BY FCS

Check Definition Section in FCS
 F.RTYP Values - Record Type Byte
 Note: These are values, not bits

```

DEFIN$ R.FIX,1           ;Fixed length records
DEFIN$ R.VAR,2           ;Variable length records
DEFIN$ R.SEQ,3           ;Sequenced records

```

F.FACC Bits - File Access Byte

```

DEFIN$ FA.RD,1           ;Set if read only
DEFIN$ FA.WRT,2          ;Set if accessed for write
DEFIN$ FA.EXT,4          ;Set if access for extend
DEFIN$ FA.CRE,10        ;Set if creating new file
DEFIN$ FA.TMP,20        ;Set if creating temp file
DEFIN$ FA.SHR,40        ;Set if shared access

```

The following two names apply to the same control bit in F.FACC:

```

FA.APD is only used if the file is an existing file
(FA.CRE=0);
FA.NSP is only used if the file is being created
(FA.CRE=1)

```

MACRO EXPANSIONS

```

DEFIN$ FA.APD,100                ;Set if appending (POSIT to EOF)
DEFIN$ FA.NSP,100                ;Set if inhibiting supersede on
                                ;file creation
DEFIN$ FO.RD,FA.RD               ;Open for read
DEFIN$ FO.WRT,FA.WRT!FA.EXT!FA.CRE ;Open for write (create)
DEFIN$ FO.APD,FA.WRT!FA.EXT!FA.APD ;Open for append
DEFIN$ FO.MFY,FA.WRT             ;Open for modify
FO.UPD,FA.WRT!FA.EXT ;Open for update

```

Bits in F.ACTL in the FDB - Several of these bits also have local definitions in FCSPRE.MAC; however, all definitions are mutually exclusive.

```

DEFIN$ FA.ENB,100000            ;Enable use of F.ACTL word
DEFIN$ FA.WCK,20000            ;Enable write check
DEFIN$ FA.SEQ,40000            ;Enable sequential processing only
DEFIN$ FA.DLK,1000             ;Enable no lock on abnormal close
DEFIN$ FA.RWD,4000             ;Enable rewind control for magtape
DEFIN$ FA.POS,10000           ;File creation position control for
                                ;magtape
DEFIN$ FA.EXC,2000             ;Exclusive use bit

```

F.CHR Bits - ACP Global Volume Characteristics

```

DEFIN$ CH.AND,1                 ;ANSI 'D' format

```

F.MBFG Bits - Multiple Buffering Flag Word -

```

DEFIN$ FD.RAH,1                 ;Read ahead if set - this or write
                                ;behind
DEFIN$ FD.WBH,2                 ;Write behind if set - not both

```

Note: More bits in F.MBFG used in FCS locally

.CTRL Function Codes

```

DEFIN$ FF.RWD,1                 ;Rewind
DEFIN$ FF.POE,2                 ;Position to end of volume (set)
DEFIN$ FF.NV,3                  ;Next volume
DEFIN$ FF.SPC,4                 ;Space
DEFIN$ FF.CHR,5                 ;Get APC characteristics
DEFIN$ FF.RWF,6                 ;ERS rewind file
.MACRO FCSBT$ ARG
.ENDM FCSBT$
.ENDM FCSBT$

```

C.5.15 FCSMC\$ Macro

FCSMC\$ executes an MCALL for all the FCS macros.

```

.MACRO FCSMC$
.MCALL OPEN$R,OPEN$W,OPEN$M,OPEN$U,OPEN$A,CLOSE$
.MCALL OPNS$R,OPNS$W,OPNS$M,OPNS$U,OPNS$A
.MCALL READ$,WRITE$,WAIT$,GET$,PUT$,DELET$,FINIT$
.MCALL FSR$SZ$,FDBF$,FDAT$,FDRC$,FDOP$,FDBF$,FDBK$A
.MCALL FDAT$R,FDRC$R,FDOP$R,FDBF$R,FDBK$R,NMBLK$
.MACRO FCSMC$
.ENDM FCSMC$
.ENDM FCSMC$

```

MACRO EXPANSIONS

C.5.16 FDAT\$A Macro

FDAT\$A initializes the file attribute section of the FDB at assembly time.

```
.MACRO FDAT$A RTYP,RATT,RSIZ,CNTG,ALOC
.MCALL FDOFF$,CBYTE$,CWORD$
FDOFF$ DEF$L
CBYTE$ <RTYP>,F.RTYP
CBYTE$ <RATT>,F.RATT
CWORD$ <RSIZ>,F.RSIZ
CWORD$ <CNTG>,F.CNTG
CWORD$ <ALOC>,F.ALOC
.ENDM FDAT$A
```

C.5.17 FDAT\$R Macro

FDAT\$R initializes the file attribute section of the FDB at run time.

```
.MACRO FDAT$R FDB,RTYP,RATT,RSIZ,CNTG,ALOC
.MCALL LDFDB$,CMOV$W,CMOV$B
LDFDB$ FDB ;Load FDB address
CMOV$B RTYP,F.RTYP
CMOV$B RATT,F.RATT
CMOV$W RSIZ,F.RSIZ
CMOV$W CNTG,F.CNTG
CMOV$W ALOC,F.ALOC
.ENDM FDAT$R
```

C.5.18 FDBDF\$ Macro

FDBDF\$ allocates space at assembly time for the FDB.

```
.MACRO FDBDF$
.MCALL FDBSZ$
FDBSZ$
...PCl=.
.BLKB S.FDB
.ENDM FDBDF$
```

C.5.19 FDBF\$A Macro

FDBF\$A initializes the buffer descriptor section of the FDB at assembly time.

```
.MACRO FDBF$A EFN,OVBS,MBCT,MBFG
.MCALL FDOFF$,CBYTE$,CWORD$
FDOFF$ DEF$L
CBYTE$ <EFN>,F.EFN
CWORD$ <OVBS>,F.OVBS
CBYTE$ <MBCT>,F.MBCT
CBYTE$ <MBFG>,F.MBFG
.ENDM FDBF$A
```

MACRO EXPANSIONS

C.5.20 FDBF\$R Macro

FDBF\$R initializes the block buffer section of the FDB at run time.

```
.MACRO FDBF$R FDB,EFN,OVBS,MBCT,MBFG
.MCALL LDFDB$,CMOV$W,CMOV$B
LDFDB$ FDB
CMOV$B EFN,F.EFN           ;Event flag to use
CMOV$W OVBS,F.OVBS        ;Size of block buffer
CMOV$B MBCT,F.MBCT        ;No. of buffers
CMOV$B MBFG,F.MBFG        ;Read ahead or write behind
.ENDM FDBF$R
```

C.5.21 FDBK\$A Macro

FDBK\$A initializes the block access section of the FDB at assembly time.

```
.MACRO FDBK$A BKAD,BKSZ,BKVB,BKEF,BKST,BKDN
.MCALL FDOFF$,CBYTE$,CWORD$
FDOFF$ DEF$L
CWORD$ <BKAD>,F.BKDS+2
CWORD$ <BKSZ>,F.BKDS
CWORD$ <BKVB>,F.BKVB+2      ;Store low order only
CBYTE$ <BKEF>,F.BKEF
CWORD$ <BKST>,F.BKST
CWORD$ <BKDN>,F.BKDN
.ENDM FDBK$A
```

C.5.22 FDBK\$R Macro

FDBK\$R initializes the block access section of the FDB at run time.

```
.MACRO FDBK$R FDB,BKAD,BKSZ,BKVB,BKEF,BKST,BKDN
.MCALL LDFDB$,CMOV$B,CMOV$W,CMOV$2
LDFDB$ FDB
CMOV$W BKAD,F.BKDS+2      ;Block address-memory buffer
CMOV$W BKSZ,F.BKDS        ;Size of transfer in bytes
CMOV$2 BKVB,F.BKVB        ;Address of 2 word virtual
                                ;Block number, move both words
CMOV$B BKEF,F.BKEF        ;Event flag
CMOV$W BKST,F.BKST        ;ADR OF I/O STATUS BLOCK
CMOV$W BKDN,F.BKDN        ;Address of I/O done AST
.ENDM FDBK$R
```

C.5.23 FDBSZ\$ Macro

FDBSZ\$ defines the size of the FDB as a local symbol (S.FDB).

```
.MACRO FDBSZ$           ;Define S.FDB as size of FDB
.IIF DF,S.FDB,.MEXIT
.MCALL FDOFF$,DEF$L
FDOFF$ DEF$N           ;Invoke offset definitions but do
                                ; not actually define the offset
                                ; names
DEF$L S.FDB           ;Now define S.FDB
.ENDM FDBSZ$
```


MACRO EXPANSIONS

C.5.24 FDOP\$A Macro

FDOP\$A initializes the file open section of the FDB at assembly time.

```
.MACRO FDOP$A LUN, FNPT, DFNB, FACC, FACTRL
.MCALL FDOFF$, CBYTE$, CWORD$
FDOFF$ DEF$L
CBYTE$ <LUN>, F.LUN
CWORD$ <FNPT>, F.DSPT
CWORD$ <DFNB>, F.DFNB
CBYTE$ <FACC>, F.FACC
CWORD$ <FACTRL>, F.ACTL
.ENDM FDOP$A
```

C.5.25 FDOP\$R Macro

FDOP\$R initializes the file open section of the FDB at run time.

```
.MACRO FDOP$R FDB, LUN, FNPT, DFNB, FACC, FACTRL
.MCALL LDFDB$, CMOV$W, CMOV$B
LDFDB$ FDB
CMOV$B LUN, F.LUN
CMOV$W FNPT, F.DSPT
CMOV$W DFNB, F.DFNB
CMOV$B FACC, F.FACC
CMOV$W FACTRL, F.ACTL
.ENDM FDOP$R
```

C.5.26 FDRC\$A Macro

FDRC\$A initializes the record access section of the FDB at assembly time.

```
.MACRO FDRC$A RACC, URBA, URBS
.MCALL FDOFF$, CBYTE$, CWORD$
FDOFF$ DEF$L
CBYTE$ <RACC>, F.RACC
CWORD$ <URBA>, F.URBD+2
CWORD$ <URBS>, F.URBD
.ENDM FDRC$A
```

C.5.27 FDRC\$R Macro

FDRC\$R initializes the record access section of the FDB at run time.

```
.MACRO FDRC$R FDB, RACC, URBA, URBS
.MCALL LDFDB$, CMOV$W, CMOV$B
LDFDB$ FDB
CMOV$B RACC, F.RACC
CMOV$W URBA, F.URBD+2
CMOV$W URBS, F.URBD
.ENDM FDRC$R
```

MACRO EXPANSIONS

C.5.28 FDOF\$L Macro

FDOF\$L calls FDOFF\$ to locally define offsets. This action occurs only once.

```
.MACRO FDOF$L
.MCALL FDOFF$
FDOFF$ DEF$L
.ENDM FDOF$L
```

C.5.29 FDOFF\$ Macro

FDOFF\$ is the File Descriptor Block definition macro. It is called with one of the following macro names: DEF\$L, DEF\$G, or DEF\$N. DEF\$L defines local offsets, DEF\$G defines total offsets, and DEF\$N defines the size of the FDB.

```
.MACRO FDOFF$ OFFSET
.MCALL OFFSET,DEF$I,NBOFF$
NBOFF$ OFFSET
DEF$I 0 ;Initialize the definition macro
```

File Attribute Section

```
OFFSET F.RTYP,1 ;Record type
OFFSET F.RATT,1 ;Record attributes
OFFSET F.RSIZ,2 ;Record size
OFFSET F.HIBK,4 ;Highest virtual block no. allocated
OFFSET F.EFBK,4 ;End of file block number
OFFSET F.FFBY,2 ;First free byte in last block
OFFSET S.FATT ;Size of file attribute section
```

Record Access Section

```
OFFSET F.RACC,1 ,Record access
OFFSET F.RCTL,1 ;Record control
OFFSET F.BKDS ;Block I/O - buffer descriptor
OFFSET F.URBD,4 ;User's record buffer descriptor
OFFSET F.BKST ;Block I/O - I/O status block address
OFFSET F.NRBD,2 ;Next record buffer descriptor
OFFSET F.BKDN ;Block I/O - I/O done AST address
OFFSET ,2 ;2nd word of NRBD
OFFSET F.OVBS ;Override block buffer size
OFFSET F.NREC,2 ;Next record address in block buffer
OFFSET F.EOBB,2 ;End of block buffer
OFFSET F.CNTG ;Size in blocks of contiguous file
OFFSET F.RCNM,2 ;Record number for random records
OFFSET F.STBK ;Address to read in statistics block
OFFSET ,2 ;2nd word of RCNM
OFFSET F.ALOC,2 ;Allocate this much space when need
;To extend, + = contig, - = not
```

File Open Section

```
OFFSET F.LUN,1 ;Logical unit number
OFFSET F.FACC,1 ;File access
OFFSET F.DSPT,2 ;File descriptor pointer
OFFSET F.DFNB,2 ;Default file name block address
```

MACRO EXPANSIONS

Block Buffer Section

```

OFFSET F.BKEF           ;Block I/O - event flag number
OFFSET F.EFN,1         ;Event flag used in QIO
OFFSET F.BKPL,1        ;Bookkeeping bits
OFFSET F.ERR,2         ;1st byte error return code
                        ;2nd byte for QIO error indicator
OFFSET F.MBCT,1        ;Number of buffers to use (desired)
OFFSET F.MBCL,1        ;Number of buffers in use
OFFSET F.MBFG,1        ;Multiple buffering control flags
OFFSET F.BGBC,1        ;Big buffer block count (size in
                        ; blocks)
OFFSET F.VBSZ,2        ;Virtual block size in bytes
OFFSET F.BBFS,2        ;Block buffer size
OFFSET F.BKVB          ;Block I/O - virtual block number
OFFSET F.VBN,4         ;Virtual block number
OFFSET F.BDB,2         ;Block buffer descriptor block
OFFSET F.SPDV,2        ;Spool device indicator
OFFSET F.SPUN,1        ;Spool unit designator
OFFSET F.CHR,1         ;ACP volume characteristics summary
                        ; byte
OFFSET F.ACTL,2        ;Access control word
OFFSET F.SEQN,2        ;Sequence number for sequenced files

```

File Name Block Section

```

OFFSET F.FNB,S.FNB     ;Beginning of file name block
.IIF IDN,<OFFSET>,<DEF$N>,.MEXIT
.IF IDN,<OFFSET>,<DEF$G>
.GLOBL F.FNAM,F.FTYP,F.FVER,F.DVNM,F.UNIT
.ENDC
F.FNAM=N.FNAM+F.FNB
F.FTYP=N.FTYP+F.FNB
F.FVER=N.FVER+F.FNB
F.DVNM=N.DVNM+F.FNB
F.UNIT=N.UNIT+F.FNB
OFFSET S.FDB           ;Size of FDB
.MACRO FDOFF$ ARG
.ENDM FDOFF$
.ENDM FDOFF$

```

C.5.30 FDSOF\$ Macro

FDSOF\$ defines offsets relative to the file descriptor pointer, F.DSPT.

```

.MACRO FDSOF$ OFFSET
.MCALL OFFSET,DEF$I,DEF$L
DEF$I 0
OFFSET N.DEVD,4 ;Device string descriptor
OFFSET N.DIRD,4 ;Directory string descriptor
OFFSET N.FNMD,4 ;File name string descriptor
OFFSET S.FIDS
.IF IDN,<DEF$N>,<OFFSET>
DEF$L S.FIDS
.MEXIT
.ENDC
.MACRO FDSOF$ ARG
.ENDM FDSOF$
.ENDM FDSOF$

```

MACRO EXPANSIONS

C.5.31 FINIT\$ and FRSZ\$ Macros

FINIT\$ and FRSZ\$ initialize the file control services.

```
.MACRO FINIT$
.GLOBL .FINIT
JSR PC,.FINIT
.ENDM FINIT$
.MACRO FRSZ$ NFILES,BFSPAC,PSECT
.MCALL BDOFF$,DEF$L
.IF NDF,S.BFHD
BDOFF$ DEF$N
DEF$L S.BFHD
.ENDC
.GLOBL .FSRCB
.PSECT $$FSR1,GBL,OVR,D
.IF NB,<BFSPAC>
.BLKB NFILES*S.BFHD+<BFSPAC>
.IFF
.BLKB NFILES*<S.BFHD+512.>
.ENDC
.PSECT PSECT
.ENDM FRSZ$
```

C.5.32 FSROF\$ Macro

FSROF\$ generates the file storage region offsets.

```
.MACRO FSROF$ OFFSET
.MCALL OFFSET,DEF$I,DEF$L
```

Define the Offsets for \$\$FSR2

```
DEF$I 0
OFFSET ,4 ;List head for allocation
OFFSET A.BFSR,2 ;First address in FSR1
OFFSET A.EFSR,2 ;Last address in FSR1

***** DO NOT SEPARATE THE FOLLOWING 2 DEFINITIONS
OFFSET A.OWUI,2;UIC of owner (from task header)
OFFSET A.FIPR,2 ;Default file protection word
***** DO NOT SEPARATE THE PRECEEDING 2 DEFINITIONS

***** DO NOT SEPARATE THE FOLLOWING 3 DEFINITIONS
OFFSET A.DPB,24. ;QI/O DPB and scratch area
OFFSET A.IOST,4 ;Scratch I/O status block
OFFSET A.DFDR,24. ;Default directory information
***** DO NOT SEPARATE THE PRECEEDING 3 DEFINITIONS

OFFSET A.DFBC,2 ;Default buffer count (multiple
;buffering)
OFFSET A.DFUI,2 ;Default UIC (task UIC)
OFFSET S.FSR2 ;Size of file storage region 2
DEF$L S.FSR2
```

Define Offsets Relative to Beginning of the Default Directory Info

```
DEF$I 0
OFFSET ,14.
OFFSET D.DFID,6 ;Default directory ID
OFFSET D.DFDV,2 ;Device name for default
;directory ID
```

MACRO EXPANSIONS

```
OFFSET D.DFUN,2 ;Unit number for default
;directory ID
.IIF IDN,<DEF$N>,<OFFSET>,.MEXIT
.MACRO FSROF$ ARG
.ENDM FSROF$
.ENDM FSROF$
```

C.5.33 GET\$ Macro

GET\$ gets a record from a file.

```
.MACRO GET$ FDB,INADR,MAXCNT,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W INADR,F.URBD+2 ;User record buffer address
CMOV$W MAXCNT,F.URBD ;User record buffer size
.GLOBL .GET
JSR PC,.GET
ERR$ ERR
.ENDM GET$
```

C.5.34 GET\$R Macro

GET\$R gets a record in random mode.

```
.MACRO GET$R FDB,INADR,MAXCNT,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W INADR,F.URBD+2 ;User record buffer address
CMOV$W MAXCNT,F.URBD ;User record buffer size
CMOV$W LRCNM,F.RCNM+2 ;Low order record number
CMOV$W HRCNM,F.RCNM ;High order record number
.GLOBL .GET
JSR PC,.GET
ERR$ ERR
.ENDM GET$R
```

C.5.35 GET\$S Macro

GET\$S gets a record in strictly sequential mode.

```
.MACRO GET$S FDB,INADR,MAXCNT,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W INADR,F.URBD+2 ;User record buffer address
CMOV$W MAXCNT,F.URBD ;User record buffer size
.GLOBL .GETSQ
JSR PC,.GETSQ
ERR$ ERR
.ENDM GET$S
```

C.5.36 NBOF\$L Macro

NBOF\$L calls NBOFF\$ to define offsets as local symbols. This action occurs only once.

MACRO EXPANSIONS

```
.MACRO NBOFF$L
.MCALL NBOFF$
NBOFF$ DEF$L
.ENDM NBOFF$L
```

If the FDB parameter is null or R0, LDFDB\$ does not move FDB to R0. Otherwise, it does move the FDB to R0.

```
.MACRO LDFDB$ FDB
.IIF B,FDB,.MEXIT ;If null then exit
.NTYPE PAR$$$ ,FDB
.IIF EQ,PAR$$$ ,.MEXIT ;If R0 then exit
MOV FDB,R0 ;Else generate the move
.ENDM LDFDB$
```

C.5.37 NBOFF\$ Macro

NBOFF\$ generates the file name block offsets.

```
.MACRO NBOFF$ OFFSET
.MCALL OFFSET,DEF$I,DEF$L
DEF$I 0
OFFSET S.FNAM ;Define as global if parameter
OFFSET S.FTYP ;Is DEF$G
OFFSET S.FNTY
OFFSET S.FNBW
S.FNAM=6 ;Size of filename in bytes
S.FTYP=2 ;Size of file type in bytes
S.FNTY=<S.FNAM+S.FTYP>/2 ;Size of filename + type in words
OFFSET N.FID,6 ;File ID
OFFSET N.FNAM,S.FNAM ;Filename
OFFSET N.FTYP,S.FTYP ;Type
OFFSET N.FVER,2 ;Version
OFFSET S.NFEN ;Size of name file entry in bytes
DEF$L S.NFEN ;Force at least a local definition
OFFSET N.STAT,2 ;Status
OFFSET N.NEXT,2 ;Temp cell for find next
OFFSET N.DID,6 ;Directory ID
OFFSET N.DVNM,2 ;Device name in ASCII
OFFSET N.UNIT,2 ;Unit number
OFFSET S.FNB ;Size of FNB in bytes
DEF$L S.FNB ;Force local definition at least
S.FNBW=S.FNB/2 ;Size of FNB in words
.IIF IDN,<DEF$N>,<OFFSET>,.MEXIT ;Don't redefine the macro
.MACRO NBOFF$ ARG
.ENDM NBOFF$
.ENDM NBOFF$
```

C.5.38 NMBLK\$ Macro

NMBLK\$ defines the file name block at assembly time.

```
.MACRO NMBLK$ FNAME,FTYPE,VERS,DEVNAM,UNIT
.MCALL RAD50$,CWORD$,NBOFF$
NBOFF$ DEF$L ;Define name block offsets locally
.IIF NDF,...PC1,...PC1=0
...PC3=...PC1 ;Preserve ...PC1
...PC1=.
...PC1+N.FNAM
RAD50$ <FNAME>,S.FNAM/2
```

MACRO EXPANSIONS

```
. = ...PC1+N.FTYP
RAD50$ <FTYPE>,S.FTYP/2
CWORD$ <VERS>,N.FVER
.IF NB,DEVNAM
. = ...PC1+N.DVNM
.WORD "DEVNAM
.ENDC
C.,ORD$ <UNIT>,N.UNIT
. = ...PC1+S.FNB
...PC1=...PC3           ;Restore ...PC1
.ENDM NMBLK$
```

C.5.39 OPEN\$ Macro

OPEN\$ opens a file.

```
.MACRO OPEN$ FDB,FACC,LUN, FNPT,DFNB,RACC,URBA,URBS,ERR
.MCALL FDOP$R,FDRC$R,ERR$,CMOV$B
FDOP$R FDB,LUN, FNPT,DFNB,FACC
FDRC$R ,RACC,URBA,URBS
.GLOBL .OPEN
JSR PC,.OPEN
ERR$ ERR
.ENDM OPEN$
```

C.5.40 OPEN\$A Macro

OPEN\$A opens a file for appending.

```
.MACRO OPEN$A FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FO.APD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OPEN$A
```

C.5.41 OPEN\$M Macro

OPEN\$M opens a file for modification.

```
.MACRO OPEN$M FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FO.MFY,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OPEN$M
```

C.5.42 OPEN\$R Macro

OPEN\$R opens a file for reading.

```
.MACRO OPEN$R, FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FO.RD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OPEN$R
```

MACRO EXPANSIONS

C.5.43 OPEN\$U Macro

OPEN\$U opens a file for updating.

```
.MACRO OPEN$U FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FO.UPD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPEN$U
```

C.5.44 OPEN\$W Macro

OPEN\$W opens a file for writing.

```
.MACRO OPEN$W FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FO.WRT,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPEN$W
```

C.5.45 OPNS\$A Macro

OPNS\$A opens a shared file for appending.

```
.MACRO OPNS$A FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FA.SHR!FO.APD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNS$A
```

C.5.46 OPNS\$M Macro

OPNS\$M opens a shared file for modification.

```
.MACRO OPNS$M FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FA.SHR!FO.MFY,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNS$M
```

C.5.47 OPNS\$R Macro

OPNS\$R opens a shared file for reading.

```
.MACRO OPNS$R FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FA.SHR!FO.RD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNS$R
```

C.5.48 OPNS\$U Macro

OPNS\$U opens a shared file for updating.

```
.MACRO OPNS$U FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OPEN$
OPEN$ FDB,#FA.SHR!FO.UPD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNS$U
```


MACRO EXPANSIONS

C.5.49 OPNS\$W Macro

OPNS\$W opens a shared file for writing.

```
.MACRO OPNS$W FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPNS$
OPNS$ FDB,#FA.SHR!FO.WRT,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNS$W
```

C.5.50 OPNT\$D Macro

Create, access, and mark for delete a new file not entered in a directory. It will be deleted when closed (deaccessed) and will be properly deleted if program should terminate abnormally.

```
.MACRO OPNT$D FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPNT$W
OPNT$W FDB,LUN, FNPT,RACC,URBA,URBS
BCS .+6
JSR PC,.MRKDL
ERR$ ERR
.ENDM OPNT$D
```

C.5.51 OPNT\$W Macro

OPNT\$W creates and accesses a new file and does enter it in a directory. The file may be closed and reopened by file ID. The file should be deleted before the program exits otherwise it remains in the index file with no directory entry.

```
.MACRO OPNT$W FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OPNS$
OPNS$ FDB,#FA.TMP!FO.WRT,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OPNT$W
```

C.5.52 OFID\$ Macro

OFID\$ opens a file by file ID.

```
.MACRO OFID$ FDB,FACC,LUN, FNPT,DFNB,RACC,URBA,URBS,ERR
.MCALL FDOP$R,FDRC$R,ERR$,CMOV$B
FDOP$R FDB,LUN, FNPT,DFNB,FACC
FDRC$R ,RACC,URBA,URBS
.GLOBL .OPFID
JSR PC,.OPFID
ERR$ ERR
.ENDM OFID$
```

C.5.53 OFID\$A Macro

OFID\$A opens a file by file ID for appending.

```
.MACRO OFID$A FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFID$
OFID$ FDB,#FO.APD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFID$A
```

MACRO EXPANSIONS

C.5.54 OFID\$M Macro

OFID\$M opens a file for modification by file ID.

```
.MACRO OFID$M FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OFID$
OFID$ FDB,#FO.MFY,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OFID$M
```

C.5.55 OFID\$R Macro

OFID\$R opens a file for reading by file ID.

```
.MACRO OFID$R FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OFID$
OFID$ FDB,#FO.RD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OFID$R
```

C.5.56 OFID\$U Macro

OFID\$U opens a file for updating by file ID.

```
.MACRO OFID$U FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OFID$
OFID$ FDB,#FO.UPD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OFID$U
```

C.5.57 OFID\$W Macro

OFID\$W opens a file for writing by file ID.

```
.MACRO OFID$W FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL OFID$
OFID$ FDB,#FO.WRT,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM OFID$W
```

C.5.58 OFNB\$ Macro

OFNB\$ opens a file by file name block.

```
.MACRO OFNB$ FDB,FACC,LUN,FNPT,DFNB,RACC,URBA,URBS,ERR
.MCALL FDOP$R,FDRC$R,ERR$,CMOV$B
FDOP$R FDB,LUN,FNPT,DFNB,FACC
FDRC$R ,RACC,URBA,URBS
.GLOBAL .OPFNB
JSR PC,.OPFNB
ERR$ ERR
.ENDM OFNB$
```

MACRO EXPANSIONS

C.5.59 OFNB\$A Macro

OFNB\$A opens a file by file name block for appending.

```
.MACRO OFNB$A FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFNB$
OFNB$ FDB,#FO.APD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFNB$A
```

C.5.60 OFNB\$M Macro

OFNB\$M opens a file by file name block for modification.

```
.MACRO OFNB$M FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFNB$
OFNB$ FDB,#FO.MFY,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFNB$M
```

C.5.61 OFNB\$R Macro

OFNB\$R opens a file by file name block for reading.

```
.MACRO OFNB$R FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFNB$
OFNB$ FDB,#FO.RD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFNB$R
```

C.5.62 OFNB\$U Macro

OFNB\$U opens a file by file name block for updating.

```
.MACRO OFNB$U FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFNB$
OFNB$ FDB,#FO.UPD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFNB$U
```

C.5.63 OFNB\$W Macro

OFNB\$W opens a file by file name block for writing.

```
.MACRO OFNB$W FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL OFNB$
OFNB$ FDB,#FO.WRT,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM OFNB$W
```

C.5.64 PUT\$ Macro

PUT\$ puts a record in random or sequential mode.

```
.MACRO PUT$ FDB,OUTADR,OUTCNT,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W OUTADR,F.NRBD+2 ;Next record buffer address
CMOV$W OUTCNT,F.NRBD ;Next record buffer size
```

MACRO EXPANSIONS

```
.GLOBL .PUT
JSR PC,.PUT
ERR$ ERR
.ENDM PUT$
```

C.5.65 PUT\$R Macro

PUT\$R puts a record in random mode.

```
.MACRO PUT$R FDB,OUTADR,OUTCNT,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W OUTADR,F.NRBD+2 ;Next record buffer address
CMOV$W OUTCNT,F.NRBD ;Next record buffer size
CMOV$W LRCNM,F.RCNM+2 ;Low order record number
CMOV$W HRCNM,F.RCNM ;High order record number
.GLOBL .PUT
JSR PC,.PUT
ERR$ ERR
.ENDM PUT$R
```

C.5.66 PUT\$\$ Macro

PUT\$\$ puts a record in strictly sequential mode.

```
.MACRO PUT$$ FDB,OUTADR,OUTCNT,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W OUTADR,F.NRBD+2 ;Next record buffer address
CMOV$W OUTCNT,F.NRBD ;Next record buffer size
.GLOBL .PUTSQ
JSR PC,.PUTSQ
ERR$ ERR
.ENDM PUT$$
```

C.5.67 RAD50\$ Macro

```
.MACRO RAD50$ STRING,MAXWRD
...PC2=.
.RAD50 /STRING/
.IF GT,<.-...PC2>-^O<2*MAXWRD>
.ERROR ;String RAD50 - string too long,
; truncated
.=^O<...PC2+<2*MAXWRD>>
.MEXIT
.ENDC
.REPT ^O<MAXWRD-<<.-...PC2>/2>>
.WORD 0
.ENDR
.ENDM RAD50$
```

MACRO EXPANSIONS

C.5.68 READ\$ Macro

READ\$ reads a virtual block.

```
.MACRO READ$ FDB,BKAD,BKSZ,BKVB,BKEF,BKST,BKDN,ERR
.MCALL FDBK$R,ERR$
FDBK$R FDB,BKAD,BKSZ,BKVB,BKEF,BKST,BKDN
.GLOBL .READ
JSR PC,.READ
ERR$ ERR
.ENDM READ$
```

C.5.69 TRUNC\$ Macro

TRUNC\$ truncates a file.

```
.MACRO TRUNC$ FDB,ERR
.MCALL LDFDB$,ERR$
LDFDB$ FDB
.GLOBL .TRUNC
JSR PC,.TRUNC
ERR$ ERR
.ENDM TRUNC$
```

C.5.70 WAIT\$ Macro

WAIT\$ waits for I/O completion after a READ\$ or WRITE\$ macro execution.

```
.MACRO WAIT$ FDB,EFN,BKST,ERR
.MCALL LDFDB$,CMOV$W,CMOV$B,ERR$
LDFDB$ FDB
CMOV$B EFN,F.EFN
CMOV$W BKST,F.BKST
.GLOBL .WAIT
JSR PC,.WAIT
ERR$ ERR
.ENDM WAIT$
```

C.5.71 WRITE\$ Macro

WRITE\$ writes a virtual block.

```
.MACRO WRITE$ FDB,BKAD,BKSZ,BKVB,BKEF,BKST,BKDN,ERR
.MCALL FDBK$R,ERR$
FDBK$R FDB,BKAD,BKSZ,BKVB,BKEF,BKST,BKDN
.GLOBL .WRITE
JSR PC,.WRITE
ERR$ ERR
.ENDM WRITE$
```

MACRO EXPANSIONS

C.6 NETWORK SYMBOL DEFINITION MACRO

C.6.1 COMDF\$ Macro

Common Symbol Module for M/D Networks Implementation

```
.MACRO COMDF$ DEF
.NLIST
.IF IDN <DEF>,<DEF$G>
.GLOBL B.OT,B.UN,B.RC,B.RN,B.RP,B.RU,B.NA,B.UA,B.MN,B.MX
.GLOBL CT.IN,CT.CN,LB.N,LB.E
.GLOBL NT.IN,NT.NS
.GLOBL CR.UR,CR.NR,CR.MX
.GLOBL CR.N0,CR.N1,CR.N2,CR.N3,CR.N4,CR.N5,CR.N6,CR.N9,
      CR.N10,CR.N11
.GLOBL CR.N7,CR.N8
.GLOBL CR.DI,CR.DT,CR.DA
.GLOBL IA.ISM,IA.ABO,IA.DIS,IA.NFW
.GLOBL CN.IN,CN.OU,CN.UT,CN.NT
.GLOBL OB.TA,OB.FS,OB.SQ,OB.BO,OB.TC,OB.RD,OB.RA,OB.ND,
      OB.DD,OB.BM
.GLOBL OB.TH,OB.TB,OB.TI,OB.CR,OB.LP,OB.PR,OB.PP,OB.PL,OB.MT
.GLOBL OB.DT,OB.CS,OB.CP,OB.FH,OB.MH,OB.FL
.ENDC
B.OT = 0                ;Object type value
B.UN = 1                ;Object type unit number
B.RC = B.OT+2          ;Offset to remote connect #
B.RP = B.RN+4          ;Offset to remote process name [RAD50]
B.RU = B.RP+4          ;Offset to remote UIC [.WORD
                        ; GROUP,USER]
B.NA = B.RU+4          ;Offset to # of user arguments
B.UA = B.NA+2          ;Offset to first user argument
B.MN = B.UA            ;Minimum size of buffer [0 arguments]
B.MX = B.MN+10        ;Maximum size of buffer [8. arguments]
NT.IN = 10             ;Send this message at AST level [XMIT]
NT.NS = 1              ;No status message required [RECV]
CR.UR = 2              ;Connect reject [remote user
                        ; rejection]
CR.NR = 3              ;Connect reject [remote NCS rejection]
CR.MX = 6              ;Maximum reason
CR.N0 = 0              ;Reject by NCS
CR.N1 = 1              ;Too many connects to remote node
CR.N2 = 2              ;Too many connects to remote process
CR.N3 = 3              ;Process does not exist on this node
CR.N4 = 4              ;Object type does not exist on this
                        ; node
CR.N5 = 5              ;Destination address in use
CR.N6 = 6              ;Node shutting down
CR.N7 = 7
CR.N8 = 8.
CR.N9 = 11             ;Invalid task name
CR.N10 = 12            ;Invalid qualifier
CR.N11 = 13            ;Bad destination parameter
CR.DI = 4              ;Disconnect initiate
CR.DT = 5              ;Disconnect terminate [CONFIRM]
CR.DA = 6              ;Disconnect abort
LB.N=0                 ;Loopback normal
LB.E=2                 ;Loopback echo mode
CT.IN = 0              ;Connect initiate
CT.CN = 1              ;Connect confirm
IA.ABO = -5            ;Disconnect abort [partner terminated]
```

MACRO EXPANSIONS

```

IA.DIS = -7           ;Disconnect requested [partner did
IA.NFW = 177673      ; disc QIO]
                     ;Topology change -- partner no longer
                     ; reachable
IA.ISM = 1           ;Interrupt semaphore from partner
CN.IN = 1            ;If set: inhibit incoming connections
CN.OU = 2            ;If set: inhibit outgoing connections
CN.UT = 4            ;If set: user tap
CN.NT = 10           ;If set: NCL tap
OB.TA = 0            ;Task
OB.FS = 1            ;File system [dap]
OB.SQ = 2            ;Sequential devices [DAP]
OB.BO = 3            ;Boot task
OB.TC = 4            ;Terminal control task
OB.RD = 5            ;Remote directives
OB.RA = 6            ;Resource allocator
OB.ND = 7            ;Network directory service
OB.DD = 10           ;DDCMP link
OB.BM = 11           ;Boot message link
;12-17 RESERVED
OB.TH = 20           ;Terminal handler
OB.TB = 21           ;Terminal : block mode
OB.TI = 22           ;Terminal : interactive
OB.CR = 23           ;Card reader
OB.LP = 24           ;Line printer
OB.PR = 25           ;PAPER TAPE READER
OB.PP = 26           ;Paper tape punch
OB.PL = 27           ;Plotter
OB.MT = 30           ;Magnetic tape
OB.DT = 31           ;DECTAPE
OB.CS = 32           ;Cassette
OB.CP = 33           ;Card punch
OB.FH = 34           ;Fixed head disk
OB.MH = 35           ;Moving head disk
OB.FL = 36           ;Floppy disk
.MACRO COMDF$ A
.ENDM
.LIST
.ENDM
.END

```

C.7 PROGRAM LOGICAL ADDRESS SPACE EXTENSION MACRO EXPANSIONS

C.7.1 ATRG\$, ATRG\$C, and ATRG\$\$ Macros

These macros generate the code to attach a region.

```

.MACRO ATRG$ RDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 57.,2
.WORD RDB
.ENDC
.IF NDF A.TRBA
OFF$
OFF$ A.TRBA,2
.ENDC
.ENDM ATRG$

```

MACRO EXPANSIONS

```
.MACRO ATRG$C RDB,CS,ERR
    .MCALL ATRG$,DIR$
    .IF NDF $$$GLB
    .PSECT $DPB$$
$$$=.
    .IFTF
    ATRG$ RDB
    .IFT
    .PSECT CS
    DIR$ #$$$ ,ERR
    .ENDC
    .ENDM ATRG$C

.MACRO ATRG$$ RDB,ERR
    .MCALL DIR$,MOV$
    MOV$ RDB
    MOV (PC)+,-(SP)
    .BYTE 57.,2
    DIR$ ,ERR
    .ENDM ATRG$$
```

C.7.2 .BLK., .BLKB., and .BLKW. Macros

These macros define the block offset symbol. The Program Logical Address Space definition macros use these macros.

```
.MACRO .BLKB. NUM,SYM,GBL
    .IF NB <SYM>
    .IF IDN <DEF$G>,<GBL>
SYM==$$$
    .IFF
SYM=$$$
    .ENDC
    .ENDC
$$$=$$$+NUM
    .ENDM .BLKB.
.MACRO .BLKW. NUM,SYM,GBL
    .IF NB <SYM>
    .IF IDN <DEF$G>,<GBL>
SYM==$$$
    .IFF
SYM=$$$
    .ENDC
    .ENDC
$$$=$$$+<2*NUM>
    .ENDM .BLKW.

.MACRO .BLK.
$$$=0
    .ENDM .BLK.
```

C.7.3 CRAW\$, CRAW\$C, CRAW\$\$ Macro

These macros generate the code to create an address window.

```
.MACRO CRAW$ WDB
    .MCALL OFF$
    .IF NDF $$$GLB
    .BYTE 117.,2
```


MACRO EXPANSIONS

```
.WORD WDB
.ENDC
.IF NDF C.RABA
OFF$
OFF$ C.RABA,2
.ENDC
.ENDM CRAW$

.MACRO CRAW$C WDB,CS,ERR
.MCALL CRAW$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CRAW$ WDB
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM CRAW$C

.MACRO CRAW$$ WDB,ERR
.MCALL DIR$,MOV$
MOV$ WDB
MOV (PC)+,-(SP)
.BYTE 117.,2
DIR$ ,ERR
.ENDM CRAW$$
```

C.7.4 CRRG\$, CRRG\$C, and CRRG\$\$ Macros

These macros generate the code to create a region.

```
.MACRO CRRG$ RDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 55.,2
.WORD RDB
.ENDC
.IF NDF C.RRBA
OFF$
OFF$ C.RRBA,2
.ENDC
.ENDM CRRG$

.MACRO CRRG$C RDB,CS,ERR
.MCALL CRRG$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
CRRG$ RDB
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM CRRG$C

.MACRO CRRG$$ RDB,ERR
.MCALL DIR$,MOV$
MOV$ RDB
MOV (PC)+,-(SP)
```

MACRO EXPANSIONS

```
.BYTE 55.,2
DIR$ ,ERR
.ENDM CRRG$$
```

C.7.5 DTRG\$, DTRG\$C, and DTRG\$\$ Macros

These macros generate the code to detach a region.

```
.MACRO DTRG$ RDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 59.,2
.WORD RDB
.ENDC
.IF NDF D.TRBA
OFF$
OFF$ D.TRBA,2
.ENDC
.ENDM DTRG$
```

```
.MACRO DTRG$C RDB,CS,ERR
.MCALL DTRG$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
DTRG$ RDB
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM DTRG$C
```

```
.MACRO DTRG$$ RDB,ERR
.MCALL DIR$,MOV$
MOV$ RDB
MOV (PC)+,-(SP)
.BYTE 59.,2
DIR$ ,ERR
.ENDM DTRG$$
```

C.7.6 ELAW\$, ELAW\$C, and ELAW\$\$ Macros

These macros generate the code to eliminate an address window.

```
.MACRO ELAW$ WDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 119.,2
.WORD WDB
.ENDC
.IF NDF E.LABA
OFF$
OFF$ E.LABA,2
.ENDC
.ENDM ELAW$
```

```
.MACRO ELAW$C WDB,CS,ERR
.MCALL ELAW$,DIR$
```

MACRO EXPANSIONS

```
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
ELAW$ WDB
.IFT
.PSECT CS
DIR$ $$$$,ERR
.ENDC
.ENDM ELAW$C

.MACRO ELAW$$ WDB,ERR
.MCALL DIR$,MOV$
MOV$ WDB
MOV (PC)+,-(SP)
.BYTE 119.,2
DIR$ ,ERR
.ENDM ELAW$$
```

C.7.7 GMCX\$, GMCX\$C, and GMCX\$\$ Macros

The macros generate the code to get the mapping context.

```
.MACRO GMCX$ WVEC
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 113.,2
.WORD WVEC
.ENDC
.IF NDF G.MCVA
OFF$
OFF$ G.MCVA,2
.ENDC
.ENDM GMCX$
.MACRO GMCX$C WVEC,CS,ERR
.MCALL GMCX$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GMCX$ WVEC
.IFT
.PSECT CS
DIR$ $$$$,ERR
.ENDC
.ENDM GMCX$C
.MACRO GMCX$$ WVEC,ERR
.MCALL DIR$,MOV$
MOV$ WVEC
MOV (PC)+,-(SP)
.BYTE 113.,2
DIR$ ,ERR
.ENDM GMCX$$
```

C.7.8 GREG\$, GREG\$C, and GREG\$\$ Macros

These macros get region parameters. They are a special case of the get partition parameters macros.

MACRO EXPANSIONS

```
.MACRO GREG$ RID,BUF
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 65.,4
.WORD 0,RID
.WORD BUF
.ENDC
.IF NDF G.RGID
.NLIST
OFF$
OFF$ ,2
OFF$ G.RGID,2
OFF$ G.RGBA,2
.IF NDF G.RGRB
$$$OST=0
.IRP X,<<G.RGRB,2>,<G.RGRS,2>,<G.RGFW,2>>
OFF$ X
.ENDM
.ENDC
.LIST
.ENDC
.ENDM GREG$
.MACRO GREG$C RID,BUF,CS,ERR
.MCALL GREG$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
GREG$ RID,BUF
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM GREG$C
.MACRO GREG$$ RID,BUF,ERR
.MCALL MOV$,DIR$,OFF$
MOV$ BUF
MOV$ RID
CLR -(SP)
MOV (PC)+,-(SP)
.BYTE 65.,4
DIR$ ,ERR
.IF NDF G.RGRB
.NLIST
$$$OST=0
.IRP X,<<G.RGRB,2>,<G.RGRS,2>,<G.RGFW,2>>
OFF$ X
.ENDM
.LIST
.ENDC
.ENDM GREG$$
```

C.7.9 MAP\$, MAP\$C, MAP\$\$ Macros

These macros generate the code to map an address window.

```
.MACRO MAP$ WDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 121.,2
.WORD WDB
.ENDC
```

MACRO EXPANSIONS

```

    .IF NDF M.APBA
    OFF$
    OFF$ M.APBA,2
    .ENDC
    .ENDM MAP$

    .MACRO MAP$C WDB,CS,ERR
    .MCALL MAP$,DIR$
    .IF NDF $$$GLB
    .PSECT $DPB$$
$$$=.
    .IFTF
    MAP$ WDB
    .IFT
    .PSECT CS
    DIR$ #$$$ ,ERR
    .ENDC
    .ENDM MAP$C

    .MACRO MAP$$ WDB,ERR
    .MCALL DIR$,MOV$
    MOV$ WDB
    MOV (PC)+,-(SP)
    .BYTE 121.,2
    DIR$ ,ERR
    .ENDM MAP$$

```

C.7.10 RDBBK\$ Macro

RDBBK\$ generates a region definition block.

```

    .MACRO RDBBK$ SIZ,NAM,PAR,STS,PRO
    .MCALL RDBDF$,R50$
    RDBDF$                                ;Define region definition block
                                         ; symbols
    .WORD 0                                ;Region ID
    .WORD SIZ                              ;Size of region (32w blocks)
    R50$ NAM                               ;Name of region (RAD50)
    R50$ PAR                               ;Region's main partition name (RAD50)
    .WORD STS                              ;Region status word
    .WORD PRO                              ;Region protection word
    .ENDM RDBBK$

```

C.7.11 RREF\$, RREF\$C, and RREF\$\$ Macros

These macros generate the code to receive by reference.

```

    .MACRO RREF$ WDB
    .MCALL OFF$
    .IF NDF $$$GLB
    .BYTE 81.,2
    .WORD WDB
    .ENDC
    .IF NDF R.REBA
    OFF$
    OFF$ R.REBA,2
    .ENDC
    .ENDM RREF$

```

MACRO EXPANSIONS

```
.MACRO RREF$C WDB,CS,ERR
.MCALL RREF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
RREF$ WDB
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM RREF$C
```

```
.MACRO RREF$$ WDB,ERR
.MCALL DIR$,MOV$
MOV$ WDB
MOV (PC)+,-(SP)
.BYTE 81.,2
DIR$ ,ERR
.ENDM RREF$$
```

C.7.12 SREF\$, SREF\$C, and SREF\$\$ Macros

These macros generate the code to send by reference.

```
.MACRO SREF$ TASK,WDB,EFN
.MCALL R50$,OFF$
.IF NDF $$$GLB
.BYTE 69.,5
R50$ TASK
.WORD EFN
.WORD WDB
.ENDC
.IF NDF S.RETN
.IRP X,<,<S.RETN,4>,<S.REEF,2>,<S.REBA,2>>
OFF$ X
.ENDM
.ENDC
.ENDM SREF$
.MACRO SREF$C TASK,WDB,EFN,CS,ERR
.MCALL SREF$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SREF$ TASK,WDB,EFN
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SREF$C

.MACRO SREF$$ TSKADR,WDB,EFN,ERR
.MCALL DIR$,MOV$,RFA$
MOV$ WDB
MOV$ EFN
RFA$ TSKADR
MOV (PC)+,-(SP)
.BYTE 69.,5
DIR$ ,ERR
.ENDM SREF$$
```

MACRO EXPANSIONS

C.7.13 SRRA\$, SRRA\$C, and SRRA\$\$ Macros

These macros generate the code for the SPECIFY RECEIVE BY REFERENCE AST directive.

```
.MACRO SRRA$ AST
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 21.,2
.WORD AST
.ENDC
.IF NDF S.RRAE
.NLIST
OFF$
OFF$ S.RRAE,2
.LIST
.ENDC
.ENDM SRRA$

.MACRO SRRA$C AST,CS,ERR
.MCALL SRRA$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
SRRA$ AST
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM SRRA$C
.MACRO SRRA$$ AST,ERR
.MCALL MOV$,DIR$
MOV$ AST
MOV (PC)+,-(SP)
.BYTE 21.,2
DIR$ ,ERR
.ENDM SRRA$$
```

C.7.14 UMAP\$, UMAP\$C, and UMAP\$\$ Macros

These macros generate the code to unmap an address window.

```
.MACRO UMAP$ WDB
.MCALL OFF$
.IF NDF $$$GLB
.BYTE 123.,2
.WORD WDB
.ENDC
.IF NDF U.MABA
OFF$
OFF$ U.MABA,2
.ENDC
.ENDM UMAP$

.MACRO UMAP$C WDB,CS,ERR
.MCALL UMAP$,DIR$
.IF NDF $$$GLB
.PSECT $DPB$$
$$$=.
.IFTF
UMAP$ WDB
```

MACRO EXPANSIONS

```
.IFT
.PSECT CS
DIR$ #$$$ ,ERR
.ENDC
.ENDM UMAP$C
```

```
.MACRO UMAP$$ WDB,ERR
.MCALL DIR$,MOV$
MOV$ WDB
MOV (PC)+,-(SP)
.BYTE 123.,2
DIR$ ,ERR
.ENDM UMAP$$
```

C.7.15 WDBBK\$ Macro

WDBBK\$ generates a window definition block.

```
.MACRO WDBBK$ APR,SIZ,RID,OFF,LEN,STS,SRB
.MCALL WDBDF$
WDBDF$ ;Define window definition block
; symbols
.BYTE 0,APR ;Window ID / base APR.
.WORD 0 ;Virtual base address (bytes)
.WORD SIZ ;Window size (32w blocks)
.WORD RID ;Region ID
.WORD OFF ;Offset in partition (32w blocks)
.WORD LEN ;Length to map (32w blocks)
.WORD STS ;Status word
.WORD SRB ;Send/receive buffer virtual address

.ENDM WDBBK$
```

C.8 RELATIVE FILES MACROS - EXPANSIONS

C.8.1 RCLOS\$ Macro

RCLOS\$ closes a file.

```
.MACRO RCLOS$ FDB,ERR
.MCALL LDFDB$,ERR$
LDFDB$ FDB
JSR PC, .RCLOS
ERR$ ERR
.ENDM RCLOS$
```

C.8.2 RFDBT\$ Macro

RFDBT\$ defines the relative file descriptor bits.

```
.MACRO RFDBT$ GLOBL
.MCALL DEFIN$
...GBL=0
.IIF IDN,<GLOBL>,<DEF$G>, ...GBL=1
```


MACRO EXPANSIONS

Relative File Flags Byte

```
DEFIN$ RF.WRT,1      ;Set if bit table record
                    ; buffer dirty
DEFIN$ RF.EXT,2      ;Set if extend is allowed
DEFIN$ RF.BEX,4      ;Set if extend attempted when
                    ; restricted
```

Relative File Value Byte

```
DEFIN$ RB.NEW,1      ;New bit value to set
DEFIN$ RB.NCH,2      ;1 if not changing value
DEFIN$ RB.VAL,4      ;Bit value read
.MACRO RFDBT$ ARG
.ENDM RFDBT$
.ENDM RFDBT$
```

C.8.3 RFIND\$ Macro

RFIND\$ finds the next existing record.

```
.MACRO RFIND$ FDB,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W LRCNM,R.RCNM+2
CMOV$W HRCNM,R.RCNM
JSR PC,.RFIND
ERR$ ERR
.ENDM RFIND$
```

C.8.4 RFOF\$L Macro

RFOF\$L locally defines relative file record offsets.

```
.MACRO RFOF$L
.MCALL RFOFF$
RFOFF$ DEF$L
.ENDM RFOF$L
```

C.8.5 RFOFF\$ Macro

RFOFF\$ defines the offsets for the relative file extension of the FDB.

```
.MACRO RFOFF$ OFFSET
.MCALL OFFSET,DEF$I,FDBSZ$
FDBSZ$
DEF$I S.FDB
```

Relative File Record Section

```
OFFSET R.RCNM,4      ;Relative record number
OFFSET R.CLSZ,4      ;Cluster size
OFFSET R.FLAG,1      ;Flags byte
OFFSET R.BTVL,1      ;Value byte
OFFSET R.BTSZ,2      ;Bit table record size
OFFSET R.BTBF,2      ;Bit table record buffer
OFFSET R.BTN,2       ;Bit table number
OFFSET R.BTRC,2      ;Bit table record number
```

MACRO EXPANSIONS

```
OFFSET S.RFD ;Size of RFDB
.IIF IDN,<DEF$N>,<OFFSET>, .MEXIT
.MACRO RFOFF$ ARG
.ENDM RFOFF$
.ENDM RFOFF$
```

C.8.6 RGET\$ Macro

RGET\$ gets a relative file.

```
.MACRO RGET$ FDB,INADR,MAXCNT,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W INADR,F.URBD+2
CMOV$W MAXCNT,F.URBD
CMOV$W LRCNM,R.RCNM+2
CMOV$W HRCNM,R.RCNM
JSR PC,.RGET
ERR$ ERR
.ENDM RGET$
```

C.8.7 ROPN\$ Macro

ROPN\$ opens a file.

```
.MACRO ROPN$ FDB,FACC,LUN,FNPT,DFNB,RACC,URBA,URBS,ERR
.MCALL FDOP$R,FDRC$R,ERR$,CMOV$B
FDOP$R FDB,LUN,FNPT,DFNB,FACC
FDRC$R ,RACC,URBA,URBS
.GLOBL .ROPEN
JSR PC,.ROPEN
ERR$ ERR
.ENDM ROPN$
```

C.8.8 ROPN\$A Macro

ROPN\$A opens a file for appending.

```
.MACRO ROPN$A FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FO.APD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPN$A
```

C.8.9 ROPN\$M Macro

ROPN\$M opens a file for modification.

```
.MACRO ROPN$M FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FO.MFY,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPN$M
```

MACRO EXPANSIONS

C.8.10 ROPN\$R Macro

ROPN\$R opens a file for reading.

```
.MACRO ROPN$R FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FO.RD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPN$R
```

C.8.11 ROPN\$U Macro

ROPN\$U opens a file for updating.

```
.MACRO ROPN$U FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FO.UPD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPN$U
```

C.8.12 ROPN\$W Macro

ROPN\$W opens a file for writing.

```
.MACRO ROPN$W FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FO.WRT,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPN$W
```

C.8.13 ROPSSA Macro

ROPSSA opens a shared file for appending.

```
.MACRO ROPSSA FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FA.SHR!FO.APD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPSSA
```

C.8.14 ROPSSM Macro

ROPSSM opens a shared file for modification.

```
.MACRO ROPSSM FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FA.SHR!FO.MFY,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPSSM
```

C.8.15 ROPSSR Macro

ROPSSR opens a shared file for reading.

```
.MACRO ROPSSR FDB,LUN,FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FA.SHR!FO.RD,LUN,FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPSSR
```

MACRO EXPANSIONS

C.8.16 ROPS\$U Macro

ROPS\$U opens a shared file for updating.

```
.MACRO ROPS$U FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FA.SHR!FO.UPD,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPS$U
```

C.8.17 ROPS\$W Macro

ROPS\$W opens a shared file for writing.

```
.MACRO ROPS$W FDB,LUN, FNPT,RACC,URBA,URBS,ERR
.MCALL ROPN$
ROPN$ FDB,#FA.SHR!FO.WRT,LUN, FNPT,,RACC,URBA,URBS,ERR
.ENDM ROPS$W
```

C.8.18 RPORT\$ Macro

RPORT\$ reports the existence of a record for a relative file.

```
.MACRO RPORT$ FDB,LRCNM,HRCNM,ERR
.MCALL RPRTC$
RPRTC$ <FDB>,#-1,<LRCNM>,<HRCNM>,<ERR>
.ENDM RPORT$
```

C.8.19 RPRTC\$ Macro

RPRTC\$ reports on and optionally changes the existence of a record.

```
.MACRO RPRTC$ FDB,CHGVAL,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,CMOV$B,ERR$
LDFDB$ FDB
CMOV$B CHGVAL,R.BTVL
CMOV$W LRCNM,R.RCNM+2
CMOV$W HRCNM,R.RCNM
JSR PC,.RPRTC
ERR$ ERR
.ENDM RPRTC$
```

C.8.20 RPUT\$ Macro

RPUT\$ puts a relative file.

```
.MACRO RPUT$ FDB,OUTADR,OUTCNT,LRCNM,HRCNM,ERR
.MCALL LDFDB$,CMOV$W,ERR$
LDFDB$ FDB
CMOV$W OUTADR,F.NRBD+2
CMOV$W OUTCNT,F.NRBD
CMOV$W LRCNM,R.RCNM+2
CMOV$W HRCNM,R.RCNM
JSR PC,.RPUT
ERR$ ERR
.ENDM RPUT$
```

MACRO EXPANSIONS

C.9 QIOMAC - QIOSYM MACRO DEFINITIONS

C.9.1 DRERR\$ Macro

Defines the directive error codes returned in the directive status word. File control services (FCS) returns these codes in the F.ERR byte of the File Descriptor Block (FDB). To distinguish them from the overlapping codes from handler and file primitives, the F.ERR+1 byte in the FDB is negative for a directive error code.

```
.MACRO DRERR$ $$$GBL
.MCALL .QIOE.,DEFIN$
.IF IDN,<$$$GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
.ENDC
.IIF NDF,$$MSG,$$MSG=0
```

Standard Error Codes Returned by Directives in the Directive Status Word

```
.QIOE. IE.UPN,-01.,<INSUFFICIENT DYNAMIC STORAGE>
.QIOE. IE.INS,-02.,<SPECIFIED TASK NOT INSTALLED>
.QIOE. IE.PTS,-03.,<PARTITION TOO SMALL FOR TASK>
.QIOE. IE.UNS,-04.,<INSUFFICIENT DYNAMIC STORAGE FOR SEND>
.QIOE. IE.ULN,-05.,<UN-ASSIGNED LUN>
.QIOE. IE.HWR,-06.,<DEVICE HANDLER NOT RESIDENT>
.QIOE. IE.ACT,-07.,<TASK NOT ACTIVE>
.QIOE. IE.ITS,-08.,<DIRECTIVE INCONSISTENT WITH TASK STATE>
.QIOE. IE.FIX,-09.,<TASK ALREADY FIXED/UNFIXED>
.QIOE. IE.CKP,-10.,<ISSUING TASK NOT CHECKPOINTABLE>
.QIOE. IE.TCH,-11.,<TASK IS CHECKPOINTABLE>
.QIOE. IE.RBS,-15.,<RECEIVE BUFFER IS TOO SMALL>
.QIOE. IE.PRI,-16.,<PRIVILEGE VIOLATION>
.QIOE. IE.RSU,-17.,<RESOURCE IN USE>
.QIOE. IE.NSW,-18.,<NO SWAP SPACE AVAILABLE>
.QIOE. IE.ILV,-19.,<ILLEGAL VECTOR SPECIFIED>
.QIOE. IE.AST,-80.,<DIRECTIVE ISSUED/NOT ISSUED FROM AST>
.QIOE. IE.MAP,-81.,<ILLEGAL MAPPING SPECIFIED>
.QIOE. IE.IOP,-83.,<WINDOW HAS I/O IN PROGRESS>
.QIOE. IE.ALG,-84.,<ALIGNMENT ERROR>
.QIOE. IE.WOV,-85.,<ADDRESS WINDOW ALLOCATION OVERFLOW>
.QIOE. IE.NVR,-86.,<INVALID REGION ID>
.QIOE. IE.NVW,-87.,<INVALID ADDRESS WINDOW ID>
.QIOE. IE.ITP,-88.,<INVALID TI PARAMETER>
.QIOE. IE.IBS,-89.,<INVALID SEND BUFFER SIZE ( .GT. 255.)>
.QIOE. IE.LNL,-90.,<LUN LOCKED IN USE>
.QIOE. IE.IUI,-91.,<INVALID UIC>
.QIOE. IE.IDU,-92.,<INVALID DEVICE OR UNIT>
.QIOE. IE.ITI,-93.,<INVALID TIME PARAMETERS>
.QIOE. IE.PNS,-94.,<PARTITION/REGION NOT IN SYSTEM>
.QIOE. IE.IPR,-95.,<INVALID PRIORITY ( .GT. 250.)>
.QIOE. IE.ILU,-96.,<INVALID LUN>
.QIOE. IE.IEF,-97.,<INVALID EVENT FLAG ( .GT. 64.)>
.QIOE. IE.ADP,-98.,<PART OF DPB OUT OF USER'S SPACE>
.QIOE. IE.SDP,-99.,<DIC OR DPB SIZE INVALID>
```

MACRO EXPANSIONS

Success Codes from Directives - Placed in the Directive Status Word

```
DEFIN$ IS.CLR,0      ;Event flag was clear
                    ; from CLEAR EVENT FLAG directive
DEFIN$ IS.SET,2     ;Event flag was set
                    ; from SET EVENT FLAG directive
DEFIN$ IS.SPD,2     ;Task was suspended
.IF EQ,$$MSG
.MACRO DRERR$ A
.ENDM DRERR$
.ENDC
.ENDM DRERR$
```

C.9.2 FILIO\$ Macro

FILIO\$ defines the general QIO function codes that are device independent.

```
.MACRO FILIO$ $$$GBL
.MCALL .WORD.,DEFIN$
.IF IDN,<$$$GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
```

General QIO Qualifier Byte Definitions

```
.WORD. IQ.X,001,000 ;No error recovery
.WORD. IQ.Q,002,000 ;Queue request in express queue
.WORD. IQ.S,004,000 ;Synonym for IQ.UMD
.WORD. IQ.UMD,004,000 ;User mode diagnostic status required
```

Express Queue Commands

```
.WORD. IO.KIL,012,000 ;Kill current request
.WORD. IO.RDN,022,000 ;I/O rundown
.WORD. IO.UNL,042,000 ;Unload I/O handler task
.WORD. IO.LTK,050,000 ;Load a task image file
.WORD. IO.RTK,060,000 ;Record a task image file
.WORD. IO.SET,030,000 ;Set characteristics function
```

General Device Handler Codes

```
.WORD. IO.WLB,000,001 ;Write logical block
.WORD. IO.RLB,000,002 ;Read logical block
.WORD. IO.LOV,010,002 ;Load overlay (disk driver)
.WORD. IO.ATT,000,003 ;Attach a device to a task
.WORD. IO.DET,000,004 ;Detach a device from a task
```

Directory Primitive Codes

```
.WORD. IO.FNA,000,011 ;Find file name in directory
.WORD. IO.RNA,000,013 ;Remove file name from directory
.WORD. IO.ENA,000,014 ;Enter file name in directory
```

File Primitive Codes

```
.WORD. IO.CLN,000,007 ;Close out LUN
.WORD. IO.ULK,000,012 ;Unlock block
.WORD. IO.ACR,000,015 ;Access for read
.WORD. IO.ACW,000,016 ;Access for write
.WORD. IO.ACE,000,017 ;Access for extend
```

MACRO EXPANSIONS

```
.WORD. IO.DAC,000,020 ;De-access file
.WORD. IO.RVB,000,021 ;Read virtual block
.WORD. IO.WVB,000,022 ;Write virtual block
.WORD. IO.EXT,000,023 ;Extend file
.WORD. IO.CRE,000,024 ;Create file
.WORD. IO.DEL,000,025 ;Delete file
.WORD. IO.RAT,000,026 ;Read file attributes
.WORD. IO.WAT,000,027 ;Write file attributes
.WORD. IO.APV,010,030 ;Privileged ACP control
.WORD. IO.APC,000,030 ;ACP control
.MACRO FILIOS$ A
.ENDM FILIOS$
.ENDM FILIOS$
```

C.9.3 .IOER. Macro

This macro defines handler error codes that are returned in the I/O status block are defined through this macro. This macro then conditionally invokes the message generating macro for the QIOSYM.MSG file.

```
.MACRO .IOER. SYM,LO,MSG
DEFIN$ SYM,LO
.IF GT,$$MSG
.MCALL .IOMG.
.IOMG. SYM,LO,<MSG>
.ENDC
.ENDM .IOER.
```

C.9.4 IOERR\$ Macro

This macro defines the error codes returned by device handlers and file primitives in the first word of the I/O status block. The File Control Services also return these codes in the F.ERR byte in the File Descriptor Block (FDB). The F.ERR+1 byte is 0 if F.ERR contains a handler or FCP error code.

```
.MACRO IOERR$ $$$GBL
.MCALL .IOER.,DEFIN$
.IF IDN,<$$$GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
.ENDC
.IIF NDF,$$MSG,$$MSG=0
```

System standard codes, used by executive and drivers:

```
.IOER. IE.BAD,-01.,<BAD PARAMETERS>
.IOER. IE.IFC,-02.,<INVALID FUNCTION CODE>
.IOER. IE.DNR,-03.,<DEVICE NOT READY>
.IOER. IE.VER,-04.,<PARITY ERROR ON DEVICE>
.IOER. IE.ONP,-05.,<HARDWARE OPTION NOT PRESENT>
.IOER. IE.SPC,-06.,<ILLEGAL USER BUFFER>
.IOER. IE.DNA,-07.,<DEVICE NOT ATTACHED>
.IOER. IE.DAA,-08.,<DEVICE ALREADY ATTACHED>
.IOER. IE.DUN,-09.,<DEVICE NOT ATTACHABLE>
.IOER. IE.EOF,-10.,<END OF FILE DETECTED>
.IOER. IE.EOV,-11.,<END OF VOLUME DETECTED>
.IOER. IE.WLK,-12.,<WRITE ATTEMPTED TO LOCKED UNIT>
```

MACRO EXPANSIONS

.IOER. IE.DAO,-13.,<DATA OVERRUN>
.IOER. IE.SRE,-14.,<SEND/RECEIVE FAILURE>
.IOER. IE.ABO,-15.,<REQUEST TERMINATED>
.IOER. IE.PRI,-16.,<PRIVILEGE VIOLATION>
.IOER. IE.RSU,-17.,<SHARABLE RESOURCE IN USE>
.IOER. IE.OVR,-18.,<ILLEGAL OVERLAY REQUEST>
.IOER. IE.BYT,-19.,<ODD BYTE COUNT (OR VIRTUAL ADDRESS)>
.IOER. IE.BLK,-20.,<LOGICAL BLOCK NUMBER TOO LARGE>
.IOER. IE.MOD,-21.,<INVALID UDC MODULE #>
.IOER. IE.CON,-22.,<UDC CONNECT ERROR>
.IOER. IE.BBE,-56.,<BAD BLOCK ON DEVICE>
.IOER. IE.STK,-58.,<NOT ENOUGH STACK SPACE (FCS OR FCP)>
.IOER. IE.PHE,-59.,<FATAL HARDWARE ERROR ON DEVICE>
.IOER. IE.EOT,-62.,<END OF TAPE DETECTED>
.IOER. IE.OFL,-65.,<DEVICE OFF LINE>
.IOER. IE.BCC,-66.,<BLOCK CHECK, CRC, OR FRAMING ERROR>

File Primitive Codes

.IOER. IE.NOD,-23.,<CALLER'S NODES EXHAUSTED>
.IOER. IE.DFU,-24.,<DEVICE FULL>
.IOER. IE.IFU,-25.,<INDEX FILE FULL>
.IOER. IE.NSF,-26.,<NO SUCH FILE>
.IOER. IE.LCK,-27.,<LOCKED FROM READ/WRITE ACCESS>
.IOER. IE.HFU,-28.,<FILE HEADER FULL>
.IOER. IE.WAC,-29.,<ACCESSED FOR WRITE>
.IOER. IE.CKS,-30.,<FILE HEADER CHECKSUM FAILURE>
.IOER. IE.WAT,-31.,<ATTRIBUTE CONTROL LIST FORMAT ERROR>
.IOER. IE.RER,-32.,<FILE PROCESSOR DEVICE READ ERROR>
.IOER. IE.WER,-33.,<FILE PROCESSOR DEVICE WRITE ERROR>
.IOER. IE.ALN,-34.,<FILE ALREADY ACCESSED ON LUN>
.IOER. IE.SNC,-35.,<FILE ID, FILE NUMBER CHECK>
.IOER. IE.SQC,-36.,<FILE ID, SEQUENCE NUMBER CHECK>
.IOER. IE.NLN,-37.,<NO FILE ACCESSED ON LUN>
.IOER. IE.CLO,-38.,<FILE WAS NOT PROPERLY CLOSED>
.IOER. IE.DUP,-57.,<ENTER - DUPLICATE ENTRY IN DIRECTORY>
.IOER. IE.BVR,-63.,<BAD VERSION NUMBER>
.IOER. IE.BHD,-64.,<BAD FILE HEADER>
.IOER. IE.EXP,-75.,<FILE EXPIRATION DATE NOT REACHED>
.IOER. IE.BTF,-76.,<BAD TAPE FORMAT>
.IOER. IE.ALC,-84.,<ALLOCATION FAILURE>
.IOER. IE.ULK,-85.,<UNLOCK ERROR>
.IOER. IE.WCK,-86.,<WRITE CHECK FAILURE>

File Control Services Codes

.IOER. IE.NBF,-39.,<OPEN - NO BUFFER SPACE AVAILABLE FOR FILE>
.IOER. IE.RBG,-40.,<ILLEGAL RECORD SIZE>
.IOER. IE.NBK,-41.,<FILE EXCEEDS SPACE ALLOCATED, NO BLOCKS>
.IOER. IE.ILL,-42.,<ILLEGAL OPERATION ON FILE DESCRIPTOR
BLOCK>
.IOER. IE.BTP,-43.,<BAD RECORD TYPE>
.IOER. IE.RAC,-44.,<ILLEGAL RECORD ACCESS BITS SET>
.IOER. IE.RAT,-45.,<ILLEGAL RECORD ATTRIBUTES BITS SET>
.IOER. IE.RCN,-46.,<ILLEGAL RECORD NUMBER - TOO LARGE>
.IOER. IE.2DV,-48.,<RENAME - 2 DIFFERENT DEVICES>
.IOER. IE.FEX,-49.,<RENAME - NEW FILE NAME ALREADY IN USE>
.IOER. IE.BDR,-50.,<BAD DIRECTORY FILE>
.IOER. IE.RNM,-51.,<CANNOT RENAME OLD FILE SYSTEM>
.IOER. IE.BDI,-52.,<BAD DIRECTORY SYNTAX>
.IOER. IE.FOP,-53.,<FILE ALREADY OPEN>
.IOER. IE.BNM,-54.,<BAD FILE NAME>
.IOER. IE.BDV,-55.,<BAD DEVICE NAME>
.IOER. IE.NFI,-60.,<FILE ID WAS NOT SPECIFIED>

MACRO EXPANSIONS

```
.IOER. IE.ISQ,-61.,<ILLEGAL SEQUENTIAL OPERATION>
.IOER. IE.NNC,-77.,<NOT ANSI 'D' FORMAT BYTE COUNT>
```

Network ACP Codes

```
.IOER. IE.AST,-80.,<NO AST SPECIFIED IN CONNECT>
.IOER. IE.NNN,-68.,<NO SUCH NODE>
.IOER. IE.NFW,-69.,<PATH LOST TO PARTNER> ;This code must be
; odd
.IOER. IE.BLB,-70.,<BAD LOGICAL BUFFER>
.IOER. IE.TMM,-71.,<TOO MANY OUTSTANDING MESSAGES>
.IOER. IE.NDR,-72.,<NO DYNAMIC SPACE AVAILABLE>
.IOER. IE.CNR,-73.,<CONNECTION REJECTED>
.IOER. IE.TMO,-74.,<TIMEOUT ON REQUEST>
.IOER. IE.NNL,-78.,<NOT A NETWORK LUN>
```

ICS/ICR Error Codes

```
.IOER. IE.NLK,-79.,<TASK NOT LINKED TO SPECIFIED ICS/ICR
INTERRUPTS>
.IOER. IE.NST,-80.,<SPECIFIED TASK NOT INSTALLED>
.IOER. IE.FLN,-81.,<DEVICE OFFLINE WHEN OFFLINE REQUEST WAS
ISSUED>
```

TTY Error Codes

```
.IOER. IE.IES,-82.,<INVALID ESCAPE SEQUENCE>
.IOER. IE.PES,-83.,<PARTIAL ESCAPE SEQUENCE>
```

Successful Return Codes:

```
DEFIN$ IS.PND,+00. ;Operation pending
DEFIN$ IS.SUC,+01. ;Operation complete, success
DEFIN$ IS.RDD,+02. ;(RX11) Floppy disk successful
; completion of a read physical
; and a deleted data mark was
; seen in the sector header of
; the last sector.
DEFIN$ IS.BV,+05. ;(A/D READ) At least one bad value
; was read (remainder may be good);
; bad channel is indicated by a
; negative value in the buffer.
```

TTY Success Codes

```
DEFIN$ IS.CR,<15*400+1> ;Carriage return was terminator
DEFIN$ IS.ESC,<33*400+1> ;Escape (ALTMODE) was terminator
DEFIN$ IS.CC,<3*400+1> ;CONTROL-C was terminator
DEFIN$ IS.ESQ,<233*400+1> ;Escape sequence was terminator
DEFIN$ IS.PES,<200*400+1> ;Partial escape sequence terminator
DEFIN$ IS.EOT,<4*400+1> ;EOT was terminator (block mode
; input)
DEFIN$ IS.TAB,<11*400+1> ;TAB was terminator (forms mode
; input)
DEFIN$ IS.TMO,+2. ;Request timed out
.IF EQ,$$MSG
.MACRO IOERR$ A
.ENDM IOERR$
.ENDC
.ENDM IOERR$
```

MACRO EXPANSIONS

C.9.5 .QIOE. Macro

This macro defines the QIO error codes. This macro then invokes the error message generating macro, ".IOMG.". The QIOSYM.MSG file uses error codes -129 through -256.

```
.MACRO .QIOE. SYM,LO,MSG
DEFIN$ SYM,LO
.IF GT,$$MSG
.MCALL .IOMG.
.IOMG. SYM,<LO-128.>,<MSG>
.ENDC
.ENDM .QIOE.
```

Conditional Data for Writing a Message File

```
.MACRO .IOMG. SYM,LO,MSG
.WORD -^O<LO>
.ASCIZ ^MSG^
.EVEN
.IIF LT,^O<$$$MAX+<LO>>,$$$MAX--^O<LO>
.ENDM .IOMG.
```

C.9.6 QIOSY\$ Macro

This macro defines standard QUEUE I/O directive function values and IOSB return values. To invoke at assembly time (with local definition) use:

```
QIOSY$ ;Define symbols
```

To obtain global definition of these symbols use:

```
QIOSY$ DEF$G ;Symbols defined globally
```

This macro can be called once only. It then redefines itself as a null.

```
.MACRO QIOSY$ $$$GBL,$$$MSG
.IIF IDN,<$$$GBL>,<DEF$G>, .GLOBL QI.VER
.IF IDN,<$$$MSG>,<DEF$S>
$$$MAX=0
$$MSG=1
.IFF
$$MSG=0
.ENDC
.MCALL IOERR$
IOERR$ $$$GBL ;I/O error codes from handlers, FCP,
; FCS
.MCALL DRERR$
DRERR$ $$$GBL ;Directive status word error codes
.IF DIF,<$$$MSG>,<DEF$S>
.MCALL FILIO$
FILIO$ $$$GBL ;Define general QIO function codes
.MCALL SPCIO$
SPCIO$ $$$GBL ;Device dependent I/O function codes
.MACRO QIOSY$ ARG,ARG1,ARG2 ;Reclaim macro storage
.ENDM QIOSY$
.ENDC
.ENDM QIOSY$
```

MACRO EXPANSIONS

C.9.7 SPCIO\$ Macro

Defines the QIO function codes that are specific to individual devices.

```
.MACRO SPCIO$ $$$GBL
.MCALL .WORD.,DEFIN$
.IF IDN,<$$$GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
.ENDC
```

QIO Function Codes for Specific Device Dependent Functions

```
.WORD. IO.WLV,100,001 ;(DECTAPE) Write logical reverse
.WORD. IO.WLS,010,001 ;(COMM.) Write preceded by sync train
.WORD. IO.WNS,020,001 ;(COMM.) Write, no sync train
.WORD. IO.WAL,010,001 ;(TTY) Write passing all characters
.WORD. IO.WMS,020,001 ;(TTY) Write suppressible message
.WORD. IO.CCO,040,001 ;(TTY) Write with cancel CONTROL-o
.WORD. IO.WBT,100,001 ;(TTY) Write with breakthrough
.WORD. IO.WLT,010,001 ;(DISK) Write last track on RK06
.WORD. IO.WLC,020,001 ;(DISK) Write logical with write check
.WORD. IO.WPB,040,001 ;(RX11 DISK) Write physical block
.WORD. IO.WDD,044,001 ;(RX11 DISK) Write physical with
; deleted data

.WORD. IO.RLV,100,002 ;(MAGTAPE,DECTAPE) Read reverse
.WORD. IO.RST,001,002 ;(TTY) Read with special terminator
.WORD. IO.RAL,010,002 ;(TTY) Read passing all characters
.WORD. IO.RNE,020,002 ;(TTY) Read without echo
.WORD. IO.RNC,040,002 ;(TTY) Read - no lower case convert
.WORD. IO.RTM,200,002 ;(TTY) Read with time out
.WORD. IO.RDB,200,002 ;(CARD READER) Read binary mode
.WORD. IO.RHD,010,002 ;(COMM.) Read, strip sync
.WORD. IO.RNS,020,002 ;(COMM.) Read, do not strip sync
.WORD. IO.CRC,040,002 ;(COMM.) Read, do not clear CRC
.WORD. IO.RPB,040,002 ;(RX11 DISK) Read physical block
.WORD. IO.ATA,010,003 ;(TTY) Attach with ASTs
.WORD. IO.GTS,000,005 ;(TTY) Get terminal support
; characteristics

.WORD. IO.R1C,000,005 ;(AFC,AD01,UDC) Read single channel
.WORD. IO.INL,000,005 ;(COMM.) Initialization function
.WORD. IO.TRM,010,005 ;(COMM.) Termination function
.WORD. IO.RWD,000,005 ;(MAGTAPE,DECTAPE) Rewind
.WORD. IO.SPB,020,005 ;(MAGTAPE) Space "N" blocks
.WORD. IO.SPF,040,005 ;(MAGTAPE) Space "N" EOF marks
.WORD. IO.STC,100,005 ;(MAGTAPE) Set characteristic
.WORD. IO.SEC,120,005 ;(MAGTAPE) Sense characteristic
.WORD. IO.RWU,140,005 ;(MAGTAPE,DECTAPE) Rewind and unload
.WORD. IO.SMO,160,005 ;(MAGTAPE) Mount & set characteristics
.WORD. IO.HNG,000,006 ;(TTY) Hangup dial-up line
.WORD. IO.RBC,000,006 ;Read multichannels (buffer
; defines channels)

.WORD. IO.MOD,000,006 ;(COMM.) Setmode function family
.WORD. IO.HDX,010,006 ;(COMM.) Set unit half duplex
.WORD. IO.FDX,020,006 ;(COMM.) Set unit full duplex
.WORD. IO.SYN,040,006 ;(COMM.) Specify sync character
.WORD. IO.EOF,000,006 ;(MAGTAPE) Write EOF
.WORD. IO.RTC,000,007 ;Read channel - time based
.WORD. IO.SAO,000,010 ;(UDC) Single channel analog output
.WORD. IO.SSO,000,011 ;(UDC) Single shot, single point
.WORD. IO.RPR,000,011 ;(TTY) Read with prompt
.WORD. IO.MSO,000,012 ;(UDC) Single shot, multi-point
```

MACRO EXPANSIONS

```

.WORD. IO.SLO,000,013 ;(UDC) Latching, single point
.WORD. IO.MLO,000,014 ;(UDC) Latching, multi-point
.WORD. IO.LED,000,024 ;(LPS11) Write LED display lights
.WORD. IO.SDO,000,025 ;(LPS11) Write digital output register
.WORD. IO.SDI,000,026 ;(LPS11) Read digital input register
.WORD. IO.SCS,000,026 ;(UDC) Contact sense, single point
.WORD. IO.REL,000,027 ;(LPS11) Write relay
.WORD. IO.MCS,000,027 ;(UDC) Contact sense, multi-point
.WORD. IO.ADS,000,030 ;(LPS11) Synchronous A/D sampling
.WORD. IO.CCI,000,030 ;(UDC) Contact int - connect
.WORD. IO.MDI,000,031 ;(LPS11) Synchronous digital input
.WORD. IO.DCI,000,031 ;(UDC) Contact int - disconnect
.WORD. IO.XMT,000,031 ;(COMM.) Transmit specified block
; with ACK
.WORD. IO.XNA,010,031 ;(COMM.) Transmit without ACK
.WORD. IO.HIS,000,032 ;(LPS11) Synchronous histogram
; sampling
.WORD. IO.RCI,000,032 ;(UDC) Contact int - read
.WORD. IO.RCV,000,032 ;(COMM.) Receive data in buffer
; specified
.WORD. IO.MDO,000,033 ;(LPS11) Synchronous digital output
.WORD. IO.CTI,000,033 ;(UDC) Timer - connect
.WORD. IO.CON,000,033 ;(COMM.) Connect function
;(VT11) - Connect task to display
; processor
.WORD. IO.CPR,010,033 ;(COMM.) Connect no timeouts
.WORD. IO.CAS,020,033 ;(COMM.) Connect with AST
.WORD. IO.CRJ,040,033 ;(COMM.) Connect reject
.WORD. IO.CBO,110,033 ;(COMM.) Boot connect
.WORD. IO.CTR,210,033 ;(COMM.) Transparent connect
.WORD. IO.GNI,010,035 ;(COMM.) Get node information
.WORD. IO.GLI,020,035 ;(COMM.) Get link information
.WORD. IO.GLC,030,035 ;(COMM.) Get link information -
; clear counters
.WORD. IO.GRI,040,035 ;(COMM.) Get remote node information
.WORD. IO.GRC,050,035 ;(COMM.) Get remote node error counts
.WORD. IO.GRN,060,035 ;(COMM.) Get remote node name
.WORD. IO.CSM,070,035 ;(COMM.) Change solo mode
.WORD. IO.CIN,100,035 ;(COMM.) Change connection inhibit
.WORD. IO.SPW,110,035 ;(COMM.) Specify network password
.WORD. IO.CPW,120,035 ;(COMM.) Check network password.
.WORD. IO.NLB,130,035 ;(COMM.) NSP loopback
.WORD. IO.DLB,140,035 ;(COMM.) DDCMP loopback
.WORD. IO.STD,000,033 ;(LPAll) Start data transfer
.WORD. IO.DTI,000,034 ;(UDC) Timer - disconnect
.WORD. IO.DIS,000,034 ;(COMM.) Disconnect function
;(VT11) - Disconnect task from
; display processor
.WORD. IO.MDA,000,034 ;(LPS11) Synchronous D/A output
.WORD. IO.RTI,000,035 ;(UDC) Timer - read
.WORD. IO.CTL,000,035 ;(COMM.) Network control function
.WORD. IO.STP,000,035 ;(LPS11) Stop in progress function
;(VT11) - Stop display processor
.WORD. IO.CNT,000,036 ;(VT11) - Continue display processor
.WORD. IO.ITI,000,036 ;(UDC) Timer - initialize

```

ICS/ICR QIO Functions

```

.WORD. IO.CTY,000,007 ;Connect to terminal interrupts
.WORD. IO.DTY,000,015 ;Disconnect from terminal interrupts
.WORD. IO.LDI,000,016 ;Link to digital interrupts
.WORD. IO.UDI,010,023 ;Unlink from digital interrupts
.WORD. IO.LTI,000,017 ;Link to counter module interrupts

```

MACRO EXPANSIONS

```
.WORD. IO.UTI,020,023 ;Unlink from counter module
; interrupts
.WORD. IO.LTY,000,020 ;Link to remote terminal interrupts
.WORD. IO.UTY,030,023 ;Unlink from remote terminal
; interrupts
.WORD. IO.LKE,000,024 ;Link to error interrupts
.WORD. IO.UER,040,023 ;Unlink from error interrupts
.WORD. IO.NLK,000,023 ;Unlink from all interrupts
.WORD. IO.ONL,000,037 ;Unit online
.WORD. IO.FLN,000,025 ;Unit offline
.WORD. IO.RAD,000,021 ;Read activating data
```

IP11 I/O Functions

```
.WORD. IO.MAO,010,007 ;Multiple analog outputs
.WORD. IO.LEI,010,017 ;Link event flags to interrupt
.WORD. IO.RDD,010,020 ;Read digital data
.WORD. IO.RMT,020,020 ;Read mapping table
.WORD. IO.LSI,000,022 ;Link to DSI interrupts
.WORD. IO.UEI,050,023 ;Unlink event flags
.WORD. IO.USI,060,023 ;Unlink from DSI interrupts
.WORD. IO.CSI,000,026 ;Connect to DSI interrupts
.WORD. IO.DSI,000,027 ;Disconnect from DSI interrupts
.MACRO SPCIO$ A
.ENDM SPCIO$
.ENDM SPCIO$
```

C.9.8 UMDIO\$ Macro

Defines the I/O codes for user-mode diagnostics. All diagnostic functions are implemented as a subfunction of I/O code 10 (octal).

```
.MACRO UMDIO$ $$$GBL
.MCALL .WORD.,DEFIN$
.IF IDN <$$$GBL>,<DEF$G>
...GBL=1
.IFF
...GBL=0
.ENDC
```

General User-mode Qualifier Bit

```
.WORD. IQ.UMD,004,000 ;User mode diagnostic request
```

User-mode Diagnostic Functions.

```
.WORD. IO.HMS,000,010 ;(DISK) Home seek or recalibrate
.WORD. IO.BLS,010,010 ;(DISK) Block seek
.WORD. IO.OFF,020,010 ;(DISK) Offset position
.WORD. IO.RDH,030,010 ;(DISK) Read disk header
.WORD. IO.WDH,040,010 ;(DISK) Write disk header
.WORD. IO.WCK,050,010 ;(DISK) Writecheck (non-transfer)
.WORD. IO.RNF,060,010 ;(DECTAPE) Read block number forward
.WORD. IO.RNR,070,010 ;(DECTAPE) Read block number reverse
.WORD. IO.LPC,100,010 ;(MAGTAPE) Read longitudinal parity
; character
.WORD. IO.ERS,110,010 ;(MAGTAPE) Erase tape
```

Macro Redefinition to Null

```
.MACRO UMDIO$ A
.ENDM
.ENDM UMDIO$
```

MACRO EXPANSIONS

C.9.9 .WORD. Macro

This macro defines the symbol SYM where LO is the low order byte and HI is the high byte.

```
.MACRO .WORD. SYM,LO,HI
DEFIN$ SYM,<^O<HI*400+LO>>
.ENDM .WORD..
```

C.10 SNAP CONTROL BLOCK AND SNAPSHOT DUMP MACROS

The following macros are the macros for the snapshot dump debugging aid.

C.10.1 SNAP\$ Macro

SNAP\$ requests a snapshot dump.

```
.MACRO SNAP$ CTL,EFN,ID,L1,H1,L2,H2,L3,H3,L4,H4,?LBL1,?LBL2
.MCALL SNPDFS,WTSE$S,SDAT$S,RQST$S,CLEF$S
SNPDF$
.IIF NB <CTL>, MOV CTL,..SPBK+SB.CTL
.IIF NB <EFN>, MOV EFN,..SPBK+SB.EFN
.IIF NB <ID>, MOV ID,..SPBK+SB.ID
...SNP = SB.LM1
.IRP X,<L1,H1,L2,H2,L3,H3,L4,H4>
.IF NB <X>
MOV X,..SPBK+...SNP
.ENDC
...SNP = ...SNP+2
.ENDM
CLEF$S ..SPBK+SB.EFN ;Clear event flag
BCS LBL2
SDAT$S #..SPBK+SB.PMD,#..SPBK ;Send snap information
BCS LBL2
RQST$S #..SPBK+SB.PMD ;Request task "PMD..."
BCC LBL1
CMP #IE.ACT,@#$DSW ;O.K. if task already
;active
BEQ LBL1
SEC
BR LBL2
LBL1: WTSE$S ..SPBK+SB.EFN ;Wait for snapshot to
;finish
LBL2:
.ENDM SNAP$
```

C.10.2 SNPBK\$ Macro

SNPBK\$ generates a snap control block.

```
.MACRO SNPBK$ DEV,UNIT,CTL,EFN,ID,L1,H1,L2,H2,L3,H3,L4,H4
```

SNPBK\$ never generates more than one block.

```
.IF DF ..SPBK
.IIF NE <..SPBK-.>, .MEXIT
.ENDC
```

MACRO EXPANSIONS

```
.MCALL SNPDF$
SNPDF$
```

The following code builds the control block.

```
..SPBK::.WORD CTL ;Control word
.ASCII /DEV/ ;Send out device name
.IF NE .-<...SPBK+SB.DEV+2>
.ERROR ;Invalid device name "DEV"
.MEXIT
.ENDC
.BYTE UNIT,0 ;Device unit number & unused
.WORD EFN ;Event flag number
.WORD ID ;Snapshot identification word
.WORD L1,H1 ;Dump address limits
.WORD L2,H2
.WORD L3,H3
.WORD L4,H4
.RAD50 /PMD.../ ;Snapshot task name
.ENDM SNPBK$
```

C.10.3 SNPDF\$ Macro

SNPDF\$ generates snap offset and bit definitions.

```
.MACRO SNPDF$ GBL
.IF IDN <GBL>,<DEF$G>
.GLOBAL SB.CTL,SB.DEV,SB.UNT,SB.EFN,SB.ID,SB.LM1,SB.PMD
.GLOBAL SC.HDR,SC.LUN,SC.OVL,SC.STK,SC.WRD,SC.BYT
.ENDC
```

C.11 STATE AND KEYWORD TABLE GENERATION MACROS

These macros initialize table generation - they are called once at the start of each finite state machine description. The user must supply labels for the state and keyword tables.

C.11.1 ISTAT\$ Macro

```
.MACRO ISTAT$ STTBL,KEYTBL
.MCALL MTRAN$
.IF DF $RONLY
.PSECT $STATE,D,RO
.IFF
.PSECT $STATE,D
.ENDC
STTBL::
.IF DF $RONLY
.PSECT $KTAB,D,RO
.IFF
.PSECT $KTAB,D
.ENDC
KEYTBL::
.IF DF $RONLY
.PSECT $KSTR,D,RO
```

MACRO EXPANSIONS

```
.IFF
.PSECT $KSTR,D
.ENDC
$$$KEY = -1
$$$FLG = -1
$EXIT = 0
$LAMDA = 300
$NUMBR = 302
$STRNG = 304
$BLANK = 306
$SUBXP = 310
$EOS = 312
$DNUMB = 314
$RAD50 = 316
$ANY = 320
$ALPHA = 322
$DIGIT = 324
.PSECT
.ENDM ISTAT$
```

C.11.2 MTRAN\$ Macro

MTRAN\$ sends out the last transition entry.

```
.MACRO MTRAN$
.PSECT $STATE
.IF EQ $$$FLG+1
$$$FLG = 0
.MEXIT
.ENDC
$$$TYP
.BYTE $$$FLG
.IF NE $$$FLG&1
$$$EXT
.ENDC
.IF NE $$$FLG&2
$$$ACT
.ENDC
.IF NE $$$FLG&10
$$$BIT
.ENDC
.IF NE $$$FLG&4
.WORD $$$STA
.IFF
.IF EQ $$$FLG&200
.ERROR "BAD DEFAULT TRANSITION"
.ENDC
.ENDC
$$$FLG = 0
.ENDM MTRAN$
.LIST
```

C.11.3 STATE\$ Macro

STATE\$ declares a state.

```
.MACRO STATE$ LABEL
.PSECT $STATE
$$$FLG = $$$FLG!200
MTRAN$
```


MACRO EXPANSIONS

```
.IF NB LABEL  
LABEL: .ENDC  
$$$FLG = -1  
.PSECT  
.ENDM STATE$
```

C.11.4 TRAN\$ Macro

TRAN\$ specifies a state transition.

```
.MACRO TRAN$ TYPE,LABEL,ACTION,MASK,ADDR  
.PSECT $STATE  
MTRAN$  
.IF NB ACTION  
$$$FLG = $$$FLG!2  
.MACRO $$$ACT  
.WORD ACTION  
.ENDM $$$ACT  
.ENDC  
.IF NB MASK  
$$$FLG = $$$FLG!30  
.IF B ADDR  
.ERROR "MASK ADDRESS NOT PRESENT"  
.ENDC  
.MACRO $$$BIT  
.WORD MASK,ADDR  
.ENDM $$$BIT  
.ENDC  
.IF NB LABEL  
$$$FLG = $$$FLG!4  
$$$STA = LABEL  
.ENDC  
.IRPC X,<TYPE>  
.IF IDN <X>,<">  
.PSECT $KSTR  
$$$TMP = .  
.ASCII TYPE<377>  
.PSECT $KTAB  
.WORD $$$TMP  
.MACRO $$$TYP  
.BYTE $$$KEY!200  
.ENDM $$$TYP  
$$$KEY = $$$KEY+1  
.IF GT $$$KEY-63.  
.ERROR "TOO MANY KEYWORDS"  
.ENDC  
.MEXIT  
.ENDC  
.IF IDN <X>,<!>  
$$$FLG = $$$FLG!1  
.MACRO $$$EXT  
.WORD 0'TYPE  
.ENDM $$$EXT  
.MACRO $$$TYP  
.BYTE $$SUBXP  
.ENDM $$$TYP  
.MEXIT  
.ENDC  
.MACRO $$$TYP  
.BYTE TYPE  
.ENDM $$$TYP  
.MEXIT
```

MACRO EXPANSIONS

```
.ENDM
.PSECT
.ENDM TRAN$
```

C.12 SUBMIT FILE TO PRINT SPOOLER (PRT...) MACRO (PRINT\$)

C.12.1 PRINT\$ Macro

The PRINT\$ macro takes the following form:

```
PRINT$ FDB,ERR
```

where:

FDB=Address of the FDB of the file to submit.
 ERR=Address of error routine.

```
.MACRO PRINT$ FDB,ERR,?LBL,A,B,C,D,E,F,?LBL2
.MCALL CALL,CLOSE$,LDRO$,GLUN$$
LDRO$ FDB ;Load FDB address into R0
TST F.BDB(R0) ;File open?
BEQ LBL ;If eq no
MOV R1,-(SP) ;Save R1 and R2
MOV R2,-(SP)
MOV R3,-(SP) ;Save R3
MOV SP,R1 ;Save pointer to end of send buffer
SUB #<8.*2>,SP ;Allocate a get LUN info buffer
MOV SP,R2 ;Save its address in R2
MOVB F.LUN(R0),R3 ;Get LUN in R3
GLUN$$ R3,R2 ;Get real device name and unit
MOV G.LUCW(R2),R3 ;Save characteristics word 1
MOV R0,R2 ;Copy FDB address
ADD #F.FNB+N.DID+6,R2 ;Point to end of directory ID
MOV -(R2),-(R1) ;Push directory ID
MOV -(R2),-(R1)
MOV -(R2),-(R1)
ADD #N.FID+6-N.DID,R2 ;Point to end of file ID
MOV -(R2),-(R1) ;Push file ID
MOV -(R2),-(R1)
MOV -(R2),-(R1)
CLRB -(R1) ;Clear LUN information flags byte
ADD #N.FVER+2-N.FID,R2 ;Point to end of filename, type,
; version
MOV -(R2),-(SP) ;Push file version number
MOV -(R2),-(SP) ;Push file type
MOV -(R2),-(SP) ;Push filename
MOV -(R2),-(SP)
MOV -(R2),-(SP)
MOV SP,R1 ;Set pointer to send buffer
CLOSE$ R0 ;Close file
BCS LBL2 ;Skip on error
BIT #FD.REC!FD.OSP,R3 ;Record oriented or spooled device?
BNE LBL2 ;If ne yes
MOV R0,R2 ;Save FDB address
MOV #RPRT,R0 ;Get "PRT" in RAD-50
CALL $DSPAT ;Send data to ...PRT or PRT...
MOV R0,F.ERR(R2) ;Get return status
MOV R2,R0 ;Restore FDB address
LBL2: ROR R1 ;Save carry
```

MACRO EXPANSIONS

```

ADD #<13.*2>,SP          ;Clean stack
ROL R1                   ;Restore carry
MOV (SP)+,R3             ;Restore R3
MOV (SP)+,R2             ;Restore R1 and R2
MOV (SP)+,R1
.IF NB ERR
BCC LBL                  ;If CC okay
CALL ERR                 ;Call error routine
.ENDC
LBL:                      ;Reference label
.ENDM PRINT$

```

C.13 SET/GET SYMBOL (TTSYM\$) MACRO EXPANSIONS

C.13.1 TTSYM\$ Macro

The TTSYM\$ macro file is used by RSX-11D, IAS, and RSX-11M terminal drivers. However, the RSX-11M terminal driver uses only a limited number of the symbols in this file. The TTSYM\$ macro defines all the symbols required for terminal SET and GET characteristics.

```

.MACRO TTSYM$ $$$GBL
.MCALL DEFIN$
...GBL=0
.IF IDN,<$$$GBL>,<DEF$G>
...GBL=1
.ENDC

DEFIN$ TC.WID,1.         ;Line width
DEFIN$ TC.LPP,2.        ;Lines per page
DEFIN$ TC.RSP,3.        ;Receiver speed
DEFIN$ TC.XSP,4.        ;Transmitter speed
DEFIN$ TC.STB,5.        ;Two stop bits
DEFIN$ TC.ISL,6.        ;Subline on interface
DEFIN$ TC.RAT,7.        ;Read-ahead type
DEFIN$ TC.TTP,8.        ;Terminal type
DEFIN$ TC.SCR,9.        ;Script line
DEFIN$ TC.SCP,10.       ;Scope
DEFIN$ TC.HFL,11.       ;Horizontal fill requirement
DEFIN$ TC.VFL,12.       ;Vertical fill
DEFIN$ TC.NL,13.        ;ASCII newline terminal
DEFIN$ TC.SFF,14.       ;Simulate formfeed and vertab
DEFIN$ TC.HFF,15.       ;HARDWARE FORMFEED AND VERTAB
DEFIN$ TC.LVF,16.       ;LA36 vertical fill
DEFIN$ TC.HHT,17.       ;Hardware horizontal tab
DEFIN$ TC.NST,18.       ;Non-standard hardware tab
DEFIN$ TC.BSP,19.       ;Hardware backspace
DEFIN$ TC.ACR,20.       ;Automatic carriage return required
DEFIN$ TC.SMR,21.       ;Small character input enabled
DEFIN$ TC.SMP,22.       ;Small character input required
                        ; (/lower case input)
DEFIN$ TC.SMO,23.       ;Small character output enabled
DEFIN$ TC.CCF,24.       ;CONTROL-C flushes type-ahead and read
DEFIN$ TC.ALT,25.       ;Alternative ALTMODE recognition
DEFIN$ TC.IMG,26.       ;IAS - messages inhibited
DEFIN$ TC.NKB,27.       ;No keyboard
DEFIN$ TC.NPR,28.       ;No printer
DEFIN$ TC.ESQ,29.       ;Escape sequence recognition
DEFIN$ TC.LCP,30.       ;Local copy line

```

MACRO EXPANSIONS

```

DEFIN$ TC.PAR,31.      ;Parity recognition/generation
                        ; required
DEFIN$ TC.EPA,32.     ;Even parity
DEFIN$ TC.DLU,33.     ;Dialup line
DEFIN$ TC.BLK,34.     ;Block mode terminal
DEFIN$ TC.FRM,35.     ;Forms mode terminal
DEFIN$ TC.HLD,36.     ;Terminal hold mode
DEFIN$ TC.TAP,37.     ;Low speed paper tape reader
DEFIN$ TC.CEQ,38.     ; compatible escape sequences
DEFIN$ TC.NEC,39.     ; terminal in no-echo mode
DEFIN$ TC.SLV,40.     ; terminal in slave mode
DEFIN$ TC.PRI,41.     ; terminal is privileged
DEFIN$ TC.UC0,42.     ;User characteristic 0
DEFIN$ TC.UC1,43.     ;User characteristic 1
DEFIN$ TC.UC2,44.     ;User characteristic 2
DEFIN$ TC.UC3,45.     ;User characteristic 3
DEFIN$ TC.UC4,46.     ;User characteristic 4
DEFIN$ TC.UC5,47.     ;User characteristic 5
DEFIN$ TC.UC6,48.     ;User characteristic 6
DEFIN$ TC.UC7,49.     ;User characteristic 7
DEFIN$ TC.UC8,50.     ;User characteristic 8
DEFIN$ TC.UC9,51.     ;User characteristic 9
DEFIN$ TC.MAX,52.     ;This must be one greater
                        ; than the highest value used
                        ; for a symbol

```

Set Characteristic Error Codes

```

DEFIN$ SE.ICN,1.      ;Wrong characteristic name
DEFIN$ SE.FIX,2.      ;Attempt to change fixed
                        ; characteristic
DEFIN$ SE.BIN,3.      ;Wrong value for binary
                        ; characteristic
DEFIN$ SE.VAL,4.      ;Wrong value for non-binary
                        ; characteristic
DEFIN$ SE.TER,5.      ;Wrong terminal type
DEFIN$ SE.SPD,6.      ;Wrong speed for interface
DEFIN$ SE.SPL,7.      ;Wrong split speed for interface
DEFIN$ SE.PAR,8.      ;Wrong parity type for interface
DEFIN$ SE.LPR,9.      ;Other wrong line parameters
DEFIN$ SE.NSC,10.     ;Interface does not have settable
                        ; characteristics
DEFIN$ SE.UPN,11.     ;No space to save default
                        ; characteristics
DEFIN$ SE.NIH,12.     ;Characteristic not assembled in
                        ; handler

```

Subfunction Codes for the Set Characteristics Function

```

DEFIN$ SF.SSC, 2400!020 ;Set single characteristic
DEFIN$ SF.SMC, 2400!040 ;Set multiple characteristics
DEFIN$ SF.RDF, 2400!060 ;Restore default
DEFIN$ SF.STT, 2400!100 ;Set terminal type
DEFIN$ SF.STS, 2400!120 ;Set terminal type and speed
DEFIN$ SF.GSC, 2400!140 ;Get single characteristic
DEFIN$ SF.GMC, 2400!160 ;Get multiple characteristics
DEFIN$ SF.GAC, 2400!200 ;Get all characteristics
DEFIN$ SF.SAC, 2400!220 ;Set all characteristics
DEFIN$ SF.DEF, 010     ;Set default characteristics

```

Speed Types

```

DEFIN$ S.0    ,1.
DEFIN$ S.50   ,2.

```

MACRO EXPANSIONS

```

DEFIN$ S.75 ,3.
DEFIN$ S.100 ,4.
DEFIN$ S.110 ,5.
DEFIN$ S.134 ,6.
DEFIN$ S.150 ,7.
DEFIN$ S.200 ,8.
DEFIN$ S.300 ,9.
DEFIN$ S.600 ,10.
DEFIN$ S.1200,11.
DEFIN$ S.1800,12.
DEFIN$ S.2000,13.
DEFIN$ S.2400,14.
DEFIN$ S.3600,15.
DEFIN$ S.4800,16.
DEFIN$ S.7200,17.
DEFIN$ S.9600,18.
DEFIN$ S.EXTA,19.
DEFIN$ S.EXTB,20.

```

Terminal Types

```

DEFIN$ T.UNK0,0. ;Unknown (unspecified)
DEFIN$ T.AS33,1. ;ASR33
DEFIN$ T.KS33,2. ;KSR33
DEFIN$ T.AS35,3. ;ASR35
DEFIN$ T.L30S,4. ;LA30S
DEFIN$ T.L30P,5. ;LA30P
DEFIN$ T.LA36,6. ;LA36
DEFIN$ T.VT05,7. ;VT05
DEFIN$ T.VT50,8. ;VT50
DEFIN$ T.VT52,9. ;VT52
DEFIN$ T.VT55,10. ;VT55
DEFIN$ T.VT61,11. ;VT61
DEFIN$ T.L180,12. ;LA180S
DEFIN$ T.SCR0,13. ;Script line
DEFIN$ T.USR0,14. ;User terminal 0
DEFIN$ T.USR1,T.USR0+1 ;User terminal 1
DEFIN$ T.USR2,T.USR1+1 ;User terminal 2
DEFIN$ T.USR3,T.USR2+1 ;User terminal 3
DEFIN$ T.USR4,T.USR3+1 ;User terminal 4

```

Bits for Return from 'GET TERMINAL SUPPORT'

```

DEFIN$ F1.ACR,000001 ;Auto CR/LF on long lines
DEFIN$ F1.BTW,000002 ;Break through write
DEFIN$ F1.BUF,000004 ;Intermediate buffering
DEFIN$ F1.UIA,000010 ;Unsolicited input ASTs
DEFIN$ F1.CCO,000020 ;Cancel CONTROL-O on write
DEFIN$ F1.ESQ,000040 ;Escape sequence support
DEFIN$ F1.HLD,000100 ;Hold screen support
DEFIN$ F1.LWC,000200 ;Lower case conversion
DEFIN$ F1.RNE,000400 ;Read no echo
DEFIN$ F1.RPR,001000 ;Read with prompt
DEFIN$ F1.RST,002000 ;Read with special terminators
DEFIN$ F1.RUB,004000 ;Scope rubouts
DEFIN$ F1.SYN,010000 ;XON/XOFF
DEFIN$ F1.TRW,020000 ;Transparent read/write
DEFIN$ F1.UTB,040000 ;Buffering in task buffer
DEFIN$ F1.VBF,100000 ;EXEC buffers are variable length
DEFIN$ F2.SCH,000001 ;Set characteristics
DEFIN$ F2.GCH,000002 ;Get characteristics
DEFIN$ F2.DCH,000004 ;Dump/restore characteristics

```

MACRO EXPANSIONS

```
DEFIN$ F2.DKL,000010 ;Historical lld/IAS IO.KIL
DEFIN$ F2.ALT,000020 ;ALTMODE is echoed
DEFIN$ F2.SFF,000040 ;Formfeed can be simulated
```

Subfunction Bits for Terminal Handler QIOs.

Note: A complete list of terminal handler function and subfunction codes appears in this appendix under QIOMAC.

```
DEFIN$ TF.RST,001 ; [IO.RLB/IO.RPR] Read with special
; terminators
DEFIN$ TF.BIN,002 ; Send prompt as PASS ALL
DEFIN$ TF.RAL,010 ; Read pass all
DEFIN$ TF.RNE,020 ; Read with no echo
DEFIN$ TF.RNC,040 ; Read with no case
; conversion
DEFIN$ TF.XOF,100 ; Send XOF after prompt
DEFIN$ TF.TMO,200 ; Read with timeout
DEFIN$ TF.WAL,010 ; [IO.WLB] Write pass all
DEFIN$ TF.WMS,020 ; Write suppressible message
DEFIN$ TF.CCO,040 ; Cancel CONTROL-O
DEFIN$ TF.WBT,100 ; Break through read
DEFIN$ TF.SYN,200 ; Synchronous write
DEFIN$ TF.AST,010 ; [IO.ATT] Specify ASTs in attach
DEFIN$ TF.ESQ,020 ; Recognise escape sequences
```

```
.MACRO TTSYM$ A
.ENDM
.ENDM TTSYM$
```

APPENDIX D

LISTING OF CONDITIONAL ASSEMBLY PARAMETERS

D.1 LISTING OF CONDITIONAL ASSEMBLY PARAMETERS

The following is a listing of all the conditional assembly parameters that are possible on the RSX-11M system. Many of these may not apply to your system but are listed here for your convenience.

Parameter	Definition
A\$\$CHK	;Address checking
A\$\$CPS	;ACP support
A\$\$D01	;AD01-D A/D convertors
A\$\$F11	;AFC11 A/D convertors
A\$\$NSI	;ANSI magtape support
A\$\$PRI	;ALTER PRIORITY directive
A\$\$R11	;AR11 laboratory peripheral systems
A\$\$RDA	;AR11 D/A option
A\$\$TRP	;AST support
B\$\$OOT	;Bootstrap ROM address
C\$\$CKP	;Checkpointing support
C\$\$INT	;Connect interrupt vector support
C\$\$LIS	;Command language interpreter support
C\$\$ORE	;Size of dynamic storage region
C\$\$OVL	;Overlaid network ACP
C\$\$R11	;CR11 card readers
C\$\$RSH	;Crash reporting
C\$\$RUN	;Crash unit number
D\$\$B11	;DA11-B parallel line interfaces
D\$\$BUG	;Panic dump routine
D\$\$E11	;DL11-E line interfaces
D\$\$H11	;DH11 asynchronous line multiplexers
D\$\$IAG	;User mode diagnostics
D\$\$ISK	;Nonresident task support
D\$\$J11	;DJ11 asynchronous line multiplexers
D\$\$L11	;DL11/A/B/C/D line interfaces
D\$\$M11	;DM11BB modem control interfaces
D\$\$P11	;DP11 line interfaces
D\$\$Q11	;DQ11 synchronous line interfaces
D\$\$SHF	;Automatic dynamic memory compaction
D\$\$U11	;DU11 line interfaces
D\$\$W11	;DU11 line interfaces
D\$\$YNC	;Dynamic checkpoint allocation
D\$\$YNM	;Dynamic memory allocation support
D\$\$Z11	;DZ11 asynchronous line multiplexers
D\$\$ZMD	;DZ11 modem support
E\$\$DVC	;Log device errors and timeout
E\$\$EAE	;EAE support
E\$\$NSI	;Log undefined interrupts
E\$\$PER	;Log parity error traps

LISTING OF CONDITIONAL ASSEMBLY PARAMETERS

Parameter	Definition
E\$\$XPR	;EXTEND PARTITION (task) directive
F\$\$AST	;Asynchronous FPP
F\$\$LPP	;Floating point processor support
F\$\$LTP	;FIS support
F\$\$LVL	;File structure level support
G\$\$TPP	;GET PARTITION PARAMETERS directive
G\$\$TSS	;GET SENSE SWITCHES directive
G\$\$TTK	;GET TASK PARAMETERS directive
G\$\$WRD	;Include \$GTWRD code
I\$\$C11	;ICS/ICR-11 industrial control subsystems
I\$\$CAD	;ICS/ICR-11 A/D convertor modules
I\$\$CDA	;ICS/ICR-11 D/A modules
I\$\$CDS	;ICS/ICR-11 digital sense input modules
I\$\$CIM	;ICS/ICR-11 digital interrupt modules
I\$\$CLK	;ICS/ICR-11 task activation from interrupts
I\$\$CLT	;ICS/ICR-11 bi-stable output modules
I\$\$CR	;ICR-11 remote units
I\$\$CRS	;ICS/ICR-11 status recovery
I\$\$CSS	;ICS/ICR-11 single shot output modules
I\$\$CTI	;ICS/ICR-11 I/O counter modules
I\$\$CWD	;ICS/ICR-11 error count
I\$\$G11	;KG11 CRC option
I\$\$RAR	;Install, request, and remove on exit
I\$\$RDN	;I/O rundown
I\$\$S11	;DRS/DSS-11 input/output modules
I\$\$SDR	;DRS-11 output modules
I\$\$SDS	;DSS-11 input modules
I\$\$SLK	;Task activation from interrupts
I\$\$SPW	;DRS-11 status restore on power recovery
I\$\$SRC	;DRS-11 command register address
I\$\$SSC	;DSS-11 command register address
K\$\$CNT	;Count register address
K\$\$CSR	;Programmable clock CSR address
K\$\$G11	;KG11 CRC option
K\$\$LDC	;Load count value
K\$\$TPS	;Ticks per second
K\$\$W11	;KW11-Y support
L\$\$11R	;Fast printer support
L\$\$50H	;50HZ line frequency
L\$\$ASG	;Logical device assignment support
L\$\$DRV	;Loadable driver support
L\$\$P11	;LP/LS/LV11/LA180 line printers
L\$\$PS1	;LPS11 laboratory peripheral systems
L\$\$SBF	;LPS11 bandwidth filtering
L\$\$SDA	;LPS11 D/A option
L\$\$SDR	;LPSDR-A present
L\$\$SGR	;LPS11 gain ranging option
L\$\$S11	;LSI-11 processor
LD\$AD	;Loadable ADDR
LD\$AF	;Loadable AFDRV
LD\$AR	;Loadable ARDRV
LD\$CR	;Loadable CRDRV
LD\$CT	;Loadable CTDRV
LD\$DB	;Loadable DBDRV
LD\$DF	;Loadable DFDRV
LD\$DK	;Loadable DKDRV
LD\$DM	;Loadable DMDRV
LD\$DP	;Loadable DPDRV
LD\$DS	;Loadable DSDRV
LD\$DT	;Loadable DTDRV
LD\$DX	;Loadable DXDRV
LD\$LP	;Loadable LPDRV

LISTING OF CONDITIONAL ASSEMBLY PARAMETERS

Parameter	Definition
LD\$LS	;Loadable LSDRV
LD\$MM	;Loadable MMDRV
LD\$MT	;Loadable MTDRV
LD\$PP	;Loadable PPDRV
LD\$PR	;Loadable PRDRV
LD\$TT	;Loadable TTDRV
LD\$XB	;Loadable XBDRV
LD\$XL	;Loadable XLDRV
LD\$XM	;Loadable XMDRV
LD\$XP	;Loadable XPDRV
LD\$XQ	;Loadable XQDRV
LD\$XU	;Loadable XUDRV
LD\$XW	;Loadable XWDRV
M\$\$CRX	;External MCR functions
M\$\$CRB	;MCR command buffer length
M\$\$EXT	;11/70 extended memory support
M\$\$FCS	;FCS/file system support
M\$\$IXD	;Mixed RH11 drive types
M\$\$MGE	;Memory management
M\$\$MUP	;Multi-user protection
M\$\$NET	;Network ACP support
M\$\$OVR	;Overlaid MCR
N\$\$CON	;Number of simultaneous logical channels
N\$\$MOV	;Size of BLXIO 'MOV' table
N\$\$MXM	;Maximum network data message size
N\$\$TMO	;Network control message timeout period
P\$\$D70	;11/70 cache parity support
P\$\$GMX	;Get mapping context
P\$\$LAS	;Program logical address extensions
P\$\$P11	;PC11 paper tape punch
P\$\$P45	;Rotating data lights
P\$\$R11	;PR11 paper tape reader
P\$\$RFL	;Powerfail recovery
P\$\$RTY	;Parity memory
P\$\$SRF	;SEND/RECEIVE by REFERENCE
P\$\$TPT	;Point-to-point network support
P\$\$WRD	;Include \$PTWRD code
Q\$\$CRC	;DQ11 CRC option support
Q\$\$HPT	;DQ11 protocol option support
Q\$\$OPT	;Pre-allocate I/O packets
R\$\$11S	;RSX-11S system
R\$\$11M	;RSX-11M system
R\$\$611	;RK611 disk cartridge controllers
R\$\$6OF	;RK06 offset recovery support
R\$\$6WC	;RK611 write check support
R\$\$DER	;CORAL deallocation error checking
R\$\$EXV	;Extend Executive to 20k
R\$\$F11	;RF11 fixed head disk controllers
R\$\$JPO	;Offset recovery support
R\$\$JS1	;RJ/RWS03-04 fixed head disk controllers
R\$\$JP1	;RJ/RWP04-05-06 disk pack controllers
R\$\$K11	;RK11 cartridge disk controllers
R\$\$KWC	;RK11 write check support
R\$\$LOD	;Remote task loading
R\$\$NDC	;Clock ticks per scheduling interval
R\$\$LKL	;RMS-11 block locking
R\$\$NDH	;Highest priority class to consider
R\$\$NDL	;Lowest priority class
R\$\$P11	;RP11-C/E disk pack controllers
R\$\$SND	;SEND/RECEIVE directives
R\$\$X11	;RX11 disk controllers
S\$\$ECC	;Shared Executive ECC code

LISTING OF CONDITIONAL ASSEMBLY PARAMETERS

Parameter	Definition
S\$\$WPC	;Clock ticks per swapping interval
S\$\$WPR	;Swapping priority
S\$\$WRG	;Console switch register present
S\$\$YSZ	;Size of physical memory in 32W blocks
T\$\$18S	;LA180S support
T\$\$30P	;LA30P support
T\$\$All	;TAll dual cassettes
T\$\$ACR	;Automatic CR/LF
T\$\$BTW	;Breakthrough write
T\$\$BUF	;Terminal input checkpointing
T\$\$Cl1	;TC11 DECTape controllers
T\$\$CCA	;Unsolicited input AST
T\$\$CCO	;Write with control-O cancellation
T\$\$CTR	;Control-R support
T\$\$ESC	;Escape sequence support
T\$\$GMC	;Get terminal characteristics
T\$\$GTS	;Get terminal driver support
T\$\$HLD	;Hold-screen support
T\$\$J16	;TJ/TWU16--45 magtape controllers
T\$\$KMG	;Task termination/device not ready messages
T\$\$LWC	;Settable case conversion (lower case)
T\$\$M11	;TM/TMA/TMB11 magtape controllers
T\$\$MAN	;DM11BB answer baud rate
T\$\$MIN	;Baseline terminal driver
T\$\$RNE	;Read with no echo
T\$\$RPR	;Read with prompt or after prompt
T\$\$RST	;Read with special terminator
T\$\$RUB	;CRT rubout
T\$\$SMC	;Set terminal characteristics
T\$\$SYN	;Terminal-host synchronization support
T\$\$TRW	;Transparent terminal read/write
T\$\$UTB	;User terminal input buffering
T\$\$VBF	;Variable length terminal input buffers
T\$\$ZAN	;DZ11 answer baud rate
U\$\$ACH	;Number of channels per module
U\$\$ADM	;ADU01 or IAD-IA module description
U\$\$AOM	;Analog output modules
U\$\$CIM	;Digital interrupt module description
U\$\$CSM	;Digital sense input module
U\$\$D11	;UDC11 Universal digital controller
U\$\$LTM	;Bi-stable output module description
U\$\$MHI	;High part of UNIBUS address
U\$\$MLO	;Low part of UNIBUS address
U\$\$MRN	;Address of next available UMR
U\$\$NIP	;Uni-polar A/D sampling
U\$\$SSM	;Single-shot output modules
U\$\$TIM	;I/O counter module description
V\$\$CTR	;Address of highest vector plus 4
V\$\$S2S	;VS60 drives two CRT monitors
V\$\$S60	;VS60 graphics display subsystem
V\$\$T11	;VT11 graphics display subsystem
X\$\$DBT	;Executive debugging tool
X\$\$M11	;DMC11 line interfaces

APPENDIX E
GENERAL FAULT ISOLATION

E.1 INTRODUCTION

This Appendix is a guide containing some general hints about fault isolation in the RSX-11M system. It is not an all-inclusive guide, but it does offer suggestions to get you started in the right direction. Other aids that may help you:

- RSX-11M Crash Dump Analyzer Manual
- IAS/RSX-11 ODT Reference Manual (for the Online Debugging Tool)
- RSX-11M User Mode Diagnostics Reference Manual
- Executive Debug Tool (XDT) and Panic Dump, described in the RSX-11M Guide to Writing an I/O Driver
- RSX-11M Task Builder Reference Manual (for Post Mortem and Snapshot Dumps)

E.2 FAULT CLASSIFICATIONS

Four possible causes can be identified when the system faults:

1. A user-state task has faulted such that it causes the system to fault (unmapped systems only)
2. A user-written driver has faulted such that it causes the system to fault
3. The RSX-11M system software itself has faulted
4. The host hardware has faulted.

You should immediately determine which of these four cases is the source of the fault. This section outlines the procedures that may help you to uncover the source of the fault. Correcting the fault is assumed to be your responsibility.

E.3 SERVICING FAULTS

Faults manifest themselves in roughly four ways. They are listed here in the order of increasing difficulty of isolation.

GENERAL FAULT ISOLATION

1. If XDT is included in your system, an unintended trap to XDT occurs.
2. The system displays text indicating a crash has occurred and halts.
3. The system halts but displays nothing.
4. The system is in an unintended loop.

Regardless of fault manifestation, you should immediately obtain pertinent fault isolation data.

Case 1--The system has trapped to XDT

The trap may or may not be intended (for example, you may not have removed a previously set breakpoint). If the trap is not intended, type the X command. This causes XDT to jump to location 40(8), from which the Executive stack and register dump routine (if present), followed by either Panic Dump or the Crash Dump Analyzer (CDA) support routine (if present), will be invoked. If, however, you have some idea of the source of the problem (for example, a recent coding change), then you can use XDT to examine pertinent data structures and code.

Case 2 - The system has displayed text indicating a crash occurred

If the text consists of output from the Executive stack and register dump routine, all the basic information describing the state of the system has been displayed. If the text has been produced by the CDA support routine, follow the procedure for obtaining and formatting a memory dump as outlined in the RSX-11M Crash Dump Analyzer Reference Manual.

Case 3--The system has halted but displays no information

Before taking any action, examine and record the following information:

- Current PS and PC
- General registers
- Executive stackpointer and stack
- Executive PARs on a mapped system
- Pertinent device registers

The procedure depends on the particular PDP-11 processor. Consult the appropriate PDP-11 Processor Handbook for details.

After preserving the PS and PC, invoke your resident debugging aid: enter 40(8) in the switch register, press LOAD ADDR, and then press START. The contents of 40(8) cause the successive invocation of:

1. The Executive stack and register dump routine (if present)
2. Either Panic Dump or CDA support routine (if present)

GENERAL FAULT ISOLATION

Case 4 - System is in an unintended loop

Proceed as follows:

1. Halt the processor
2. Record PC and PS and any pertinent device registers, as in Case 3 above.

You may then want to step through a number of instructions in an attempt to locate the loop. For this attempt to be meaningful you must first disable the system clock. Proceed as follows:

1. Examine the contents of word 777546 (if your system has a line frequency clock) or word 772540 (if it has a programmable clock).
2. Clear bit 6 in this word and redeposit the word. This disables the clock.
3. The system will not run unless you reenables the clock.
4. After trying to locate the loop and reenabling the clock, transfer to location 40(8) as in Case 3.

E.3.1 Gathering Pertinent Fault Isolation Data

Before proceeding with locating the fault, you should to dump the system common (SYSCM). SYSCM contains a number of critical pointers and listheads. Find the file SYSCM in the Executive memory allocation map listing, then enter the appropriate limits into the Panic Dump Routine.

Alternately, you can run the Crash Dump Analyzer as a task; or, if XDT is included in your system and a program condition causes a processor trap, control transfers automatically to XDT and typing X at your terminal causes control to be transferred to the crash dump routine.

In addition, you should to dump the Dynamic Storage Region and the device tables. The Dynamic Storage Region starts in the module INITL and the device tables are in SYSTB.

At this point, you have the following data:

- PS
- PC
- The stack
- R0 through R6

GENERAL FAULT ISOLATION

- The Dynamic Storage Region
- The Device Tables
- System common

These data are the minimum required to trace the fault effectively.

E.3.2 Tracing Faults

Three pointers in SYSCM are critical in fault tracing. These pointers are described below:

\$STKDP - Stack Depth Indicator

This data item indicates which stack was being used at the time of the crash. \$STKDP plays an important role in determining the origin of a fault. The following values apply.

- +1 -- User (task-state) stack
- 0 -- System stack (system processing)
- 1 -- Interrupt processing only

\$TKTCB - Pointer to the Current Task Control Block (TCB)

The current TCB is the TCB of the user-level task in control of the CPU.

\$HEADR - Pointer to the Current Task Header

The \$HEADR word points to the header of the task currently running. The task header provides additional data to help isolate the fault. Figures E-1 and E-2 show the layout of task headers for unmapped and mapped systems.

The first word in the header is the user task's stack pointer (SP) the last time it was saved. If the user task branches wildly into the Executive, the Executive terminates the user task, but the system continues to function (possibly erroneously on an unmapped system). On a mapped system, only a privileged task can enter and corrupt the Executive. Knowing the user task's stack pointer provides one more link in the chain that may lead to the resolution of the fault.

The header (as pointed to by \$HEADR) also contains the last-saved register set, just before the header guard word (the last word in the header, pointed to by H.GARD).

GENERAL FAULT ISOLATION

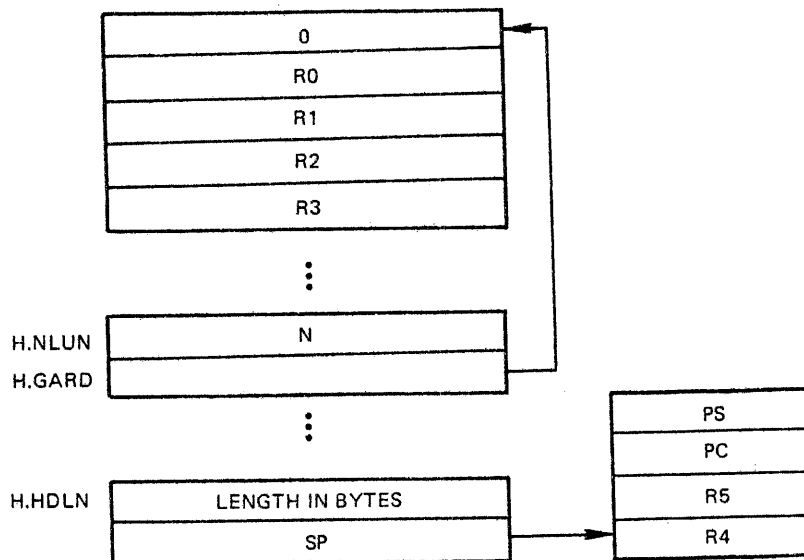


Figure E-1 Task Header on an Unmapped System

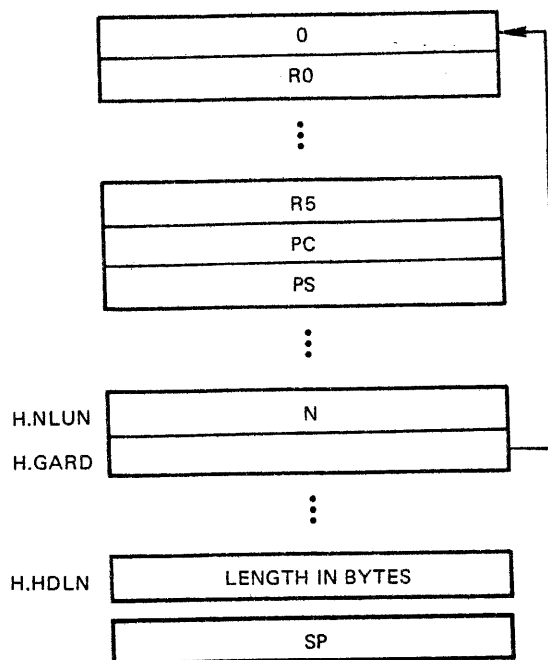


Figure E-2 Task Header on a Mapped System

E.3.2.1 Tracing Faults Using a Display of the Executive Stack and Registers - To trace a fault after a display of the Executive stack and register contents, first examine the system stack pointer. Usually an Executive failure is the result of an SST-type trap within the Executive. If an SST does occur within the Executive, the origin to the call on the crash reporting routine is in the SST service module. (The crash call is initiated by issuing an IOT at a stack depth of zero or less.)

GENERAL FAULT ISOLATION

A call to crash also occurs in the Directive Dispatcher when an EMT is issued at a stack depth of zero or less, or a trap instruction is executed at a depth of less than zero. The stack structure in the case of an internal SST fault is shown in Figure E-3.

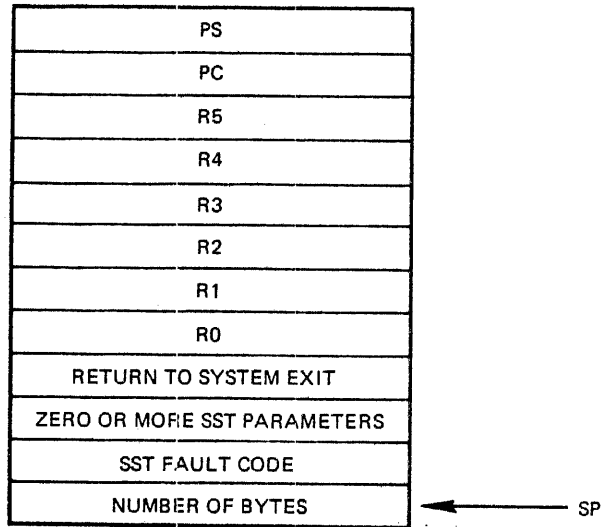


Figure E-3 Stack Structure: Internal SST Fault

The fault codes are:

0	Odd address and traps to 4
2	Memory protect violation
4	Break point or trace trap
6	IOT instruction
10	Illegal or reserved instruction
12	Non-RSX EMT instruction
14	Trap instruction
16	11/40 floating point exception
20	SST abort - bad stack
22	AST abort - bad stack
24	Abort via directive
26	Task load read failure
30	Task checkpoint read failure
32	Task exit with outstanding I/O
34	Task memory parity error

The PC points to the instruction following the one which caused the SST failure. The number of bytes is the number normally transferred to the user task's stack when the particular type of SST occurs. If the number is 4, an abnormal SST fault occurred, and only the PS and PC are transferred. There are no SST parameters.

If the failure is detected in \$DRDSP the stack is the same as that shown in Figure E-3, except the number of bytes, the SST fault code (the fault codes are listed above), and the SST parameters are not present. The crash report message, however, will indicate that the failure occurred in \$DRDSP.

GENERAL FAULT ISOLATION

One SST-type failure, stack underflow, does not result in the stack structure of Figure E-3. To determine where the crash occurred, first establish the stack structure; this can be deduced by the value of the SP and the contents of the top word on the stack. If the stack structure is that of Figure E-3, then the failure occurred in \$DRDSP, or was a normal SST crash. If the stack structure is that of Figure E-4, then an abnormal SST crash has occurred.

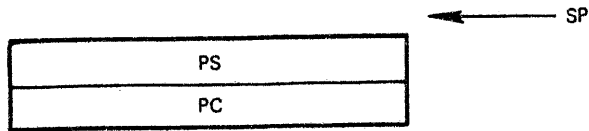


Figure E-4 Stack Structure: Abnormal SST Fault

Abnormal SST failures occur when it is not possible to push information on the stack without forcing another SST fault. When this situation occurs, a direct jump to the crash reporting routine occurs rather than an IOT crash. The PS and PC on the stack are those of the actual crash, and the address printed out by the crash-reporting routine is the address of the fault rather than the address of the IOT that crashes the system. Note that the crash reporting routine removes the PC and PS of the IOT instruction from the stack, which in this case is incorrect. Thus, the SP appears to be 4 bytes greater than it really is (as in Figure E-4).

You now have all the information needed to isolate the cause of the failure.

E.3.2.2 Tracing Faults When the Processor Halts Without Display - To trace a fault when the processor halts but displays no information (Case 3 as described above), first examine \$STKDP, \$TKTCB, and \$HEADR. Tracing failures in this case is difficult because the system stack is not directly associated with the cause of a failure.

By examining \$STKDP, you can determine the system state at the time of the failure. If the system was in user state, examine the user task's stack. The examination focuses on scanning the stack for addresses that may be subroutine links that can ultimately lead to a thread of events isolating the fault. This is essentially the aim of looking at the system stack if \$STKDP is zero or less.

Frequently, a fault can occur that causes the SP to point to the Top of Stack (TOS)+4. This fault results from issuing an RTI when the top two items on the stack are data instead of the return address (PC) and processor status (PS) for the routine being returned to. A wild branch occurs followed, most probably, by a halt.

Figure E-5 shows a case in which two data items are on the stack when the program executes an RTI. TOS points to a word containing 40100. This word will be the new PC. Suppose that location 40100 contains a halt. After the RTI and the halt occurs, the SP points four bytes above the previous location, and fault tracing should begin from the previous SP.

GENERAL FAULT ISOLATION

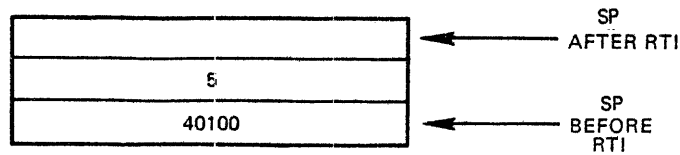


Figure E-5 Stack Structure: Data Items on Stack

This type of fault also occurs when an RTS instruction is executed with an inconsistent stack; that is, a stack with the wrong contents for the correct execution of the RTS instruction. The RTS instruction pops the top element off the stack. The top element of the stack becomes the new PC or occupies a register depending on the form of the RTS instruction. However, in any case, SP points to TOS+2 after an RTS instruction.

A scan of the contents of the general registers may give some hint as to the neighborhood in which a fault (or the sequence of events leading up to the fault) occurred.

If the fault occurred in a new driver, the buffer address and count words in the UCB (U.BUF, U.BUF+2, U.CNT), and the activity flags (US.BSY and S.STS) are frequent sources of clues. Other locations in both the UCB and SCB may also provide information that may help locate the source of the fault.

E.3.2.3 Tracing Faults After an Unintended Loop - To trace a fault when an unintended loop has occurred, first halt the processor. After you halt the processor, the same state exists as in tracing faults when the processor halts without display (as described above). Follow the same tracing procedure described there. A specific suggestion is to check for a stack overflow loop. Patterns of data successively duplicated on the stack indicate a looping failure.

E.3.2.4 Additional Hints for Tracing Faults -

E.3.2.5 I/O Packets - Another item to check is the current (or last) I/O packet, the address of which is found in S.PKT of the SCB. The packet function (I.FCN) defines the last activity performed on the unit.

If trouble occurred in terminating an I/O request, a scan of the free block list for the system Dynamic Storage Region may provide some insight. This list starts at the address contained in \$CRAVL, a cell in SYSCM. Because all I/O packets are built in the system's Dynamic Storage Region, their memory is returned to the Dynamic Storage Region when they are successfully terminated. Following the link pointers in this region may reveal whether I/O completion proceeded to that point. In systems with QIO optimization, \$PKAVL (SYSCM) points to a list of I/O packet-sized blocks of dynamic memory that are not linked into the \$CRAVL chain. Normally, the last completed I/O packet is the first in the \$PKAVL list.

GENERAL FAULT ISOLATION

A frequent error for an interrupt-driven device is to terminate an I/O packet twice when the device is not properly disabled on I/O completion and an unexpected interrupt occurs. This action ultimately produces a double deallocation of the same packet of dynamic memory. Double deallocation of a dynamic buffer in RSX-11M causes a loop in the routine \$DEACB (in the CORAL module) on the next deallocation (of a block of higher address) after the second deallocation of the same block. At that time, R2 and R3 both contain the address of the I/O packet memory that has been doubly deallocated. If XDT has been included in the system, the deallocation routine checks for bad deallocation and crashes the system if it occurs. Note, that queue I/O optimization, which employs pre-allocated I/O packets, may cause a double-deallocation error to get by. This may happen if the I/O packet in question does not become deallocated and is put in a separate list. To avoid this, use the SET /MAXPKT=0 command to temporarily void the optimization.

E.3.2.6 System Stack Pointer - Use the system stack pointer to find the current position of the system stack. The system stack will contain addresses or instructions of code that was being executed prior to the crash (this is only true if the \$STKDP is less than 1). From this point, all call activity can be traced, with the help of the listings, back to the original \$\$WSTK or EMT377 call.

APPENDIX F

SYSTEM TUNING

This appendix is a brief synopsis of ways to improve the performance of an RSX-11M system. There may be many conflicting trade-offs and requirements in your particular installation; therefore, some techniques may not be applicable to your situation. This appendix is not intended to be the last word in system tuning, but is written in the nature of a general approach to system tuning and contains some of the techniques and concepts you can use to improve performance.

F.1 HARDWARE CONSIDERATIONS

The most obvious approach to improving system performance is to upgrade the hardware to faster and/or larger capacity devices. If your applications make heavy demands upon CPU time, it makes sense to upgrade to a faster processor. If your application involves many disk transfers, more CPU memory may be needed to reduce the number of transfers or fixed head disks may be needed to reduce transfer time. Heavily overlaid tasks require many disk transfers. If this constantly occurs, more memory may allow fewer task overlays. The use of memory management directives instead of overlaid tasks could improve system performance in this case. Also, consider the use of resident overlays or multiple copies of F11ACP, one for each volume, if you must use overlaid tasks with high I/O requirements. However, resident overlays require more memory.

F.2 MEMORY LAYOUT

The memory layout of your particular system is an important factor that affects system throughput. The following procedures improve throughput:

- If your CPU has enough memory, put the following system programs in separate partitions to avoid their checkpointing each other:
MCR, F11ACP, TKTN, SHUFFLER
- Load loadable drivers into special driver partitions to avoid fragmenting memory.
- Arrange to have PLAS dynamic regions in their own special partitions.

SYSTEM TUNING

F.3 EXECUTIVE SOFTWARE OPTIONS

The options listed below are covered in detail in the RSX-11M System Generation Manual. The most important options that affect system performance include:

- Choice of FllACP - If your memory is big enough, install BIGFCP as memory-resident to greatly improve disk access time. Install separate copies of FllACP for each disk if file access is frequent and occurs to many different disks. The /ACP switch in the Mount command selects the copy of FllACP for that particular disk.
- Dynamic checkpoint allocation on the fastest system disk on your system.
- Shuffling - The Shuffler, a nonresident privileged task for mapped systems compacts memory in a system-controlled partition. This task minimizes memory fragmentation and provides better memory usage. The Shuffler is a SYSGEN option.
- Round-robin scheduling - Selection of the round-robin scheduling option during SYSGEN ensures that tasks in a range of priorities share the use of memory. This is important when many CPU-bound tasks within a specified range of priorities are running at the same time.
- Swapping - This SYSGEN option allows tasks to be checkpointed by tasks of equal priority when many tasks of equal priority are competing for memory. The checkpointing option is a pre-requisite.
- Memory management directives - Allows tasks to access a greater span of physical memory than their normal 32K addressing capability allows.
- Loadable device drivers - Loadable drivers may slow down I/O a bit, however, they occupy memory only when needed.
- Queue I/O speed optimizations - I/O throughput is increased at the expense of additional Executive code. Because of this option, I/O packets are pre-allocated, which makes them quickly available for QIOs.
- Install, request, and remove on exit support - This option conserves Dynamic Storage Region space by allowing tasks to be memory resident only when active, but does not increase throughput.
- Checkpointing on terminal input - This SYSGEN option allows more copies of terminal I/O-bound tasks to be run than normally could be run in a given amount of memory.
- Error logging - This option can be useful when information about system errors (device errors, timeouts, cache and memory parity errors) is needed. However, file space is needed for the records and this option automatically includes support for checkpointing and the Send/Receive directives.
- Size of dynamic storage region - It is possible to exhaust dynamic memory during the running of RSX-11M. This can occur if a large number of tasks are installed, if many volumes are mounted, or if a number of other conditions are present. This

SYSTEM TUNING

problem can manifest itself in the lack of any response from the system. Although the system looks like it is operating successfully, it cannot accept any MCR commands or complete the execution of presently running tasks because they all require dynamic memory. Another manifestation of the problem may be I/O errors being received by tasks or failures in the Loader due to pool allocation failures. The only solution to this problem is to bootstrap the system and make more dynamic space available (SET /POOL), remove tasks, mount fewer volumes, rebuild FCP with space for more file control blocks in its partition, or rebuild the system with more dynamic memory. See the RSX-11M System Generation Manual for a complete and detailed discussion of dynamic memory - its uses and its users.

- VMR installable Loader task - The Loader task occupies 2200(8) bytes and is one of the Executive tasks. Installing and fixing the Loader in its own partition releases the 2200(8) bytes for use in the Dynamic Storage Region.

F.4 FILE SYSTEM OPTIONS

Many options of the INI and MOU commands can change performance. The more important options are:

- /EXT default extension block count - Unless this block count is specified, the file has a default block count of five, whether it uses all five blocks or not. If the default is used and the file is larger than five blocks, the file is not continuous and more accesses are required for the file. If the file uses less than five blocks, all five blocks are reserved for the file and no other file can use them. On the other hand, if you specify one as the block count, the file gets only the number of blocks it needs but it may require many accesses to access the whole file. To conserve space and reduce the number of file accesses, be as specific as possible about the number of blocks that your files need.
- /INDX index file position - The position of the index file in large volumes is important because of seek time. Rather than have the index file at the beginning or end of a volume, position it, either by block number or the MID key word, at the mid-point of the volume. In the case of small volumes, such as a floppy-disk, putting the index file at the mid-point will limit the maximum size of the work files on the disk. In this case, put the index file at the beginning or end of the volume.
- /LRU memory buffers to speed up directory searches - LRU specifies the number of 512-byte buffers to be maintained in memory. The buffers contain only the most recently accessed directories. The default is three buffers. If your application is working with a small number of directories, the defaults may be sufficient. However, if many directories are being scanned frequently, access time will improve if you specify a higher number than three. However, more buffers use more memory.
- /WIN mapping pointer count - WIN specifies the number of mapping pointers to be allocated for file windows. The default is seven pointers. The pointers point to contiguous blocks of the file on the disk. Access to fragmented files may be optimized by increasing the number of pointers. However, additional memory may be freed by reducing the number of pointers for files with little or no fragmentation.

SYSTEM TUNING

F.5 HELPFUL HINTS

- Avoid entering CTRL/C to explicitly request MCR. Typing CNTRL/C slows down response to other users needing MCR facilities.
- Use indirect command files to MAC and TKB whenever possible, especially if the commands are long. If you do this, MAC or TKB spend less time competing for resources.
- When task building or assembling large programs, insert all modules in a created library and specify each module by name. This reduces the number of times that TKB or MAC have to open files.
- If you are doing SYSGEN with a mapped baseline system and your system has enough memory, create a separate 32K partition and install PIP, BIGTKB, and BIGMAC in this partition; this shortens the time taken for SYSGEN.
- If you have the room, install BIGTKB and BIGMAC with a large /INC. This reduces the system overhead that TKB and MAC would cause when they executed Extend Task directives.
- Install PIP with an increment (/INC). In general, run utilities in the largest possible partition.
- If your system uses RMS, build the library modules as memory-resident.
- Use the following form, shown as an example, to call utilities whenever you want to use a utility once:

```
>PIP command <CR>
```

Using this form reduces the time that PIP (or any other utility) spends competing for the memory resource. However, if you intend to use a utility to perform multiple operations, use:

```
>PIP  
PIP> command <CR>  
PIP> command <CR>  
PIP> command <CR>  
CTRL/Z  
>
```

- Use the print spooler as much as possible. This avoids leaving utilities in memory while printing on the line printer.
- To save memory space by reducing task size, you may want to build a resident (sharable) library containing frequently-used FORTRAN routines. For the same reason, you may want to produce a position-independent library containing all the FCS modules required by a set of programs that are intended to run simultaneously. FCSRES.MAC and FORRES.MAC files are provided on the distribution disk. Building a FORTRAN and FCS resident library is described in the RSX-11M System Generation Manual.
- Plan the overlay structure of your tasks. You may possibly achieve a large reduction in disk accesses by careful planning.

SYSTEM TUNING

- Avoid putting all the files needed by an application in a single UIC. Having all files in a single UIC slows down directory searches as each block can hold 32 entries only (8 words per entry). Many more disk accesses are required to search for the 300th file in a single UIC than the 30th file in the last of 10 UICs.
- RMDEMO does add some overhead to the system's operation. It is not necessary to run RMDEMO continuously, only when the information that it supplies is required. However, if you do not run RMDEMO continuously, you may not be able to observe the system when a problem occurs (for example, dangerously low and decreasing pool space).

F.6 SOME USEFUL COMMANDS

There are four commands that can give you a description of your system that may be useful when you tune your system. They are the Tasklist, the Partition Definitions, the Task List, and the Active Task List commands. These commands are more fully described in the RSX-11M Operator's Procedures Manual.

TASK LIST - Short Version

The Tasklist command displays on the entering terminal a brief description of each installed task. The display contains, from left to right:

1. Task name
2. Task version identification
3. Partition name
4. Task priority
5. Size of task in bytes (8)
6. Load device identification
7. Disk address logical block number (8)
8. Task memory state

Format:

TAS

Example:

```
...LDR 07.05  LDRPAR 248. 00002200 LB0:-01035303 FIXED
TKTN  03.3   SYSPAR 248. 00010000 LB0:-01126742
RMDEMO X03.03 GEN    225. 00013700 SY0:-00352100
MTAACP 0006  GEN    200. 00013000 DS0:-00000777
FllMSG V0010 GEN    200. 00005400 SY0:-01053030
...MCR 01    GEN    160. 00025000 LB0:-01051676
      .
      .
      .
      ETC.
```


SYSTEM TUNING

PARTITION DEFINITIONS

The Partition Definitions command displays on the entering terminal a description of each memory partition in the system.

The display consists of five columns that specify (in order from left to right):

1. Partition name
2. Partition base address (8)
3. Partition size (8)
4. Partition kind: main partition (MAIN) or subpartition (SUB)
5. Partition type

TASK for user-controlled
COM for common
DEV for device registers
SYS for system-controlled
Taskname for task region
D ...MIC for dynamically created region
DRIVER for region occupied by a loadable driver

Format:

PAR

Example:

Name	Base	Size	Type
LDR	000000	000000	MAIN TASK
SYS PAR	120000	010000	MAIN TASK
FCPPAR	130000	026000	MAIN TASK
PMDPAR	156000	020000	MAIN TASK
SPLPAR	156000	010000	SUB TASK
DRV PAR	176000	014000	MAIN SYS
	176000	001600	SUB DRIVER - DB:
	177600	000500	SUB DRIVER - DS:
	200300	001000	SUB DRIVER - DK:
	201300	001100	SUB DRIVER - DT:
	202400	001000	SUB DRIVER - LP:
	203400	003100	SUB DRIVER - MM:
GEN	212000	546000	MAIN SYS
	212000	013400	SUB (RMDEMO)
	225400	045700	SUB (...EDI)
		.	
		.	
		.	
		ETC.	

TASK LIST - Long Version

The Task List command displays on the entering terminal the names, descriptions, and status of all installed tasks or of a specific installed task in the system.

SYSTEM TUNING

The display contains, reading from left to right, the following information for each task:

- Task name
- Task control block physical address (8)
- Partition name
- Partition control block physical address (8)
- Partition base and limit physical address (8)
- Task's running priority and default priority
- Task status flags
- TI terminal physical device unit
- I/O count (10)
- Task local event flags
- Task registers and processor status word (memory-resident tasks only)

Format:

TAL [taskname]

Example:

```
. LDR. 052220 LDR 052164 00000000-00000000 PRI - 248. DPRI - 248.
STATUS: -CHK FXD STP PRV
TI - COO: IOC - 0. EFLG - 000001 000000 PS - 170000 PC - 041350
REGS 0-6 000162 004030 177777 105312 064254 105260 052132
TKTN 105010 SYSPAR 107734 00110000-00120000 PIR - 248. DPRI - 248.
STATUS: -EXE OUT -CHK -PMD PRV
TI - COO: IOC - 0. EFLG - 000001 000000
      .
      .
      .
      ETC.
```

ACTIVE TASK LIST

This command displays on the entering terminal the names, descriptions, and status of all active tasks in the system or the status of a specified active task (taskname). The display is the same as the Task List command.

Format:

ATL [taskname]

SYSTEM TUNING

F.7 A USEFUL TOOL

RMDEMO is a privileged task that displays in a highly visual and active manner information concerning task activity in an RSX-11M system. The display usually runs continuously on a video terminal with cursor control (the VT05B or the VT52). Certain versions of the task allow a once-only display on any record-oriented device.

The display features include:

1. Current date and time;
2. The currently active task;
3. All tasks, loaded drivers, and common blocks that are currently in memory, displayed in a graphic fashion to show their individual memory requirements and location relative to other tasks;
4. The number of active tasks currently in memory, the number not currently resident, and the total amount of memory occupied by each group;
5. The current amount of available system dynamic memory (POOL space), including the largest available block and the number of fragments;
6. A graphical display of the partition information;
7. The number of hours that have elapsed since RMDEMO was initiated (this is system up time if the STARTUP_CMD file automatically runs RMDEMO when your system comes up).
8. The number of free blocks available on the system device and one or more additional mounted FILES-11 volumes;
9. The system error sequence count.

RMDEMO scales the display to the memory size of the computer on which it is executing. The amount of memory displayed on the screen is always the next largest power of 2 in K-words. For a 28K system, it displays 32K. For a 256K system, 256K is displayed and the available memory just fills the screen.

You should consider task locations and sizes that RMDEMO displays to be approximate only. RMDEMO provides a visual display and is a system debugging tool; however, it should not be used for accurate measurement of task size and location.

A complete description of RMDEMO can be found in the RSX-11M System Generation Manual.

INDEX

A

- Abort codes, task 8-47
- ABRT\$ macro C-6
- ABRT\$C macro C-5
- ABRT\$\$ macro C-6
- ACP codes, network C-119
- \$ACTHD pointer 8-5
- Active Task List 8-11
- Active Task List 8-5
- Active task, fixing 1-11
- Active tasks 1-10
- Address space, logical 1-14
 - logical 1-15, 2-3
 - logical, extended 1-14
 - physical 1-15
 - virtual 1-15, 2-3
- Addresses, logical 2-3
 - physical 1-3, 2-3
 - virtual 1-3, 2-3
- Addressing 1-3
- Addressing scheme 1-3
- Addressing, logical 2-2
 - memory 2-1
 - overlaid task 2-1
 - virtual 2-2
- \$ALCLK diagram 2-25
- Allocation, memory 2-10
- \$ALOCB diagram 2-26
- Alter command 1-12
- Altering priority 1-12
- ALTP\$ macro C-7
- ALTP\$C macro C-6
- ALTP\$\$ macro C-7
- ALUN\$ macro C-9
- ALUN\$C macro C-8
- ALUN\$\$ macro C-8
- Argument passing B-8
- Assembly parameter definition
 - listing, conditional C-1
- Assigning processor control by priority 1-11
- Assigning task priority 1-11
- AST Control Block 8-14, 8-28
- AST queue 8-14
- ASTX\$ macro C-10
- ASTX\$\$ macro C-10
- ASTX\$C macro C-10
- Asynchronous events 1-2
- Asynchronous System Trap Control Block 8-28
- Asynchronous traps 1-18
- ALT 8-5
- ALT list 8-11
- ATRG\$ macro C-101
- ATRG\$C macro C-101

- ATRG\$\$ macro C-101
- Attached terminals 1-26
- Attaching to regions 2-9
- Attachment descriptor offsets 8-43
- Attachment descriptor status byte bit definitions 8-43

B

- BDOFF\$ macro C-79
- BFCTL module description 7-2
- BIGFCP global cross-reference 9-54
- BIGFCP segment cross-reference 9-69
- .BLK macro C-102
- .BLKB macro C-102
- .BLKW macro C-102
- Blocked tasks 1-10
- Boot command, 1-2
- Bootstrap 1-2

C

- CALL macro C-71
- CALLR macro C-71
- CBYTE\$ macro C-80
- Checkpoint space allocation, for tuning F-2
- Checkpointable task 1-7
- Checkpointable task, fixing a 1-11
- Checkpointing 1-7, 1-12, 2-10, 2-13
- Checkpointing on terminal input for tuning F-2
- Checkpointing, disk space for 1-13
 - during terminal input wait 2-10
 - dynamic 1-13
 - in system-controlled partitions 2-11
 - in user-controlled partitions 2-10
- \$CHKPT diagram 2-28
- \$CHKPT routine 2-22
- CINT\$ macro C-12
- CINT\$C macro C-11
- CINT\$\$ macro C-11
- Classification of faults E-1
- CLEF\$ macro C-14
- CLEF\$C macro C-13
- Clock queue 8-16

INDEX (Cont.)

- Clock Queue Control Block 8-31
 - CLOSE\$ macro C-80
 - CMKT\$ macro C-15
 - CMKT\$C macro C-14
 - CMKT\$S macro C-15
 - CMOV\$2 macro C-80
 - CMOV\$B macro C-80
 - CMOV\$W macro C-81
 - Co-routines B-14
 - Code with interrupts inhibited B-11
 - Coding standards and conventions B-1
 - Coding standards and conventions, argument passing B-8
 - co-routines B-14
 - code with interrupts inhibited B-11
 - comments B-2
 - common exits B-10
 - device registers B-2
 - displaying the version identifier B-13
 - exiting B-8
 - formatting B-9
 - general purpose registers B-2
 - global symbols B-3, B-4
 - instruction usage, forbidden B-11
 - intra-module calling conventions B-8
 - line format B-1
 - macro names B-4
 - modularity B-8
 - module checking routines B-8
 - module preface B-5
 - naming B-2
 - other symbols B-2
 - processor priority B-2
 - program modules B-5
 - program source files B-11
 - program-local symbols B-4
 - registers B-2
 - standard symbols B-3
 - success or failure indications B-8
 - version number standard B-12
 - version number usage B-14
 - COMDF\$ macro C-100
 - Commands for tuning F-5
 - Commands, Alter 1-12
 - Boot 1-2
 - Fix 1-11
 - Install 1-10
 - Install 1-3
 - privileged 1-25
 - Remove 1-11
 - Save 1-2
 - Unfix 1-11
 - Comments B-2
 - Common exits in coding B-10
 - Communications Control Block 8-32
 - Communications Vector 8-33
 - Conditional assembly parameter definition listing D-1
 - Conditional assembly parameter to module cross-reference 9-69
 - Control blocks 8-1
 - Control blocks, system, overview 8-3
 - Conventions and standards for coding B-1
 - CORAL module description 7-4
 - CRRG\$C macro C-103
 - CRRG\$S macro C-103
 - CRASH macro C-72
 - CRASH module description 7-6
 - \$CRAVL pointer 8-6
 - CRAW\$ macro C-102
 - CRAW\$C macro C-102
 - CRAW\$S macro C-102
 - Cross-reference, conditional assembly parameter to module 9-69
 - Executive module to routine 9-1
 - module to conditional assembly parameter 9-75
 - CRRG\$ macro C-103
 - CRI\$ macro C-4
 - CSI\$1 macro C-1
 - CSI\$2 macro C-1
 - CSI\$SV macro C-3
 - CSI\$SW macro C-2
 - CSI\$ND macro C-3
 - CSRQ\$ macro C-16
 - CSRQ\$C macro C-15
 - CSRQ\$S macro C-16
 - CVRTM module description 7-7
 - CWORD\$ macro C-81
- ## D
- Data areas 8-1
 - Data structure, I/O 8-27
 - DCB, linkage of 8-4
 - SCB, UCB, LCB relationship 8-18
 - UCB, SCB relationship 8-24

INDEX (Cont.)

- DCB-UCB relationship 8-4
 - \$DEACB diagram 2-30
 - DECL\$ macro C-18
 - DECL\$C macro C-17
 - DECL\$\$ macro C-17
 - \$DECLK diagram 2-30
 - DEF\$G macro C-81
 - DEF\$I macro C-81
 - DEF\$L macro C-81
 - DEF\$N macro C-82
 - Default priority, task 1-11
 - DEFIN\$ macro C-82
 - DELET\$ macro C-82
 - \$DEPKT diagram 2-30
 - \$DEVHD pointer 8-4
 - Device Control Block 6-1, 6-2, 8-34
 - Device Control Block pointer 8-3
 - Device drivers 1-5
 - Device handler codes, general C-116
 - Device name, in DCB 8-4
 - Device registers B-2
 - Device table status definitions 8-53
 - Devices, private 1-27
public 1-27
 - Diagnostic functions, user-mode C-123
 - Diagnostic tasks 1-28
 - DIR\$ macro C-18
 - Directive macro expansions C-5
 - Directive Parameter Block 6-3
 - Directive processing routines, interrupt processing 3-9
 - Directive processing, QIO 6-5
stack state upon entry into 3-10
 - Directives, Executive 1-19
memory management 2-2, 1-15
 - Directory primitive codes C-116
 - \$DIRSV routine 3-3
 - \$DIRSV routine operation 3-7
 - \$DIRSV routine, example use of 3-7
 - \$\$DIRSV routine, use of by EMTRP 3-8
 - DIRSV\$ macro C-72
 - \$DIRXT routine 3-3
 - \$DIRXT routine processing 3-16
 - Disk controller control block structure 8-25
 - Disk space for checkpointing 1-13
 - Disk swapping 2-12
 - Dormant tasks 1-10
 - Dormant tasks, activating 1-10
 - DRABO module description 7-7
 - DRASG module description 7-8
 - DRATX module description 7-8
 - DRCIN module description 7-9
 - DRDMI module description 7-11
 - DRDAR module description 7-12
 - DRDCP module description 7-13
 - DRDSP module description 7-14
 - DREIF module description 7-15
 - DRERR\$ macro C-115
 - DREXP module description 7-17
 - DRGCL module description 7-18
 - DRGLI module description 7-19
 - DRGPP module description 7-19
 - DRGSS module description 7-20
 - DRGTK module description 7-21
 - DRGTP module description 7-21
 - Driver Dispatch Table offsets 8-36
 - Driver standard codes C-117
 - DRMAP module description 7-22
 - DRMKT module description 7-29
 - DRPUT module description 7-31
 - DRQIO module description 7-33
 - DRRAS module description 7-34
 - DRREG module description 7-36
 - DRREQ module description 7-39
 - DRRES module description 7-40
 - DRSED module description 7-42
 - DRSST module description 7-45
 - DRSTS macro C-72
 - DRVPAR partition 1-8
 - DSAR\$ macro C-19
 - DSAR\$\$ macro C-19
 - DSAR\$C macro C-19
 - DSCP\$ macro C-20
 - DSCP\$C macro C-20
 - DSCP\$\$ macro C-20
 - DSR (See Dynamic Storage Region)
 - DTRG\$ macro C-104
 - DTRG\$C macro C-104
 - DTRG\$\$ macro C-104
 - Dynamic checkpoint space 1-13
 - Dynamic region 2-8
 - Dynamic Storage Region 1-4, 8-6
 - Dynamic Storage Region free
block queue 8-23
- E**
- ECSBT\$ macro C-82
 - ELAW\$ macro C-104
 - ELAW\$C macro C-104
 - ELAW\$\$ macro C-104

INDEX (Cont.)

EMT trap processing routine 3-8	CVRTM 7-7
ENAR\$ macro C-21	DASTT 7-72
ENAR\$C macro C-21	DEAC1 7-5
ENAR\$\$ macro C-21	DEACB 7-5
ENCP\$ macro C-22	DECLK 7-5
ENCP\$C macro C-22	DEPKT 7-6
ENCP\$\$ macro C-22	DETRG 7-39
ERR\$ macro C-23	DEUMR 7-53
Error codes, ICR C-119	DEVTE 7-83
Set characteristics C-130	DIRSV 7-84
TTY C-119	DIRXT 7-87
Error logging 1-27, F-2	DISIN 7-10
Error Message Block 8-36	DIV 7-61
ERROR module description 7-47	DQAC 7-72
Event flags 1-24	DQUMR 7-54
Events, asynchronous 1-2	DRABO 7-7
synchronous 1-2	DRASG 7-8
Executive code 1-4	DRATP 7-41
Executive directive summary 1-19	DRATR 7-38
Executive directives 1-19	DRATX 7-9
Executive global cross-	DRCEF 7-42
reference 9-18	DRCIN 7-9
Executive macro expansions C-71	DRCMT 7-11
Executive module descriptions	DRCRR 7-36
7-1	DRCRW 7-23
Executive routines, ABCTK 7-70	DRCRSR 7-11
ABTSK 7-70	DRDAR 7-12
ACHCK 7-51	DRDCP 7-13
ACHK2 7-51	DRDSE 7-42
ACHKB 7-51	DRDTR 7-38
ACHKP 7-51	DREAR 7-12
ACHKW 7-51	DRECP 7-14
ACTRM 7-73	DREIF 7-16
ACTTK 7-71	DRELW 7-24
ALCLK 7-5	DREXP 7-17
ALEB1 7-47	DREXT 7-16
ALEMB 7-47	DRFEX 7-31
ALOC1 7-4	DRGCL 7-18
ALOCB 7-4	DRGLI 7-19
ALPKT 7-5	DRGMX 7-28
ASUMR 7-51	DRGPP 7-19
BILDS 7-70	DRGSS 7-20
BLKCL 7-53	DRGTP 7-21
BLKCK 7-53	DRGTX 7-21
BLXIO 7-4	DRMAP 7-25
BMSET 7-47	DRMKT 7-30
CEFI 7-52	DRPUT 7-32
CEFN 7-52	DRQIO 7-33
CHKPT 7-75	DRQRQ 7-34
CKACC 7-65	DRRAF 7-43
CKINT 7-89	DRRCV 7-33
CLINS 7-68	DRREC 7-35
CLRMV 7-68	DRREQ 7-39
CRASH 7-6	DRRES 7-40
CRATT 7-65	DRRRA 7-32
CRPAS 7-61	DRRRF 7-28

INDEX (Cont.)

DRRUN	7-30	NS0	7-49
DRSDV	7-45	NS1	7-49
DRSEF	7-43	NS2	7-49
DRSND	7-35	NS3	7-49
DRSPN	7-41	NS4	7-49
DRSRF	7-27	NS5	7-49
DRSTV	7-46	NS6	7-49
DRUNM	7-26	NS7	7-49
DRWFL	7-44	NXTSK	7-74
DRWFS	7-44	PARER	7-63
DRWSE	7-44	POWER	7-67
DTOER	7-48	PTBYT	7-3
DVCER	7-48	PTWRD	7-3
DVMSG	7-52	QASTC	7-72
ECCOR	7-60	QASTT	7-72
EMSST	7-77	QEMB	7-49
EMTRP	7-15	QINSF	7-68
EXRQF	7-75	QINSP	7-68
EXRQN	7-75	QMCR	7-69
EXRQP	7-75	QRMVF	7-69
FINBF	7-88	QRMVT	7-69
FLTRP	7-77	RELOC	7-58
FLTRP	7-78	RELOM	7-58
FNDSP	7-74	RELOP	7-61
FORK	7-85	RLCH	7-59
FORK0	7-85	RLMCB	7-18
FORK1	7-85	RLPAR	7-73
FORK2	7-85	RLPR1	7-73
GTBYT	7-2	RQCH	7-59
GTCWD	7-4	SAVNR	7-88
GTPKT	7-53	SCDV1	7-59
GTWRD	7-3	SCDVT	7-59
ICHP	7-75	SETCR	7-71
ILINS	7-78	SETF	7-71
INITL	7-50	SETM	7-71
INTSC	7-86	SETRQ	7-71
INTSE	7-86	SETRT	7-71
INTSV	7-86	SGFLT	7-79
INTX1	7-87	SRATT	7-65
INTXT	7-86	SRNAM	7-64
IOALT	7-54	SRSTD	7-73
IODON	7-54	SRWND	7-66
IOFIN	7-54	SSTXT	7-79
IOKIL	7-55	STMAP	7-60
IOTRP	7-78	STPCT	7-73
LCKPR	7-55	STPTK	7-73
LOADR	7-62	SWSTK	7-88
LOADT	7-75	TKWSE	7-43
MAPTK	7-76	TRACE	7-79
MPLNE	7-56	TRP04	7-79
MPLUN	7-56	TRTRP	7-15
MPPHY	7-57	TSKRP	7-76
MPPKT	7-57	TSKRQ	7-76
MPUBM	7-57	TSKRT	7-76
MPVBN	7-58	TSTCP	7-75
MUL	7-61	UISET	7-76

INDEX (Cont.)

UNMAP 7-66
 WTUMR 7-62
 Executive standard codes C-117
 Executive, contents 1-4
 EXIF\$ macro C-24
 EXIF\$C macro C-23
 EXIF\$\$ macro C-24
 EXIT\$ macro C-25
 EXIT\$C macro C-24
 EXIT\$\$ macro C-25
 Exiting B-8
 Exiting system state 3-15
 Expansions, macro C-1
 Express queue commands C-116
 /EXT option, for tuning F-3
 Extended arithmetic element
 registers 8-39
 External interrupts 1-17, 3-2
 EXTK\$ macro C-26
 EXTK\$C macro C-25
 EXTK\$\$ macro C-26

F

F11ACP choice, for tuning F-2
 Fault classification E-1
 Fault isolation data, gathering
 of E-3
 Fault servicing E-1
 Fault tracing E-4
 Fault tracing, additional hints
 E-8
 I/O packets E-8
 processor halt without display
 E-7
 system stack pointer E-9
 unintended loop E-8
 using the Executive stack E-5
 Fault, abnormal SST, stack
 structure E-7
 internal SST, stack structure
 E-6
 FCB 8-37
 FCPNMH 1-7
 FCPPAR partition 1-9
 FCS (See File Control Services)
 FCSCM macro C-84
 FDAT\$A macro C-85
 FDAT\$R macro C-85
 FDBF\$A macro C-85
 FDBF\$R macro C-86
 FDBDF\$ macro C-85
 FDBK\$A macro C-86
 FDBK\$R macro C-86
 FDBSZ\$ macro C-86

FDOF\$L macro C-88
 FDOFF\$ macro C-88
 FDOP\$A macro C-87
 FDOP\$R macro C-87
 FDRC\$R macro C-87
 FDRC\$A macro C-87
 FDSOF\$ macro C-89
 Feature symbol definitions 8-40
 FHD01\$ macro C-76
 FHD0F\$ macro C-77
 File Control Block 8-37
 File Control Services 1-6
 File Control Services, codes
 C-118
 macro expansions C-79
 File primitive codes C-118,
 C-116
 File system 1-6
 Files-11 header offsets, macro
 definitions C-76
 Files-11 system 1-6
 FILIO\$ macro C-116
 FINIT\$/FSRSZ\$ macros C-90
 Fixed tasks 1-11
 Fixing a checkpointable task
 1-11
 Fixing a task in memory 1-11
 Fixing an active task 1-11
 Flags, event 1-24
 Flow diagrams, memory allocation
 2-24
 \$FNDSPP, diagram 2-35
 routine 2-22
 Fork processing 3-13
 Fork queue 8-17
 \$FORK routine 3-13
 \$FORK, calling 3-13
 using 3-15
 \$FORK1, USING 3-15
 Formatting standards for coding
 B-9
 Formatting the module preface,
 rules for B-6
 Free block list 8-6
 \$FRKHD pointer 8-6
 FSROF\$ macro C-90
 Function codes for specific
 device dependent functions
 C-121

G

GEN partition 1-8
 General fault isolation E-1
 General purpose registers B-2

INDEX (Cont.)

Generation, system 1-1
 Get Command Line Control Block 8-38
 Get Terminal support, bits for return from C-131
 GET\$ macro C-91
 GET\$R macro C-91
 GET\$\$ macro C-91
 Global cross-reference, BIGFCP 9-54
 Executive 9-18
 MCRMU 9-30
 SYS 9-42
 Global symbols B-3
 GLUN\$ macro C-28
 GLUN\$C macro C-27
 GLUN\$\$ macro C-27
 GMCR\$ macro C-29
 GMCR\$C macro C-29
 GMCX\$ macro C-105
 GMCX\$C macro C-105
 GMCX\$\$ macro C-105
 GPRT\$ macro C-31
 GPRT\$C macro C-30
 GPRT\$\$ macro C-30
 GREG\$C macro C-105
 GREG\$\$ macro C-105
 GREG\$ macro C-105
 GSSW\$\$ macro C-32
 GSSW\$C macro C-32
 GSSW\$ macro C-32
 GTIM\$ macro C-34
 GTIM\$C macro C-33
 GTIM\$\$ macro C-33
 GTSK\$ macro C-35
 GTSK\$\$ macro C-35
 GTSK\$C macro C-34
 GTUCB\$ macro C-72

H

Hardware definitions 8-39
 Hardware register addresses 8-39
 Hardware register status codes 8-39
 \$HEADR pointer 8-7
 Hints for tuning F-4
 HMBOF\$/HMBO1\$ macros C-78

I

I/O control block linkage 8-26
 I/O data structure 8-27
 I/O data structures 6-1
 I/O functions IP11 C-123

I/O implementation 6-1
 I/O Packet queue 8-21
 I/O Packet offset definitions 8-30
 I/O processing 6-1
 \$ICHKP diagram 2-36
 ICR error codes C-119
 ICR QIO functions C-122
 ICS error codes C-119
 ICS QIO functions C-122
 IHAR\$ macro C-37
 IHAR\$C macro C-37
 IHAR\$\$ macro C-37
 /INDX option, for tuning F-3
 INITL module description 7-49
 Install command 1-3
 Install request and remove on exit support for tuning F-2
 Installed priority 1-12
 Installed priority, establishing, with Install 1-12
 Installed task, STD entry 1-10
 Installed tasks 1-10
 Instruction usage, forbidden B-11
 Interface, MCR 1-25
 Interrupt conventions, for drivers 3-13
 Interrupt locations 1-18
 Interrupt mechanism, for RSX-11M 3-1
 hardware 3-1
 Interrupt processing 1-2, 1-15, 3-1
 Interrupt processing code 3-17
 Interrupt processing routines, DIRSV 3-3
 DIRXT 3-3
 INTSV 3-2
 INTXT 3-3
 for drivers 3-3
 Interrupt routine, driver, example of 3-15
 Interrupt routines, processing within 3-12
 Interrupt Transfer Block 8-40
 Interrupt vectors 1-16
 Interrupts, classes of 3-1
 device 3-2
 Executive processing 3-2
 external 1-17
 external 3-2
 external, from system state 3-5
 from task state 3-4
 flow of control in 3-18
 loadable drivers 3-3

INDEX (Cont.)

- processing summary 3-19
 - queued on the sytem stack 3-12
 - resident drivers 3-3
 - stack processing 3-2
- Interrupt processes 3-2
- INTLB macro C-72
- Intra-module calling conventions B-8
- INTSE\$ macro C-73
- \$INTSV routine 3-2
- \$INTSV routine operation 3-5
- INTSV\$ macro 3-3
- INTSV\$ macro 3-5
- INTSV\$ macro C-73
- INTSV\$ macro expansion 3-6
- INTSV\$ macro format 3-4
- INTSV\$ macro, example of a
 - driver using 3-6
 - used in a driver 3-13
- \$INTSV, calling from driver 3-12
- \$INTXT routine 3-3
- \$INTXT routine processing 3-16
- .IOER macro C-117
- IOERR\$ macro 117
- IOSUB module description 7-50
- IP11 I/O functions C-123

K

- Keyword table generation macros C-125
- KT11 hardware 1-15
- KT11 memory management unit 1-3

L

- Label block offsets 8-51
- LCB list 8-19
- LCB, DCB, SCB, UCB relationship 8-18
- LDD macro C-76
- LDFPS macro C-76
- LDR0\$ macro C-4
- \$LDRPT pointer 8-6
- Library list entry flags 8-50
- Line format B-1
- Linkage, for I/O control blocks 8-26
- Linkages, system 8-1
- Linked lists on RSX-11M 8-2
- List, ATL 8-11
 - LCB 8-19
 - PCB 8-10

- STD 8-11
- Loadable device drivers for tuning F-2
- Loaded task 1-11
- Loader 1-5, 2-15
- Loader diagram 2-43
- Loader functions 2-15
- Loader queue 8-15
- Loader, defined 1-5
 - VMR installable, for tuning F-3
- LOADR module description 7-62
- Locked task 1-11
- Logical address space 1-14, 1-15
- Logical addresses 2-3
- Logical addresses space 2-3
- Logical addressing 2-2
- Logical assignment control block 8-41
- Logical assignment control block list 8-19
- LOWCR module description 7-62
- /LRU option, for tuning F-3

M

- Macro definitions, files-11
 - header offsets C-76
- Macro expansions C-1
- Macro expansions (see the specific macro name)
- Macro expansions, directive C-5
 - Executive C-71
 - file control services C-79
 - QIOMAC C-115
 - QIOSYM C-115
 - relative file C-110
- Macro names B-4
- Main partition 1-4
- Management, resource, memory 1-2
- MAP\$ macro C-106
- MAP\$C macro C-106
- MAP\$S macro C-106
- Mapped system, example of 1-8
- Mapping 1-3
- Mapping assignment block 8-46
- Mapping, 4k nonprivileged user task, in mapped system 4-12
- 8k nonprivileged user task, in mapped system 4-13
 - defined 1-3
 - in a mapped system 1-3
 - in an unmapped system 1-3

INDEX (Cont.)

- user task in unmapped system 4-11
- MCR 1-7
- MCR and terminal driver relationship 5-3
- MCR commands 5-1
- MCR dispatcher 5-5
- MCR interface 1-25
- MCR interface 5-1
- MCR operation 5-5
- MCR queue 8-20
- MCR task 1-5
- MCR, buffer processing 5-6
 - command overlay 5-7
 - common overlay 5-7
 - dispatcher 5-8
 - error overlay 5-10
 - explained 5-1
 - final exit 5-10
 - function tables 5-7
 - input queue entry 5-6
 - operating environment of 5-2
 - overlays 5-7
 - parser functions 5-9
 - parser overlay 5-7
 - parser table entry 5-8
 - structure of 5-2
- MCRMU 1-9
- MCRMU global cross-reference 9-30
- MCRMU, segment cross-reference 9-41
- Memory 1-2
- Memory addressing 2-1
- Memory allocation 2-10
- Memory allocation data structures 2-61
- Memory compaction 2-13
- Memory management directives 2-2
- Memory management for tuning F-2
- Memory management overview 2-1
- Memory management register addresses 8-39
- Memory management register status codes 8-39
- Memory management directives 1-15
- Memory management unit 1-3
- Memory partitions 1-2
- Memory resource allocation 2-1
- Memory resource management 1-2
- Memory structure 1-4
- MFPS/MTPS macros C-74
- Modularity B-8
- Module preface B-5
- Module preface, formatting the B-6
- Module to conditional assembly parameter cross-reference 9-75
- Module to routine cross-reference, Executive, BFCTL 9-1
- Executive,
 - CORAL 9-1
 - CRASH 9-1
 - CTDRV 9-1
 - CVRTM 9-2
 - DBDRV 9-2
 - DLDRV 9-2
 - DMDRV 9-2
 - DPDRV 9-2
 - DRABO 9-2
 - DRASG 9-2
 - DRATX 9-2
 - DRCIN 9-3
 - DRCMT 9-3
 - DRDAR 9-3
 - DRDCP 9-3
 - DRDSP 9-3
 - DREIF 9-3
 - DREXP 9-3
 - DRGCL 9-3
 - DRGLI 9-3
 - DRGPP 9-3
 - DRGSS 9-3
 - DRGTP 9-4
 - DRGTX 9-3
 - DRMAP 9-4
 - DRMKT 9-4
 - DRPUT 9-4
 - DRQIO 9-4
 - DRRAS 9-5
 - DRREG 9-5
 - DRREQ 9-5
 - DRRES 9-5
 - DRSED 9-6
 - DRSST 9-6
 - DTDRV 9-6
 - DXDRV 9-6
 - ERROR 9-6
 - INITL 9-7
 - IOSUB 9-7
 - LOADR 9-8
 - LOWCR 9-8
 - LPDRV 9-8
 - MMDRV 9-9
 - MTDRV 9-9
 - NLDRV 9-9
 - PARTY 9-9
 - PLSUB 9-10
 - POWER 9-10
 - PPTAB 9-10
 - PRDRV 9-10

INDEX (Cont.)

QUEUE 9-10
 REQSB 9-10
 SSTSR 9-11
 SYSCM 9-11
 SYSDF 9-12
 SYSTB 9-12
 SYSXT 9-12
 SYTAB 9-13
 TDSCH 9-13
 TTDRV 9-13
 XBDRV 9-14
 XMDRV 9-15
 XPDRV 9-16
 XQDRV 9-16
 XUDRV 9-16
 XWDRV 9-17
 MOV\$ macro C-38
 MRKT\$ macro C-39
 MRKT\$C macro C-38
 MRKT\$\$ macro C-39
 MTRAN\$ macro C-126
 Multiuser protection 1-27
 MVB\$ macro C-40

N

Naming B-2
 NBOF\$L macro C-91
 NBOFF\$ macro C-92
 Network ACP codes C-119
 Network symbol definition macros C-100
 NMBLK\$ macro C-92
 Nonprivileged tasks, in a mapped system 1-3
 \$NXTSK diagram 2-37
 \$NXTSK routine 2-13
 \$NXTSK, functions 2-17
 inputs 2-17
 operation in a system-controlled partition 2-18
 operation in a user-controlled partition 2-17
 routine 2-17
 routines that call 2-18
 routines that call, description of 2-20
 routines that it calls 2-22

O

OFF\$ macro C-40
 OFID\$ macro C-95
 OFID\$A macro C-95

OFID\$M macro C-96
 OFID\$R macro C-96
 OFID\$U macro C-96
 OFID\$W macro C-96
 OFNB\$ macro C-96
 OFNB\$A macro C-97
 OFNB\$M macro C-97
 OFNB\$R macro C-97
 OFNB\$U macro C-97
 OFNB\$W macro C-97
 OPEN\$ macro C-93
 OPEN\$A macro C-93
 OPEN\$M macro C-93
 OPEN\$R macro C-93
 OPEN\$U macro C-94
 OPEN\$W macro C-94
 Operation of a terminal 1-26
 OPNS\$A macro C-94
 OPNS\$M macro C-94
 OPNS\$R macro C-94
 OPNS\$U macro C-94
 OPNS\$W macro C-95
 OPNT\$D macro C-95
 OPNT\$W macro C-95
 Overlaid task addressing 2-1
 Overlaid tasks 1-14
 Overlays 1-14

P

PAR command for tuning F-6
 \$PARHD pointer 8-5
 Partition control block 1-10, 2-61, 8-5, 8-10, 8-41
 Partition status word bit definitions 8-42
 Partition types 1-3
 Partition wait queue, example of 8-9
 Partitions 1-1
 Partitions, defined 1-2
 in mapped systems 1-3
 in unmapped system 1-3
 memory 1-2
 system-controlled 1-4
 task, user 1-7
 user-controlled 1-4
 PARTY module description 7-63
 Passing arguments B-8
 PCB (see partition control block)
 PCB list for checkpoint files 8-10
 Physical address space 1-15
 Physical addresses 1-3, 2-3
 Physical addresses, range of 1-3

INDEX (Cont.)

- PLAS macro definitions C-101
 - PLSUB module description 7-64
 - Pointers, ACTHD 8-5
 - CRAVL 8-6
 - DEVHD 8-4
 - FRKHD 8-6
 - HEADR 8-7
 - LDRPT 8-6
 - PARHD 8-5
 - RQSCH 8-6
 - TKNPT 8-6
 - TKTCB 8-6
 - TSKHD 8-5
 - system 8-1
 - Power failure restart 1-29
 - POWER module description 7-66
 - Powerfail processing 3-11
 - Pre-allocated I/O packet queue 8-21
 - Print spooler 1-7
 - Print spooler, macro for submitting a file to C-128
 - PRINTS macro C-128
 - Priority 1-11
 - Priority, altering 1-12
 - swapping 1-14
 - tasks with equal 1-12
 - Private devices 1-27
 - Privileged commands 1-25
 - Privileged task execution, logging off during 4-1
 - processor trap during 4-1
 - Privileged task hazards 4-1
 - Privileged task mapping 4-2, 4-3, 4-4, 4-10, 4-15
 - Privileged task, /PR:0 switch 4-2
 - /PR:4 switch 4-2
 - /PR:5 switch 4-4
 - reading or writing to a volume 4-1
 - specifying a task as a 4-2
 - using commons 4-2
 - Privileged tasks 4-1
 - Privileged tasks, accessing the I/O page 4-2
 - in a mapped system 1-3
 - rebuilding, in a mapped system 1-3
 - using SWSTK\$ 4-5
 - writing 4-4
 - writing logical block I/O 4-2
 - Processing within interrupt routines 3-12
 - Processing, interrupt 1-2
 - task 1-2
 - Processor control, assigned by priority 1-11
 - Processor priority B-2
 - Processor trap 1-17
 - Processor trap during privileged task execution 4-1
 - Processor trap, from the system state 3-9
 - Processor traps 1-15, 3-1
 - Processor traps, from task state 3-7
 - Program logical address space extension, macro definitions C-101
 - Program modules B-5
 - Program-local sysmbols B-4
 - Protection, multiuser 1-27
 - region 2-9
 - Pseudo device DCBs 8-4
 - Public devices 1-27
 - PUT\$ macro C-97
 - PUT\$R macro C-98
 - PUT\$\$ macro C-98
- Q**
- QDPB\$\$ macro C-41
 - QDPB\$S macro C-41
 - QIO directive proccessing 6-5
 - QIO function codes for specific device dependent functions C-121
 - QIO functions, ICR C-122
 - ICS C-122
 - QIO processing 6-1
 - QIO qualifier bit definitions, general C-116
 - QIO\$ macro C-43
 - QIO\$C macro C-42
 - QIO\$\$ macro C-42
 - .QIOE macro C-120
 - QIOMAC macro definitions C-115
 - QIOSY\$ macro C-120
 - QIOSYM macro definitions C-115
 - QIOW\$ macro C-44
 - QIOW\$C macro C-43
 - QIOW\$\$ macro C-44
 - Queue I/O speed optimizations for tuning F-2
 - QUEUE module description 7-67
 - Queue, AST 8-14
 - clock 8-16
 - dynamic storage region free block 8-23
 - fork 8-17

INDEX (Cont.)

I/O packet 8-21
 Loader 8-15
 MCR 8-20
 Send/Receive by Reference 8-16
 Send/Receive Data 8-15
 STD, in scheduling 1-12
 TKTN 8-22

R

R50\$ macro C-45
 RREF\$C macro C-107
 RREF\$\$ macro C-107
 RAD%50\$ macro C-98
 RCLOS\$ macro C-110
 RCVD\$ macro C-46
 RCVD\$C macro C-45
 RCVD\$\$ macro C-46
 RCVX\$ macro C-48
 RCVX\$C macro C-47
 RCVX\$\$ macro C-47
 RDAF\$ macro C-49
 RDAF\$C macro C-48
 RDAF\$\$ macro C-49
 RDBBK\$ macro C-107
 READ\$ macro C-99
 Ready-to-run tasks 1-10
 Rebuilding privileged tasks in
 a mapped system 1-3
 Record management services 1-6
 Region definition block 2-10
 Region definition block 8-43
 Region protection 2-9
 Region status word symbols 8-43
 Region, creating a 1-15
 dynamic 2-8
 mapping to 1-15
 static common 2-8
 task 2-8
 Regions 2-8
 Regions, attaching to 2-9
 shared 2-9
 Register standards B-2
 Registers B-2
 Relative file macro C-110
 REQSB module description 7-69
 Resident library descriptor
 offsets 8-50
 Resident tasks 1-10
 Resource management, memory 1-2
 Restart, power failure 1-29
 RETURN macro C-74
 RFA\$ macro C-50
 RFDBT\$ macro C-110
 RFIND\$ macro C-111

RFOFF\$ macro C-111
 RGET\$ macro C-112
 RMDEMO, for tuning F-5, F-8
 RMS (see record management
 services)
 ROPN\$ macro C-112
 ROPN\$R macro C-113
 ROPN\$A macro C-112
 ROPN\$M macro C-112
 ROPN\$U macro C-113
 ROPN\$W macro C-113
 ROPS\$R macro C-113
 ROPS\$A macro C-113
 ROPS\$M macro C-113
 ROPS\$U macro C-114
 ROPS\$W macro C-114
 Round-robin scheduling 1-12
 Round-robin scheduling for
 tuning F-2
 Round-robin scheduling, use of
 STD 1-12
 RPORT\$ macro C-114
 RPRTC\$ macro C-114
 RPUT\$ macro C-114
 \$RQSCH pointer 3-17
 \$RQSCH pointer 8-6
 RQST\$ macro C-51
 RQST\$\$ macro C-51
 RQST\$C macro C-50
 RREF\$ macro C-107
 RSUM\$ macro C-53
 RSUM\$\$ macro C-52
 RSUM\$C macro C-52
 RSX-11M supported devices a-1
 RSX-11M system 1-1
 RUN\$ macro C-54
 RUN\$C macro C-53
 RUN\$\$ macro C-54
 RVP\$ macro C-55

S

Save command 1-2
 SAVNR macro C-74
 SCB, DCB, UCB, LCB relationship
 8-18
 SCB, UCB, DCB relationship 8-24
 SCBLB macro C-74
 Scheduling 1-2, 1-12
 \$SCNDT routine 8-4
 SDAT\$ macro C-57
 SDAT\$\$ macro C-56
 SDAT\$C macro C-56
 Segment cross-reference, BIGFCP
 9-69

INDEX (Cont.)

- MCRMU 9-41
- SYS 9-54
- Send/Receive by Reference queue 8-16
- Send/Receive Data queue 8-15
- Servicing faults E-1
- Set characteristic error codes C-130
- Set characteristics, speed types C-130
 - subfunction codes for C-130
 - terminal types C-130
- SET/GET symbol macro expansions C-129
- SETD macro C-75
- SETF\$ macro C-58
- SETF\$\$ macro C-58
- SETF\$C macro C-57
- SFPA\$ macro C-59
- SFPA\$\$ macro C-59
- SFPA\$C macro C-59
- Shared regions 2-9
- Shuffler 1-4, 1-9, 2-13
- Shuffler diagram 2-51
- Shuffler task 1-14
- Shuffler, algorithm of 2-14
 - basic operation of 1-14
 - first pass of 2-14
 - second pass of 2-14
- Shuffling, for tuning F-2
- Size of dynamic storage region, for tuning F-2
- Slave terminals 1-27
- Snap block 8-46
- Snap control block macro C-124
- SNAP\$ macro C-124
- Snapshot sump macro C-124
- SNPBK\$ macro C-124
- SNPDF\$ macro C-125
- SOB macro C-75
- Source files B-11
- SPCIO\$ macro C-121
- Speed types, for set characteristics C-130
- SPND\$ macro C-61
- SPND\$\$ macro C-60
- SPND\$C macro C-60
- Spooler, print 1-7
- SPRA\$ macro C-62
- SPRA\$\$ macro C-61
- SPRA\$C macro C-61
- SRRA\$\$ macro C-109
- SRRA\$C macro C-109
- SRDA\$ macro C-63
- SRDA\$\$ macro C-63
- SRDA\$C macro C-62
- SREF\$ macro C-107 8
- SREF\$\$ macro C-108
- SREF\$C macro C-108
- SRRA\$ macro C-109
- SSTSR module description 7-77
- Stack structure, fault tracing E-8
- Stack, system 1-16
 - system, queuing interrupts on 3-12
 - upon entry into directive processing 3-10
- Standard codes, driver C-117
- executive C-117
- Standard symbols, using the B-3
- Standards and conventions, coding B-1
- State generation macros C-125
- STATES macro C-126
- Static common region 2-8
- Status control block 6-1, 6-2, 8-5, 8-44
- Status return, directive processing routines 3-11
- STD (see system task directory)
- STD list 8-11
- STD macro C-75
- STFPS macro C-75
- \$STKDP<=0 3-5
- \$STKDP=1 3-4, 3-7
- STST macro C-75
- Subfunction codes for the set characteristics C-130
- Subpartitions 1-4, 1-7
- Success codes, TTY C-119
- Supported devices, card readers
 - A-2
 - communications A-2
 - data acquisition A-2
 - disk devices A-3
 - laborator/industrial control A-3
 - printers A-4
 - processors and options A-1
 - RSX-11M A-1
 - tape devices, magnetic A-4
 - paper A-5
 - terminals A-5
- SVDB\$ macro C-65
- SVDB\$\$ macro C-64
- SVDB\$C macro C-64
- SVTK\$ macro C-66
- SVTK\$\$ macro C-66
- SVTK\$C macro C-65
- Swapping algorithm 2-12
- Swapping for tuning F-2

INDEX (Cont.)

- Swapping interval 2-12
- Swapping priority 1-14
- Swapping priority range 2-12
- Swapping tasks 1-13
- Swapping, disk 2-12
- SWSTK\$ in a mapped system, explained 4-5
- SWSTK\$ macro C-76
- SWSTK\$ routine, described for a mapped system 4-5
- described for an unmapped system 4-8
- SWSTK\$, used by privileged tasks 4-5
- Symbols B-2
- Synchronous events 1-2
- Synchronous traps 1-18
- SYS global cross-reference 9-42
- SYS segment cross-reference 9-54
- SYSCM module description 7-80
- SYSDF module description 7-82
- SYSPAR 1-7
- SYSPAR partition 1-9
- SYSTB 8-4
- SYSTB module description 7-83
- System common data 1-4
- System controlled partitions 1-4
- System generation 1-1
- System generation, phase one 1-1
- phase two 1-1
- System linkages 8-1
- System linkages 8-7
- System maintenance 1-27
- System pointers 8-1
- System stack 1-4, 1-16
- System stack, queuing interrupts on 3-12
- System state, exiting 3-15
- System task directory 1-10
- System task directory list 8-11
- System traps 1-18
- System tuning F-1
- System, RSX-11M 1-1
- System-controlled partition control blocks 8-13
- System-controlled partition linkage 8-5

- T**

- TAL command for tuning F-7
- TAS command for tuning F-5
- Task abort codes 8-47
- Task attributes, changing 1-3
- Task control block 2-65, 8-5, 8-47
- Task execution, scheduling of 1-26
- Task header, field definitions 8-49
- mapped system E-5
- unmapped system E-5
- window block 8-49
- Task mapping 4-10
- Task mapping, 4k nonprivileged user, in mapped system 4-12
- 8k nonprivileged user, in mapped system 4-13
- in mapped system using PLAS 4-14
- nonprivileged user, in mapped system 4-12
- user, in unmapped system 4-11
- Task partitions, user 1-7
- Task priority 1-11
- Task priority, assigning 1-11
- establishing 1-12
- Task processing 1-10, 1-2
- Task region 2-8
- Task scheduling 1-2
- Task states 1-10
- Task status definitions 8-48
- Task swapping 1-13
- Task termination notification message codes 8-51
- Task termination notification queue 8-22
- Task window block 2-7
- Task windows 2-4
- Task, shuffler 1-14
- Tasks, active 1-10
- blocked 1-10
- diagnostic 1-28
- dormant 1-10
- fixed 1-11
- installed 1-10
- overlaid 1-14
- privileged 4-1
- privileged, PR:0 switch 4-2
- PR:4 switch 4-2
- PR:5 switch 4-3
- accessing the I/O page 4-2
- hazards of using 4-1
- mapping of 4-15
- specifying 4-2
- using commons 4-2
- virtual address space 4-3, 4-4
- writing a 4-4
- writing logical block I/O 4-2
- ready-to-run 1-10
- unshufflable 2-13

INDEX (Cont.)

- with equal priorities 1-12
- Task image file label block 8-50
- \$TCHKP routine 2-22
- TDSCH module description 7-89
- Terminal input wait, checkpointing during 2-11
- Terminal operation 1-26
- Terminal types, for SET characteristics C-130
- Terminals, attached 1-26
 - slave 1-27
- Throughput improvement F-1
- Throughput improvement, executive software opinions for F1
 - memory layout for F-1
- \$TKNPT pointer 8-6
- \$TKTCB pointer 8-6
- TKTN 1-7
- TKTN queue 8-22
- TKTN task 1-5
- Tracing faults E-4
- TRAN\$ macro C-127
- Trap processing, for traps in system state 3-9
- Trap vectors 1-4
- Trap vectors 3-1
- Trap, processor 1-17
- Traps, asynchronous 1-18
 - processor 3-1
 - synchronous 1-18
 - system 1-18
- TRUNC\$ macro C-99
- \$TSKHD pointer 8-5
- TSTAT\$ macro C-125
- \$TSTCP diagram 2-42
- \$TSTCP routine 2-23
- TTSYM\$ macro C-129
- TTY error codes C-119
- TTY success codes C-119
- Tuning, Executive software
 - options for F-1
 - hardware considerations F-1
 - helpful hints F-4
 - memory layout F-1
 - system F-1

U

- UCB 8-51
- UCB, DCB, SCB, LCB relationship 8-18
- UCB, SCB, DCB relationship 8-24
- UCB-DCB relationship 8-4
- UMAP\$ macro C-109

- UMAP\$C macro C-109
- UMAP\$\$ macro C-109
- UMDIO\$ macro C-123
- Unit Control Block 6-1, 6-2, 8-4, 8-51
- Unmapped system, example of 1-7
- Unshufflable tasks 2-13
- User controlled partitions 1-4
- User-controlled partition, control blocks 8-12
 - linkage 8-5
- User-mode diagnostic functions C-123

V

- Vector locations 1-18
- Vectors, interrupt 1-16
- Version identifier, displaying the B-13
- Version number standard B-12
- Version number, use of in the program B-14
- Virtual address mapping 2-6
- Virtual address space 1-15, 2-3
- Virtual addresses 1-3, 2-2, 2-3
- VMR installable loader, for tuning F-2
- Volume control block 8-56

W

- WAIT\$ macro C-99
- WDBBK\$ macro C-110
- /WIN option, for tuning F-3
- Window block, task 2-7
- Window definition block 8-56
- Window Definition Block 2-7
- Window status word symbols 8-57
- Windows, memory management
 - directives 2-4
 - task 2-4
- .WORD macro C-124
- WRITE\$ macro C-99
- Writing a privileged task 4-4
- WSIG\$ macro C-67
- WSIG\$\$ macro C-67
- WSIG\$C macro C-67
- WTLO\$ macro C-69
- WTLO\$C macro C-68
- WTLO\$\$ macro C-68
- WTSE\$ macro C-70
- WTSE\$\$ macro C-70

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please cut along this line.

Please indicate the type of reader that you most nearly represent.

- Assembly language programmer
- Higher-level language programmer
- Occasional programmer (experienced)
- User with little programming experience
- Student programmer
- Other (please specify) _____

Name _____ Date _____
Organization _____
Street _____
City _____ State _____ Zip Code _____
or
Country _____

- Fold Here -

- Do Not Tear - Fold Here and Staple -

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML 5-5/E39
Maynard, Massachusetts 01754

