MPX-32 ™

System Tables and Tasks

Revision 3.5

Technical Manual Volume I

April 1990

322-001551-500

ENCORE

Limited Rights

This manual is supplied without representation or warranty of any kind. Encore Computer Corporation therefore assumes no responsibility and shall have no liability of any kind arising from the supply or use of this publication or any material contained herein.

Proprietary Information

The information contained herein is proprietary to Encore Computer Corporation and/or its vendors, and its use, disclosure, or duplication is subject to the restrictions stated in the standard Encore Computer Corporation License terms and conditions or the appropriate third-party sublicense agreement.

Restricted Rights

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at 252.227.7013.

Encore Computer Corporation
6901 West Sunrise Boulevard
Fort Lauderdale, Florida 33313

History

The MPX-32 Release 3.2 Technical Manual, Publication Order Number **322-001550-000**, was printed September, 1983.

Publication Order Number **322-001550-100**, (Revision 1, Release 3.2B) was printed March, 1985.

Publication Order Number **322-001550-101**, (Change 1 to Revision 1, Release 3.2C) was printed December, 1985.

The MPX-32 Release 3.3 Technical Manual Volume I, Publication Order Number **322-001551-200**, was printed December, 1986.

Publication Order Number **322-001551-300**, (Revision 3, Release 3.4) was printed January, 1988.

Publication Order Number **322-001551-400**, (Revision 4, Release 3.4U03) was printed October, 1989.

Publication Order Number **322-001551-500**, (Revision 5, Release 3.5) was printed April, 1990.

This manual contains the following pages:

Title page
Copyright page
iii/iv through xxv/xxvi
1-1 through 1-69/1-70
2-1 through 2-206
3-1 through 3-60
4-1 through 4-27/4-28
5-1 through 5-31/5-32
6-1 through 6-25/6-26
7-1 through 7-17/7-18
8-1 through 8-14
9-1 through 9-29/9-30
10-1 through 10-6
A-1 through A-3/A-4

Contents

Page

Documentation Conventions xxiii

1 System Description

1.1 Naming Conventions 1-1
1.1.1 Communications Region 1-1
1.1.2 Task Service Area (TSA) 1-1
1.1.3 Entry Variables 1-1
1.1.4 System Modules and Interrupt Handlers 1-2
1.1.5 Common System Subroutines 1-2
1.1.6 System Macros 1-3
1.1.7 System Task Load Module Files 1-3
1.1.8 Batch Task Load Module and Executable Image Files 1-3
1.1.9 System Permanent Files 1-3
1.2 Scheduler - IOCS Interface 1-4
1.2.1 I/O Initiation 1-4
1.2.1.1 Wait I/O Postprocessing 1-4
1.2.1.2 No-wait I/O Postprocessing 1-4
1.2.1.3 No-wait I/O Completion Task Interrupt Service 1-4
1.2.1.4 No-wait I/O Restrictions for System Services 1-5
1.3 Scheduler - Task Termination Interface 1-15
1.3.1 Exit Task 1-15
1.3.1.1 Outstanding I/O (Exit) 1-15
1.3.1.2 Messages in Receiver Queue (Exit) 1-15
1.3.1.3 Outstanding Run Requests (Exit) 1-15
1.3.1.4 Run Requests in Receiver Queue (Exit) 1-15
1.3.1.5 Task Abort Receiver (Exit) 1-15
1.3.1.6 Files (Exit) 1-15
1.3.1.7 Resources (Exit) 1-15
1.3.2 Abort Task 1-16
1.3.2.1 Asynchronous Abort 1-16
1.3.2.2 Synchronous Aborts 1-17
1.3.3 Delete Task 1-18
1.3.3.1 Asynchronous Delete 1-18
1.3.3.2 Synchronous Deletes 1-18
1.4 Scheduler-Debug Interface 1-19
1.4.1 Design Goals 1-19

Contents

Page

1.4.2 Debug Entry Points 1-19
1.4.3 Task Interrupt Status 1-20
1.4.4 TSA Stack Pushdown Level Interpretation 1-20
1.4.5 Exit from AIDDB Mode 1-20
1.4.6 Entry Point 1 - Start-up 1-20
1.4.6.1 AIDDB Activated with User Task 1-20
1.4.6.2 AIDDB Activated by Load and Execute SVC 1-21
1.4.7 Entry Point 2 - Reserved 1-21
1.4.8 Entry Point 3 - Trap/Break 1-21
1.4.9 Entry Point 4 - User Break Exit 1-21
1.4.10 Entry Point 5 - Abort 1-21
1.4.10.1 Wait I/O Operation Status on Abort Receiver 1-21
1.4.10.2 No-Wait I/O Operation Status on Abort Receiver 1-22
1.4.10.3 File Status on Abort Receiver Entry 1-22
1.4.10.4 Inhibit of Abort Receiver Entry 1-22
1.4.10.5 Re-use of Abort Receiver 1-22
1.5 Task Interrupts 1-23
1.5.1 Task Interrupt Priorities 1-23
1.5.2 Task Interrupt Receivers 1-23
1.5.3 Task Interrupt Scheduling 1-23
1.5.4 System Service Calls from Task Interrupt Levels 1-23
1.5.5 Task Interrupt Context Storage 1-23
1.5.6 Task Interrupt Level Gating 1-24
1.5.7 User Break Interrupt Receivers 1-24
1.5.8 User End-Action Receivers 1-24
1.5.9 User Message Receivers 1-25
1.5.10 User Run Receivers 1-25
1.5.11 User Abort Receivers 1-26
1.6 Send/Receive Facilities 1-27
1.6.1 Receiving Task Services 1-27
1.6.1.1 Establishing Message and Run Receiver Capability 1-27
1.6.1.2 Execution of Message and Run Receiver Programs 1-27
1.6.1.3 Obtaining the Passed Parameters 1-28
1.6.1.4 Exiting the Receiver Program 1-28
1.6.1.5 Waiting for the Next Request 1-29
1.6.2 Sending Task Services 1-29
1.6.2.1 Sending the Request 1-29
1.6.2.2 Waiting for Request Completion 1-30
1.6.2.3 End-Action Processing 1-30
1.6.2.4 Parameter Send Block (PSB) 1-30

Page

1.6.2.5 Parameter Receive Block (PRB) 1-36
1.6.2.6 Receiver Exit Block (RXB) 1-37
1.6.2.7 Message or Run Request Queue Entry (MRRQ) 1-38
1.6.2.8 Messages and Run Request Services Summary 1-41
1.7 Device Address Specification 1-42
1.8 CPU Scheduling 1-45
1.8.1 Execution Priorities 1-45
1.8.2 Real-Time Priority Levels (1 to 54) 1-45
1.8.3 Time-Distribution Priority Levels (55 to 64) 1-45
1.8.4 Priority Migration 1-45
1.8.4.1 Situational Priority Increments 1-46
1.8.5 Time-Quantum Controls 1-46
1.8.6 State Chain Management 1-46
1.9 FAT/FPT and Blocking Buffer Allocation 1-48
1.9.1 FAT/FPT Area 1-48
1.9.2 Blocking Buffer Area 1-48
1.10 Indirectly Connected Interrupts 1-49
1.10.1 Connect Task to Interrupt Service (M.CONN) 1-49
1.10.2 Disconnect Task from Interrupt Service (M.DISCON) 1-49
1.10.3 Indirectly Connected Task Linkage Table (ITLT) 1-49
1.10.4 Indirectly Connected Task Linkage Block (ITLB) 1-50
1.10.5 Indirectly Connected Interrupt Program (H.ICP) 1-52
1.11 Miscellaneous System Macros 1-53
1.11.1 M.BACK 1-53
1.11.2 M.CALL 1-53
1.11.3 M.CLSE 1-54
1.11.4 M.DFCB 1-54
1.11.5 M.DFCBE 1-55
1.11.6 M.EIR 1-56
1.11.7 M.FCBEXP 1-56
1.11.8 M.FWRD 1-57
1.11.9 M.INIT 1-57
1.11.10 M.INITX 1-58
1.11.11 M.IOFF 1-58
1.11.12 M.IONN 1-58
1.11.13 M.IPUOFF 1-58
1.11.14 M.IPUON 1-58
1.11.15 M.IPURTN 1-59
1.11.16 M.IVC 1-59
1.11.17 M.KILL 1-59

Page

1.11.18 M.MODT 1-59
1.11.19 M.OPEN 1-60
1.11.20 M.RTNA 1-60
1.11.21 M.RTRN 1-61
1.11.22 M.RTRNOS 1-61
1.11.23 M.SHUT 1-61
1.11.24 M.SPAD 1-61
1.11.25 M.SVCP 1-62
1.11.26 M.SVCP2 1-62
1.11.27 M.SVCT 1-63
1.11.28 M.SVCT2 1-63
1.11.29 M.TRAC 1-63
1.11.30 M.TRPINT 1-63
1.11.31 M.TSAD 1-64
1.11.32 M.TYPE 1-64
1.11.33 M.USHUT 1-64
1.11.34 M.XIR 1-64
1.11.35 DCA.DATA 1-65
1.11.36 DCA.INI1 1-65
1.11.37 DCA.INI2 1-66
1.11.38 HMP.INIT 1-66
1.11.39 IB.INIT 1-66
1.12 Extended MPX-32 Macros 1-67
1.12.1 MBR_DBG (Calls to System Debugger) Macro 1-68
1.12.2 MBR_DSCT (DSECT Data Separation) Macro 1-68
1.12.3 MBR_ENT (Extended Code Routine Entry) Macro 1-68
1.12.4 MBR_INIT (Module Initialization) Macro 1-68
1.12.5 MBR_SSCT (System Code Separation) Macro 1-69

2 System Tables and Variables

2.1 Overview 2-1
2.2 Memory Layout 2-4
2.3 Communications Region 2-6
2.4 Allocated Resource Table (ART) 2-34
2.5 Blocking Buffer Control Cells 2-36
2.5.1 Blocking Buffer Head Cells 2-37
2.6 Caller Notification Packet (CNP) 2-38
2.7 Channel Definition Table (CHT) 2-39

Page

2.8 Controller Definition Table (CDT) 2-41
2.9 Device Context Area (DCA) 2-43
2.10 Device Type Table (DTT) 2-45
2.11 Directory Entry Table (M.DN.TEQ) 2-46
2.12 Dispatch Queue Area 2-48
2.13 Dispatch Queue Entry (DQE) 2-48
2.14 Dispatch Queue Address Table (DAT) 2-63
2.15 File Assignment Table (FAT) 2-64
2.16 File Control Block (FCB) 2-67
2.17 File Control Block (8 Word Compatible Mode) 2-75
2.18 File Pointer Table (FPT) 2-82
2.19 I/O Queue (IOQ) Entry 2-83
2.20 M.KEY Entry Format 2-89
2.21 M.PRJCT Format 2-89
2.22 Map Image Descriptor List (MIDL) 2-90
2.22.1 Halfword MIDL Entries 2-90
2.22.2 Fullword MIDL Entries 2-91
2.23 Memory Allocation Pointer List (MPTL) 2-92
2.24 Memory Allocation Table (MATA) 2-93
2.25 Memory Attribute List (MEML) 2-94
2.25.1 Halfword MEML Format 2-94
2.25.2 Fullword MEML Format 2-95
2.26 Memory Pool Management 2-96
2.27 Memory Resident Descriptor Table (MDT) 2-99
2.28 Message or Run Request Queue (MRRQ) 2-99
2.28.1 Remote Messaging Request Queue 2-101
2.29 Module Address Table 2-103
2.30 Mounted Volume Table (MVT) 2-103
2.31 Physical Shared Memory Table (PSM) 2-105
2.32 Resource Create Block (RCB) 2-107
2.33 Resource Inquiry Table (M.RIQ) 2-109
2.34 Resource Logging Block (RLB) 2-110
2.35 Resource Requirement Summary (RRS) Entries 2-111
2.36 Shared Memory Table (SMT) 2-117
2.37 Spooled File Data Structures 2-120
2.37.1 J.SSIN Run Request 2-121
2.37.2 J.TSM Run Request 2-121
2.37.3 J.SOEX Run Request 2-122
2.37.4 J.SOUT Run Request 2-123
2.38 System Master Directory (SMD) 2-124

Contents

Page

2.39 Task Service Area (TSA) 2-126
2.40 Terminal Line Buffer 2-140
2.41 Timer Table 2-141
2.42 Type Control Parameter Block (TCPB) 2-143
2.43 Unit Definition Table (UDT) 2-145
2.44 Volume Assignment Table (VAT) 2-148
2.45 Disk Resident Resource Descriptors (RD) 2-149
2.45.1 Resource Descriptor (M.RDCOM) 2-150
2.45.2 Resource Descriptor Space Definition (M.RDSPD) 2-154
2.45.3 Bad Block Descriptor (M.BB.DEQ) 2-155
2.45.4 Descriptor Allocation Map Descriptor (M.DM.DEQ) 2-155
2.45.5 Descriptors Descriptor (M.DD.DEQ) 2-155
2.45.6 Descriptor Map (DMAP) Deallocation File Descriptor (M.BD.DEQ) 2-156
2.45.7 Directory Descriptor (M.DI.DEQ) 2-157
2.45.8 File Descriptor (M.FI.DEQ) 2-158
2.45.9 Memory Partition Descriptor (M.ME.DEQ) 2-159
2.45.10 Space Allocation Map Descriptor (M.SM.DEQ) 2-160
2.45.11 Space Map (SMAP) Deallocation File Descriptor (M.BS.DEQ) 2-160
2.45.12 Volume Descriptor (M.VO.DEQ) 2-162
2.45.13 Segment Definitions (RD.SEGDF) 2-164
2.45.14 User Area (RD.USER) 2-165
2.46 Disk Resident Structures 2-165
2.46.1 Volume Format 2-166
2.46.2 Load Module Structure 2-167
2.46.3 Load Module Preamble 2-167
2.46.4 Executable Image Structure 2-175
2.46.5 Executable Image Preamble 2-176
2.46.6 Shared Executable Image Structure 2-182
2.46.7 Shared Executable Image Preamble 2-183
2.46.8 Shared Image Descriptors 2-189
2.46.9 COFF Load Module Structure 2-190
2.46.10 COFF Executable Image Preamble 2-191
2.46.11 COFF Shared Image Preamble 2-198
2.47 Internal J.VFMT Structures 2-204
2.47.1 Newboot Macro Offsets (M.BO.EQU) 2-204
2.47.2 Disk Parameter Table Structures 2-205
2.47.2.1 Disk Parameter Table Offsets (M.DPT) 2-205
2.47.2.2 Disk Parameter Table Format (SJ.VFDPT) 2-206

Page

3 System Task Descriptions

3.1 Swap Scheduler Task (J.SWAPR) 3-1
3.1.1 J.SWAPR Processing 3-4
3.1.1.1 Dispatch Processing 3-5
3.1.1.2 Inswap Processing 3-5
3.1.1.3 Shared Memory Request (SISHR) Processing 3-5
3.1.1.4 No Memory Available (NOMEM) Processing 3-5
3.1.1.5 Outswap Processing 3-6
3.1.1.6 I/O Error Handling Processing 3-8
3.1.1.7 Initialization Processing 3-8
3.1.2 J.SWAPR Internal Subroutines 3-9
3.1.3 J.SWAPR Memory Request Functions 3-10
3.1.3.1 Memory Expansion Request 3-11
3.1.3.2 Memory Deallocation Request 3-11
3.1.3.3 Inswap Request (Memory Roll-in) 3-11
3.1.3.4 Change in Task Status Request 3-11
3.1.3.5 Shared Memory Include Request 3-11
3.1.3.6 Exit Conditions 3-12
3.1.4 Managing Swap Space Entries 3-12
3.1.5 Swap Context Area 3-13
3.1.6 Swap Activity Table 3-14
3.1.7 Shadow Memory Outswap Tables 3-14
3.2 Terminal Services Manager Task (J.TSM) 3-16
3.2.1 Functional Description 3-16
3.2.2 Operational Design 3-16
3.2.2.1 Base Level 3-16
3.2.2.2 Message Level 3-17
3.2.2.3 End Action Level 3-18
3.2.2.4 Break Level 3-19
3.2.2.5 Abort Level 3-19
3.2.3 Data Structures 3-19
3.2.3.1 Terminal Context Area (TCA) Table 3-21
3.2.3.2 Nested Context Area (NCA) Table 3-34
3.2.3.3 TSM Procedure Call Buffer (TPCB) 3-35
3.2.4 Intertask Communications 3-35
3.2.5 TSM Command Line Recall and Edit (CLRE) Processing 3-35
3.3 System Mount Task (J.MOUNT) 3-38
3.3.1 Run Request Interface 3-38
3.3.1.1 Formatted Mount Requests 3-38

Page

3.3.1.2 Unformatted Mount Requests 3-39
3.3.1.3 Formatted and Unformatted Dismount Requests....3-39
3.3.2 Mount Messages and Requests 3-39
3.3.3 Checks on Mounted Volumes 3-40
3.3.4 Dismount Messages and Requests 3-42
3.3.5 Error Status Return 3-43
3.4 Multiprocessor Recovery Task (J.UNLOCK) 3-43
3.4.1 Structure 3-43
3.4.2 Entry Conditions 3-43
3.4.3 Exit Conditions 3-44
3.4.4 Multiprocessor Recovery 3-44
3.4.5 Error Status Return 3-45
3.5 System Spooled Output Tasks (J.SOUT and J.SOEX) 3-46
3.5.1 Functional Description 3-46
3.5.2 Operational Design 3-46
3.5.2.1 J.SOEX Message Receiver 3-46
3.5.2.2 Call Back Information 3-48
3.5.2.3 Return Status 3-49
3.5.2.4 Break Receiver 3-50
3.6 Online Help Facility 3-50
3.6.1 Online Help Tasks 3-50
3.6.1.1 HELP Task 3-51
3.6.1.2 J.HLP Task 3-51
3.6.1.3 HELPT Task 3-54
3.6.2 Data Structures 3-54
3.6.2.1 Terminal Context Area (TCA) 3-55
3.6.2.2 Topic Name Table List (TNTL) 3-59
3.6.2.3 Keyword List (KWLI) 3-59
3.6.2.4 Positional Information List (PILI) 3-59
3.6.2.5 Print Screen Audit Trail (PSAT) 3-59
3.6.3 Interfacing J.HLP with Other Tasks 3-60
3.6.3.1 Sending Message Requests Via the Interface 3-60

4 System Generation Task Description

4.1 Task Structure and Functional Organization 4-1
4.2 SYSGEN Components 4-14
4.2.1 DID and DTT Definitions 4-14
4.2.1.1 Device Type Table 4-15

Page

4.2.1.2 Device ID Table 4-16
4.2.2 SYSGEN Scanner 4-17
4.2.2.1 Directive Definition List 4-19
4.3 Table Building 4-20
4.3.1 System Tables 4-20
4.3.1.1 Tables Referenced in SYSGEN 4-20
4.3.2 Internal Tables 4-21
4.3.2.1 SYSGEN Internal Tables 4-22
4.4 Handler and Module Loading and Initialization 4-24
4.5 SYSGEN Load Map Descriptions 4-25
4.6 Special Considerations 4-25
4.6.1 MAPTGT/MAPHOST Routines 4-25
4.6.2 Special Case Activation 4-25
4.6.3 SYSINIT Loading 4-26

5 Batch Task Descriptions

5.1 Cataloger 5-1
5.1.1 Introduction 5-1
5.1.1.1 Exit Conditions 5-1
5.1.2 Processing Regions 5-1
5.1.2.1 X Region 5-2
5.1.2.2 M Region 5-2
5.1.2.3 C Region 5-2
5.1.3 SYMTAB Entries 5-4
5.1.3.1 Linkback Entries 5-4
5.1.3.2 Segment (Module) Entry 5-4
5.1.3.3 Defined Entry Point 5-5
5.1.3.4 Common Entry 5-6
5.1.3.5 Section Entry 5-6
5.1.3.6 Program Name 5-7
5.1.3.7 Control Entry 5-8
5.1.3.8 B Region 5-8
5.1.4 Load Module Structure 5-9
5.1.5 Symbol Table Output Format 5-10
5.1.6 Object Language 5-10
5.1.6.1 Object Module Records 5-11
5.1.7 Object Commands 5-11
5.1.7.1 Absolute Data 5-11

Page

5.1.7.2 Program Origin 5-12
5.1.7.3 Absolute Data Repeat 5-12
5.1.7.4 Transfer Address 5-12
5.1.7.5 Relocatable Data 5-12
5.1.7.6 Program Name 5-13
5.1.7.7 Relocatable Data Repeat 5-13
5.1.7.8 External Definition 5-13
5.1.7.9 Forward Reference 5-14
5.1.7.10 External Reference 5-14
5.1.7.11 Common Definition 5-14
5.1.7.12 Common Reference 5-15
5.1.7.13 Datapool Reference 5-15
5.1.7.14 Escape to Extended Functions 5-15
5.1.7.15 Common Origin 5-16
5.1.7.16 Object Termination 5-16
5.1.8 Extended Object Commands 5-16
5.1.8.1 Section Definition 5-16
5.1.8.2 Section Origin 5-17
5.1.8.3 Section Relocatable Reference 5-17
5.1.8.4 Section Transfer Address 5-17
5.1.8.5 Section External Definition 5-18
5.1.8.6 Section External Reference 5-18
5.1.8.7 Section Forward Reference 5-19
5.1.8.8 Large Common Definition 5-19
5.1.8.9 Large Common Origin 5-19
5.1.8.10 Large Common Reference 5-20
5.1.8.11 Debugger Information 5-20
5.1.8.12 Object Creation Date/Time 5-21
5.1.8.13 Product Identification Information Leader 5-22
5.1.8.14 Multiple Datapool Reference 5-22
5.1.9 Assembler Instructions and Generated Object Commands 5-22
5.2 AIDDB 5-25
5.2.1 The AIDDB Environment 5-25
5.2.2 Entry Points 5-26
5.2.2.1 Entry Point 1 - Start-up 5-26
5.2.2.2 Entry Point 2 - Reserved 5-27
5.2.2.3 Entry Point 3 - Trap/Break Receiver 5-27
5.2.2.4 Entry Point 4 - M.BRKXIT Receiver 5-27

Page

5.2.2.5 Entry Point 5 - Abort Receiver 5-28
5.2.2.6 Entry Point 6 - User Overlay Load Courtesy Call 5-28
5.2.3 H.EXEC Calls 5-29
5.2.4 H.REXS Calls 5-29
5.2.5 File Code Usage 5-30
5.2.6 TSA References 5-31
5.2.7 Communication Region References 5-31
5.2.8 Dispatch Queue Entry (DQE) References 5-31

6 System Trace

6.1 Introduction 6-1
6.2 Trace Type 1 - Task Activation 6-3
6.3 Trace Type 2 - Task Termination 6-4
6.4 Trace Type 3 - Dispatch CPU to Task 6-5
6.5 Trace Type 4 - Task Relinquishes CPU 6-6
6.6 Trace Type 5 - Queue I/O 6-7
6.7 Trace Type 6 - End I/O 6-8
6.8 Trace Type 7 - Interrupt/Trap Handler Entry 6-9
6.9 Trace Type 8 - Interrupt/Trap Handler Exit 6-10
6.10 Trace Type 9 - M.SHUT 6-11
6.11 Trace Type 10 - M.OPEN 6-12
6.12 Trace Type 11 - M.IOFF or BEI 6-13
6.13 Trace Type 12 - M.ION or UEI 6-14
6.14 Trace Type 13 - M.CALL 6-15
6.15 Trace Type 14 - SVC Type 1 6-16
6.16 Trace Type 15 - M.RTRN or M.RTNA 6-17
6.17 Trace Type 16 - Inswap Task 6-18
6.18 Trace Type 17 - Outswap Task 6-19
6.19 Trace Type 18 - Dispatch IPU Task 6-20
6.20 Trace Type 19 - Relinquish IPU Task 6-21
6.21 Trace Type 20 - Reserved 6-22
6.22 Trace Type 21 - Mobile Event Trace 1 6-22
6.23 Trace Type 22 - Mobile Event Trace 2 6-23
6.24 Trace Type 23 - SVC Type 15 6-24
6.25 Trace Type 24 - SVC Type 2 6-25

Contents

Page

7 System Initializers and Builders

7.1 Introduction 7-1
7.2 SDT Loader 7-4
7.2.1 Activating 7-4
7.2.2 Required Input 7-4
7.2.3 Processing 7-4
7.2.4 Results 7-4
7.3 The DBOOT Program Section 7-5
7.3.1 Activating 7-5
7.3.2 Processing 7-5
7.4 The SYSINIT Program Section 7-5
7.4.1 Activating 7-5
7.4.2 Processing 7-5
7.4.2.1 Memory Initialization 7-6
7.4.2.2 System Date and Time 7-7
7.4.2.3 Disk Start-up Final Initialization 7-8
7.4.2.4 Tape Start-up Final Initialization 7-8
7.4.2.5 Master SDT 7-9
7.4.3 Autodisk Subroutine 7-13
7.4.4 Memory Disk 7-15
7.5 Online RESTART 7-16
7.5.1 Activating 7-16
7.5.2 Required Input 7-16
7.5.3 Processing 7-16

8 Internal Processing Unit (IPU)

8.1 Overview 8-1
8.1.1 IPU - Memory Interface 8-1
8.1.2 IPU - CPU Interface 8-1
8.2 Task Scheduling and Execution 8-2
8.2.1 Task Biasing 8-2
8.2.2 Standard CPU/IPU Scheduling 8-3
8.2.3 Optional CPU/IPU Scheduling 8-3
8.2.4 Standard Scheduling of IPU-Biased Tasks 8-3
8.2.5 Optional Scheduling of IPU-Biased Tasks 8-3

Page

8.2.6 Scheduling Unbiased Tasks 8-4
8.2.7 Scheduling CPU Only Tasks 8-4
8.2.8 IPU Task Execution 8-4
8.3 IPU Executive Module Description 8-5
8.3.1 Entry Point 1 - IPU Executive 8-5
8.3.2 Entry Point 2 - Undefined IPU Instruction 8-5
8.3.3 Entry Point 3 - Memory Parity Error 8-5
8.3.4 Entry Point 4 - Nonpresent Memory 8-5
8.3.5 Entry Point 5 - Undefined Instruction 8-5
8.3.6 Entry Point 6 - Privilege Violation 8-5
8.3.7 Entry Point 7 - Map Fault 8-6
8.3.8 Entry Point 8 - SVC Trap Handler 8-6
8.3.9 Entry Point 9 - Arithmetic Exception Trap Handler 8-6
8.3.10 Entry Point 10 - Privilege Mode Halt 8-6
8.3.11 Entry Point 11 - Address Specification 8-7
8.3.12 Entry Point 12 - Cache Fault 8-7
8.3.13 Entry Point 13 - Machine Check 8-8
8.3.14 Entry Point 14 - System Check 8-8
8.3.15 Entry Point 15 - Power Fail Trap 8-9
8.3.16 Subroutine S.IPU1 - Perform Stack Push 8-9
8.3.17 Subroutine S.IPU2 - IPU Initialization 8-9
8.3.18 Subroutine S.IPU3 - Terminate IPU Execution 8-10
8.3.19 Subroutine S.IPU4 - Generate IPU History Buffer 8-10
8.4 IPU Auto Start Trap Processor - H.IPUAS 8-11
8.5 IPU Task Scheduler - H.CPU/H.CPU2 8-11
8.5.1 Entry Point 1 - Field IPU Halt 8-11
8.5.2 Entry Point 2 - Schedule IPU Biased Tasks 8-12
8.5.3 Entry Point 3 - Schedule Unbiased Tasks 8-12
8.5.4 Subroutine S.CPU1 - Link Task to IPU Request State 8-12
8.5.5 Subroutine S.CPU2 - IPU Eligibility Test 8-12
8.6 IPU Accounting Module Descriptions 8-13
8.6.1 Entry Point 1 - Field Interval Timer Interrupt 8-13
8.6.2 Subroutine S.IPUIT1 - Perform Accounting After IPU Trap 8-13
8.6.3 Subroutine S.IPUIT2 - Perform Accounting Before Starting IPU 8-13
8.7 IPU SYSGEN Directives 8-13
8.8 SVCs Executable by an IPU 8-14

Page

9 Converting Modules for Extended MPX-32

9.1 General Information 9-1
9.2 Programming Considerations 9-2
9.3 Macros for Extended MPX-32 9-3
9.3.1 MBR_BL - Branch and Link Macro 9-4
9.3.2 MBR_BU - Branch Unconditional Macro 9-7
9.3.3 MBR_B*xx* - Conditional Branch Macro 9-9
9.3.4 MBR_DBG - Calls to System Debugger Macro 9-11
9.3.5 MBR_DEF - Identify Linkage Symbols Macro 9-11
9.3.6 MBR_DSCT - DSECT Data Separation Macro 9-11
9.3.7 MBR_ENT - Extended Code Routine Entry Macro 9-12
9.3.8 MBR_EXT - Identify External Linkage Symbols Macro 9-12
9.3.9 MBR_INIT - Module Initialization Macro 9-12
9.3.10 MBR_OFFS - Offset Mode Macro 9-13
9.3.11 MBR_REL - Relative Mode Macro 9-13
9.3.12 MBR_SSCT - System Code Separation Macro 9-13
9.3.13 MBR_TRSW - Transfer Register Status Word Macro 9-14
9.4 Macro Assembler and Extended MPX-32 9-15
9.5 Macro Assembler Directives for Extended MPX-32 9-15
9.5.1 OPTR Directive 9-16
9.5.2 OPTS Directive 9-16
9.5.3 OPTT Directive 9-16
9.5.4 SDEF Directive 9-17
9.5.5 SEXT Directive 9-17
9.5.6 SORG Directive 9-19
9.5.7 SSECT Directive 9-19
9.5.8 SSECT FLG_MPX Directive 9-20
9.6 Macro Assembler Options for Extended MPX-32 9-25
9.7 Macro Assembler Errors and Aborts for Extended MPX-32 9-25
9.8 Extended MPX-32 Examples 9-25
9.8.1 Nonextended SVC (H.NONEXT) 9-26
9.8.2 Extended MPX-32 SVC (H.EXTMOD) 9-27
9.8.3 Assemble Assignment for Extended MPX-32 SVC 9-28
9.8.4 JH.32_E File Sample 9-29
9.8.5 JCL for Compressing the Extended MPX-32 SVC 9-29
9.8.6 JCL for SYSGENing an Extended MPX-32 Operating System 9-29

Page

10 RTOM Interval Timer

10.1 General Information 10-1
10.2 SYSGENing RTOM 10-1
10.3 Frequency Rate of the Interval Timer 10-2
10.4 Controlling the Interval Timer 10-3
10.5 Examples 10-4
10.5.1 Example 1: Enabling and Reading the Timer 10-4
10.5.2 Example 2: Reading the Timer 10-5

A System Tables and Variables A-1

List of Figures

Figure		Page
1-1	Scheduler - IOCS Interface - IOCS I/O SVC Processing Overview	1-6
1-2	Scheduler - IOCS Interface - IOCS No-Wait I/O Postprocessing Overview	1-7
1-3	Scheduler - IOCS Interface - IOCS Initiate I/O Procedure	1-8
1-4	Scheduler - IOCS Interface - IOCS Postprocessing Procedure	1-9
1-5	Scheduler - I/O Interrupt Interface Overview	1-10
1-6	Scheduler - I/O Interrupt - Interface, Procedures	1-11
1-7	Scheduler - I/O Interrupt Interface, Re-entrant Subroutines	1-12
1-8	Pre-emptive System Service List Entry Header Format	1-13
1-9	I/O Overview from User Request to I/O Complete	1-14
2-1	I/O Table Linkages	2-87
2-2	Handler Tables and Corresponding Hardware	2-88
2-3	Memory Pool Diagram	2-98
2-4	Spooled File Data Structures	2-120
2-5	TSA Structure	2-127
3-1	System Swap Scheduler	3-1
3-2	Mapping of Candidate Task's TSA (an overview)	3-2
3-3	Mapping of a Candidate Task During Roll-out	3-3
4-1	SYSGEN Output File Format	4-27
5-1	General Table Area	5-3
5-2	Sample Source Listing	5-23
5-3	Sample Object Code Dump	5-24
7-1	Components and Functions in Boot from an SDT	7-1
7-2	Components and Functions in Boot from IOP Console	7-2
7-3	Components and Functions in Boot from Online RESTART	7-3
9-1	Adaptive Sequence Generated By a Branch and Link From a Nonextended to an Extended MPX-32 Module for Extended MPX-32	9-5
9-2	Adaptive Sequence Generated By a Branch and Link From Extended to a Nonextended MPX-32 Module	9-6

List of Figures

Figure		Page
9-3	Adaptive Sequence Generated By an Unconditional Branch From Nonextended to an Extended MPX-32 Module	9-7
9-4	Adaptive Sequence Generated By an Unconditional Branch From Extended to a Nonextended MPX-32 Module	9-8
9-5	Adaptive Sequence Generated By a Conditional Branch From a Nonextended to an Extended MPX-32 Module	9-10
9-6	Adaptive Sequence Generated By a Conditional Branch From an Extended to a Nonextended MPX-32 Module	9-10

List of Tables

Table		Page
1-1	Device Type Mnemonics and Codes	1-44
2-1	Special Control Flags	2-70
2-2	Special Control Flags (8-word FCB)	2-78
3-1	Memory Request Function Codes for J.SWAPR	3-11
4-1	SYSGEN Overlays - Overview of Functions	4-2
4-2	SYSGEN Loading Sequence	4-3
8-1	IPU Trap Structure	8-2
9-1	Conditional Branch Macros for Extended MPX-32	9-9
10-1	RTOM Frequency Rates and Jumper Addresses	10-2

Documentation Conventions

Conventions used in directive syntax, messages, and examples throughout the MPX-32 documentation set are described below.

Messages and Examples

Text shown in `this distinctive font` indicates an actual representation of a system message or an example of actual input and output. For example,

```
VOLUME MOUNT SUCCESSFUL
```

or

```
TSM>!ACTIVATE MYTASK
TSM>
```

Lowercase Italic Letters

In directive syntax, lowercase italic letters identify a generic element that must be replaced with a value. For example,

$NOTE *message*

means replace *message* with the desired message. For example,

```
$NOTE 10/12/89 REV 3
```

In system messages, lowercase italic letters identify a variable element. For example,

`**BREAK** ON:`*taskname*

means a break occurred on the specified task.

Uppercase Letters

In directive syntax, uppercase letters specify the input required to execute that directive. Uppercase bold letters indicate the minimum that must be entered. For example,

$ASSIGN *lfc* **TO** *resource*

means enter $AS or $ASSIGN followed by a logical file code, followed by TO and a resource specification. For example,

```
$AS OUT TO OUTFILE
```

In messages, uppercase letters specify status or information. For example,

```
TERMDEF HAS NOT BEEN INSTALLED
```

Brackets [ ]

An element inside brackets is optional. For example,

$CALL *pathname* [*arg*]

means supplying an argument (*arg*) is optional.

Multiple items listed within brackets means enter one of the options or none at all. The choices are separated by a vertical line. For example,

$SHOW [**CPU**TIME | **JOB**S | **USE**RS]

means specify one of the listed parameters, or none of them to invoke the default.

Items in brackets within encompassing brackets or braces can be specified only when the other item is specified. For example,

BACKSPACE **F**ILE [[**FILES=**] *eofs*]

indicates if *eofs* is supplied as a parameter, FIL= or FILES= can precede the value specified.

Commas within brackets are required only if the bracketed element is specified. For example,

LIST [*taskname*][,[*ownername*] [,*pseudonym*]]

indicates that the first comma is required only if *ownername* and/or *pseudonym* is specified. The second comma is required only if *pseudonym* is specified.

Braces { }

Elements listed inside braces specify a required choice. Choices are separated by a vertical line. Enter one of the arguments from the specified group. For example,

[**BLO**CKED=**{Y | N}**]

means Y or N must be supplied when specifying the BLOCKED option.

Horizontal Ellipsis ...

The horizontal ellipsis indicates the previous element can be repeated. For example,

$DEFM [*par*] [,*par*] ...

means one or more parameters (*par*) separated by commas can be entered.

Vertical Ellipsis

The vertical ellipsis indicates directives, parameters, or instructions have been omitted. For example,

```
$DEFM SI,ASSEMBLE,NEW,OP
    .
    .
    .
$IFA %OP ASSM
```

means one or more directives have been omitted between the $DEFM and $IFA directives.

Parentheses ( )

In directive syntax, parentheses must be entered as shown. For example,

(*value*)

means enter the proper value enclosed in parentheses; for example, (234).

Special Key Designations

The following are used throughout the documentation to designate special keys:

<ctrl>	control key
<ret> or <CR>	carriage return/enter key
<tab>	tab key
<break>	break key
<bck>	backspace key
<del>	delete key

When the <ctrl> key designation is used with another key, press and hold the control key, then press the other key. For example,

<ctrl>C

means press and hold the control key, then press the C.

Change Bars

Change bars are vertical lines (|) appearing in the right-hand margin of the page for your convenience in identifying the changes made in MPX-32 Revision 3.5.

When an entire chapter has been changed or added, change bars appear at the chapter title only. When text within figures has changed, change bars appear only at the top and bottom of the figure box.

1 System Description

1.1 Naming Conventions

MPX-32 software and documentation use the following naming conventions for system components.

1.1.1 Communications Region

Names of variables within the MPX-32 communications region are prefixed by the characters "C.". The general form is C.*x* where *x* is a string of one to six characters.

1.1.2 Task Service Area (TSA)

Names of variables within the TSA associated with each task are prefixed by the characters "T.". The general form is T.*x* where *x* is a string of one to six characters.

1.1.3 Entry Variables

Names of variables within table and file entries consist of characters which identify the table or file and the variable. The general form is *x.y* where *x* consists of two to four characters which identify the table and *y* consists of three to six characters which identify the variable. Table or file name prefixes (*x*) are as follows:

ART	Allocated Resource Table
CDT	Controller Definition Table
CHT	IOP Channel Definition Table
DAT	Dispatch Queue Address Table
DCA	Device Context Area
DFT	Disk File Assignment Table
DQE	Dispatch Queue Entry Table
DTT	Device Type Table
FCB	File Control Block
FPT	File Pointer Table
ICB	Interrupt Control Block
IOQ	I/O Queue Entry
JOB	Job Table
MEM	Memory Allocation Table
MEML	Memory Attribute List
MIDL	Map Image Descriptor List
MQ	Message or Run Request Queue Entry
MVT	Mounted Volume Table
PRB	Parameter Receive Block
PSB	Parameter Send Block
RCB	Resource Create Block
RD	Resource Descriptor
RLB	Resource Logging Block
RRS	Resource Requirement Summary Entry
RXB	Receiver Exit Block
SMD	System Master Directory Entry
SMT	Shared Memory Table
TCA	Terminal Context Area
TCP	Type Control Parameter Block
UDT	Unit Definition Table
VAT	Volume Assignment Table

1.1.4 System Modules and Interrupt Handlers

Names of system modules and interrupt handlers are prefixed by the characters "H.". The general form is H.*x* where *x* is a string of one to six characters. Entry points in system modules are identified by the module name, followed by the entry point's numeric identifier. Entry point names are of the general form H.*x,n*, where *n* is the numeric entry point identifier.

1.1.5 Common System Subroutines

Common system subroutines are subroutines contained within modules intended for use by other modules. Their names are prefixed by the characters "S.". The general form is S.*xn*, where *x* is the one to four character module identifier and *n* is the subroutine numeric identifier. For example, S.EXEC1 is the first subroutine in the H.EXEC module.

1.1.6 System Macros

Names of nonbase mode system macros are prefixed by the characters "M.". Names of base mode system macros are prefixed by "M_". The general form is M.*x* or M_*x*, where *x* is a string of one to six characters for nonbase mode or one to fourteen characters for base mode.

1.1.7 System Task Load Module Files

Names of system task load module files are prefixed by the characters "J.". The general form is J.*x*, where *x* is a string of one to six characters.

1.1.8 Batch Task Load Module and Executable Image Files

Names of system batch task load module files are identical to the names of the tasks contained on the files.

1.1.9 System Permanent Files

Names of system permanent files not containing load modules are prefixed by the characters "M.". The general form is M.*x*, where *x* is a string of one to six characters. M.ERR, M.CNTRL, and M.KEY are examples of system permanent files.

1.2 Scheduler - IOCS Interface

1.2.1 I/O Initiation

A task issues an SVC to enter IOCS. I/O services for pretransfer processing are then executed at the software priority level of the requesting task. Once the I/O request is initiated (or queued for initiation), an H.EXEC entry point is called to report the event to the CPU and swapping scheduler:

Entry Point	Event
H.EXEC,1	interactive input starting
H.EXEC,2	terminal output starting
H.EXEC,3	wait I/O starting
H.EXEC,4	no-wait I/O starting

1.2.1.1 Wait I/O Postprocessing

A return is made to IOCS from H.EXEC,1, 2, or 3 only when the I/O request completes. Post transfer processing may then occur at the software priority level of the requesting task.

1.2.1.2 No-Wait I/O Postprocessing

A return from H.EXEC,4 is made immediately after recording the no-wait I/O event. Since IOCS also makes an immediate return to the user task, no-wait I/O post transfer processing occurs as a task interrupt service.

1.2.1.3 No-Wait I/O Completion Task Interrupt Service

When the I/O handler interrupt service routine fields a completion interrupt for a no-wait I/O request, it calls the executive subroutine S.EXEC4 to report the event. The I/O queue entry associated with the call is then linked to the task interrupt list in the DQE of the task that made the I/O request. When the scheduler attempts to dispatch control to the task, it finds that a task interrupt is outstanding. Task interrupts are inhibited during execution of any system service for a task. No task interrupt is honored while a higher priority task interrupt is active. When the task interrupt is honored, control is transferred to the IOCS routine specified in the pre-emptive system service header of the I/O queue entry. Post transfer processing then occurs at the software priority level of the requesting task. When postprocessing of the no-wait I/O request is complete, the task interrupt service is exited by a call to S.EXEC6 or H.EXEC,12.

1.2.1.4 No-Wait I/O Restrictions for System Services

Post transfer processing for a no-wait I/O request is processed as a task interrupt. Task interrupts are not honored while the task is executing in a system service (PC .LE. TSA address). An exception is made for a task that is in a wait for any no-wait I/O completion state. A task interrupt generated by the completion of no-wait I/O is honored if the task is in the wait for any no-wait I/O completion state. A system service that wants no-wait I/O can issue a series of no-wait calls followed by a wait-for-any call. Be careful that all outstanding calls are completed appropriately.

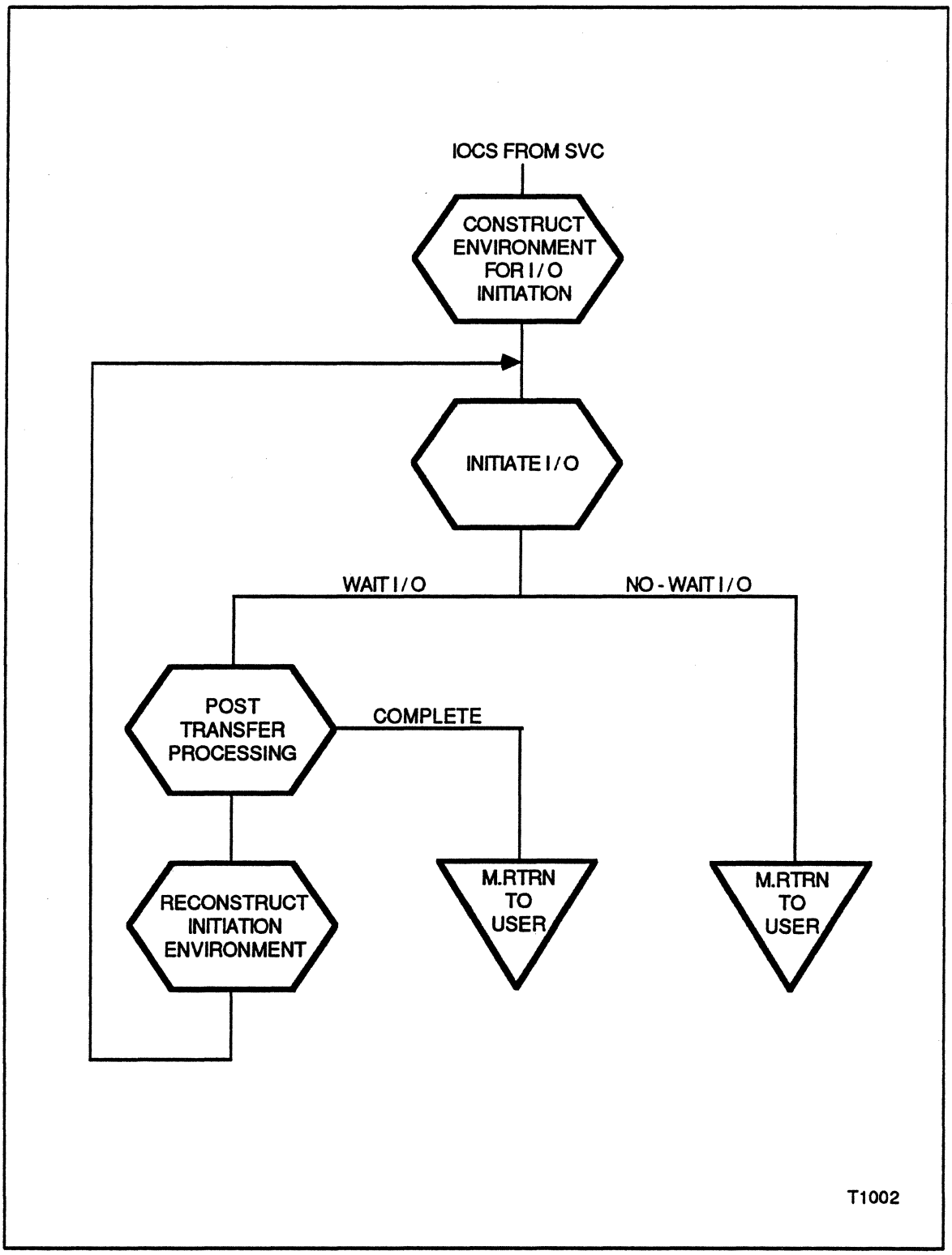


Figure 1-1
Scheduler - IOCS Interface - IOCS I/O SVC Processing Overview

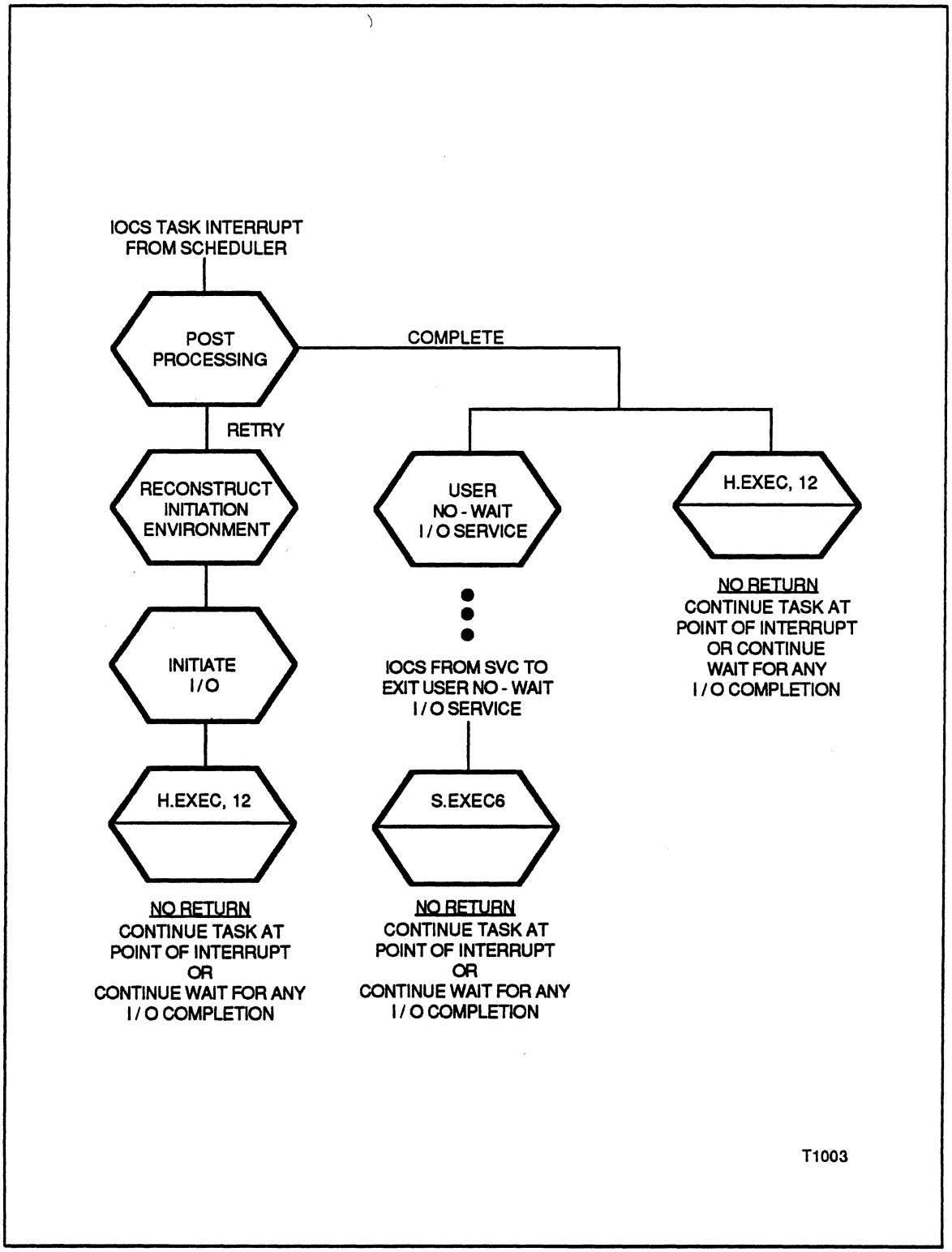


Figure 1-2
Scheduler - IOCS Interface - IOCS No-Wait I/O Postprocessing Overview

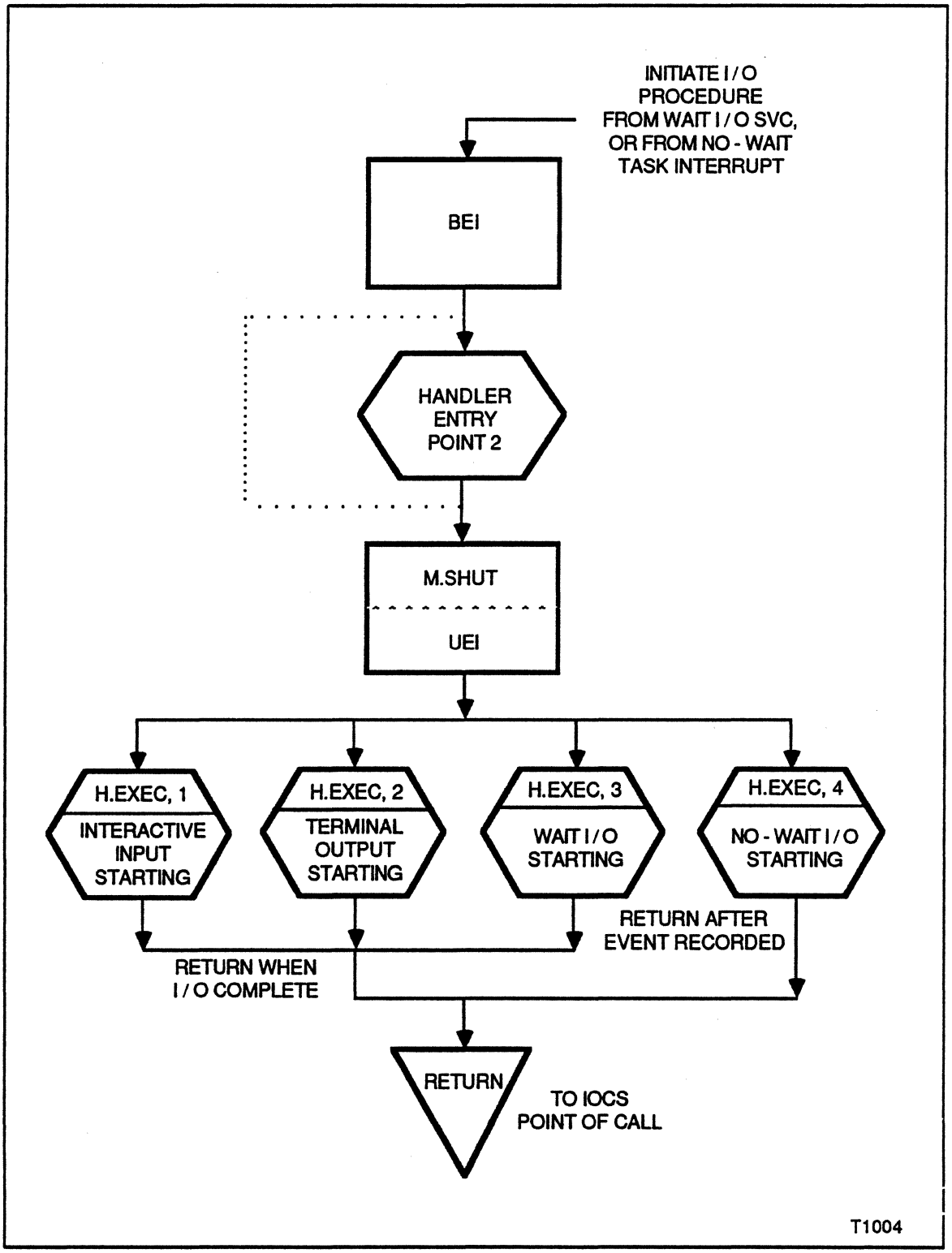


Figure 1-3
Scheduler - IOCS Interface - IOCS Initiate I/O Procedure

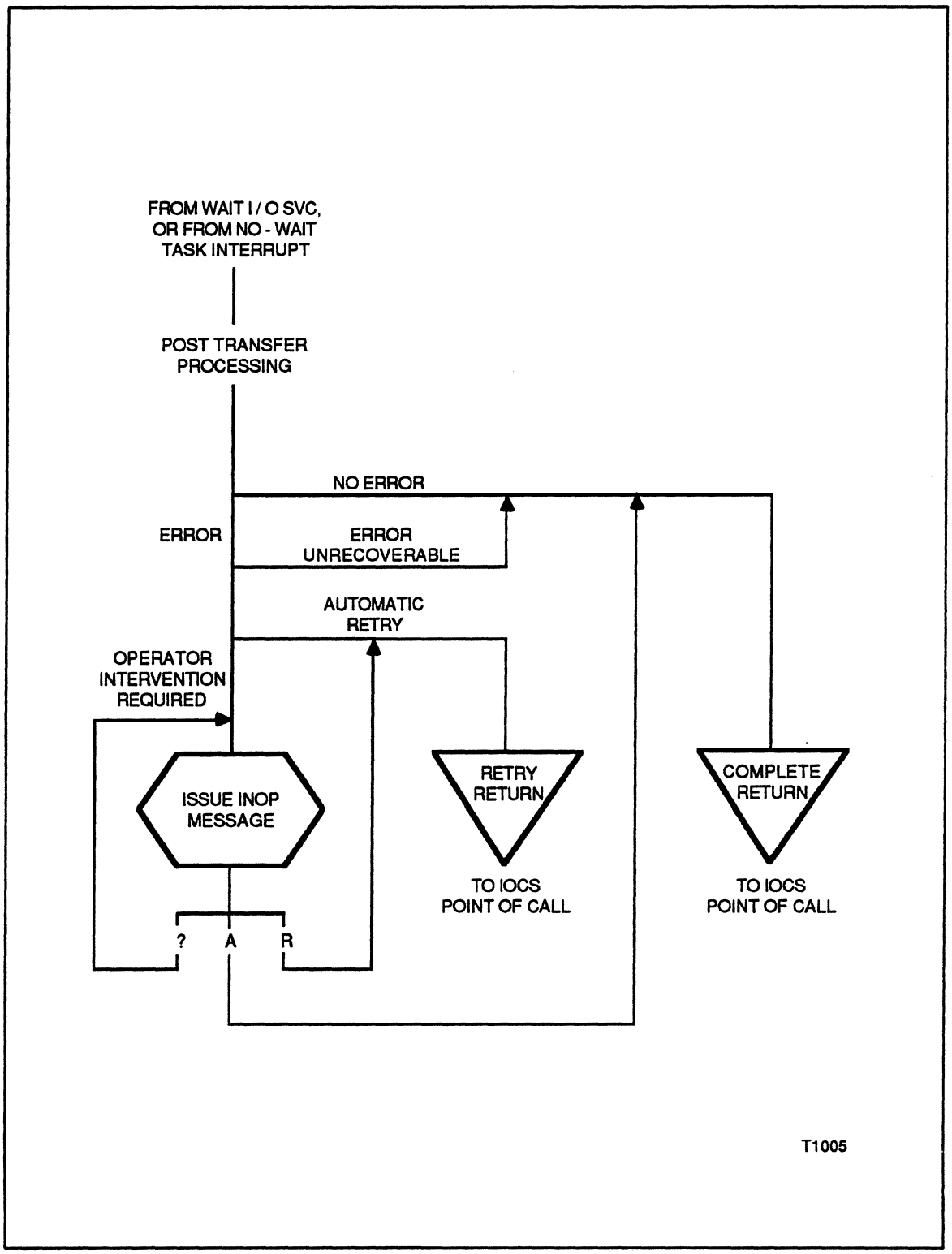


Figure 1-4
Scheduler - IOCS Interface - IOCS Postprocessing Procedure

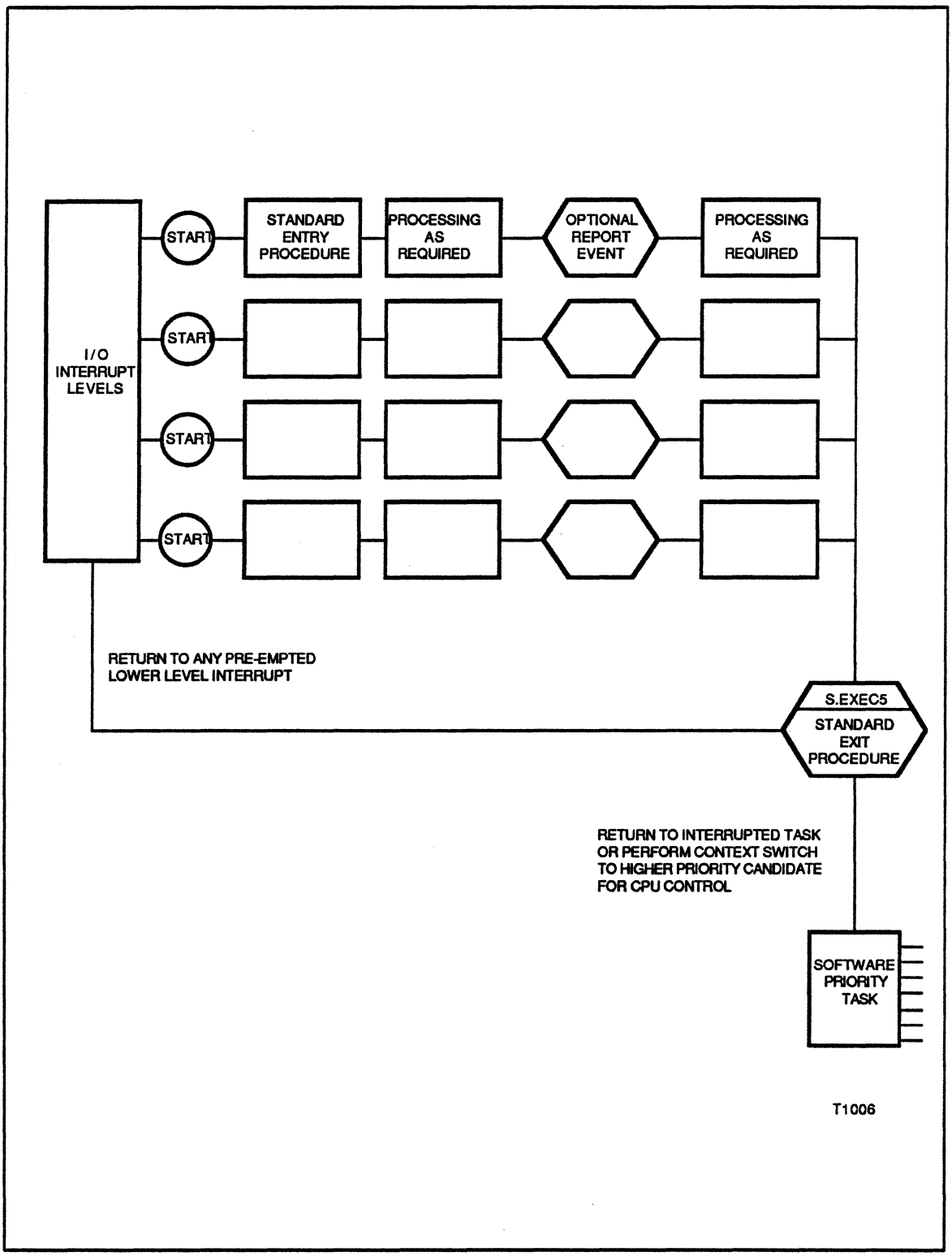


Figure 1-5
Scheduler - I/O Interrupt Interface Overview

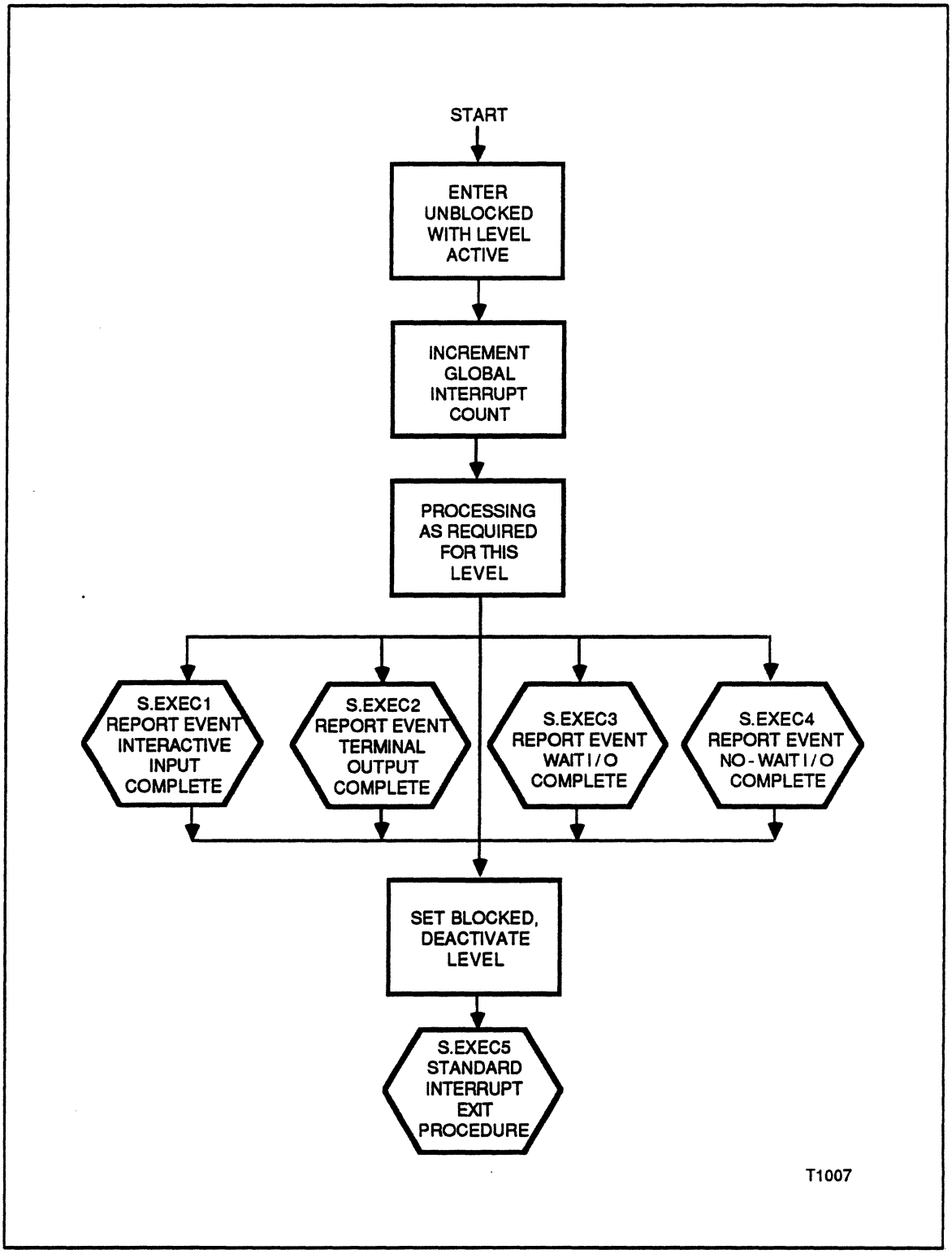


Figure 1-6
Scheduler - I/O Interrupt - Interface, Procedures

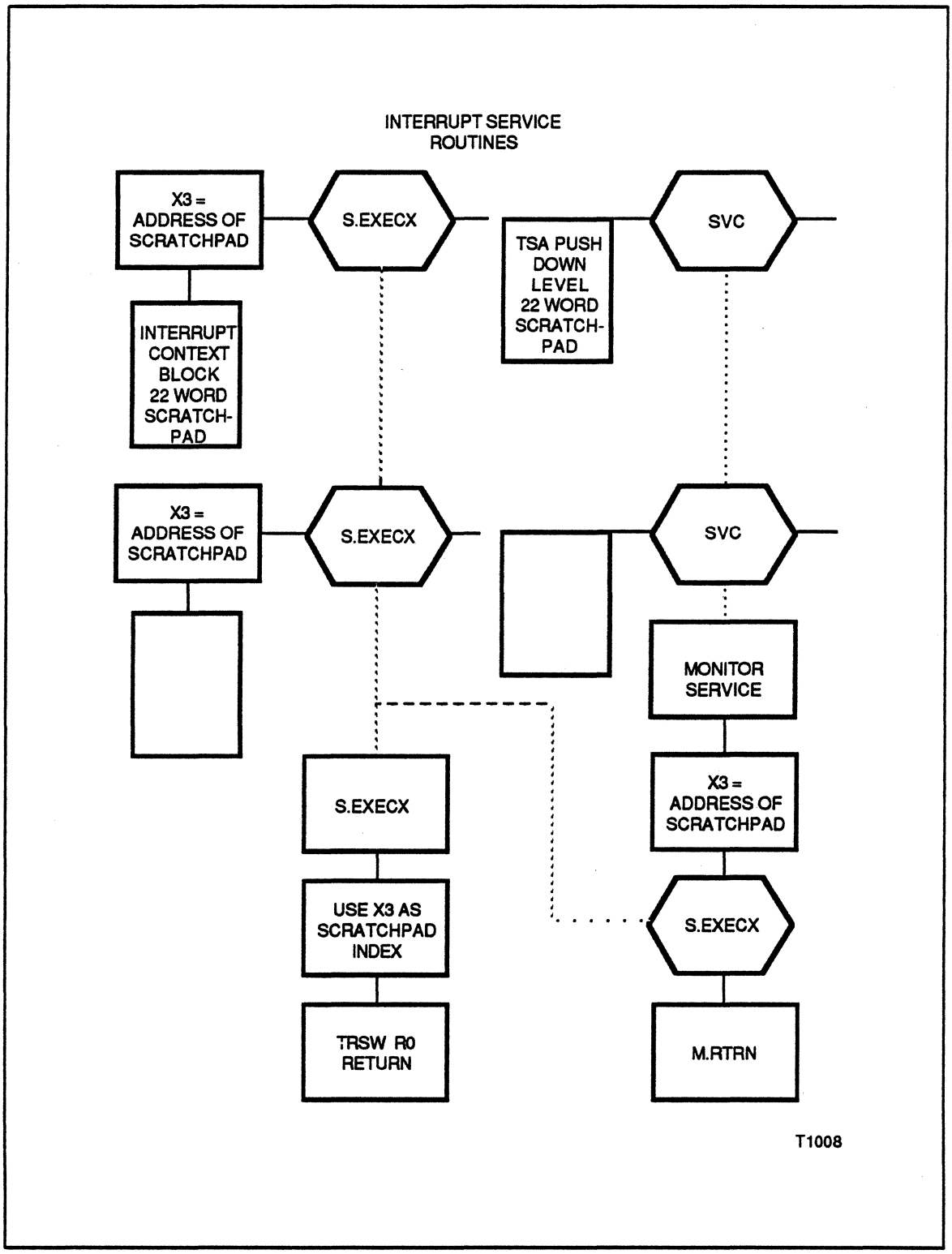


Figure 1-7
Scheduler - I/O Interrupt Interface, Re-entrant Subroutines

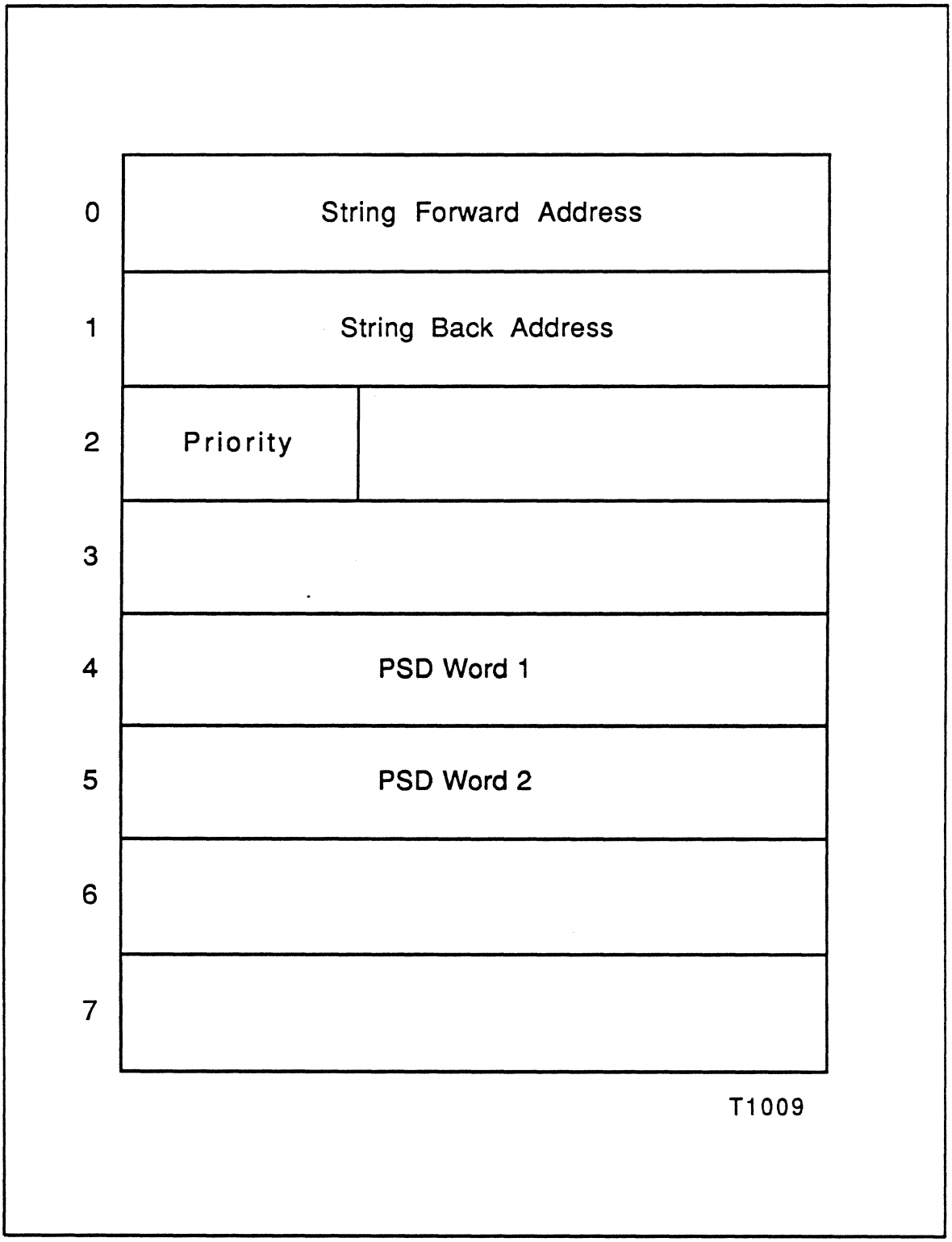


Figure 1-8
Pre-emptive System Service List Entry Header Format

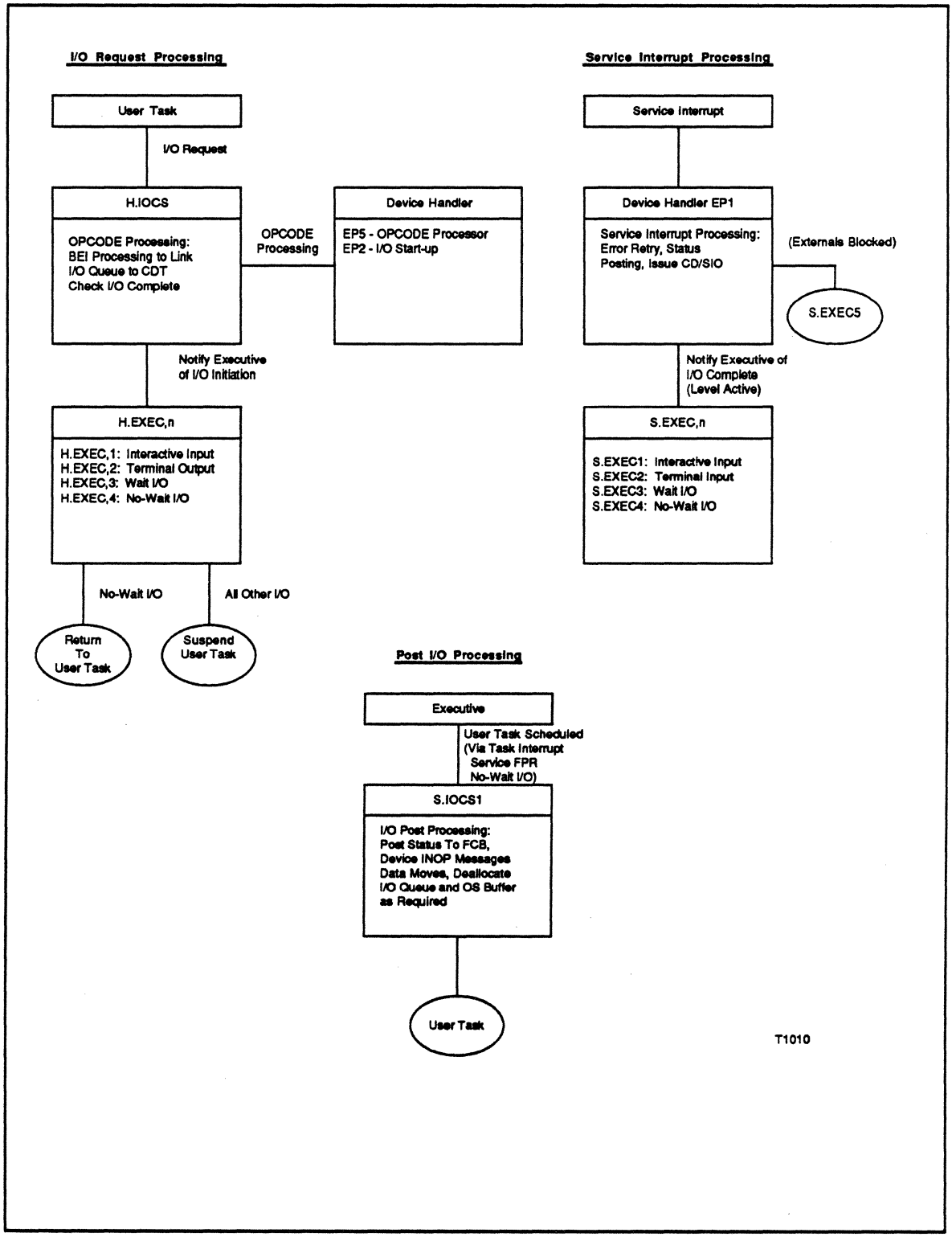


Figure 1-9
I/O Overview from User Request to I/O Complete

1.3 Scheduler - Task Termination Interface

Three types of task termination are provided in the MPX-32 system: exit, abort, and delete task execution.

1.3.1 Exit Task

The exit task service is called by a task that needs to terminate its execution normally.

1.3.1.1 Outstanding I/O (Exit)

If an exiting task has outstanding I/O, further exit processing is deferred until all outstanding I/O is complete. Any user end-action routines associated with no-wait I/O which completes while a task is exiting result in a task abort.

1.3.1.2 Messages in Receiver Queue (Exit)

All outstanding messages sent to an exiting task are unlinked from the message receiver queue and treated as complete with abnormal status.

1.3.1.3 Outstanding Run Requests (Exit)

A task attempting to exit with outstanding no-wait run requests (with call back) for other tasks is aborted.

1.3.1.4 Run Requests in Receiver Queue (Exit)

If an exiting task has requests in its run receiver queue, the current run request is terminated and the appropriate status is posted in the run request parameter block. If any additional run requests are queued, a new copy of the task is activated.

1.3.1.5 Task Abort Receiver (Exit)

A task abort receiver is not processed on task exit.

1.3.1.6 Files (Exit)

All open files associated with a task are automatically closed during task exit processing.

1.3.1.7 Resources (Exit)

All resources associated with a task are automatically deallocated during task exit processing.

1.3.2 Abort Task

The abort task service is called by a task that needs to terminate its execution abnormally. It is also initiated by the system when a task encounters a system trap condition, such as undefined instruction, privilege violation, nonpresent memory, or by a system service because of a parameter validation error. This service is asynchronously initiated by another task or by operator communications. If the OWNERNAME restriction is set in T.ACCESS, only a task of the same owner name can initiate the abort.

1.3.2.1 Asynchronous Abort

When a task needs to abort another task it calls the asynchronous abort service. If the OWNERNAME restriction is set in T.ACCESS, only a task of the same owner name can initiate the abort. The task to be aborted is in a ready-to-run state or one of the following wait states:

1. Waiting for execution signal:
   - timed suspend
   - message receive
   - run request receive
   - interrupt receive
2. Waiting for resource:
   - device
   - disk space
   - memory
   - memory pool
3. Waiting for operation complete:
   - interactive input
   - low speed output
   - any no-wait I/O
   - wait I/O
   - any no-wait message
   - wait message
   - any no-wait run request
   - wait run request

If the specified task to be aborted is waiting for an execution signal, an abort request bit is set in the DQE. The DQE is unlinked from its current state queue and linked to the ready-to-run list at its current priority. When it is selected for execution by the CPU scheduler, the abort request processing then proceeds for the aborting task.

If the specified task is waiting for a resource or operation complete, the abort requested bit is set in its DQE. The task remains linked to its current list, and abort processing does not proceed until outstanding operations are complete and the task is ready to run.

1.3.2.2 Synchronous Aborts

When the currently executing task encounters an abort condition, the abort bit is set in the DQE. The CPU scheduler then processes the abort request. The following is an outline of synchronous abort processing.

Outstanding I/O — If the aborting task has outstanding I/O, further abort processing is deferred until all outstanding I/O is complete. End-action routine execution is inhibited, and task abort status is reflected in the FCB.

Messages in Receiver Queue — All outstanding messages sent to an aborting task are unlinked from the message receiver queue and treated as complete with abnormal status.

Outstanding Run Requests — If the aborting task has outstanding run requests (with call back) for other tasks, further abort processing is deferred until completion of all such requests. End-action routine execution is inhibited, and task abort status is reflected in the run request block.

Run Requests in Receiver Queue — If the aborting task has requests in its run receiver queue, the current run request is terminated and the appropriate status is posted in the run request parameter block. If any additional run requests are queued, a new copy of the task is activated.

Abort Receiver — If the aborting task has an abort receiver, control is transferred to it. All outstanding operation or resource waits have been completed, and all no-wait I/O or no-wait run requests (with call back) have been completed when the abort receiver is entered. End-action routines associated with no-wait operations that completed while the abort request was outstanding have not been executed. Status bits reflecting this are posted in the appropriate FCBs and PSBs. Any files open when the abort request was received remain open on an abort receiver entry. Any resources allocated when the abort request was received remain allocated when the abort receiver is executed.

Open Files — If the aborting task has no intercepting abort receiver, all files open when the abort request was encountered are automatically closed.

Resources — If an aborting task has no intercepting abort receiver, all previously allocated resources are deallocated and the task is no longer active in the system.

1.3.3 Delete Task

The delete task service is called by the system for a task that encounters a second abort condition during processing of an initial abort request. This service is asynchronously initiated by another task or by operator communications. If the OWNERNAME restriction is set in T.ACCESS, only a task of the same owner name can initiate the task delete request.

1.3.3.1 Asynchronous Delete

When a task needs to delete another task of the same owner name, it calls the asynchronous delete service. The task to be deleted can be in a ready-to-run state or a wait state, such as wait for execution signal, wait for resource, or wait for operation complete. In any case, the delete task bit is set in the DQE, and the task is linked to the ready-to-run list or to the memory request queue for inswap. An exception is made for a task already in the memory request queue. In this case, the task is not linked into the ready-to-run queue until memory scheduler processing is complete.

1.3.3.2 Synchronous Deletes

When the currently executing task encounters a delete condition, the delete task bit is set in the DQE. The CPU scheduler then processes the delete request. The following is an outline of synchronous delete processing.

Outstanding I/O — Delete processing causes all outstanding I/O to be terminated (killed).

Messages in Receiver Queue — All outstanding messages sent to a task being deleted are unlinked from the message receiver queue and treated as complete with abnormal status.

Outstanding Run Requests — If the task being deleted has outstanding run requests for other tasks, any call back is ignored.

Run Requests in Receiver Queue — If the task being deleted has requests in its run receiver queue, the current run request is terminated and the appropriate status is posted in the run request parameter block. If any additional run requests are queued, a new copy of the task is activated.

Abort Receiver — Abort receivers are not processed for tasks being deleted.

Open Files — Files associated with a task being deleted are not automatically closed.

Resources — All resources associated with a task being deleted are deallocated, and the task is no longer active in the system.

1.4 Scheduler-Debug Interface

1.4.1 Design Goals

The structure of the scheduler-debug interface is dictated by the following major design goals:

- AIDDB can be associated with a task at task activation time, or subsequently associated with a terminal task when the break key is pressed. AIDDB can also be associated with a task dynamically through a system service call.
- When a task that has AIDDB associated with it is executing, two methods of entering AIDDB are provided: the executing task encounters a previously set AIDDB trap instruction, or the terminal operator presses the break key.
- Entering AIDDB mode by a trap or break is allowed during execution of software (task) interrupt receivers like message, end action, and break.
- AIDDB intercepts any task aborts, automatically enters the AIDDB mode, and informs the operator of the abort reason.
- System entry into the abort receiver is soft (outstanding I/O requests are completed, and files remain open and allocated). This allows the operator to correct and proceed from the environment that caused the abort condition.

1.4.2 Debug Entry Points

AIDDB has five entry points. These entry points are reflected by the halfword address table (HAT) at the beginning of the AIDDB program. When AIDDB is loaded, the address of the AIDDB HAT is stored in T.DBHAT in the TSA. The first word of the HAT contains the number of AIDDB entry points. Subsequent words contain the address of the individual AIDDB entry points. The entry points provided are:

Entry Point	Description
1	debug start-up
2	reserved
3	trap/break
4	user break exit
5	abort

1.4.3 Task Interrupt Status

To determine the status of task interrupts, AIDDB examines a byte (DQE.ATI) in the dispatch queue entry. When AIDDB is entered, DQE.ATI contains the definition of all active task interrupts.

Bit	Meaning
0	reserved
1	active end action interrupt 1 (DQE.AEA1)
2	active debug mode interrupt (DQE.ADM)
3	active user break interrupt (DQE.AUB)
4	active end action interrupt 2 (DQE.AEA)
5	active message interrupt (DQE.AMI)
6-7	reserved

1.4.4 TSA Stack Pushdown Level Interpretation

For all AIDDB entry points except restart, the context associated with the most recently interrupted task level is contained in T.CONTXT. Nested levels of task interrupt are contained in the TSA stack. Unless one of the task interrupt levels (other than DQE.ADM) is active, the TSA stack is empty on entry to AIDDB. If task interrupts are active, the context storage in the TSA is in reverse order of priority. For example, highest priority is the most recent. In the active task interrupt bit assignments, bit zero is the lowest priority.

1.4.5 Exit from AIDDB Mode

When AIDDB is executing (regardless of the entry point) the task is in the AIDDB mode. The AIDDB mode is exited by calling one of the following H.EXEC entry points:

Entry Point	Description
H.EXEC,22	go to specified task context
H.EXEC,23	run user break receiver

1.4.6 Entry Point 1 - Start-up

This entry point is entered in one of two methods: AIDDB is activated with the user task, or the user task issues an SVC call to load and execute AIDDB.

1.4.6.1 AIDDB Activated with User Task

The program activation service that runs for the task being activated detects that AIDDB is to be activated with the task. After the task is loaded, a special service is called to load AIDDB. Once AIDDB is loaded, the service stores the normal start-up registers and PSD in an AIDDB context block in the TSA (T.CONTXT). The service then adjusts the stack in the TSA to enter AIDDB at the AIDDB start-up entry point. When AIDDB is entered the stack is empty, AIDDB mode is set, and T.CONTXT contains the user task start-up registers and PSD.

1.4.6.2 AIDDB Activated by Load and Execute SVC

When the user task issues a load and execute AIDDB SVC, the system service loads AIDDB, stores the user's registers and PSD in T.CONTXT, sets AIDDB mode, and adjusts the TSA stack for entry at AIDDB's start-up entry point.

1.4.7 Entry Point 2 - Reserved

1.4.8 Entry Point 3 - Trap/Break

This entry point is entered when a hardware break or M.INT is received by the user task being debugged. It is also entered when a trap SVC is executed. On entry, T.CONTXT contains the interrupted context, and the AIDDB mode task interrupt flag is set.

1.4.9 Entry Point 4 - User Break Exit

This entry point is executed when the user task being debugged executes a break exit. A user task being debugged can only execute its break receiver by giving a break command to AIDDB. AIDDB in turn calls H.EXEC,23. Normal break receiver entry is reserved for AIDDB use when AIDDB is associated with a task. When AIDDB's user break exit entry point is entered, T.CONTXT contains the most recent level of pushdown from the TSA stack. The number of pushdowns in the TSA stack varies based on the number of active task interrupts like message and end action.

1.4.10 Entry Point 5 - Abort

This entry point is executed when an abort request is received for the user task and no user abort receiver has been specified. When the abort is received, the user task context is in T.CONTXT of the TSA. If a task interrupt like message or break receiver was in effect when the abort request was received, the TSA stack is at the associated level of pushdown. Otherwise, the TSA stack is empty.

1.4.10.1 Wait I/O Operation Status on Abort Receiver

When the abort receiver is entered, any wait I/O operation is completed first. If an abort request is received for a task with wait I/O outstanding, abort processing is deferred until the wait I/O is complete. A service is provided by operator communications to terminate (kill) outstanding I/O requests associated with the specified task. When an I/O request is terminated, appropriate status is posted in the FCB.

1.4.10.2 No-wait I/O Operation Status on Abort Receiver

When the abort receiver is entered, all no-wait I/O operations is complete. If an abort request is received for a task with no-wait I/O outstanding, abort processing is deferred until all no-wait I/O requests are complete. User end-action routine processing is inhibited for no-wait I/O completions when the task is aborting. Task abort status is posted in the FCB.

1.4.10.3 File Status on Abort Receiver Entry

All user files remain open on entry to the task abort receiver.

1.4.10.4 Inhibit of Abort Receiver Entry

If an abort condition is detected during abort processing for a previously detected abort condition, all outstanding I/O is terminated, no status is posted, abort receiver entry is inhibited, resources are deallocated, and the task is removed from the system.

1.4.10.5 Re-use of Abort Receiver

Privileged tasks can re-establish an abort receiver from within an abort receiver, allowing privileged tasks to enter their abort receiver more than once. Unprivileged tasks are aborted if an attempt is made to re-establish this receiver.

1.5 Task Interrupts

In addition to the 64 levels of execution priority available for task execution, the MPX-32 scheduler provides a software interrupt facility within the individual task environment.

1.5.1 Task Interrupt Priorities

Individual tasks operating in the MPX-32 environment can be organized to take advantage of the task unique software interrupt levels. Each task in the MPX-32 system has six levels of software interrupt:

Level Priority	Description
0	reserved for operating system use
1	AIDDB
2	break
3	end action
4	message
5	normal execution (run request)

1.5.2 Task Interrupt Receivers

An individual task is allowed to issue system service calls to establish interrupt receiver addresses for both break and message interrupts. The AIDDB interrupt level is reserved for system use by tasks running in AIDDB mode. The end-action interrupt level is used for system postprocessing of no-wait I/O, message, or run requests. It also executes user-task specified end-action routines. The normal execution level is used for run request processing and general base level task execution.

1.5.3 Task Interrupt Scheduling

Task interrupt processing is gated by the MPX-32 scheduler during system service processing. If a task interrupt request occurs while the task is executing in a system service, the scheduler defers the interrupt until a return is made to the user task execution area.

1.5.4 System Service Calls from Task Interrupt Levels

A task can utilize the complete set of system services from any task interrupt level. It is prohibited, however, from making a wait for any no-wait completion call (M.ANYW) from an end-action routine. It is illegal to issue an I/O request on any FCB that is busy or has postprocessing outstanding.

1.5.5 Task Interrupt Context Storage

When a task interrupt occurs, the scheduler automatically stores the interrupted context into the TSA pushdown stack. This context is automatically restored when the task exits from the active interrupt level.

1.5.6 Task Interrupt Level Gating

When a task interrupt occurs, the level is marked active. Additional interrupt requests for that level are queued until the level active status is reset by the appropriate level exit system service call. When the level active status is reset, any queued request is processed.

In addition, the following services can inhibit higher priority task interrupts:

M.ASYNCH	resets the asynchronous task interrupt mode back to the default environment
M.DSMI	disables the task interrupts for messages sent to the calling task
M.DSUB	deactivates the user break interrupt and allows user breaks by the terminal break key to be acknowledged
M.ENMI	enables task interrupts for messages sent to the calling task
M.ENUB	activates the user break interrupt and causes further user breaks by the terminal break key to be ignored
M.SYNCH	causes message and task interrupts to be deferred until the user makes a call to M.ANYW, M.ASYNCH, M.EAWAIT, or M.WAIT. Any deferred task interrupts are processed when a lower level task interrupt calls the M.ANYW, M.EAWAIT, or M.WAIT services.

1.5.7 User Break Interrupt Receivers

A task can enable the break interrupt level by calling the M.BRK monitor service to establish a break interrupt receiver address. The level becomes active as a result of a break interrupt request generated either from a hardware break or from an M.INT service call that specified this task. When the break level is active, end action, message, and normal execution processing is inhibited. The level active status is reset by calling the M.BRKXIT monitor service to exit from the pseudointerrupt (break) level.

1.5.8 User End-Action Receivers

When a task issues a no-wait I/O, send message, or send run request, a user-task end-action routine address can be specified. If specified, the routine is entered at the end-action priority level from the appropriate system postprocessing routine. When the end-action level is active, processing at the message or normal execution level is inhibited. The level active status is reset by calling the appropriate end-action service:

End-action Type	End-action Exit Service
I/O	H.IOCS,34
Send message	M.XMEA
Send run request	M.XREA

All types of user end-action exits provide a return or a continue-wait for any option. An interrupt exit normally returns to the interrupted context. A task can issue a series of no-wait request calls followed by a wait for any completion service call from the base level. This wait service (M.ANYW) places the task in an interruptive wait state, allowing the execution of postprocessing and end-action routines associated with the no-wait call. The return or continue wait end-action exit options allow the exiting end-action routine to return to the point following the wait for any call or to continue the wait for any state.

Note: A task is prohibited from making a wait for any service call from an end-action routine.

1.5.9 User Message Receivers

A task can enable the message interrupt level by calling the M.RCVR system service to establish a message interrupt receiver address. The level becomes active as the result of a message send request specifying this task as the destination task. When the message level is active, normal execution processing is inhibited. On entry to the message interrupt receiver, register one contains the address of the queue entry (MRRQ) in memory pool. The receiver can call a service M.GMSGP to store the message in a user receiver buffer. No-wait I/O is permitted with the M.WAIT service. After appropriate processing, the message interrupt level can be reset by calling the M.XMSGR system service to exit from the message interrupt receiver.

1.5.10 User Run Receivers

User run receivers execute at the normal task execution (base) level. The cataloged transfer address is used as the run receiver execution address. The run receiver mechanism is provided by the system to allow queued requests for task execution with optional parameter passing. When a run request is issued, the task load module name is used to identify the task to be executed. If a task of that load module name is currently active, the run request is queued from the DQE of the specified task. If the specified task is not active, it is first activated. When a task begins execution as the result of a run request, register one contains the address of the run request queue entry. The receiver can call a service M.GRUNP to store the run parameters in a user-receiver buffer. After appropriate processing, the run receiver task exits by calling the M.XRUNR system service. Any queued run requests are then processed.

1.5.11 User Abort Receivers

User abort receivers execute at the normal task execution (base) level. The user task establishes an abort receiver by calling the M.SUAR monitor service. If an abort condition is encountered during task operation, control is transferred to it. On entry, any active software interrupt level is reset, all outstanding operations or resource waits are complete, and all no-wait requests were processed. End-action routines associated with no-wait requests that completed while the abort was outstanding were not executed. Status bits reflecting this are posted in the appropriate FCBs and PSBs. Any files opened or resources allocated when the abort condition was encountered remain opened and/or allocated when the abort receiver is executed. The TSA stack is clean, and the context when the abort condition was encountered is stored in T.CONTXT. When the abort receiver is entered, register six contains a status byte reflecting task interrupt status when the abort condition was encountered.

Bit	Meaning if Set
24	N/A
25	N/A
26	user break interrupt active
27	end action interrupt active
28	message interrupt active

The standard exit service is used to exit from an abort receiver. If another abort condition is encountered while a task is in an abort receiver, the task is deleted.

1.6 Send/Receive Facilities

MPX-32 provides both message and run request send/receive processing. Run request services allow a task to queue an execution request (with optional parameter pass) for another task. Message services allow a task to send a message to another active task. The services provided for use by the destination tasks are called receiving task services. Those provided for tasks that issue the requests are called sending task services.

1.6.1 Receiving Task Services

1.6.1.1 Establishing Message and Run Receiver Capability

Establishing Message Receivers — To receive messages sent from other tasks, a task must be active and have a message receiver established. A message receiver is established by calling the system service M.RCVR, and providing the receiver routine address as an argument with the call.

Establishing Run Receivers — Any valid task can be a run receiver. Although a set of special run receiver services are provided, in the most simple case they are not needed. The run receiver mechanism is provided by the system to allow queued requests for task execution with optional parameter passing. The cataloged transfer address is used as the run receiver execution address. The task load module name is used to identify the task to be executed. If a task of that load module name is currently active and is a single-copied task, the run request is queued until the task exits. If a task of that load module name is currently active but is not a single-copied task, the load module is activated (multicopied) to process the request. If a multicopied task is waiting for a run request, the task number is used to activate the load module to process the request. When a single-copied task exits, any queued run requests are executed. If a run request is issued for a task that is not currently active, the task is activated automatically.

1.6.1.2 Execution of Message and Run Receiver Programs

Execution of Message Receiver Programs — When a task is active and has a message receiver established, it can receive messages sent from other tasks. A message sent to this task causes a software (task) interrupt entry to the established message receiver.

Execution of Run Receiver Programs — When a valid task is executed as a result of a run request sent by another task, it is entered at its cataloged transfer address. A run receiver executes at the normal task execution (base) level.

1.6.1.3 Obtaining the Passed Parameters

Obtaining Message Parameters — When the message receiver is entered, register one contains the address of the message queue entry in memory pool. The task can retrieve the message directly from memory pool or call a receiver service (M.GMSGP) to store the message into the designated receiver buffer. If the M.GMSGP service is utilized, the task must present the address of a five word parameter receive block (PRB) as an argument with the call.

Obtaining the Run Request Parameters — When the run receiver is entered, register one contains the address of the run request queue entry in memory pool. The task can retrieve the run request parameters directly from memory pool or call a receiver service (M.GRUNP) to store the run request parameters into the designated receiver buffer. If the M.GRUNP service is utilized, the task must present the address of a five word parameter receive block (PRB) as an argument with the call.

1.6.1.4 Exiting the Receiver Program

Exiting the Message Receiver — When processing of the message is complete, the message interrupt level must be exited by calling the M.XMSGR service. When M.XMSGR is called, the address of a two word receiver exit block (RXB) must be provided. The RXB contains the address of the return parameter buffer and the number of bytes (if any) to be returned to the sending task. The RXB also contains a return status byte to be stored in the parameter send block (PSB) of the sending task. After message exit processing is complete, the message-receiver queue for this task is examined for any additional messages to process. If none exist, a return to the base level interrupted context is performed.

Exiting the Run Receiver Task — When run request processing is complete, the task uses either the standard exit call (M.EXIT) or the special run receiver exit service (M.XRUNR). If the standard exit service (M.EXIT) is used to exit the run receiver task, no user status or parameters are returned. Only completion status is posted (in the scheduler status word) of the parameter send block (PSB) in the sending task. After completion processing for the run request is accomplished, the run receiver queue for this task is examined, and any queued run request causes the task to be re-executed. If the run receiver queue for this task is empty, a standard exit is performed.

If the special exit (M.XRUNR) is used to exit the run receiver task, the address of a two word receiver exit block (RXB) must be provided as an argument with the call. The RXB contains the address of the return parameter buffer and the number of bytes (if any) to be returned to the sending task. The RXB also contains a return status byte to be stored in the PSB of the sending task. After completion processing for the run request is accomplished, the exit control options in the RXB are examined. If the wait exit option is used, the run receiver queue for this task is examined for any additional run requests to be processed. If none exist, the task is put into a wait state, waiting for the receipt of new run requests. Execution of the task does not resume until such a request is received. If the terminate exit option is used, any queued run requests are processed. If the run receiver is empty, however, a standard exit is performed.

1.6.1.5 Waiting for the Next Request

In addition to the wait options described under the previous section, Exiting the Receiver Program, a special message-wait call is provided. When operating at the base execution level, a task that has established a message receiver can invoke a service call (M.SUSP) to enter a wait state until the next message is received.

A task can also make use of the M.ANYW service from the base software level. The M.ANYW service is similar to M.SUSP. However, the M.SUSP wait state is ended only on receipt of a message interrupt, timer expiration, or resume. The M.ANYW wait state is ended upon receipt of any message, end-action, or break software interrupt.

1.6.2 Sending Task Services

1.6.2.1 Sending the Request

Message Send Service — A task can send a message to another active task that has a message receiver established. The sending task must identify the destination task by task activation sequence number. When the send message service (M.SMSGR) is called, the address of a parameter send block (PSB) must be provided as an argument. The PSB format allows for the specification of the message to be sent, any parameters to be returned, scheduler and user status, and the address of a user end-action routine. No-wait and no call back mode control options are also provided.

Send Run Request Service — A task can send a run request to any active or inactive task, identifying the task by load module name or task number if the task is multicopied and waiting for a run request. When the run request service (M.SRUNR) is called, the doubleword-bounded address of a parameter send block (PSB) must be provided as an argument. The PSB format allows for the specification of the run request parameters to be sent, any parameters to be returned, scheduler and user status, and the address of a user end-action routine. No-wait and no call back mode control options are also provided.

1.6.2.2 Waiting for Request Completion

Waiting for Message Completion — A message can be sent in the wait or no-wait mode. If the wait mode is used, execution of the sending task is deferred until processing of the message by the destination task is complete. If the no-wait mode is used, execution of the sending task continues immediately after the request is queued. The operation in progress bit in the scheduler status field of the PSB is examined to determine completion. A sending task issues a series of no-wait mode messages followed by a call to the M.ANYW system wait service. This allows a task to wait for the completion of any no-wait mode messages previously sent. The completion of such a message causes resumption at the point after the M.ANYW call.

Waiting for Run Request Completion — Waiting for a run request completion follows the same form and has the same options as waiting for message completion.

1.6.2.3 End-Action Processing

Message End-Action Processing — User-specified end-action routines associated with no-wait mode message-send requests are entered at the end-action software interrupt level when the requested message processing is complete. Status and return parameters are posted as appropriate. When end-action processing is complete, the M.XMEA service must be called to exit the end-action software interrupt level.

Run Request End-Action Processing — Run request end-action processing follows the same form and has the same options as message end-action processing. The only difference is that the M.XREA service is used instead of M.XMEA.

1.6.2.4 Parameter Send Block (PSB)

The parameter send block (PSB) describes a send request issued from one task to another. The same PSB format is used for both message and run requests. The address of the PSB (word bounded) must be specified when invoking the M.SMSGR or M.SRUNR services, but is optional when invoking the M.PTSK service.

When a load module name is supplied in words 0 and 1 of the PSB, the operating system searches the system directory only. For activations in directories other than the system directory, a pathname or RID vector must be supplied.

When activating a task with the M.SRUNR or M.PTSK service, the value specified in byte 0 of PSB word 2 (PSB.PRI) is used to determine the task's execution priority. This value overrides the cataloged priorities of the sending and receiving tasks and the priority specified in the PTASK parameter block. However, priority clamping is used to prevent time-distribution tasks from using this value to execute at a real-time priority, and real-time tasks from executing at a time-distribution priority. Values that can be specified in PSB.PRI are 1-64 (to be the task priority), zero (to use the base priority of the sending task), and X'FF' (to ignore the PSB priority field).

A PSB can be specified as a parameter for the M.PTSK service, along with the required task activation (PTASK) block. The PTASK block also contains a priority specification field. The PSB priority value always overrides the PTASK block priority value.

	0 7	8 15	16 23	24 31
Word 0	Load module or executable image name (PSB.LMN) or zero if activation (or task number (PSB.TSKN) if message or run request to multicopied task)			
1	Load module or executable image name, pathname vector, or RID vector if activation (or zero if message or run request to multicopied task)			
2	Priority (PSB.PRI)	Reserved	Number of bytes to be sent (PSB.SQUA)	
3	Reserved	Send buffer address (PSB.SBA)		
4	Return parameter buffer length in bytes (PSB.RPBL)		Number of bytes actually returned (PSB.ACRP)	
5	Reserved	Return parameter buffer address (PSB.RBA)		
6	Reserved	No-wait request end action address (PSB.EAA)		
7	Completion status (PSB.CST)	Processing start status (PSB.IST)	User status (PSB.UST)	Options (PSB.OPT)

Word 0

Bits 0-31 Load module or executable image name — contains characters 1 through 4 of the name of the load module or executable image to receive the run request or

Task number — contains the task number of the task to receive the message or the task number of the multicopied load module or executable image to receive the run request.

Word 1

Bits 0-31 Load module or executable image name — contains characters 5 through 8 of the name of the load module or executable image to receive the run request, or zero if the message or run request is sent to multicopied load module or executable image.

Word 2

Bits 0-7 Contains the priority at which the receiver task is expected to be activated. Valid values are 1-64, zero, (for base priority of the sending task) and X'FF', which generates activation priority based on a combination of values that can be specified during task activation.

The following tables show how the priority of a receiver task is determined when activated with M.SRUNR or with M.PTSK.

When Activating with M.SRUNR

Send Task	Cataloged Priority of Receive task	Priority in PSB	Activates Receive task at
1-54	1-54	0	Send task cat. priority
1-54	55-64	0	55 (time-dist. clamp)
55-64	1-54	0	54 (real-time clamp)
55-64	55-64	0	Send task cat. priority
*	1-54	1-54	PSB priority
*	1-54	55-64	54 (real-time clamp)
*	55-64	1-54	55 (time-dist. clamp)
*	55-64	55-64	PSB priority
*	*	X'FF'	Receive task cat. priority

* not specified

When Activating with M.PTSK

Send Task	Cataloged Priority of Receive task	Priority in PTASK block	PSB	Activates Receive task at
1-54	1-54	0	0	Send task cat. priority
1-54	55-64	0	0	55 (time-dist. clamp)
1-54	*	1-54	0	Send task cat. priority
1-54	*	55-64	0	55 (time-dist. clamp)
55-64	1-54	0	0	54 (real-time clamp)
55-64	55-64	0	0	Send task cat. priority
55-64	*	1-54	0	54(real-time clamp)
55-54	*	55-64	0	Send task cat. priority
*	1-54	0	1-54	PSB priority
*	1-54	0	55-64	54 (real-time clamp)
*	55-64	0	1-54	55 (time-dist.clamp)
*	55-64	0	55-64	PSB priority
*	*	1-54	1-54	PSB priority
*	*	1-54	55-64	54 (real-time clamp)
*	*	1-54	X'FF'	PTASK block priority
*	*	55-64	1-54	55 (real-time clamp)
*	*	55-64	55-64	PSB priority
*	*	55-64	X'FF'	PTASK block priority
*	*	0	X'FF'	Receive task cat. priority

* not specified

Bits 8-15	reserved
Bits 16-31	Number of bytes to be sent — specifies the number of bytes to be passed (0 to 768) with the message or run request.

Word 3

Bits 0-7	reserved
Bits 8-31	Send buffer address — contains the word address of the buffer containing the parameters to be sent.

Word 4

Bits 0-15	Return parameter buffer length — contains the maximum number of bytes (0 to 768) that may be accepted as returned parameters.
Bits 16-31	Number of bytes actually returned — set by the send message or run request service upon completion of the request.

Word 5

Bits 0-7	reserved
Bits 8-31	Return parameter buffer address — contains the word address of the buffer where any returned parameters are stored.

Word 6

Bits 0-7	reserved
Bits 8-31	No-wait request end-action address — contains the address of a user routine to be executed at a software interrupt level upon completion of the request.

Word 7

Bits 0-7 Completion status — contains completion status information posted by the operating system as follows:

Bit	Meaning if Set
0	operation in progress (PSB.OIP)
1	destination task was aborted before completion of processing for this request (PSB.DTA)
2	destination task was deleted before completion of processing for this task (PSB.DTD)
3	return parameters truncated — attempted return exceeds return parameter buffer length (PSB.RPT)
4	send parameters truncated — attempted send exceeds destination task receiver buffer length (PSB.SPT)
5	user end-action routine not executed because of task abort outstanding for this task (can be examined in abort receiver to determine incomplete operation) (PSB.EANP)
6-7	reserved

Bits 8-15 Processing start (initial) status — contains initial status information posted by the operating system as follows:

Bit	Meaning if Set
0	normal initial status (PSB.IST)
1	message request task number invalid (PSB.TSKE)
2	run request load module or executable image name not found (PSB.LMNE)
3	reserved
4	file associated with run request load module or executable image name does not have a valid load module or executable image format (PSB.LMFE)
5	dispatch queue entry (DQE) space is unavailable for activation of the load module or executable image specified by a run request (PSB.DQEE)
6	an I/O error was encountered while reading the directory to obtain the file definition of the load module or executable image specified in a run request (PSB.SMIO)
7	an I/O error was encountered while reading the file containing the load module or executable image specified in a run request (PSB.LMIO)
8	memory unavailable
9	invalid task number for run request to module or executable image in RUNW state
10	invalid priority specification. An unprivileged task can not specify a priority which is higher than its own execution priority (PSB.PRIE).
11	invalid send buffer address or size (PSB.SBAE)
12	invalid return buffer address or size (PSB.RBAE)
13	invalid no-wait mode end action routine address (PSB.EAE)
14	memory pool unavailable (PSB.MPE)
15	destination task receiver queue is full (PSB.DTQF)

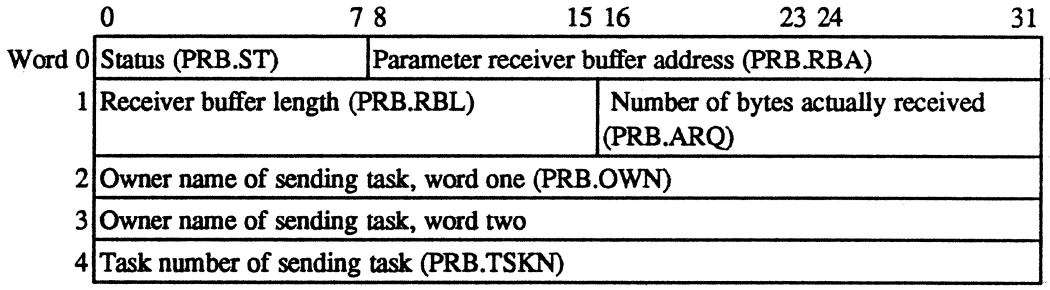
Bits 16-23 User status — defined by the destination task.

Bits 24-31 Options — contains user-request control specification as follows:

Bit	Meaning if Set
24	request is to be issued in no-wait mode (PSB.NWM)
25	do not post completion status or accept return parameters. This bit is examined only if bit 24 is set. When this bit is set, the request was issued in the no call back mode. (PSB.NCBM).

1.6.2.5 Parameter Receive Block (PRB)

The parameter receive block (PRB) is used to control the storage of passed parameters into the receiver buffer of the destination task. The same format PRB is used for message and run requests. The address of the PRB must be presented when the M.GMSGP or M.GRUNP services are invoked by the receiving task.



Word 0

Bits 0-7 Status — contains status as follows:

Bit	Meaning if Set
0	normal status
1	invalid PRB address
2	invalid receiver buffer address or size detected during parameter validation (PRB.RBAE)
3	no active send request (PRB.NSRE)
4	receiver buffer length exceeded (PRB.RBLE)
5-7	reserved

Bits 8-31 Parameter receiver buffer address — contains the word address of the buffer where any returned parameters are stored.

Word 1

Bits 0-15 — Receiver buffer length — contains the length of the receiver buffer (0 to 768 bytes).

Bits 16-31 — Number of bytes actually received — set by the operating system and is a maximum equal to the receiver buffer length.

Words 2 to 3

Bits 0-63 — Owner name of sending task — set by the operating system to contain the owner name of the task which issued the parameter send request.

Words 4

Bits 0-31 — Task number of sending task — set by the operating system to contain the task activation sequence number of the task which issued the parameter send request.

1.6.2.6 Receiver Exit Block (RXB)

The receiver exit block (RXB) is used to control the return of parameters and status from the destination (receiving) task to the task that issued the send request. It is also used to specify receiver exit options. The same format RXB is used for both messages and run requests. The address of the RXB must be presented as an argument when either the M.XMSGR or M.XRUNR services are called.

	0 7	8 15	16 23	24 31
Word 0	Return status (RXB.ST)	Return parameter buffer address (RXB.RBA)		
1	Options (RXB.OPT)	Reserved	Number of bytes to be returned (RXB.RQ)	

Word 0

Bits 0-7 — Return status — contains status as defined by the receiver task. Used to set the user status byte in the parameter send block (PSB) of the task which issued the send request.

Bits 8-31 — Return parameter buffer address — contains the word address of the buffer containing the parameters which are to be returned to the task that issued the send request.

Word 1

Bits 0-7 Options — contains receiver exit control options as follows:

Value	Meaning
0	wait for next run request (M.XRUNR). Return to point of task interrupt (M.XMSGR)
1	exit task, process any additional run requests. If none exist, perform a standard exit (M.XRUNR) not applicable for M.XMSGR

Bits 8-15 reserved

Bits 16-31 Number of bytes to be returned — contains the number of bytes (0 to 768) to be returned on a message or receiver run exit.

1.6.2.7 Message or Run Request Queue Entry (MRRQ)

The message or run request queue entry (MRRQ) is generated by the system to process a send request. After the MRRQ has been generated by the send service, it is attached to the appropriate queue slot in the DQE of the destination task. When the receiver program is entered, R1 contains the address of the MRRQ in memory pool. The receiver program can reference the MRRQ directly, without issuing a M.GRUNP or M.GMSGP service call. The same format MRRQ is used for both messages and run requests.

	0 7	8 15	16 23	24 31
Word 0	String forward address (MQ.SF)			
1	String backward address (MQ.SB)			
2	Priority (MQ.PR)	Address of parameter send block (PSB) (MQ.PSBA)		
3	Task number of sending task (MQ.TNST)			
4	Sending task owner name word one			
5	Sending task owner name word two			
6	Passed parameter quantity in bytes or number of bytes of storage space (MQ.PPQ)		Return parameter buffer length in bytes or number of actual return parameters (MQ.RBL)	
7	Completion status-- PSB format (MQ.CST)	Initial status-- PSB format (MQ.IST)	User status-- PSB format (MQ.UST)	Options-- PSB format (MQ.OPT)
8-9	End action PSD (MQ.EAPSD)			
10	Parameter area pointer (MQ.PPTR)			
11	Reserved			
n	Variable length storage area for passed and returned parameters			

Word 0

Bits 0-31 — String forward address — contains the address of next entry of top-to-bottom list.

Word 1

Bits 0-31 — String backward address — contains address of next entry in bottom-to-top list.

Word 2

Bits 0-7 — Priority — contains the priority (1 to 64) of this request.

Bits 8-31 — Address of parameter pend block (PSB) — contains the logical address of the PSB in the address space of the task which initiated the request.

Word 3

Bits 0-31 — Task number of requesting task — contains the task activation sequence number of the task which issued the request.

Words 4-5

Bits 0-63 — Send task's owner name.

Word 6

Bits 0-15 — Passed parameter quantity — contains the number of bytes sent to the destination task.

Bits 16-31 — Return parameter buffer length — contains the length in bytes of the return parameter buffer in the task which issued the request.

Word 7

Bits 0-15 — Scheduler status — contains status information to be posted in the scheduler status field of the PSB upon request completion. See PSB format.

Bits 16-23 — User status — contains status as defined by the destination task.

Bits 24-31 — Options — contains user request control specifications as follows:

Bit	Meaning if Set
24	request is in no-wait mode
25	request is in no call back mode (no wait, no status, no return parameters)

Words 8-9

Bits 0-31	End action PSD (words 1 & 2) (post processing service PSD).

Word 10

Bits 0-31	Pointer to variable length parameter area.

Word 11

Bits 0-31	Reserved

1.6.2.8 Messages and Run Request Services Summary

The following table lists the message and run request services provided by the MPX-32 system.

Run Request Services	Message Services	Function
Receiver Services:		
N/A	M.RCVR *recvaddr*	Establish receiver address
M.GRUNP *prbaddr*	M.GMSGP *prbaddr*	Get parameters
M.XRUNR *rxbaddr* or M.EXIT	M.XMSGR *rxbaddr*	Exit receiver
N/A	M.ANYW *time1* or M.SUSP *taskno,time1*	Wait for receipt of next message
Sender Services:		
M.SRUNR *psbaddr*	M.SMSGR *psbaddr*	Send request
M.ANYW *time1*	M.ANYW *time1* M.EAWAIT *time1*	Wait for any request completion
M.XREA	M.XMEA	Exit user end action service

Argument	Description
recvaddr	address of receiver
prbaddr	address of parameter receive block (PRB)
rxbaddr	address of receiver exit block (RXB)
psbaddr	address of parameter send block (PSB)
taskno	contains zero
time1	contains zero if indefinite wait, or contains a negative number of time units to be used as a wait time-out value

1.7 Device Address Specification

Device addresses are specified using a combination of three levels of identification: device type, device channel/controller address, and device address/subaddress.

A device may be specified using the generic device type mnemonic only, which will result in allocation of the first available device of the type requested. Device type mnemonics are listed in Table 1-1.

A second method of device specification is achieved by using the generic device type and specifying the channel/controller address which results in allocation of the first available device of the type requested on the channel/controller specified.

The third method of device selection requires specification of the device type mnemonic, channel/controller, and device address/subaddress. This method allows specification of a specific device.

Examples

Type 1 - Generic device class:

```
ASSIGN OUT TO DEV=M9
```

In this example, the file associated with logical file code `OUT` is allocated to any `9`-track tape unit on any channel.

Type 2 - Generic device class and channel/controller:

```
ASSIGN OUT TO DEV=M910
```

In this example, the file associated with logical file code `OUT` is allocated to the first available `9`-track tape unit on channel `10`. The specification is invalid if a `9`-track tape unit does not exist on the channel.

Type 3 - Specific device request:

```
ASSIGN OUT TO DEV=M91001
```

In this example, the file associated with logical file code `OUT` is allocated to the `9`-track tape unit `01` on channel `10`. The specification is invalid if unit `01` on channel `10` does not exist or is not a `9`-track tape.

GPMC/GPDC devices are specified in keeping with the general structure as defined. For instance, the CRT at subaddress `04` on GPMC 01 whose channel address is `20` would be identified as follows:

```
ASSIGN OUT TO DEV=TY2004
```

A special device type, NU, is available for null device specifications. Files accessed using this device type generate an end-of-file upon attempt to read and normal completion upon attempt to write.

Assignment of logical file codes to the operator console is achieved through usage of the device type CT.

A description of device selection possibilities is constructed as follows:

Disk

DC	any disk except memory disk
DM	any moving head or memory disk
DM08	any moving head disk on channel 08
DM0801	moving head disk 01 on channel 08
DM0002	memory disk 02 on channel 00
DF	any fixed head disk
DF04	any fixed head disk on channel 04
DF0401	fixed head disk 01 on channel 04

Tape

MT	any magnetic tape
M9	any 9-track magnetic tape
M910	any 9-track magnetic tape on channel 10
M91002	9-track magnetic tape 02 on channel 10

Card Equipment

CR	any card reader
CR78	any card reader on channel 78
CR7800	card reader 00 on channel 78

Line Printer

LP	any line printer
LP7A	any line printer on channel 7A
LP7A00	line printer 00 on channel 7A
LP7EA0	serial printer A0 on ACM channel 7E

Table 1-1
Device Type Mnemonics and Codes

Device Type Code	Device Type Mnemonic	Device Description
00	CT	operator console (not assignable)
01	DC	any disk unit except memory disk
02	DM	any moving head or memory disk
03	DF	any fixed head disk
04	MT	any magnetic tape unit
05	M9	any 9-track magnetic tape unit*
06	M7	any 7-track magnetic tape unit*
08	CR	any card reader
0A	LP	any line printer
0B	PT	any paper tape reader-punch
0C	TY	any teletypewriter (other than console)
0D	CT	operator console (assignable)
0E	FL	floppy disk
0F	NU	null device
10	CA	communications adapter (binary synchronous/asynchronous)
11	U0	available for user-defined applications
12	U1	available for user-defined applications
13	U2	available for user-defined applications
14	U3	available for user-defined applications
15	U4	available for user-defined applications
16	U5	available for user-defined applications
17	U6	available for user-defined applications
18	U7	available for user-defined applications
19	U8	available for user-defined applications
1A	U9	available for user-defined applications
1B	LF	line printer/floppy controller (used only with SYSGEN)
N/A	ANY	any nonfloppy disk except memory disk

* When both 7- and 9-track magnetic tape units are configured, the designation must be 7-track.

1.8 CPU Scheduling

The MPX-32 CPU scheduler allocates CPU execution time to active tasks. Tasks are allocated CPU time based on execution priority and execution eligibility. Execution priority is specified when a task is cataloged into the system. Execution eligibility is determined by the task's readiness to run.

1.8.1 Execution Priorities

The MPX-32 system provides 64 levels of execution priority. These priority levels are divided into two categories. Real-time tasks operate in the priority range 1 to 54. Time-distribution tasks operate in the priority range 55 to 64.

1.8.2 Real-Time Priority Levels (1 to 54)

Real-time tasks are scheduled on a strict priority basis. The system does not impose time-slice, priority migration, or any other scheduling algorithm that interferes with the execution priority of a real-time task. Execution of an active real-time task at its specified priority level is inhibited only when it is ineligible for execution (not ready to run). Execution of a real-time task can always be pre-empted by a higher priority real-time task that is ready to run.

1.8.3 Time-Distribution Priority Levels (55 to 64)

For tasks that execute at priority levels 55 to 64, MPX-32 provides a full range of priority migration, situational priority increment, and time-quantum control.

1.8.4 Priority Migration

The specified execution priority of a time-distribution task is used as the task's base execution priority. Each time-distribution task's current execution priority is determined by the base priority level as adjusted by any situational priority increment. The current execution priority is further adjusted by increasing the priority by one level whenever execution is pre-empted by a higher priority time-distribution task, and decreasing the priority whenever the task gains CPU control. The highest priority achievable by a time-distribution task is priority level 55. The lowest priority is the task's base execution priority level.

1.8.4.1 Situational Priority Increments

Time-distribution tasks are given situational priority increments to increase program responsiveness. The effect of situational priority increments is to give execution preference to tasks that are ready to run after having been in a natural wait state. A task that is CPU bound migrates toward its base execution priority. Situational priority increments are invoked when a task is unlinked from a wait-state list and relinked to the ready-to-run list.

Situation	Priority Increment
Terminal input wait complete	Base level + 2
I/O wait complete	Base level + 2
Message (send) wait complete	Base level + 2
Run request (send) complete	Base level + 2
Memory (inswap) wait complete	Base level + 3
Pre-empted by real-time task	Level 55

1.8.5 Time-Quantum Controls

Two time-quantum values can be specified at system generation. If these values are not specified, system default values are used. The two quantum values are provided for scheduling control of time-distribution tasks.

The first quantum value (stage 1) indicates the minimum amount of CPU execution time guaranteed to a task before pre-emption by a higher priority time-distribution task. The stage 1 quantum value is also used as a swap inhibit quantum after inswap. The second quantum value represents the task's full-time quantum. The difference between the first and second quantum values defines the execution period called quantum stage 2. During quantum stage 2, a task is pre-empted and/or out-swapped by any higher priority task. When a task's full time-quantum has expired, it relinks to the bottom of the priority list at base execution priority.

Time-quantum accumulation is the accumulated sum of actual execution times used by this task. A task's quantum accumulation value resets when the task voluntarily relinquishes CPU control; for example, suspend, wait I/O, etc.

1.8.6 State Chain Management

The current state of a task, such as ready to run or waiting for I/O, is reflected by the linkage of the dispatch queue entry associated with the task into the appropriate state chain. The state queues are divided into two major categories: ready to run and waiting. The ready-to-run category is subdivided by priority, with a single queue for the real-time priorities and a separate queue for each of the time-distribution priority levels. The waiting category is subdivided according to the resource or event required to make the task eligible for execution.

MPX-32 State Queues

Ready-to-Run Queues

1. Current CPU task (in execution) — CURR
2. Current IPU task (in execution) — CIPU
3. IPU requesting state — RIPU
4. Real-time priority levels (1-54) — SQRT
5. Time-distribution priority level 55 — SQ55
6. Time-distribution priority level 56 — SQ56
7. Time-distribution priority level 57 — SQ57
8. Time-distribution priority level 58 — SQ58
9. Time-distribution priority level 59 — SQ59
10. Time-distribution priority level 60 — SQ60
11. Time-distribution priority level 61 — SQ61
12. Time-distribution priority level 62 — SQ62
13. Time-distribution priority level 63 — SQ63
14. Time-distribution priority level 64 — SQ64

Wait Mode Operation Queues

15. Wait mode interactive input — SWTI
16. Wait mode I/O — SWIO
17. Wait mode send message — SWSM
18. Wait mode send run request — SWSR
19. Wait mode low speed output (not implemented) — SWLO

Execution Wait Queues

20. Suspended waiting for message interrupt, timer expiration, or resume — SUSP
21. Waiting for run request or timer expiration — RUNW
22. Operator hold, waiting for continue — HOLD

Wait for Any Operation Complete Queue

23. Waiting for completion of any no-wait I/O, no-wait message, no-wait run request, or any message interrupt or break — ANYW

Waiting for Resource Queues

24. Waiting for disk space — SWDC
25. Waiting for peripheral device — SWDV
26. Reserved
27. Waiting for memory — MRQ
28. Waiting for memory pool — SWMP

1.9 FAT/FPT and Blocking Buffer Allocation

During the task allocation process, separate areas are reserved in a task's TSA for FAT/FPT pairs and blocking buffers. The size of each area is fixed for the duration of a task's execution. The size of the FAT/FPT area limits the number of file codes that a task can have allocated concurrently. The size of the blocking buffer area limits the number of file codes assigned to blocked devices or files that a task can allocate concurrently. The number of entries in each area is established as follows.

1.9.1 FAT/FPT Area

Nonshared task — one FAT and FPT entry for each cataloged assignment, plus one entry for each TSM assignment that does not override a cataloged assignment, plus the number specified on the cataloger FILES directive.

Shared task — the number specified on the cataloger FILES directive.

1.9.2 Blocking Buffer Area

Nonshared task — from the assignments resulting from merging cataloger and TSM assignments, one buffer for each ASSIGN; plus one buffer for each ASSIGN to a magnetic tape or disk unit on which the unblocked option is not specified, plus one buffer for each ASSIGN plus the number specified on the cataloger BUFFERS directive.

Shared task — the number specified on the cataloger BUFFERS directive.

Cataloger and TSM ASSIGN directives are modified by the addition of an unblocked specification as follows:

```
ASSIGN lfc TO file BLOCKED=N
```

The following cataloger directives are added:

FILES *number* — *number* specifies the maximum number of dynamically allocated file codes that a nonshared task can allocate concurrently. It specifies the maximum number of file codes that a shared task can have allocated concurrently.

BUFFERS *number* — *number* specifies the maximum number of dynamically allocated file codes assigned to blocked files or devices that a nonshared task can allocate concurrently. It specifies the maximum number of file codes assigned to blocked files or devices that a shared task can allocate concurrently.

FILES and BUFFERS override parameters are specified in the parameter task activation (M.PTSK) system service. These parameters allow addition of TSM FILES and BUFFERS directives if required by a future "load and go" capability.

1.10 Indirectly Connected Interrupts

An indirectly connected interrupt is an interrupt that is associated with an MPX-32 task. When the interrupt occurs, the associated task is resumed. An interrupt is declared as indirectly connected at system generation (SYSGEN) time. This declaration causes SYSGEN to generate an indirectly connected task linkage block (ITLB). The ITLB is permanently associated with the specified interrupt level, but only becomes associated with an MPX-32 task when the M.CONN system service is invoked. A task can be disconnected from an interrupt level by invoking the M.DISCON system service.

1.10.1 Connect Task to Interrupt Service (M.CONN)

The M.CONN system service associates an MPX-32 task with an external interrupt that was declared at SYSGEN as indirectly connected. When called, M.CONN is presented the priority level of the interrupt and the task activation sequence number of the task. The task number is first validated to ensure that it is currently active and has either the same owner name as the calling task, or the owner name of the calling task is privileged or is not restricted from access to tasks of a different owner. If so, the M.CONN service next checks to see if the specified task is already connected to an interrupt. DQE.ILN in the DQE contains the interrupt priority level if the task is already connected. If the task is not previously connected, the M.CONN service searches the indirectly connected task linkage table (ITLT) to find the linkage block (ITLB) associated with this interrupt. If one exists and is not already connected, the DQE address of the task being linked is stored in word one of the ITLB to reflect the linkage. DQE.ILN in the DQE is set to contain the interrupt priority level.

Note: The task is automatically disconnected from the interrupt on abort, delete, or exit.

1.10.2 Disconnect Task from Interrupt Service (M.DISCON)

The M.DISCON system service disconnects an MPX-32 task from an external interrupt it had previously been connected to. When called, M.DISCON is presented the task activation sequence number of the task as an argument with the call. If the specified task is not connected to an interrupt, DQE.ILN in the DQE is equal to zero and the request is ignored. Otherwise, DQE.ILN contains the external interrupt priority level. M.DISCON uses this priority level to locate the linkage block (ITLB) in the linkage table (ITLT). The DQE address (word one of the ITLB) is cleared to mark the level as disconnected. DQE.ILN is cleared in the DQE of the specified task.

1.10.3 Indirectly Connected Task Linkage Table (ITLT)

The indirectly connected task linkage table (ITLT) is a variable length table built by SYSGEN. It contains an entry for each interrupt specified as indirectly connectable. An entry is called an indirectly connected task linkage block (ITLB) and is 24 words in length. The address of the ITLT is contained in C.ITLT. The number of entries in ITLT is contained in C.NITI. Both C.ITLT and C.NITI are initialized by SYSGEN.

1.10.4 Indirectly Connected Task Linkage Block (ITLB)

An entry in the indirectly connected task linkage table is called an indirectly connected task linkage block (ITLB). An ITLB is 24 words long and is used to associate an external interrupt with an indirectly connected task.

	0	31
Word 0	Priority level	[DATAW X'YY']
1	DQE address of indirectly connected program	[DATAW 0]
2	Old PSD word one	[DATAW 0]
3	Old PSD word two	[DATAW 0]
4	New PSD word one	[GEN 1/1, 12/0, 19/W($+2W)]
5	New PSD word two	[GEN 1/1, 14/0, 1/1, 1/0, 1/0, 14/0]
6	Increment global interrupt count instruction	[ABM 31,C.GINT]
7	Save register instruction	[STF R0,$+9W]
8	Branch and link to ICP routine	[BL ICP]
9	Address of register save area for S.EXEC5 call	[LA X2, $+7W]
10	Old PSD for S.EXEC5 call	[LD R6, -8W]
11	Reserved	
12	Deactivate interrupt	[DAI X'YY']
13	Branch back for S.EXEC5 call	[BL S.EXINT]
14-15	Reserved for future use	
16 - 23	Register save area	

Word 0

Bits 0-31 Priority level — set by SYSGEN to contain the priority level of the associated interrupt.

Word 1

Bits 0-31 DQE address of indirectly connected program — contains the dispatch queue entry (DQE) address of the task to be resumed on occurrence of this interrupt. Initially set to zero by SYSGEN. Initialized by M.CONN system service.

Words 2 to 3

Bits 0-63 Old PSD — contains the old PSD slot of the interrupt control block. Used to store the PSD associated with the interrupted context. Initially set to zero by SYSGEN. The dedicated interrupt location (IVL) is initialized by SYSGEN to contain the address of word two of the ITLB.

Words 4 to 5

Bits 0-63 — New PSD — contains the new PSD slot of the interrupt control block to be used on occurrence of this interrupt. Causes execution to begin at ITLB word 6 in privileged mode, unblocked state, with old map status retained.

Word 6

Bits 0-31 — Increment global interrupt instruction — contains an add bit in memory instruction to increment the global interrupt count. Execution begins at this location when the associated interrupt occurs. It must be the first instruction executed in ICP. This location is initialized by SYSGEN to contain an ABM 31,C.GINT.

Word 7

Bits 0-31 — Save registers instruction — contains a store file instruction to save all eight registers in words 16 to 23 of the ITLB. This location is initialized by SYSGEN to contain an STF R0,$+9W.

Word 8

Bits 0-31 — Branch and link to ICP routine — executed after the register save instruction on occurrence of the associated interrupt. Transfers control to the single-copied ICP routine. This location is initialized by SYSGEN to contain a BL ICP.

Words 9 to 13

Branch back for S.EXINT call returns control to this location after S.EXEC14 is called in the ICP routine. Set-up is made for exiting the interrupt; then control is transferred back to ICP for the S.EXEC5 call.

Words 14 to 15

reserved for future use

Words 16 to 23

register save area

1.10.5 Indirectly Connected Interrupt Program (H.ICP)

The indirectly connected interrupt program (H.ICP) is a single-copied routine that processes all indirectly connected external interrupts. It is entered in blocked, unmapped mode with the end address (+1W) of the linkage block (ITLB) in R0. The global interrupt count is incremented within the ITLB and the registers from the interrupted context are stored in words 16 to 23 of the block. When H.ICP is entered, it checks ITLB word one to verify connection of the interrupt to an MPX-32 task. If the interrupt is not connected, it is ignored and H.ICP transfers back to the ITLB to exit the interrupt.

When ITLB word 1 contains a valid DQE address, H.ICP performs the following checks to determine if the task resumption time can be optimized:

- is the task a real-time task
- does the task have any system action requests or task interrupts pending
- is the task outswapped
- are interrupts nested
- is context switching inhibited
- can the task run in the CPU

If the task meets these checks, its resumption can be optimized and dispatched directly from H.ICP.

If the task does not meet the checks, S.EXEC14 links the task to the ready to run queue. The task then exits the interrupt level via S.EXEC5.

1.11 Miscellaneous System Macros

1.11.1 M.BACK

This macro backspaces the current address of a blocked file by the specified number of file or record marks.

Calling Sequence

M.BACK *addr*,[R] [,*num*]

addr is the FCB address

R specifies record. If not specified, the default is file.

num is the number of record or file marks to be backspaced. The number specified must be word scaled, for example, one word for one record. If not supplied, the current contents of register four are used.

1.11.2 M.CALL

This macro generates a supervisor call instruction. If code has been assembled with an MPX-32 PRE file, interrupts are unblocked by default (SVC '6' call). If code has not been assembled with an MPX-32 PRE file, interrupts are blocked by default (SVC '0' call). Each of the three standard MPX-32 PRE files contain the symbol MPX_SVC. This symbol is set to 6 so MPX-32 uses the optimized M.CALL macro, which increases the performance of MPX-32. Defaults can be overridden with the *state* parameter on individual macro calls.

Calling Sequence

M.CALL *name,num*[,*state*]

name is the name of a system module

num is an entry point number (1,2,3,...) within the system module

state is the state of processing:

RETAIN generates an SVC '0' call with blocked interrupts

UEI generates an SVC '6' call with unblocked interrupts

1.11.3 M.CLSE

This macro marks a file closed to subsequent service. An end-of-file (EOF) mark can be written and a rewind can be performed.

Calling Sequence

M.CLSE *addr*,[EOF],[REW]

addr	FCB address
EOF	specifies an end-of-file mark is to be written
REW	specifies the file is to be rewound

1.11.4 M.DFCB

This macro creates a file control block (FCB). It also sets the appropriate parameters and specifications that are common to I/O requests issued for the file.

Calling Sequence

M.DFCB *label*,*lfc*,[*count*],[*addr1*],[*addr2*],[*addr3*],

[NWT],[NER],[DFI],[NST],[RAN]

[ASC|BIN],[LDR|NLD],[INT|PCK],[EVN|ODD] [,556,800]

label	is the ASCII character string to be used as the symbolic label for the address of the FCB
lfc	is the 1- to 3-character ASCII string to be used as the logical file code in the FCB
count	is the transfer count in bytes
addr1	is the data transfer address
addr2	is the error return address
addr3	is the random access address expressed as the hexadecimal block number (zero origin) relative to the base of the random access file
NWT	is the no-wait I/O specification indicator
NER	is the inhibit peripheral error processing indicator
DFI	is the inhibit data formatting indicator
NST	is the inhibit status testing indicator
RAN	is the random access mode indicator
ASC or BIN	is the forced ASCII or forced binary mode specification for read or punch operations performed when the file code for this file is assigned to a card reader
LDR or NLD	is the skip leader or do not skip leader specification when the file code for this file is assigned to a paper tape reader/punch device

INT or PCK — is the interchange or packed mode specification when the file code for this file is assigned to a magnetic tape device

EVN or ODD — is the even or odd parity specification when the file code for this file is assigned to a magnetic tape device

556 or 800 — is the 556 or 800 BPI tape density specification when the file code for this file is assigned to a magnetic tape device

1.11.5 M.DFCBE

This macro creates an expanded file control block (FCB). It also sets the appropriate parameters and specifications that are common to I/O requests issued for the file.

Calling Sequence

M.DFCBE *label,lfc*,[*count*],[*addr1*],[*addr2*],[*addr3*],

[NWT],[NER],[DFI], [NST], [RAN],

[ASC|BIN],[LDR|NLD],[INT|PCK],[EVN|ODD],[556,800]

[*addr4*],[*addr5*],[*addr6*]

label — is the ASCII character string to be used as the symbolic label for the address of the FCB

lfc — is the 1- to 3-character ASCII string to be used as the logical file code in the FCB

count — is the transfer count in bytes

addr1 — is the data transfer address

addr2 — is the wait I/O error return address

addr3 — is the random access address expressed as the hexadecimal block number (zero origin) relative to the base of the random access file

NWT — is the no-wait I/O specification indicator

NER — is the inhibit peripheral error processing indicator

DFI — is the inhibit data formatting indicator

NST — is the inhibit status testing indicator

RAN — is the random access mode indicator

ASC or BIN — is the forced ASCII or forced binary mode specification for read operations performed when the file code for this file is assigned to a card reader

LDR or NLD — is the specify skip leader or do not skip leader specification when the file code for this file is assigned to a paper tape reader/punch device

INT or PCK	is the interchange or packed mode specification when the file code for this file is assigned to a magnetic tape device
EVN or ODD	is the even or odd parity specification when the file code for this file is assigned to a magnetic tape device
556 or 800	is the 556 or 800 BPI tape density specification when the file code for this file is assigned to a magnetic tape device
addr4	is the no-wait I/O normal end-action service address
addr5	is the no-wait I/O error end-action service address
addr6	is the user-supplied blocking buffer

1.11.6 M.EIR

This macro is called by the resident system module's initialization entry points at entry. It stores R0 for later recall by M.XIR, the initialization entry point exit macro.

Calling Sequence

M.EIR

1.11.7 M.FCBEXP

This macro defines a file control block (FCB) to be used for an execute channel program request.

Calling Sequence

M.FCBEXP *label,lfc*[, [*cpaddr*],[*tout*], [PCP],[NWI], [NST],
[*ssize*],[*sbuffer*], [*nowait*],[*nowaiterror*],[*waiterror*],[*psize*],[*ppciadr*]

label	is the ASCII string to use as the symbolic label for the address of the FCB
lfc	is the logical file code, word 0, bits 8 to 31 of the FCB
cpaddr	the logical address of the channel program to be executed
tout	a time-out value specified in seconds
PCP	specifies physical channel program
NWI	specifies no-wait I/O request
NST	specifies status checking not requested
ssize	the size of the user-specified sense buffer
sbuffer	the address of the user-specified sense buffer
nowait	normal no-wait end-action return address

nowaiterror no-wait end-action error return address

waiterror wait end-action error return address

psize size of PPCI status buffer to use

ppciaddr PPCI end-action address

1.11.8 M.FWRD

This macro advances the current address of a blocked file by the number of file or record marks specified.

Calling Sequence

M.FWRD *addr*, [R] [,*num*]

addr is the FCB address

R specifies record. If not specified, the default is file.

num is the number of record or file marks to be advanced, one word for one record.

1.11.9 M.INIT

This macro initializes device handler parameters through entry point eight. The code generated by this macro is executed by SYSGEN and overlayed.

Calling Sequence

M.INIT *label*,[NOP][,SPA1, [SPA2]...[,SPA15]]

label is the entry point truncated label; for example, MT0 for magnetic tape handler. This argument must be three ASCII characters. The first two represent the device mnemonic and the third is zero.

NOP specifies that TD 2000 level device status testing is not to be performed

SPA1-SPA15 are the SPA parameters to be initialized. A maximum of 15 parameters can be specified.

Usage:

```
M.INIT MT0,,SPA1,,SPA3
```

When placed as the last source statement in the device handler, this macro provides the necessary code to initialize the handler. The HAT must be modified to specify entry point eight and an additional entry must be made in the table (ACH MT00.8).

1.11.10 M.INITX

This macro is called by the handler initialization macros to combine basic instruction and commands with priority levels and device addresses for later execution within the handler. When this macro is called, R5 must be preloaded with the properly positioned priority level or device address.

Calling Sequence

M.INITX *cmd,mask*

cmd is the basic instruction or command

mask is a mask which is ORed with command

1.11.11 M.IOFF

This macro generates a block external interrupt (BEI) instruction that prevents the CPU from sensing all external interrupt requests generated by the I/O channel and RTOM.

Calling Sequence

M.IOFF

1.11.12 M.IONN

This macro generates an unblock external interrupt (UEI) instruction that causes the CPU to sense all external interrupt requests generated by the I/O channel and RTOM.

Calling Sequence

M.IONN

1.11.13 M.IPUOFF

This macro causes the IPU to be put offline in software by setting bit C.IPUOFF.

Calling Sequence

M.IPUOFF

1.11.14 M.IPUON

This macro causes the IPU to be put online in software by resetting bit C.IPUOFF.

Calling Sequence

M.IPUON

1.11.15 M.IPURTN

This macro allows an IPU executable system module to return to the caller with registers preserved. The system service performs a register pop-up, except for those registers to be preserved, and returns to the location specified by the saved program status word (PSW).

Calling Sequence

M.IPURTN *regn* [,*regn*]...

regn is a list of register numbers (0 to 7) identifying the registers to be preserved through the register pop-up. Any register not specified is not preserved.

1.11.16 M.IVC

This macro connects a handler entry point to an interrupt vector location.

Calling Sequence

M.IVC *num,addr*

num is the register number containing the interrupt level

addr is the handler entry point address label

1.11.17 M.KILL

This macro disables the CPU Halt Trap Processor (H.IPHT) and halts the system.

Calling Sequence

M.KILL *addr*

addr is the address of a 4-character ASCII crash code

1.11.18 M.MODT

This macro builds an entry in the module address table.

Calling Sequence

M.MODT *addr,num*

addr is the address label of the module's HAT table

num is the module number

1.11.19 M.OPEN

This macro controls gating. If code has been assembled with an MPX-32 PRE file, context switch inhibit is reset and an SVC '3' call is issued only if a scheduling event occurred while M.SHUT was in effect. If code has been assembled without an MPX-32 PRE file, context switch inhibit is removed by issuing an SVC '3' call with each M.OPEN call. Each of the three standard MPX-32 PRE files contain the symbol MPX_SVC. This symbol is set to 6 so MPX-32 uses the optimized M.OPEN macro, which increases the performance of MPX-32.

These processing states can be altered on a call by call basis by specifying the *state* parameter.

Entry Conditions

Calling Sequence

M.OPEN [*state*]

state specifies the state of processing:

- RETAIN retains original functionality of the M.OPEN call that removes the task context switch inhibit state set by M.SHUT by issuing the SVC '3' call.
- FAST provides the optimized state of resetting context switch inhibit and issuing the SVC '3' call only if a scheduling event occurred while M.SHUT was in effect. The optimized M.OPEN cannot be used in mapped out tasks.

1.11.20 M.RTNA

This macro provides the facility to return to the caller from a system module to an address other than that specified by the saved PSW. It is used primarily for denial returns. It operates like the M.RTRN macro. The interrupt handler tests for the presence of an address specification in the parameter and replaces the saved program status word (PSW) if an address is found.

Calling Sequence

M.RTNA *addr*,*regn* [,*regn*]...

addr is the register number of the register containing the address where return control resumes

regn is a list of register numbers (0 to 7) identifying the registers to be preserved through the register pop-up. Any register not specified is not preserved.

1.11.21 M.RTRN

This macro is the complement of M.CALL and allows a system module to return to the caller with registers preserved. The system service performs a register pop-up (except for those registers to be preserved) and returns to the location specified by the saved program status word (PSW).

Calling Sequence

M.RTRN *regn* [,*regn*]...

regn is a list of register numbers (0 to 7) identifying the registers to be preserved through the register pop-up. Any register not specified is not perserved.

1.11.22 M.RTRNOS

This macro is used to return control from the task level debugger to the MPX-32 Operating System.

Calling Sequence

M.RTRNOS

1.11.23 M.SHUT

This macro is used to control gating. It results in context switching being inhibited. This macro should not be used in a user task which is eligible for IPU execution. See M.USHUT.

Calling Sequence

M.SHUT

1.11.24 M.SPAD

At each register push-down level, 22 scratchpad storage cells are provided for the use of re-entrant system modules. The scratchpad storage macro, M.SPAD, provides a convenient means of referencing the current level of scratchpad storage. The M.SPAD macro performs any memory reference operating on at least a word boundary (LW, STF, ARMD, DVMW), or any bit in memory operation (TBM, SBM, ABM, ZBM).

Calling Sequence

M.SPAD *mnem,reg,spad,index*

mnem is an instruction mnemonic defining the operation to be performed

reg is the register number (0 to 7) or bit position (0 to 31) on which the operation is to be performed, or null

spad is the scratchpad cell number (1 to 22) to be referenced by the operation

index is an index register number (1, 2, or 3) that is used to perform the operation

1.11.25 M.SVCP

This macro establishes any required protect bits in the high order byte of the SVC '1' table. A table is supplied containing 16 bit entries aligned on a halfword boundary. Each entry contains the SVC number in byte 0 and the required protect bits in byte 1. The following protect bits are defined:

Bit	Meaning if Set
0	privileged SVC
1	IPU
2	base mode tasks executable
3-7	reserved

Calling Sequence

M.SVCP *addr,num*

addr is the address of the data table

num is the number of entries in the table

1.11.26 M.SVCP2

This macro establishes any required protect bits in the high order byte of the SVC '2' table. A table is supplied containing 16 bit entries aligned on a halfword boundary. Each entry contains the SVC number in byte 0 and the required protect bits in byte 1. The following protect bits are defined:

Bit	Meaning if Set
0	privileged SVC
1	IPU
2	base mode tasks executable
3-7	reserved

Calling Sequence

M.SVCP2 *addr,num*

addr is the address of the data table

num is the number of entries in the table

1.11.27 M.SVCT

This macro builds one entry in the SVC '1' table for each SVC type one defined in the calling module's prototype SVC table. Each one word entry contains the address of the corresponding SVC; i.e., the 20th entry contains the address of the 20th SVC.

Calling Sequence

M.SVCT *addr,num*

addr is the address label for the calling module's prototype SVC table

num is the number of SVC entries in the module's prototype SVC table

1.11.28 M.SVCT2

This macro builds one entry in the SVC '2' table for each SVC type two defined in the calling module's prototype SVC table. Each one word entry contains the address of the corresponding SVC. For example, the 20th entry contains the address of the 20th SVC.

Calling Sequence

M.SVCT2 *addr,num*

addr is the address label for the calling module's prototype SVC table

num is the number of SVC entries in the module's prototype SVC table

1.11.29 M.TRAC

See Chapter 6 - System Trace.

1.11.30 M.TRPINT

This macro generates an entry in the trap vector table.

Calling Sequence

M.TRPINT *rpl,tcb*

rpl is the hexadecimal trap priority level

tcb is the address of the trap context block of the user trap handler

1.11.31 M.TSAD

This macro allows resident modules to obtain the TSA address regardless of which processor they may be executing in.

Calling Sequence

M.TSAD *regn* [,*proc*]

regn the general purpose register R0 through R7 to which the TSA address will be returned.

proc CPU or IPU. Omission of this field indicates either processor may be executing this macro call.

1.11.32 M.TYPE

This macro types a user-specified message and performs an optional read on the system console teletype.

Calling Sequence

M.TYPE *outaddr,outcnt* [, *inaddr*] [,*incnt*]

outaddr is the output message address

outcnt is the output transfer count

inaddr is the input message address

incnt is the input transfer count

1.11.33 M.USHUT

This macro is used to inhibit context switching of a user task. It should be used in user tasks which are eligible for IPU execution. See M.SHUT.

Calling Sequence

M.USHUT

1.11.34 M.XIR

This macro is called by the resident system module's initialization entry points right before they exit. It decrements the number of entry points in the calling module by one so the initialization entry point is no longer included, and returns to the SYSGEN processor.

Calling Sequence

M.XIR *addr*

addr is the address label of the module's HAT table

1.11.35 DCA.DATA

This macro is used within the SYSGEN entry point of F class handlers to reserve device context area (DCA) space for the number of DCAs specified by the repeat (REPT) count. During SYSGEN execution, one DCA is initialized for each unit definition table (UDT) entry containing the name of the handler. The unused DCAs and the code contained within the SYSGEN entry point are overlayed following execution.

Calling Sequence

DCA.DATA *sbuf* [,[*xwrds*] [,*time0,...,timeF*]]

sbuf	specifies the sense buffer size (bytes) for automatic sense retrieval by the extended I/O (XIO) common subroutines following an I/O error indication
xwrds	is the number of extra words to reserve for each DCA. If not specified, the standard DCA size is used.
time0-timeF	specifies the time-out value in seconds for each input/output control system (IOCS) opcode, hexadecimal 0 through F; for example, open, rewind, read, write, etc. If not specified or if zero is specified, no time out is associated with the I/O request.

1.11.36 DCA.INI1

This macro is used within the SYSGEN entry point of F-class handlers to initialize areas of the device context area (DCA), controller definition table (CDT) and unit definition table (UDT) associated with the particular handler. The code generated by this macro is overlayed following SYSGEN execution.

Calling Sequence

DCA.INI1 *hname* [,[OPIN] ,[IOQCDT] [,COM]]

hname	specifies the handler name, e.g., H.DCXIO for F class disk handlers
OPIN	specifies operator intervention is applicable for this handler
IOQCDT	specifies I/O queue entries are to be linked to the CDT. If not specified, I/O queue entries are linked to the UDT. Because many standard handlers assume I/O queue entries are linked a certain way, this parameter must be used with caution. This parameter is available to allow users flexibility when building handlers.
COM	specifies the handler interfaces with the XIO common subroutines

1.11.37 DCA.INI2

This macro is used within the SYSGEN entry point of F-class handlers to restore the working environment within the SYSGEN entry point following any user added executable code.

Calling Sequence

DCA.INI2

1.11.38 HMP.INIT

This macro initializes multiplexed I/O processor (MIOP) device handler parameters with entry point eight. The code generated by this macro is executed by SYSGEN and overlayed.

Calling Sequence

HMP.INIT *label*

label is the entry point truncated label; for example, AS0 for the asynchronous communications handler. This argument must be 3 ASCII characters. The first two represent the device mnemonic and the third is zero.

1.11.39 IB.INIT

This macro initializes multiplexed I/O processor (MIOP) device handler parameters with entry point 8, where R7 contains the controller definition table (CDT) address and R2 contains the address of the current context block.

Calling Sequence

IB.INIT

1.12 Extended MPX-32 Macros

The extended MPX-32 macros allow existing user modules and service routines to run in the extended mode. These macros select the appropriate coding, extended or nonextended, for a task by testing the state of the Macro Assembler option 16. (For example, if the MBR_DEF macro is specified, the coding for a DEF or SDEF directive is supplied depending on the state of option 16.)

The following are extended macros that directly replace the corresponding Macro Assembler directive:

Macro	Assembler Directive
MBR_BEQ	BEQ
MBR_BGE	BGE
MBR_BGT	BGT
MBR_BL	BL
MBR_BLE	BLE
MBR_BLT	BLT
MBR_BNE	BNE
MBR_BNS	BNS
MBR_BS	BS
MBR_DEF	DEF
MBR_EXT	EXT
MBR_TRSW	TRSW

For descriptions of these macros, see the corresponding Macro Assembler directive description in the MPX-32 Utilities Manual.

The following macros do not have corresponding Macro Assembler directives, and must be placed within the code that is to operate in the extended mode:

Macro	Description
MBR_DBG	calls the system debugger
MBR_DSCT	directs data into the DSECT
MBR_ENT	generates the adaptation sequence required to reference a routine from a nonextended module
MBR_INIT	tests the state of Macro Assembler option 16
MBR_SSCT	returns to a local code section in the system section (SSECT) area after an MBR_DSCT has been specified

1.12.1 MBR_DBG (Calls to System Debugger) Macro

The MBR_DBG macro calls the system debugger from the target extended module. This macro references the system debugger extended code entry within the adaptation code sequence.

Syntax

MBR_DBG <*symbol*>

1.12.2 MBR_DSCT (DSECT Data Separation) Macro

The MBR_DSCT macro specifies that the following code is data, and directs the data into the DSECT. All data and variable constants must have been separated for inclusion in the DSECT section.

Syntax

MBR_DSCT

1.12.3 MBR_ENT (Extended Code Routine Entry) Macro

The MBR_ENT macro generates the adaptative sequence required to reference a routine from a nonextended module. Each entry point must have an MBR_ENT macro before the first instruction.

Syntax

MBR_ENT <*symbol*> Replaces <*symbol*> EQU $

1.12.4 MBR_INIT (Module Initialization) Macro

The MBR_INIT macro tests the state of option 16. If option 16 is set, MBR_INIT initializes the code location to SSECT EXT_MPX. This macro is required after the program statement of an extended module.

Syntax

MBR_INIT

1.12.5 MBR_SSCT (System Code Separation) Macro

The MBR_SSCT macro specifies that the following code is executable data, and returns to a local code section in the system section (SSECT) area after an MBR_DSCT macro has been specified.

Syntax

MBR_SSCT

Usage:

```
        MBR_DSCT                        SEND DATA TO DATA SECTION
J.MOUNT DATAD   C'J. MOUNT      '       SYSTEM MOUNT TASK
OPCOM   DATAD   C'OPCOM         '       OPERATOR COMMUNICATIONS TASK
SYS.LFC DATAW   X'00AA532A      '     SYSTEM LFC '?T*'  (H.TAMM)
LFC3    DATAW   G'(3)           '     LFC FOR SYSTEM FCB3   (H.VOMM)
J.ATAPE DATAD   C'J.ATAPE       '     ANSI TAPE HANDLER TASK
        LPOOL
        MBR_SSCT                      RETURN TO CODE SECTION
```

2 System Tables and Variables

2.1 Overview

This chapter contains descriptions and format layouts for the tables and variables used by the MPX-32 operating system.

The MPX-32 table structure consists of the following categories:

Batch processing data area which contains the following:

- Job table
- Link file format (batch SLO and SBO)
- Run request format (J.SOEX)
- Spooled file directories

Executive (H.EXEC) data area which contains the following:

- Central Processing Unit (CPU)
- Dispatch Queue Entry (DQE)
- Dispatch Queue Address Table (DAT)

Input/output data area which contains the following:

- Blocking buffer control cells
- Controller Definition Table (CDT)
- Device Context Area (DCA)
- File Assignment Table (FAT)
- File Control Block (FCB)
- File Pointer Table (FPT)
- I/O Queue (IOQ) entry
- I/O table linkages
- Type Control Parameter Block (TCPB)
- Unit Definition Table (UDT)
- XIO Channel Definition Table (CHT)

Memory Management data area which contains the following:

- Map Image Descriptor List (T.MIDL)
- Memory Allocation Table (MATA)
- Memory Attribute List (T.MEML)
- Memory pool management
- Physical Shared Memory Table (PSM)
- Shared Memory Table (SMT)

Resource Management data area which contains the following:

- Allocated Resource Table (ART)
- Device Type Table (DTT)
- Mounted Volume Table (MVT)
- Resource Inquiry Table (M.RIQ)
- Resource Requirement Summary (RRS) entries
- Task Service Area (TSA)
- Volume Assignment Table (VAT)

Status Management data area which contains the following:

- Caller Notification Packet (CNP)

Terminal Services data area which contains the following:

- Terminal Line Buffer

Volume Management data area which contains the following:

- Bad Block Descriptor (M.BB.DEQ)
- Descriptor Allocation Map Descriptor (M.DM.DEQ)
- Descriptor Map (DMAP) Deallocation File Descriptor (M.BD.DEQ)
- Descriptors Descriptor (M.DD.DEQ)
- Directory Descriptor (M.DI.DEQ)
- Directory Entry Table (M.DN.TEQ)
- DQE Address Table (DAT)
- Memory Partition Descriptor (M.ME.DEQ)
- Resource Create Block (RCB)
- Resource Descriptor (M.RDCOM)
- Resource Descriptor Space Definition (M.RDSPD)
- Resource Logging Block (RLB)
- Space Allocation Map Descriptor (M.SM.DEQ)
- Space Map (SMAP) Deallocation File Descriptor (M.BS.DEQ)
- System Master Directory (SMD)
- Volume Descriptor (M.VO.DEQ)

Disk resident structures are:

- Volume format
- Load module structure
- Load module preamble
- Nonshared executable image structure
- Nonshared executable image preamble
- Shared executable image structure
- Shared executable image preamble
- Shared image descriptor

The resident system memory layout and utilization structure is described first.

The communications region is described next.

The table formats are then described, arranged in alphabetical order by the table name.

The disk resident resource descriptors are described in alphabetical order after the tables.

The disk resident structures are described last.

2.2 Memory Layout

Resident system memory layout and utilization structures are described below.

# Entries in Table	Table Address	CONCEPT/32
	0-1C	Not used
	20-60	IPU trap vectors
	64-7C	Not used
	80-FC	Trap vectors
	100-2FC	Interrupt vectors
	300-6FC	CPU scratchpad save area
	700-7FC	IOCD emulation area
	800 C.TABLES	Communication Region (C.)
C.TMCC	C.MATA	Memory Allocation Table (MEM.) 1 byte/map configured (word bound)
C.NITI	C.ITLT	Indirectly connected interrupt 24 words/entry (file bound)
	C.MPAA	Patch area user defined (word bound)
C.SMTN	C.SMTA	Shared Memory Table (SMT.) variable - C.SMTS contains the number of bytes/entry (file bound)
C.TENT	C.TTAB	Interrupt Timer Table (ITT.) 5 words/entry (word bound)
C.ARTN	C.ARTA	Allocated Resource Table (AR.) 8 words/entry (doubleword bound)
C.MVTN	C.MVTA	Mounted Volume Table (MV.) 40 words/entry (doubleword bound)
C.RMTM	C.RMTA	Resourcemark Table (RMT.) 1 byte/entry (word bound)
C.ACTN	C.ACTA	Activation Table 4 words/entry (doubleword bound)
C.SEQN	C.SEQA	Sequence Table 4 words/entry (doubleword bound)
C.NQUE	C.DQUE	Dispatch Queue (DQE.) 58 words/entry (file bound)
	C.ADAT+1W	DQE Address Table 1 word/DQE (word bound)
C.CDTN	C.CDTA	Controller Definition Table (CDT.) 24 words/entry (word bound)

C.UDTN	C.UDTA	Unit Definition Table (UDT.) 16 words/entry (word bound)
C.DTTN	C.DTTA	Device Type Table (DTT.) 2 words/entry (doubleword bound)
C.CHTN	C.CHTA	Channel Definition Table (CHT.) 40 words/entry (file bound)
	C.MPL	Master process list 2 words/entry (doubleword bound)
	C.MIDL	Map image list for operating system 1 halfword/operating system map (doubleword bound)
	C.SPAD	CPU scratchpad image 256 words (word bound)
C.SVTN	C.SVTA	SVC Type 1 Vector Table 128 words or user defined (word bound)
	C.SVTA2	SVC Type 2 Vector Table 128 words (word bound)
	C.MIOP	GPMC Jump Table 16 words (file bound)
C.MODN	C.MODD	Module Address Table 16 words (word bound)
Start of resident code. All programs are file bound.		Trap processors
		Interrupt processors
		I/O processors
		System modules: H.ALOC, H.BKDM, H.EXEC, H.FISE, H.IOCS, H.MEMM, H.MONS, H.MVMT, H.REMM, H.REXS, H.SOUT, H.TAMM, H.TSM, H.VOMM
		User operating system resident modules and tasks, if any
		System debugger (H.DBUG1)
		Swapper (H.SWAPR)
	C.SBUFA	IOQ memory pool (doubleword bound)
	C.SBUFB	MSG memory pool (doubleword bound)
	C.SBUF C.POOL	Memory pool area (doubleword bound, map block bound)
End of resident system		
	C.LOSEND	Logical end of the operating system + 1 byte
		User task space

2.3 Communications Region

The communications region is an area of main memory reserved for MPX-32 to store common data. This data is referenced by symbols that are equated to absolute memory locations. With each symbol is the length of the variable associated with the symbol. The length is in units, which is also the minimum boundary on which the variable resides.

Bit variables are contained in a set of contiguous words with the symbol C.BIT or C.BIT1 equated to the address of the first word. Bit variables are equated to bit positions relative to C.BIT or C.BIT1. Bit variables are referenced by a combination of the variable symbol and C.BIT or C.BIT1; for example, TBM C.AFLK,C.BIT.

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
512-513	800	C.DATE
514	808	C.CAL
515	80C	C.INTC
516-517	810	C.TIME
518-519	818	C.LODC
520-521	820	C.SYMTAB
522-523	828	C.PODC
524-525	830	C.SBUF
526-527	838	C.SIDV
528	840	C.TMAC | C.EMAC
529	844	C.HMAC | C.SMAC
530	848	C.TMCC | C.EMCC
531	84C	C.HMCC | C.SMCC
532-533	850	C.SYSTEM
534-537	858	C.SYPATH
538-539	868	C.PCHFLE
540-541	870	C.TRACE
542-543	878	C.DBGLM
544	880	C.SWPRD
545	884	C.SWPDEV
546	888	C.IREGS
547	88C	C.ITSAD
548	890	C.LOSEND
549	894	C.POSEND
550-553	898	C.HLPVOL
554-557	8A8	C.HLPDIR
558-560	8B8	C.CIPU
561-563	8C4	C.RIPU
564-566	8D0	C.FREE
567-569	8DC	C.PREA
570-572	8E8	C.CURR
573-575	8F4	C.SQRT
576-578	900	C.SQ55
579-581	90C	C.SQ56
582-584	918	C.SQ57
585-587	924	C.SQ58

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
588-590	930	C.SQ59
591-593	93C	C.SQ60
594-596	948	C.SQ61
597-599	954	C.SQ62
600-602	960	C.SQ63
603-605	96C	C.SQ64
606-608	978	C.SWTI
609-611	984	C.SWIO
612-614	990	C.SWSM
615-617	99C	C.SWSR
618-620	9A8	C.SWLO
621-623	9B4	C.SUSP
624-626	9C0	C.RUNW
627-629	9CC	C.HOLD
630-632	9D8	C.ANYW
633-635	9E4	C.SWDC
636-638	9F0	C.SWDV
639-641	9FC	C.SWFI
642-644	A08	C.MRQ
645-647	A14	C.SWMP
648-650	A20	C.SWGQ
651-671	A2C	C.SPCH
672	A80	C.TSAD
673	A84	C.ACTSEQ
674	A88	C.ADAT
675-676	A8C	C.BIT
677	A94	C.CDTA
678	A98	C.CPRI
679	A9C	C.DQUE
680	AA0	C.DTTA
681	AA4	C.FADR
682	AA8	C.FGONR
683	AAC	C.GINT
684	AB0	C.IDLA

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
685	AB4	C.IDLC
686	AB8	C.ITLT
687	ABC	C.BATSEQ
688	AC0	C.JOBN
689	AC4	C.MGRAN
690	AC8	C.MIDL
691	ACC	C.MIOP
692	AD0	C.MODD
693	AD4	C.MPL
694	AD8	C.MSD
695	ADC	C.MTIM
696	AE0	C.NTIM
697	AE4	C.PATCH
698	AE8	C.POOL
699	AEC	C.SGOS
700	AF0	C.SICTD
701	AF4	C.SMTA
702	AF8	C.ARTA
703	AFC	C.SPAD
704	B00	C.SVTA
705	B04	C.SVTA2
706	B08	C.SWAP
707	B0C	C.SYCS
708	B10	C.TSKN
709	B14	C.TSMDQA
710-717	B18	C.TTBT
718	B38	C.UDTA
719	B3C	C.TTAB
720	B40	C.MATA
721	B44	C.MPAA
722	B48	C.MPAC
723	B4C	C.MPAH
724	B50	C.RMTA
725	B54	C.EMTA
726	B58	C.REV
727	B5C	C.DEBUG

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
728	B60	C.TDQ1
729	B64	C.TDQ2
730	B68	C.TDQ3
731	B6C	C.REGS
732	B70	C.MVTA
733	B74	C.ACTA
734	B78	C.SEQA
735	B7C	C.SCDIPU
736	B80	C.CHTA
737	B84	C.ETLOC
738	B88	C.ADMASK
739	B8C	C.IDLA1
740	B90	C.IDLC1
741	B94	C.IPUIT1
742	B98	C.IPUIT2
743	B9C	C.BTIME
744	BA0	C.BDATE
745	BA4	C.TCORR
746	BA8	C.FSSP
747	BAC	C.DPTIMO
748	BB0	C.MDTA
749	BB4	C.MDTE
750	BB8	C.SWPLIM
751	BBC	C.PDQE
752-759	BC0	C.MPXBR
760	BE0	C.MPXBRD
761	BE4	C.IP00
762-763	BE8	C.PSDBRE
764-765	BF0	C.PSDBRX
766-767	BF8	C.PSDMSE
768-769	C00	C.PSDMSX
770-771	C08	C.PSDEAE
772-773	C10	C.PSDEAX
774	C18	C.DSECT
775	C1C	C.ADAPT
776	C20	C.TDEFA

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
777	C24	C.SWIOCL			
778	C28	C.CRDUMP			
779	C2C	C.HSTADR			
780	C30	C.CDTN		C.ITRS	
781	C34	C.SMVTI		C.SVTN	
782	C38	C.UDTN		C.RMTM	
783	C3C	C.EMTM		C.NOS	
784	C40	C.NRST		C.MVTN	
785	C44	C.ARTN		C.CHTN	
786	C48	C.HIMAP		C.SVTN2	
787	C4C	C.MDTN		C.MDTAV	
788-789	C50	C.RMS			
790	C58	C.GSLEMC			
791	C5C	C.PTRINT			
792-793	C60	C.BDBUG			
794	C68	C.PET			
795-797	C6C-C74	C.TSMIR			
798-800	C78-C80	C.TSMIA			
801-802	C84-C88	C.SBUFA			
803-804	C8C-C90	C.SBUFB			
805	C94	C.IPUAE			
806	C98	C.HLPDQA			
807	C9C	C.NODEID			
808	CA0	C.PSMA			
809	CA4	C.RMSS			
810	CA8	C.RCASIZ		C.PSMSIZ	
811	CAC	C.DPTRY		C.SMAPS	
812	CB0	C.BPRI	C.DTTN	C.FSFLGS	C.MODN
813	CB4	C.NITI	C.NQUE	C.RRUN	C.SMTN
814	CB8	C.TSMCNT	C.TSMPRI	C.TSMTOT	C.TENT
815	CBC	C.RMTL	C.EMTL	C.CONF	C.MACH
816	CC0	C.ACTN	C.DBTLC	C.SMTS	C.SEQN
817	CC4	C.IPUHIS			
818	CC8	C.SHRHI (C.UNCAHI)		C.SHRLO (C.UNCALO)	
819	CCC	C.MAXSWP	C.NLOAD2	C.PDPEND	C.PSMN
820	CD0	C.CDTSIZ	C.DFTSIZ	C.DTTSIZ	C.FPTSIZ

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
821	CD4	C.IOQSIZ	C.DPGPRI	C.NFRAME	C.MRQLEN
822	CD8	C.MMSG	C.MRUN	C.MNWI	C.GSLEGI
823	CDC	C.GSLEPR	C.ADAFL	C.TKILL	C.DELTA
824	CE0	C.MPXBRN	C.DBMAPS	C.SWAPSZ	
825	CE4	C.DTSAVE			
826	CE8	C.SHCPU			
827	CEC	C.SHIPU			
828	CF0	C.UPDT			
829	CF4	C.SWPBUF			
830	CF8	C.MRQTMR			
831	CFC	C.SHBTH			
832	D00	C.TABLES			
833-864	D04-D80	C.USER			
865	D84	C.USERVA			
866	D88	C.TPVA			
867	D8C	C.EXEND		C.FRAME	
868	D90	C.SCOFDQ			
869	D94	C.CTSAD			
870	D98	C.AGE			
871	D9C	C.EFRPG			
872	DA0	C.HFRPG			
873	DA4	C.SFRPG			
874	DA8	C.DFRPG			
875	DAC	C.MPFRPG			
876	DB0	C.MPTLA			
877	DB4	C.CREGS			
878	DB8	C.PTEA			
879	DBC	C.PSTA			
880	DC0	C.CHTSIZ		C.DQESIZ	
881	DC4	C.RDSIZ		C.MPMAC	
882	DC8	C.BEGPGO		C.ENDPGO	
883	DCC	C.DMCC		C.DMAC	
884-885	DD0-DD4	C.BIT1			
886-887	DD8-DDC	C.SPGOL			
888-889	DE0-DE4	C.COMM			

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
890-946	DE8-ED4	Reserved for MPX-32			
947	ED8	Reserved		C.TPGOC	
948	EDC	C.MVTSIZ		C.DCASIZ	
949	EE0	C.RCBSIZ	C.UDTSIZ	C.DPTSIZ	C.TIQSIZ
950	EE4	C.MRQSIZ	Reserved	C.PCBSIZ	C.ARTSIZ
951	EE8	C.DETSIZ	C.BBFSIZ	C.VATSIZ	
952-956	EEC-EFC	Reserved for MPX-32			

Byte (Hex)	Symbol	Description
800	C.DATE	current date (Gregorian) as input by operator
808	C.CAL	calendar devices:

Byte	Description
0	current century in binary (C.CENT)
1	current year in binary (C.YEAR)
2	current month in binary (C.MONTH)
3	current day in binary (C.DAY)

80C	C.INTC	interrupt counter (number of interrupts from zero which is midnight) used for time-of-day calculations
810	C.TIME	the system start-up values from C.BTIME and C.BDATE
818	C.LODC	the system listed output device used as a default in operator communications commands:

Byte	Description
0-1	ASCII device type code
2-3	ASCII channel number
4-5	ASCII subaddress
6-7	reserved

820	C.SYMTAB	name of the symbol table file
828	C.PODC	the system punched output device used as a default in operator communications commands:

Byte	Description
0-1	ASCII device type code
2-3	ASCII channel number
4-5	ASCII subaddress
6-7	reserved

830	C.SBUF	first word contains address of memory pool. Second word is set by S.MEMM9 to the number of words in memory pool.

Byte (Hex)	Symbol	Description
838	C.SIDV	the system input device used as a default in operator communications commands: Byte — Description 0-1 ASCII device type code 2-3 ASCII channel number 4-5 ASCII subaddress 6-7 reserved
840	C.TMAC	total count in halfwords of all E, H, and S memory modules available
842	C.EMAC	total count of valid E type memory modules available
844	C.HMAC	total count of valid H type memory modules available
846	C.SMAC	total count of valid S type memory modules available
848	C.TMCC	total count of all valid E, H, and S memory modules configured (less the size of the unmapped portion of the system debugger if it is present, and any maps used for static partitions or extended MPX-32)
84A	C.EMCC	total count of valid E type memory modules configured (minus one if swap device is E-class and extended memory is present in the system, minus any E-class map blocks allocated for the unmapped portion of the system debugger, and any E class map blocks allocated for static partitions or extended MPX-32)
84C	C.HMCC	total count of valid H type memory modules configured (minus any H-class map blocks allocated for the unmapped portion of the system debugger and any H class map blocks allocated for static partitions or extended MPX-32)
84E	C.SMCC	total count of valid S type memory modules configured (minus any S-class map blocks allocated for the unmapped portion of the system debugger and any S class map blocks allocated for static partitions or extended MPX-32)

Byte (Hex)	Symbol	Description
850	C.SYSTEM	name of current system image
858	C.SYPATH	system pathname prototype
868	C.PCHFLE	patch file name
870	C.TRACE	system trace (M.TRAC) control word
878	C.DBGLM	debugger load module name
880	C.SWPRD	absolute block number of the resource descriptor of the swap file
884	C.SWPDEV	MVTE address of the actual swap volume
888	C.IREGS	logical address of the start of register save area in the TSA for task running in IPU only
88C	C.ITSAD	logical address of the TSA for the task running in the IPU only
890	C.LOSEND	logical end of MPX-32 (+1B) for the current task
894	C.POSEND	physical end of non-split MPX-32 (+1B)
898	C.HLPVOL	name of the volume used to store the HELP files
8A8	C.HLPDIR	name of the directory used to store the HELP files

Byte (Hex)	Symbol	Description
8B8	C.CIPU	standard format linked list head cell for all IPU tasks ineligible for CPU control, waiting in general queue. C.CIPU is the first of a set of communications region variables which are contiguous in memory. These variables, listed in the order that they appear in memory, are as follows: C.CIPU C.RIPU C.FREE C.PREA C.CURR C.SQRT C.SQ55 C.SQ56 C.SQ57 C.SQ58 C.SQ59 C.SQ60 C.SQ61 C.SQ62 C.SQ63 C.SQ64 C.SWTI C.SWIO C.SWSM C.SWSR C.SWLO C.SUSP C.RUNW C.HOLD C.ANYW C.SWDC C.SWDV C.SWFI C.MRQ C.SWMP C.SWGQ C.SPCH
8C4	C.RIPU	standard format linked list head cell for all IPU tasks ready to run, waiting in general queue
8D0	C.FREE	standard format linked list head cell for free entries in the CPU dispatch queue
8DC	C.PREA	standard format linked list head cell for CPU dispatch queue entries that are in the preactivation state

Byte (Hex)	Symbol	Description
8E8	C.CURR	standard format linked list head cell for the CPU dispatch queue entry of the currently executing task. This list can have a maximum of two entries: one for the current real-time task (if any) and one for the current time-distribution task (if any).
8F4	C.SQRT	standard format linked list head cell for the list of ready-to-run real-time (priority level 1 to 54) tasks
900	C.SQ55	standard format linked list head cell for the list of ready-to-run priority level 55 time-distribution tasks
90C	C.SQ56	standard format linked list head cell for the list of ready-to-run priority level 56 time-distribution tasks
918	C.SQ57	standard format linked list head cell for the list of ready-to-run priority level 57 time-distribution tasks
924	C.SQ58	standard format linked list head cell for the list of ready-to-run priority level 58 time-distribution tasks
930	C.SQ59	standard format linked list head cell for the list of ready-to-run priority level 59 time-distribution tasks
93C	C.SQ60	standard format linked list head cell for the list of ready-to-run priority level 60 time-distribution tasks
948	C.SQ61	standard format linked list head cell for the list of ready-to-run priority level 61 time-distribution tasks
954	C.SQ62	standard format linked list head cell for the list of ready-to-run priority level 62 time-distribution tasks
960	C.SQ63	standard format linked list head cell for the list of ready-to-run priority level 63 time-distribution tasks
96C	C.SQ64	standard format linked list head cell for the list of ready-to-run priority level 64 time-distribution tasks
978	C.SWTI	standard format linked list head cell for all tasks waiting for the completion of wait mode interactive (terminal) input
984	C.SWIO	standard format linked list head cell for all tasks waiting for the completion of wait mode I/O requests
990	C.SWSM	standard format linked list head cell for all tasks waiting for the completion of wait mode send message request
99C	C.SWSR	standard format linked list head cell for all tasks waiting for the completion of wait mode send run request
9A8	C.SWLO	standard format linked list head cell for all tasks waiting for the completion of low speed output

Byte (Hex)	Symbol	Description
9B4	C.SUSP	standard format linked list head cell for all tasks that are in an execution suspend mode, waiting for a message interrupt, a timer expiration, or a resume task request
9C0	C.RUNW	standard format linked list head cell for all tasks that are ineligible for CPU control, waiting for a run request to be received, or for the expiration of a timer
9CC	C.HOLD	standard format linked list head cell for all tasks that are ineligible for CPU control, waiting for a continue request to be received
9D8	C.ANYW	standard format linked list head cell for all tasks that are ineligible for CPU control, waiting for the completion of any no-wait mode I/O request, any no-wait mode send message request, any no-wait mode send run request, or any message or break interrupt
9E4	C.SWDC	standard format linked list head cell for all tasks ineligible for CPU control, waiting for disk space to become available
9F0	C.SWDV	standard format linked list head cell for all tasks ineligible for CPU control, waiting for a peripheral device to become available
9FC	C.SWFI	reserved
A08	C.MRQ	standard format linked list head cell for all tasks ineligible for CPU control, waiting for memory to become available
A14	C.SWMP	standard format linked list head cell for all tasks ineligible for CPU control, waiting for memory pool to become available
A20	C.SWGQ	standard format linked list head cell for all tasks ineligible for CPU control, waiting in general queue
A2C	C.SPCH	reserved
A80	C.TSAD	address of the TSA for a CPU or IPU task running in mapped in mode with its TSA at MINADDR (for compatibility only)
A84	C.ACTSEQ	running count of task activations, used to form right-most 24 bits of task number when a task is activated. SYSGEN initializes this word to zero
A88	C.ADAT	address of the DQE address table (DAT)

Byte (Hex)	Symbol	Description
A8C	C.BIT	symbol associated with the beginning of the bit variables:

Bit	Meaning if Set
0	accounting file lock indicator (C.AFLK)
1	swap volume is E-class disk (C.ESWAP)
2	dump real-time tasks on abort (C.FGPM)
3	indicates user is using CPU scratchpad for his own needs. IPL alters SPAD locations defined by SYSGEN. Reset indicates SPAD locations not defined by SYSGEN are to be set to zero. (C.SPADOK)
4	list patches indicator (C.LSPT)
5	online restart in progress (C.RSTRT)
6	reserved
7	continuous batch mode indicator (C.SCBT)
8	J.SOUT banner page inhibit (C.SIBP)
9	SSIN device density is 556 (7-track) (C.SIDD)
10	SSIN device parity is odd (7-track) (C.SIDP)
11	inhibit context switching in IPU (C.ICSIPU)
12	task context switch inhibited (C.CSWI)
13	activation from tape (C.TAPACT)
14	static IOQ indicator (C.PIOQ)
15	reserved
16	inhibit magnetic tape mount message (C.SIMM)
17	memory error detected by H.IP02 (C.MERR1)
18	memory parity error detected during memory initialization (C.MERR2)
19	nonpresent memory detected (C.MERR3)
20	module cannot be loaded (C.NOLOAD)
21	SYSINIT active - IPL or restart (C.SYSB)
22	IPU is offline (C.IPUOFF)
23	IPU accounting timer present (C.IPUIT)
24	inhibit operator intervention (C.NOP)
25	reserved for RJE
26	reserved
27	shadow memory configuration error (C.SMERR)
28	reserved
29	dual-port disk mounted (C.DPMT)
30	activating tasks specified in the SYSGEN SEQUENCE directive (C.SEQUEN)
31	reserved for ICS (C.ICS)
32	reserved
33	exclusive ANSI tape drive is configured (C.ANSI)
34	Development System (C.DEV)
35	group swap limit exceeded (C.GSLE)

Byte (Hex)	Symbol	Description Bit	Meaning if Set
		36	no terminal definition (C.NOTDEF)
		37	no ANSI (C.NOANSI)
		38	H.PTRACE is present (C.PTRACE)
		39	returns to implicit physical mount functionality of MPX-32 Revision 3.3 (C.CMIMM)
		40	disables public volume dismounts (C.CMPMM)
		41	real-time accounting disabled (C.RTACC)
		42	H.IPCL needs to send break to J.TSM (C.TSMLC1)
		43	reserved
		44	only the system administrator can execute the PASSWORD task (C.SAPSWD)
		45	passwords are required (C.PASSWD)
		46	SYSTEM is the only valid ownername if no M.KEY file (C.SAONLY)
		47	no rollover allowed from MSGPOOL (C.MSGNR)
		48	no rollover allowed from IOQPOOL (C.PIONR)
		49	rollover occurred from IOQPOOL (C.ROLIOQ)
		50	rollover occurred from MSGPOOL (C.ROLMSG)
		51	inhibit write to system volume descriptor during shutdown process (C.IWSYSV)
		52	inhibit volume mounts during shutdown process (C.IVM)
		53	reserved
		54	system volume is quiescent (C.SQUIET)
		55	read and lock specified (C.RLWU)
		56	image down loaded for host (C.IMAGDL)
		57	remote task activation allowed (C.REMTSK)
		58	all configured memory is physically shadow memory (C.ALLSHD)
		59	allow multiple logins using same owner name (C.MLOGIN)
		60	no system volume (C.NOSVOL)
		61	mapped out image (C.MAPOUT)
		62	inhibit echoing of owner name at logon (C.NOECHO)
		63	system default move TSA (C.TSA)

Byte (Hex)	Symbol	Description
A94	C.CDTA	address of controller definition table
A98	C.CPRI	task execution bytes:

Byte	Description
0	current execution priority of currently executing task (C.CUP)
1	base execution priority of currently executing task (C.BUP)
2	I/O priority of currently executing task (C.IOP)
3	state chain index of currently executing task (C.US)

A9C	C.DQUE	address of CPU dispatch queue area. The CPU dispatch queue area is a variable length table built by SYSGEN. It contains the number of 64-word dispatch queue entries (DQEs) specified at system generation time.
AA0	C.DTTA	address of device type table
AA4	C.FADR	reserved
AA8	C.FGONR	reserved
AAC	C.GINT	contains the count of all outstanding interrupts and traps (except SVC). It is incremented as the first instruction of every interrupt or trap service routine, and decremented by S.EXEC5, the standard interrupt and trap exit routine.
AB0	C.IDLA	CPU idle time accumulation value in seconds, cleared by SYSGEN. This value is incremented when the countdown value in C.IDLC expires.
AB4	C.IDLC	CPU idle time countdown value, cleared by SYSGEN. This value is used to load the interval timer when there are no tasks ready to run. When a task becomes ready to run, the interval timer is read and the value is stored in this word.
AB8	C.ITLT	address of indirectly connected task linkage table (ITLT). Initialized by SYSGEN.
ABC	C.BATSEQ	next batch sequence number
AC0	C.JOBN	maximum number of concurrent batch jobs
AC4	C.MGRAN	machine dependent map granularity

Byte (Hex)	Symbol	Description
AC8	C.MIDL	address of the list of map registers used by the operating system
ACC	C.MIOP	address of first entry of MIOP jump table
AD0	C.MODD	address of variable length module address table. Initialized by SYSGEN. The module address table contains entries in module sequence. Each entry consists of one word that contains the address of the entry point transfer list (HAT) of the associates module.
AD4	C.MPL	address of master process list. Length of list in words is contained in C.NDQE plus one word. First entry points to C.MSD (hardware requirement).
AD8	C.MSD	contains map segment descriptor for operating system (BPIX). It points to C.MIDL (hardware requirement).
ADC	C.MTIM	number of clock interrupts per second. Initialized by SYSGEN.
AE0	C.NTIM	number of clock interrupts per time unit. Initialized by SYSGEN.
AE4	C.PATCH	system debug patch area
AE8	C.POOL	address of memory pool
AEC	C.SGOS	contains the default SGO size of 32 blocks. This is included for compatibility purposes only and is not examined during job processing.
AF0	C.SICTD	address of MIOP test device status processor, H.SICTD.
AF4	C.SMTA	address of shared memory table area. Size is determined by SYSGEN SHARE directive.
AF8	C.ARTA	address of allocated resource table
AFC	C.SPAD	address of CPU scratchpad image
B00	C.SVTA	address of variable length SVC '1' table. Initialized by SYSGEN. Each entry consists of one word which contains the address of the service associated with the SVC number.
B04	C.SVTA2	address of variable length SVC '2' table. Initialized by SYSGEN. Each entry consists of one word which contains the address of the service associated with the SVC number.

Byte (Hex)	Symbol	Description
B08	C.SWAP	contains the swapper's status and DQE address. (If bit 0 equals zero, the swapper is active. If bit 0 equals one, the swapper is inactive). Bits 8 through 31 contain the address of the swapper's DQE.
B0C	C.SYCS	contains the default SYC size of 32 blocks. This is included for compatibility only and is not examined during job processing.
B10	C.TSKN	task activation sequence number of currently executing task

Byte	Description
0	contains the DQE entry number of the currently executing task in the range of 1 to 255; when word format is adjusted, it may be used as an index to the DQE address table (DAT) to obtain the DQE for the associated task. The address of the DAT is contained in C.ADAT. (C.PRNO)
1-3	activation sequence number of currently executing task

B14	C.TSMDQA	address of DQE for J.TSM or 0 if J.TSM has exited. Required for ring processing and message sending.
B18	C.TTBT	task timer bit table containing 256 bits. Each bit corresponds to a C.DQE entry and is accessed by the DQE entry number (1 to 255). A bit set in this table indicates the associated DQE has an active task timer.
B38	C.UDTA	address of unit definition table
B3C	C.TTAB	address of timer table
B40	C.MATA	address of memory tables
B44	C.MPAA	low address of the patch area
B48	C.MPAC	current address of the patch area
B4C	C.MPAH	high address of the patch area
B50	C.RMTA	address of resourcemark table
B54	C.EMTA	address of eventmark table
B58	C.REV	MPX-32 revision and interim revision
B5C	C.DEBUG	address location of debugger

Byte (Hex)	Symbol	Description
B60	C.TDQ1	time-distribution quantum stage one, in interval timer units. Initialized by SYSGEN. This value is used to load the interval timer when CPU control is dispatched to a time-distribution task under one of the following conditions: • a task is initially selected after activation • a task is initially selected after the termination of a voluntary wait state (e.g., wait I/O or timed suspend) • a task is initially selected after in-swap • a task is reselected after completion of its full quantum During the quantum stage one interval, the currently executing task is not eligible for out-swap, and may not be pre-empted from CPU control by a higher priority time-distribution task.
B64	C.TDQ2	time-distribution quantum stage two, in interval timer units. Initialized by SYSGEN. This value is used to load the interval timer when the stage one quantum for the currently executing task expires. (The quantum stage two value may be added to the quantum stage one value to define the full task quantum.)
B68	C.TDQ3	time-distribution full quantum value, in interval timer units. Initialized by SYSGEN. This value is the sum of the quantum stage one and stage two values.
B6C	C.REGS	TSA address of current task in the CPU (for compatibility only)
B70	C.MVTA	address of mounted volume table
B74	C.ACTA	address of activation table
B78	C.SEQA	address of sequence table
B7C	C.SCDIPU	schedule IPU routine address
B80	C.CHTA	address of channel definition table
B84	C.ETLOC	address of event trace logic
B88	C.ADMASK	maximum address bit mask for machine
B8C	C.IDLA1	IPU idle time accumulation value in seconds, cleared by SYSGEN. This value is incremented when the countdown value in C.IDLC1 expires.

Byte (Hex)	Symbol	Description
B90	C.IDLC1	IPU idle time countdown value, cleared by SYSGEN. This value is used to load IPU accounting interval timer (if present) when there are no tasks ready to run on the IPU. When a task becomes ready to run, the IPU accounting interval timer is read and the value is stored in this word.
B94	C.IPUIT1	address of the IPU accounting routine, S.IPUIT1, which performs accounting functions after an IPU trap is fielded. Initialized by SYSGEN.
B98	C.IPUIT2	address of the IPU accounting routine, S.IPUIT2, which performs accounting functions prior to the starting of the IPU. Initialized by SYSGEN.
B9C	C.BTIME	the current time (in binary) kept as the number of 100 microsecond units
BA0	C.BDATE	the current date (in binary) kept as the number of days since January 1, 1960
BA4	C.TCORR	the correction factor (in 100 microsecond units) which must be subtracted from C.BTIME to get the correct local time. This value is determined by the daylight savings and time zone parameters specified (if any) at IPL.
BA8	C.FSSP	file system stack frame pointer
BAC	C.DPTIMO	default time-out value applied to dual-processor, shared volume resource assignments
BB0	C.MDTA	physical starting address of the memory resident descriptor table (MDT)
BB4	C.MDTE	physical ending address of the MDT
BB8	C.SWPLIM	minimum number of maps to be swapped at any one time
BBC	C.PDQE	address of DQE of a partially swapped task
BC0	C.MPXBR	base registers save area (eight words--one file each)
BE0	C.MPXBRD	default logical map address
BE4	C.IP00	address of the A.IP00 module

Byte (Hex)	Symbol	Description
BE8	C.PSDBRE	break entered PSD for base mode task (two words)
BF0	C.PSDBRX	break exited PSD for base mode task (two words)
BF8	C.PSDMSE	message entered PSD for base mode task (two words)
C00	C.PSDMSX	message exited PSD for base mode task (two words)
C08	C.PSDEAE	end action entered PSD for base mode task (two words)
C10	C.PSDEAX	end action exited PSD for base mode task (two words)
C18	C.DSECT	start address of DSECT for extended MPX-32
C1C	C.ADAPT	start address of the adapter code region for extended MPX-32
C20	C.TDEFA	address of TERMPART, if present
C24	C.SWIOCL	swapper's IOCL address
C28	C.CRDUMP	address of the crash dump routine
C2C	C.HSTADR	address of the optional CPU/IPU state chain history buffer
C30	C.CDTN	number of entries in controller definition table
C32	C.ITRS	interval timer resolution, in tenths of microseconds, as derived from the SYSGEN ITIM directive
C34	C.SMVTI	mounted volume table (MVT) index of swap device
C36	C.SVTN	number of entries in the SVC '1' table. Initialized by SYSGEN.
C38	C.UDTN	number of entries in unit definition table
C3A	C.RMTM	maximum number of resourcemarks
C3C	C.EMTM	maximum number of eventmarks
C3E	C.NOS	number of blocks required for SYSGEN code
C40	C.NRST	number of blocks required for restart code
C42	C.MVTN	number of entries in mounted volume table
C44	C.ARTN	number of entries in allocated resource table
C46	C.CHTN	number of entries in channel definition table

Byte (Hex)	Symbol	Description
C48	C.HIMAP	number of the last map block of logical address space available to a task
C4A	C.SVTN2	number of entries in the SVC '2' table. Initialized by SYSGEN.
C4C	C.MDTN	total hexadecimal number of entries in the MDT. This number is larger than the number specified at SYSGEN time because it includes an extra 25% for collision resolution.
C4E	C.MDTAV	hexadecimal number of entries currently available in the MDT
C50	C.RMS	reserved for RMS (two words)
C58	C.GSLEMC	group swap limit exceeded map count
C5C	C.PTRINT	PTRACE task activation control address
C60	C.BDBUG	base task debugger name
C68	C.PET	PET patch address table pointer
C6C-C74	C.TSMIR	TSM input request head cell
C78-C80	C.TSMIA	TSM input active head cell
C84-C88	C.SBUFA	first word contains the address of the IOQ memory pool. Second word is set by S.MEMM9A to the number of words in the IOQ memory pool
C8C-C90	C.SBUFB	first word contains the address of the MSG memory pool. Second word is set by S.MEMM9B to the number of words in the MSG memory pool
C94	C.IPUAE	address of the arithmetic exception handler
C98	C.HLPDQA	address of DQE for J.HLP. Required for J.TSM
C9C	C.NODEID	RMSS node identifier
CA0	C.PSMA	address of PSM table
CA4	C.RMSS	reserved for RMSS
CA8	C.RCASIZ	size of remote context area
CAA	C.PSMSIZ	number of context areas
CAC	C.DPTRY	decimal number of tries to access a dual-port resource
CAE	C.SMAPS	number of MAPS used by the system (i.e., J.SWAPR)
CB0	C.BPRI	default software priority level at which batch jobs execute

Byte (Hex)	Symbol	Description
CB1	C.DTTN	number of entries in device type table
CB2	C.FSFLGS	reserved
CB3	C.MODN	entry number of last entry in module address table. Initialized by SYSGEN.
CB4	C.NITI	contains the number of 24-word indirectly connected task linkage block (ITLB) entries in the indirectly connected task linkage table (ITLT). Initialized by SYSGEN.
CB5	C.NQUE	number of entries (255 maximum) in CPU dispatch queue.
CB6	C.RRUN	contains the count of memory release events. It is incremented by H.EXEC,9 when a memory scheduler event is reported. It is cleared by the memory scheduler (swapper) when processing of the memory request queue begins. It is decremented by the swapper when memory is deallocated by the swapper. It is cleared by the swapper before H.EXEC,8 is called. H.EXEC,8 will rerun the swapper if C.RRUN is not equal to zero.
CB7	C.SMTN	number of entries in shared memory table
CB8	C.TSMCNT	number of currently active TSM devices. Maintained by J.TSM.
CB9	C.TSMPRI	priority default for TSM-activated tasks. Overrides cataloged priority.
CBA	C.TSMTOT	number of TSM devices. Initialized by entry point eight of all terminal device handlers.
CBB	C.TENT	number of timer table entries
CBC	C.RMTL	low address of user resourcemark area
CBD	C.EMTL	low address of eventmark area
CBE	C.CONF	configuration flags:

Bit	Meaning if Set
0	CPU accelerator present (C.CPUACC)
1	IPU accelerator present (C.IPUACC)
2	IPU present (C.IPU)
3	memory-only system (not valid in Revision 2.*x* series) (C.MEMNLY)
4	base code removed (C.NOBASE)
5	Ada support module present (C.ADA)
6	shadow memory configured (C.SHMEM)
7	shadow memory reserved (C.SHRSV)

Byte (Hex)	Symbol	Description
CBF	C.MACH	machine type currently in use:

Value	Description
0	CONCEPT 32/2000
1	reserved
2	CONCEPT 32/27
3	CONCEPT 32/67
4	CONCEPT 32/87
5	CONCEPT 32/97
6-7	reserved

CC0	C.ACTN	number of entries in activation table
CC1	C.DBTLC	channel address used for system debugger
CC2	C.SMTS	shared memory table entry size in bytes
CC3	C.SEQN	number of entries in sequence table
CC4	C.IPUHIS	address of IPU history buffer
CC8	C.SHRHI	interprocessor memory high bound
CCA	C.SHRLO	interprocessor memory low bound
CCC	C.MAXSWP	maximum swap size in megabytes
CCD	C.NLOAD2	SYSGEN error code
CCE	C.PDPEND	number of public dismounts pending
CCF	C.PSMN	number of entries in PSM table
CD0	C.CDTSIZ	size of controller definition table in bytes
CD1	C.DFTSIZ	size of disk file assignment table in bytes
CD2	C.DTTSIZ	size of device type table in bytes
CD3	C.FPTSIZ	size of file pointer table in bytes
CD4	C.IOQSIZ	size of I/O queue entry table in bytes
CD5	C.DPGPRI	demand page base priority
CD6	C.NFRAME	number of frames in TSA register stack
CD7	C.MRQLEN	length in bytes of the MRRQ fixed header area
CD8	C.MMSG	nonprivileged task's no-wait message count
CD9	C.MRUN	nonprivileged task's no-wait run request count
CDA	C.MNWI	nonprivileged task's no-wait I/O count
CDB	C.GSLEGI	group ID of a task whose group outswap limits have been exceeded

Byte (Hex)	Symbol	Description
CDC	C.GSLEPR	hexadecimal priority of a task whose group outswap limits have been exceeded
CDD	C.ADAFL	control flag for Ada run-time system
CDE	C.TKILL	number of seconds before an abort becomes a kill
CDF	C.DELTA	delta value for real-time IPU tasks
CE0	C.MPXBRN	number of base registers to load
CE1	C.DBMAPS	number of MAPS used by the system debugger
CE2	C.SWAPSZ	swapfile size in megabytes (halfword)
CE4	C.DTSAVE	elapsed time before J.DTSAVE resumes
CE8	C.SHCPU	CPU shadow memory. Starting map block number in first halfword. Number of map blocks in second halfword. These fields include any C.SHBTH area.
CEC	C.SHIPU	IPU shadow memory. Starting map block number in first halfword. Number of map blocks in second halfword. These fields include any C.SHBTH area.
CF0	C.UPDT	MPX-32 patch or replacement release
CF4	C.SWPBUF	logical address of J.SWAPR's dedicated system buffer
CF8	C.MRQTMR	timer used by J.SWAPR
CFC	C.SHBTH	shadow memory in both CPU and IPU. Starting map block number in first halfword. Number of map blocks in second halfword.
D00	C.TABLES	symbol equated to the absolute memory location at which SYSGEN-built tables begin if no user communication region is SYSGENed. This location is on a word boundary.
D04	C.USER	symbol equated to the absolute memory location where the fixed portion of the user communication region begins.
D84	C.USERVA	address of the variable size user region. This region is defined by the CDOTS directive to SYSGEN and is on a word boundary. If the CDOTS directive is specified, the SYSGEN-built tables begin at the first word location following this user region.
D88	C.TPVA	address of the 16 word area reserved for third party vendor products
D8C	C.EXEND	address of the end of executable memory
D8E	C.FRAME	size of each stack frame in TSA non-base register stack push-down area
D90	C.SCOFDQ	CPU/IPU scratchpad offset for task DQE address

Byte (Hex)	Symbol	Description
D94	C.CTSAD	logical address of TSA for the current task running in the CPU
D98	C.AGE	virtual time before page considered aged
D9C	C.EFRPG	free page head cell for E class
DA0	C.HFRPG	free page head cell for H class
DA4	C.SFRPG	free page head cell for S class
DA8	C.DFRPG	free page head cell for D class
DAC	C.MPFRPG	free page head cell for multi processer memory
DB0	C.MPTLA	physical memory allocation pointer list address
DB4	C.CREGS	logical address of the TSA stack for current CPU task
DB8	C.PTEA	physical page table entry address
DBC	C.PSTA	physical page state table address
DC0	C.CHTSIZ	size of channel definition table in bytes
DC2	C.DQESIZ	size of dispatch queue entry in bytes
DC4	C.RDSIZ	size of resouce descripter in bytes
DC6	C.MPMAC	total valid configured multiprocessor memory
DC8	C.BEGPGO	number of mapblocks to page out
DCA	C.ENDPGO	number of mapblocks to stop page out
DCC	C.DMCC	total configured DRAM (CONCEPT 32/2000)
DCE	C.DMAC	total available DRAM (CONCEPT 32/2000)
DD0-DD4	C.BIT1	symbol associated with the beginning of the bit variables:

Bit	Meaning if Set
0	inhibit batch messages to all terminals except system console (C.TERM)
1	inhibit batch messages to system console (C.CONS)
2	restrict ownername "SYSTEM" from multiple logons (C.NOSYS)
3	SYSGEN request to run all SYSMAP tasks with MPX-32 mapped out (C.TSKOUT)
4	system specification for task (C.TSKOFL)
5	no last access update (C.NOLACC)
6	image supports demand page (C.DPGSYS)
7	TSM will exit when not in use (C.TSMXIT)
8	TSM reactivated (C.RACTSM)
9	activate TSM (C.ACTSM)
10	page out to swap file in progress (C.PGOPRG)
11	high priority task is ready to run (C.HIPRI)
12-63	reserved

Byte (Hex)	Symbol	Description
DD8-DDC	C.SPGOL	shared page out queue head cell string forward address (MAP.SF); 1 HW string backward address (MAP.SB); 1 HW number of pages in queue (SPGO.CNT); 1 HW reserved; 1HW
DE0-DE4	C.COMM	reserved for COMM-32
DE8-ED8		reserved for MPX-32
EDA	C.TPGOC	total number of pages in the system ready for pageout
EDC	C.MVTSIZ	size of mounted volume table in bytes
EDE	C.DCASIZ	size of device context area table in bytes
EE0	C.RCBSIZ	size of resource create block table in bytes
EE1	C.UDTSIZ	size of unit definition table in bytes
EE2	C.DPTSIZ	size of disk parameter table in bytes
EE3	C.TIQSIZ	size of terminal input queue table in bytes
EE4	C.MRQSIZ	size of message or run request queue table in bytes
EE5		reserved for MPX-32
EE6	C.PCBSIZ	size of procedure call block in bytes
EE7	C.ARTSIZ	size of allocated resource table in bytes
EE8	C.DETSIZ	size of directory entry table in bytes
EE9	C.BBHSIZ	size of blocking buffer head cell size in bytes
EEA	C.VATSIZ	size of volume assignment table in bytes
EEB-EFC		reserved for MPX-32

2.4 Allocated Resource Table (ART)

The allocated resource table (ART) is a system resident structure that provides a central mechanism to control the manipulation of all allocated resources. An entry is made for a resource when it is allocated, and remains while there are active assignments to that resource. Shared resources are given an entry in the ART by the first process to allocate them. The table is linked to each task's service area file assignment table (FAT) entry for the respective resource.

When the ART entry is made at assignment, the resource assign count is incremented. The assign count is decremented when the task deallocates the resource. The resource is not physically deallocated until the assign count equals zero; the physical deallocation of the resource is not performed while it is in use.

Other information is kept in the ART when a resource is determined to be implicitly shared. For files, pointers are kept to indicate the position of writers on the file by the current end-of-file and end-of-medium positions. These pointers are identified by relative block number. The current allowable access modes are also noted in the ART entry when the resource is implicitly shared.

The size of the ART is determined at SYSGEN by the ARTSIZE directive.

	0 7	8 15	16 23	24 31
Word 0	Resource index (AR.UDTI). See Note 1.	Resource descriptor block address (AR.BLOCK). See Note 2.		
1	Current access mode (AR.CACM). See Note 3.	Resource Pointer. See Note 4.		
2	Resource allocation flags (AR.FLAGS). See Note 5.		DQE index of exclusive lock owner (AR.XRL)	DQE index of synchronous lock owner (AR.SRL)
3	Number of active assignments (AR.ASSNS)	Number of users/allocations of resource (AR.USERS)	Number of multiprocessor requests queued for this resource (AR.QUE)	Number of readers currently on this resource minus the number of writers, appenders, modifiers and updaters (AR.RDRS)
4	Current EOF position in this file (AR.EOF)			
5	Current EOM position in this file (AR.EOM)			
6	Port number of multiport resource lock owner (AR.MPID)	DQE index of task locking a multiport resource	Resource reserve count (AR.RCNT)	
7	Reserved			

Notes:

1. Resource index corresponds to a UDT index in most cases or to an SMT entry index when bit 5 of AR.FLAGS is set.
2. Resource descriptor block address field contains a shared memory table entry pointer when bit 5 of AR.FLAGS is set.
3. Bits in AR.CACM are assigned as follows (implicit shared use only):

Bit	Meaning if Set
0	read access (RD.READ)
1	write access (RD.WRITE)
2	modify access (RD.MODFY)
3	update access (RD.UPDAT)
4	append access (RD.APPDN)
5-7	reserved

4. Resource pointer is as follows:

Resource	Pointer
Volume	mounted volume table entry pointer (AR.MVTA)
Segment definition	Number of blocks in segment definition (AR.NBLKS)
Partition	shared memory table entry pointer (AR.SMTA)
Device	unit definition table entry pointer (AR.UDTA)

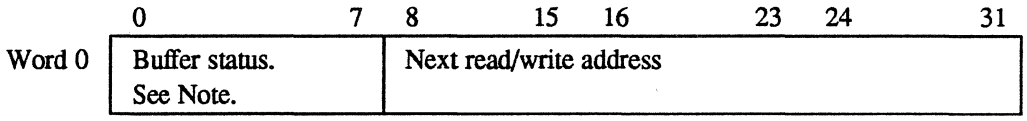
5. Bits in AR.FLAGS are assigned as follows:

Bit	Meaning if Set
0	allocated for explicit shared use (AR.EXSHR)
1	allocated for implicit shared use (AR.IMSHR)
2	allocated as mount device (AR.MNT)
3	marked for deletion (AR.DELET)
4	segment definition (AR.SPACE)
5	memory partition (AR.PART)
6	device (AR.DEVC)
7	entry is active (AR.ACTV)
8	resource marked for truncation (AR.TRUNC)
9-10	reserved
11	dual-processor resource is being appended by a task in this system environment (AR.WOWN2)
12	dual-processor lock is in effect on this resource (AR.DPLK)
13	dual-processor resource is being written to by a task in this system environment (AR.WOWN)
14	multiprocessor volume flag (AR.DUALP)
15	port designation for resource lock owner when resource is treated as dual processor (AR.PORT). This bit is used only when the system is SYSGENed to be compatible to a previous release.

2.5 Blocking Buffer Control Cells

Blocking buffer control cells are built by IOCS for blocked files as the file is written and they become a permanent part of the file. This information is then used by IOCS as the file is read to unblock individual records within the file.

Blocking Buffer Control Word

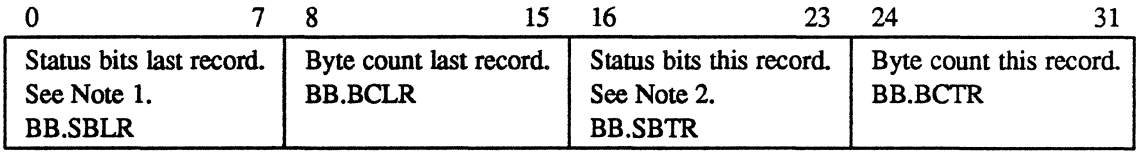


Notes:

Bits in buffer status are assigned as follows:

Bit	Status
0	reserved
1	buffer is empty
2	buffer is output active
3	reserved
4	buffer is free to allocate
5-7	reserved

Record Control Bytes



For the last record in a block, bytes 2 and 3 -- status bits this record and byte count this record -- are omitted.

Notes:

1. Bits in this field are assigned as follows:

Bit	Meaning if Set
0	end-of-file (SB.EOF)
1	beginning-of-block (SB.BOB)
2	end-of-block (SB.EOB)
3	end of medium (SB.EOM)
4-7	reserved

2. Bits in this field are assigned as follows:

Bit	Meaning if Set
0	end-of-file (SB.EOF)
1-7	reserved

2.5.1 Blocking Buffer Head Cells

The DFT.BBA field of the FAT contains the address of the 8 word blocking buffer head cell. Head cells are built in the TSA. The total number, as well as the address, of the first head cell are contained in T.BBHCA. The head cell includes the following information:

	0 7	8 15	16 23	24 31
Word 0	Status bits (BB.SW). See Note.			
1	Address of first buffer (BB.FIRST)			
2	Address of current buffer (BB.CURR)			
3	Block number in first buffer (BB.FBLK)			
4	Number of buffers in big blocking buffers (BB.SIZE)	Buffer number being read/ written (BB.NBUF)	Reserved	
5-7	Reserved			

Notes:

Status bits in BB.SW are assigned as follows:

Bit	Meaning if Set
0	blocking buffer status word (BB.SW)
1	buffer is empty (SW.EMP)
2	buffer is output active (SW.OUT)
3	buffer is in use (SW.BBB)
4	buffer is free to allocate (SW.FRE)
5	buffer is allocated by H.BKDM (SW.ALL)
6	user-supplied buffer is in use (SW.UBB)
7	reserved for S.BKDM9 (SW.SVC)
8-31	read/write address

2.6 Caller Notification Packet (CNP)

The caller notification packet (CNP) is the mechanism used by the Resource Management Module (H.REMM) and the Volume Management Module (H.VOMM) for handling abnormal conditions that may result during resource requests. All or part of this structure can be used by a particular service being called. The CNP must be on a word boundary.

	0 7	8 15	16 23	24 31
Word 0	Time-out value (CP.TIMO)			
1	Abnormal return address (CP.ABRET)			
2	Option field (CP.OPTS). See Note 1.		Status field (CP.STAT). See Note 2.	
3	Actual file size created (CP.FSIZ)			
4	Reserved (See Note 3.)			
5	Automatic open FCB address (CP.FCBA)			

Notes:

1. A bit sequence and/or value used to provide additional information that can be necessary to fully define the calling sequence for a particular service.
2. A right-justified numeric value identifying the return status for this call.
3. Refer to the individual system service description in the MPX-32 Reference Manual Volume I for interpretation of word 4.

2.7 Channel Definition Table (CHT)

The channel definition table (CHT) is a system resident structure applicable only to F-class and extended I/O devices. The CHT is built by the SYSGEN process, one for each extended I/O channel configured in the system. It serves as a register save area, contains the interrupt context block associated with extended I/O protocol, identifies CDTs linked to the channel, and defines other pertinent channel information.

	0 7 8 15 16 23 24 31		
Word 0-7	Register save area (CHT.REGS). See Note 1.		
8-9	Old PSD1/old PSD2 (CHT.OPSD)		
10-11	New PSD1/new PSD2 (CHT.NPSD)		
12	IOCL address (CHT.IOCL)		
13	Status address (CHT.STAD)		
14	Flag word (CHT.FLGS). See Note 2.		
15	Channel spurious interrupt count (CHT.SPUR)	Channel interrupt priority* (CHT.IPL)	Channel address* (CHT.CHAN)
16	CDT address unit 0* (CHT.CDT0). See Note 3.		
17-31	CDT address unit 1* (CHT.CDT1). through CDT address unit 15* (CHT.CDTF). See Note 3.		
32	IOP status doubleword (CHT.STDW) (or) Subaddress Real IOCD address (CHT.RIOA) (CHT.SUBA)		
33	Channel status (CHT.CHST)	Cont/device status (CHT.CDST)	Residual byte count (CHT.RBC)
34	Address of XIO.SUB exit entry point (CHT.EXIT). See Note 4.		
35	Address of H.IFXIO initialization entry point (CHT.INCH). See Note 5.		
36	SIO status stored return address (CHT.RTN)		
37	HIO status stored return address (CHT.HRTN)		
38-39	Reserved for future development use.		

* Initialized by SYSGEN

Notes:

1. CHT.REGS must begin on a register file boundary.
2. Bits in CHT.FLGS are assigned as follows:

Bit	Meaning if Set
0	INCH (initialize channel) has been performed
1	status stored response for SIO or HIO instruction
2	SI. routine was called from LI.XIO routine (common XIO routines)
3	interrupt level was activated by IQ.XIO routine (common XIO routines)
4	cache controller (CHT.CAC)
5	SCSI controller (CHT.SCSI)
6-31	reserved

3. These fields contain the addresses of the CDT entries for controllers connected to the corresponding XIO channel. Entries for unimplemented controllers are set to zero.
4. CHT.EXIT contains the address of the exit procedure within the common XIO subroutines.
5. CHT.INCH contains the address of the initialization procedure used to initialize the corresponding XIO channel.

2.8 Controller Definition Table (CDT)

The controller definition table (CDT) is a system resident structure used to identify information required by handlers and the I/O processor for a specific controller. The CDT is built by the SYSGEN process, one for each controller configured on the system. The CDT identifies devices (UDTs) associated with the controller, the handler address associated with the controller, and defines other pertinent controller information.

	0 7	8 15	16 23	24 31
Word 0	String forward address (CDT.FIOQ)			
1	String backward address (CDT.BIOQ)			
2	Link priority (CDT.LPRI). See Note 1.	Number of entries in list (CDT.IOCT). See Note 2.	Class (CDT.CLAS). See Note 3.	Flags (CDT.FLG2). See Note 4.
3	CDT index (CDT.INDX)		Device type code (CDT.DTC) See Note 5.	Interrupt priority level (CDT.IPL)
4	Number units on controller (CDT.NUOC)	Number requests outstanding (CDT.IORO)	Channel number (CDT.CHAN)	Subaddress of first device (CDT.SUBA)
5	Program number if reserved (CDT.PNRC)	Interrupt handler address (CDT.SIHA) or controller information block (CDT.CIF)		
6	Flags (CDT.FLGS). See Note 6.	UDT address of first device on controller (CDT.UDTA)		
7	I/O status (CDT.IOST). See Note 7.	TI address (CDT.TIAD) or SI address if extended I/O (CDT.SIAD)		
8	UDT address unit 0* (CDT.UT0)			
9-23	UDT address unit 1* (CDT.UT1) through UDT address unit 15* (CDT.UTF)			

*Initialized by SYSGEN

Notes:

1. Always zero (head cell)
2. Number of entries in list (zero if none)
3. Values in CDT.CLAS are assigned as follows:

Value	Meaning
X'0D'	TCW type with extended addressing capability
X'0E'	TCW type
X'0F'	extended I/O

4. Bits in CDT.FLG2 are assigned as follows:

Bit	Meaning if Set
0	SCSI device (CDT.SCSI)
1-7	reserved for future use

5. For example, 01 for any disk, 04 for any tape, etc. Valid device type codes are listed in Chapter 1 of this reference manual.

6. Bits in CDT.FLGS are assigned as follows:

Bit	Meaning if Set
0	extended I/O device (CDT.FCLS)
1	I/O outstanding (set by handler, reset by IOCS) (CDT.IOU1)
2	GPMC device (CDT.GPMC)
3	initialization (INC) needs to be performed for this controller (CDT.FINT)
4	D-class (CDT.XGPM)
5	used only when IOQs are linked to the CDT. Set when SIO is accepted by the controller. Reset when IOQ is unlinked from the CDT or when I/O is reported complete to IOCS in the case of operator intervention type errors (CDT.IOU5).
6	IOP controller (CDT.IOP)
7	controller malfunction (CDT.MALF)

7. Bits in CDT.IOST are assigned as follows:

Bit	Meaning if Set
0	IOQ linked to UDT (CDT.NIOQ)
1	multiplexing controller (CDT.MUXC)
2	use standard XIO interface
3	16MB GPMC (CDT.XGPS)
4	cache controller (CDT.CAC)
5	H.F8XIO has determined if the controller is pre-8512-2 or not (CDT.CKFL)
6	controller not pre-8512-2 (CDT.FLOW)
7	reserved for FMS

2.9 Device Context Area (DCA)

A device context area (DCA) exists for each active subchannel and serves as a storage area for information regarding the subchannel and its operation. The DCAs are physically located at the end of each device-dependent handler (H.??XIO) and must be doubleword bounded. The first 33 words of each DCA are identical; however, additional words can be added to suit the needs of the particular device. The following represents the first 33 words of each DCA.

	0 7 8 15	16 23	24 31
Word 0	DCA size (DCA.SIZE)		
1	Device address (DCA.UADD)	Reserved	
2	CHT address (DCA.CHTA)		
3	CDT address (DCA.CDTA)		
4	UDT address (DCA.UDTA)		
5	IOQ address (DCA.IOQA)		
6	Lost interrupt count (DCA.LINC)		
7	Spurious interrupt count (DCA.SINC)		
8	Total retry count this device (DCA.RETC)		
9	Flags (DCA.FLAG). See Note 1.		Retry count this request (DCA.RCNT)
10	UDT address (DCA.NUDT). See Note 2.		
11	Status word one (DCA.WST1)		
12	Status word two (DCA.WST2)		
13	Number of reserves outstanding (DCA.RESC)		
14	Time-out value opcode 0 (DCA.TIMO). See Note 3.		
15	Time-out value opcode 1. See Note 3.		
29	Time-out value opcode F. See Note 3.		
30-31	Sense IOCD (DCA.SENI)		
32	Sense buffer (DCA.SENS)		

Notes:

1. Bits in DCA.FLAG are assigned as follows:

Bit	Meaning if Set
0	interrupts not expected
1	HIO issued at LI.XIO
2	HIO needs to be reissued
3	device rewinding or seeking
4	sense issued without an IOQ
5	device is an XIO magnetic tape
6-15	reserved for common subroutine usage
16-23	reserved for device dependent handler usage

2. This UDT address is the UDT address of the device for which an SIO or HIO was issued when a status stored response was generated for this device. It indicates the need to reissue the I/O request for that device.
3. Time-out values corresponding to opcodes 0 through F (16 entries).

2.10 Device Type Table (DTT)

The device type table (DTT) is a system resident structure used to identify device types that are configured in the system and their associated controllers. The DTT is built by the SYSGEN process and its entries are linked to the associated controller definition table (CDT).

Valid device type codes are listed in Chapter 1 of this manual.

	0 7	8 15	16 23	24 31
Word 0	Device type code (DTT.COD) See Note 1	Address of first CDT entry of this type (DTT.CDTA)		
1	Number of controller entries (DTT.CNT)	Flags (DTT.FLGS). See Note 2.	ASCII device mnemonic (DTT.NAM). See Note 3.	

Notes:

1. For example, 01 = any disk, 04 = any magnetic tape, 08 = any reader card, and 0A = any line printer.
2. Used by job control and cataloger to validate ASSIGN3 statements with bits assigned as follows:

Bit	Meaning if Set
0	entry of device address not legal
1	entry of size or reel ID required
2	entry of reel ID required
3-7	reserved

3. For example, X'4443' (DC) = any disk; X'4D54' (MT) = any tape

2.11 Directory Entry Table (M.DN.TEQ)

The directory entry table (M.DN.TEQ) contains information pertinent to resources defined in a directory. Each resource defined in a directory has an M.DN.TEQ associated with it.

	0 7 8 15 16 23 24 31	
Word 0-3	Resource name (DN.IDNAM)	
4	Binary creation date (DN.DATE)	
5	Binary creation time (DN.TIME)	
6	Absolute block number of resource descriptor (DN.DOFF)	
7	Resource ID flags (DN.RDFLG). See Note 1.	Resource type (numeric value) (DN.RTYPE). See Note 2.
8	Number of entries that collided with this entry (DN.COLCT)	
9	Number of hashes required to locate this entry (DN.HSHCT)	
10	Directory entry flags (DN.FLAGS). See Note 3.	
11	Directory entry index (DN.DIRI)	
12-13	Owner name of directory entry creator (DN.OWNR)	
14-15	Filler (DN.FILL)	

Notes:

1. Internal flags reserved for MPX-32
2. Values for DN.RTYPE are as follows:

Value	Meaning
1	volume type (DN.VOL)
2	resource descriptor description (DN.RESRC)
3	descriptor map descriptor (DN.DMAP)
4	space map descriptor (DN.SMAP)
5	root directory descriptor (DN.ROOT)
6	system image descriptor (DN.IMAGE)
7	bad block descriptor (DN.BDBLK)
8	value for spool file descriptor (DN.SYM)
9	extra segment definition descriptor (DN.XSEGD)
10	permanent file (DN.FILE)
11	permanent directory (DN.DIR)
12	temporary file (DN.TFILE)
13	temporary directory (DN.TDIR)
14	static memory partition (DN.MEM)
15	dynamic memory partition (DN.TMEM)
16	device descriptor (DN.DEVC)
17	resource descriptor for the DMAP bad block deallocation file (DN.BDMAP)
18	resource descriptor for the SMAP bad block deallocation file (DN.BSMAP)

3. Bits in DN.FLAGS are assigned as follows:

Bit	Meaning if Set
0	active entry (DN.ACTIV)
1-31	reserved

2.12 Dispatch Queue Area

The dispatch queue area is a variable length doubleword-bounded table built by SYSGEN. It contains a maximum of 255 dispatch queue entries (DQEs). The address of the dispatch queue area is contained in C.DQUE. The number of DQE entries is contained in C.NQUE. Free DQE entries are linked into the C.FREE head cell in the standard linked list format. When a task is activated, a DQE is obtained from the free list and is used to contain all of the core-resident information necessary to describe the task to the system. Additional (swappable) information is maintained in the task service area (TSA). While a task is active, its DQE is linked to one of the various ready-to-run or wait state chains provided by the scheduler to describe the task's current status. When a task exits, its DQE is again linked to the free list.

2.13 Dispatch Queue Entry (DQE)

The dispatch queue entry (DQE) contains all of the core-resident information required to describe an active task to the system. It is always linked to the CPU scheduler state chain that describes the current execution status of the associated task.

Dispatch Queue Entry (DQE) Table

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31			
0	0	DQE.SF			
1	4	DQE.SB			
2	8	DQE.CUP	DQE.BUP	DQE.IOP	DQE.US
3	C	DQE.NUM/DQE.TAN			
4-5	10	DQE.ON			
6-7	18	DQE.LMN			
8-9	20	DQE.PSN			
10	28	DQE.USW			
11	2C	DQE.USHF			
12	30	DQE.MSD			
13	34	DQE.KCTR			
14	38	DQE.MMSG	DQE.MRUN	DQE.MNWI	DQE.GQFN
15	3C	DQE.UF2	DQE.IPUF	DQE.NWIO	DQE.SOPO
16	40	DQE.CQC			
17	44	DQE.SH	DQE.SHF	DQE.TIFC	DQE.RILT
18	48	DQE.UTS1			
19	4C	DQE.UTS2			
20	50	DQE.DSW			
21	54	DQE.PRS			
22	58	DQE.PRM			
23	5C	Reserved	DQE.TSKF	DQE.MSPN	DQE.MST
24	60	DQE.PSSF			
25	64	DQE.PSSB			
26	68	DQE.PSPR	DQE.PSCT	DQE.ILN	DQE.RESU
27	6C	DQE.TISF			
28	70	DQE.TISB			
29	74	DQE.TIPR	DQE.TICT	DQE.SWIF	DQE.UBIO
30	78	DQE.RRSF			
31	7C	DQE.RRSB			
32	80	DQE.RRPR	DQE.RRCT	DQE.NSCT	
33	84	DQE.MRSF			
34	88	DQE.MRSB			

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
35	8C	DQE.MRPR	DQE.MRCT	DQE.NWRR	DQE.NWMR
36	90	DQE.RTI	DQE.NWLM	DQE.ATI	Reserved
37	94	DQE.SAIR/DQE.TAD			
38-40	98	DQE.ABC			
41	A4	DQE.TSAP			
42-43	A8	DQE.SRID/DQE.PGOL			
	AC	DQE.SRID/DQE.PGOC		DQE.SRID/Reserved	
44-51	B0	DQE.CDIR/DQE.CVOL			
52	D0	DQE.GID	Reserved	DQE.ASH	
53	D4	DQE.ACX2			
54	D8	DQE.MRQ	DQE.MEM	DQE.MEMR	
55	DC	DQE.MRT	Reserved	DQE.RMMR	
56	E0	DQE.MAPN		DQE.CME	
57	E4	DQE.CMH		DQE.CMS	
58-63	E8-FC	Reserved			

Byte (Hex)	Symbol	Description
0	DQE.SF	String forward linkage address; Field length = 1W; Standard linked list format; Contains address of next (top-to-bottom) entry in chain.
4	DQE.SB	String backward linkage address; Field length = 1W; Standard linked list format; Contains address of next (bottom-to-top) entry in chain.
8	DQE.CUP	Current user priority; Standard linked list format; This priority is adjusted for priority migration based on situational priority increments. Situational priority increments are based on the base level priority (DQE.BUP) of the task.
	DQE.BUP	Base priority of user task; Field length = 1B; Used by scheduler to generate DQE.CUP (current priority) based on any situational priority increments.
	DQE.IOP	I/O priority; Field length = 1B; Initially set from base priority; Used for I/O queue priority.

Byte (Hex)	Symbol	Description
	DQE.US	State chain index for this user task; Field length = 1B; Range: zero through X'1E'; Indicates current state of this task, such as ready-to-run priority, I/O wait, resource block, etc.

Label	Index	Task description
FREE	00	DQE is available (in free list)
PREA	01	activation in progress
CURR	02	currently executing task or is pre-empted time-distribution task in quantum stage one
SQRT	03	ready to run (priority level 1 to 54)
SQ55	04	ready to run (priority level 55)
SQ56	05	ready to run (priority level 56)
SQ57	06	ready to run (priority level 57)
SQ58	07	ready to run (priority level 58)
SQ59	08	ready to run (priority level 59)
SQ60	09	ready to run (priority level 60)
SQ61	0A	ready to run (priority level 61)
SQ62	0B	ready to run (priority level 62)
SQ63	0C	ready to run (priority level 63)
SQ64	0D	ready to run (priority level 64)
SWTI	0E	waiting for terminal input
SWIO	0F	waiting for I/O
SWSM	10	waiting for message complete
SWSR	11	waiting for run request complete
SWLO	12	waiting for low speed output
SUSP	13	waiting for timer expiration, resume request, or message interrupt
RUNW	14	waiting for timer expiration, or run request
HOLD	15	waiting for a continue request
ANYW	16	waiting for timer expiration, no-wait I/O complete, no-wait message complete, no-wait run request complete, message interrupt, or break interrupt
SWDC	17	waiting for disk space
SWDV	18	waiting for device allocation
SWFI	19	waiting for file system
MRQ	1A	waiting for memory
SWMP	1B	waiting for memory pool
SWGQ	1C	waiting in general wait queue
CIPU	1D	current IPU task in execution
RIPU	1E	IPU requesting state

Byte (Hex)	Symbol	Description
C	DQE.NUM	DQE entry number; Field length = 1B; Used as an index to DQE address table (DAT); Range: one through "N"(for MPL index compatibility); Used by scheduler to set C.PRNO to reflect the currently executing task. This value is also used as the MPL index. It is used by the scheduler to initialize the CPIX in the PSD before loading the map for this task.
	DQE.TAN	Task activation sequence number; Field length = 1W; This number is assigned by the activation service and uniquely identifies a task. **Note:** The most significant byte of this value is the DQE entry number and is accessible as DQE.NUM.
10	DQE.ON	Owner name; Field length = 1D.
18	DQE.LMN	Load module name; Field length = 1D.
20	DQE.PSN	Pseudonym associated with task; Field length = 1D; This parameter is an optional argument accepted by the pseudo task activation service. It can be used to uniquely identify a task within a subsystem, such as multibatch. It contains descriptive information useful to the system operator or to other tasks within a subsystem. Conventions used to generate a pseudonym are determined by the associated subsystem. A system-wide convention should be used to establish pseudonym prefix conventions to avoid confusion between subsystems.
28	DQE.USW	User status word; Field length = 1W.
2C	DQE.USHF	Scheduling flags; Field length = 1W; Used by the scheduler to indicate special status conditions.

Byte (Hex)	Symbol	Description

Bit	Meaning When Set
00	load protection image requested (DQE.LPI)
01	single copy load module (DQE.SING)
02	task is indirectly connected (DQE.INDC)
03	task is privileged (DQE.PRIV)
04	task has message receiver (DQE.MSGR)
05	task has break receiver (DQE.BRKR)
06	task quantum stage one expired (DQE.QS1X)
07	task quantum stage two expired (DQE.QS2X)
08	in-swap I/O error (DQE.INER)
09	wait I/O request outstanding (DQE.WIOA)
10	wait I/O complete before in-progress notification (DQE.WIOC)
11	inhibit message pseudointerrupt (DQE.INMI)
12	batch origin task (DQE.BAOR)
13	running in TSM environment (DQE.TMOR)
14	task abort in progress (DQE.ABRT)
15	task is in pre-exit state (DQE.PRXT)
16	run receiver mode (DQE.RRMD)
17	wait send message outstanding (DQE.WMSA)
18	wait message complete before link to wait queue (DQE.WMSC)
19	wait mode send run request outstanding (DQE.WRRA)
20	wait mode send run request complete before link to wait queue (DQE.WRRC)
21	debug associated with task (DQE.DBAT)
22	real-time task (DQE.RT)
23	time-distribution task initial dispatch (DQE.TDID) Set by: • H.ALOC1 on activation of T/D task. • S.EXEC51 when task is linked to wait state. • H.EXEC7 on completion of inswap or other memory request. Reset by: • S.EXEC20 on initial dispatch of task after activation • Wait state termination • In-swap
24	task delete in progress (DQE.DELP)
25	task abort (with abort receiver) in progress (DQE.ABRA)
26	abort receiver established (DQE.ABRC)
27	asynchronous abort/delete inhibited (DQE.ADIN)
28	asynchronous delete deferred (DQE.ADDF)
29	task is inactive (DQE.INAC)
30	asynchronous abort deferred (DQE.AADF)
31	activation timer in effect (DQE.ACTT)

Byte (Hex)	Symbol	Description
30	DQE.MSD	Physical address of MIDL in TSA; Field length = 1W.
34	DQE.KCTR	Kill/abort timer; Field length = 1W.
38	DQE.MMSG	Maximum number of no wait messages allowed to be sent by this task; Field length = 1B.
	DQE.MRUN	Maximum number of no-wait run requests allowed to be sent by this task; Field length = 1B.
	DQE.MNWI	Maximum number of no-wait I/O requests allowed to be concurrently outstanding for this task; Field length = 1B.
	DQE.GQFN	Contains the generalized queue (SWGQ) function code; Field length = 1B; Function codes are queued as follows:

Code	Meaning
01	volume resource (QVRES)
02	ART space (QART)
03	mount in progress (QMNT)
04	resourcemark lock (QRSM)
05	reserved for eventmark (QEVM)
06	read wait for writer (QGEN)
07	shared memory table (QSMT)
08	synchronous resource lock (QSRL)
09	mounted volume table (QMVT)
0A	dual-port lock (QDPLK)
0B	suspend dual-port lock (QSUSP)
0C	debug wait (QDBGW)
0D	remote message area (QMSG)
0E	remote message event (QSER)
0F	remote allocate area (QASMP)
10	remote deallocate area (QDSMP)
11	remote abort area (QAMSG)
12	remote enable/disable area (QOMSG)
13	wait for TSM (QWTSM)

Byte (Hex)	Symbol	Description
3C	DQE.UF2	Scheduling flags; Field length = 1B;

Bit	Meaning if Set
0	enable debug mode break (DQE.EDB)
1	generalized wait queue time-out (DQE.GQTO)
2	task interrupts are synchronized (DQE.SYNC)
3	task is part of a job (DQE.JOB)
4	ACX-32 task flag (DQE.ACX)
5	special arithmetic function requested (DQE.AF)
6	reserved
7	run request terminated (DQE.RRT)

DQE.IPUF — IPU flag byte;
Field length = 1B;

Bit	Meaning if Set
0	IPU inhibit flag (DQE.IPUH)
1	IPU bias flag (DQE.IPUB)
2	CPU only (DQE.IPUR)
3	OS execution direction flag (set when PSD is in user area) (DQE.OSD)
4	base register task (DQE.BASE)
5	Ada task (DQE.ADA)
6	PTRACE debugger task (DQE.PDBG)
7	H.PTRAC task association control bit (DQE.PTRA)

DQE.NWIO — Number of no-wait I/O requests;
Field length = 1B.

DQE.SOPO — Priority bias only swapping control flags;
Field length = 1B;

Bit	Meaning if Set
0	SWGQ state priority-based swapping (DQE.GQPO)
1	swap inhibit due to bit map access (DQE.BMAP)
2	inhibit swap device while accessing MDT (DQE.MDTA)
3	user swap inhibit flag (DQE.USWI)
4	user swap on priority only flag (DQE.USPO)
5-7	reserved

Byte (Hex)	Symbol	Description
40	DQE.CQC	Current quantum count; Field length = 1W; Used by the scheduler to accumulate elapsed execution time for the task to compare the level unique stage one and stage two time-distribution values.
44	DQE.SH	Used by J.SWAPR to swap shadow memory; Field length = 1B.
	DQE.SHF	Shadow memory flag; Field length = 1B; Bit — Meaning if Set 0 — task requests shadow memory (DQE.SHAD) 1 — IPU shadow memory requested (DQE.SHI) 2 — IPU/CPU Common Shadow Memory requested (DQE.SHB).
	DQE.TIFC	Timer function code; Field length = 1B; Value — Meaning 00 — not active 01 — request interrupt 02 — resume program from suspend (SUSP) queue 03 — resume program from any-wait (ANYW) queue 04 — resume program from run-request-wait (RUNW) queue 05 — resume program from generalized (SWGQ) queue 06 — resume program from peripheral device (SWDV) queue 07 — resume program from disk space (SWDC) queue
	DQE.RILT	Request Interrupt (RI) level for timer; Field length = 1B; Identifies the interrupt level to be requested upon timer expiration.
48	DQE.UTS1	User timer slot word 1; Field length = 1W; Current timer value; Contains negative number of timer units before time out.

Byte (Hex)	Symbol	Description
4C	DQE.UTS2	User timer slot word 2; Field length = 1W; Reset timer value; Contains negative number of timer units; Used to reset the current timer value when it expires.
50	DQE.DSW	Base mode debugger status word (PCALL); Field length = 1W.
54	DQE.PRS	Peripheral requirement specification; Field length = 1W;

Bit	Description
0-7	reserved
8-15	device type code
16-23	channel address
24-31	subchannel address or contains first word of SWGQ ID.

Byte (Hex)	Symbol	Description
58	DQE.PRM	Peripheral requirements mask; Field length = 1W;

Value	Meaning
X'00FF0000'	any device of this type code
X'00FFFF00'	any device of the specified type code on the specified channel
X'00FFFFFF'	the specified device as described by type code, channel, and subchannel address, or contains second word of SWGQ ID.

Byte (Hex)	Symbol	Description
5C	Reserved	Field length = 1B
	DQE.TSKF	Task flags; Field length = 1B;

Bit	Meaning if Set
0	real-time accounting disabled (DQE.RTAC)
1-2	reserved for RMSS
3	task is running with MPX-32 mapped out (DQE.MAPO)
4	reserved for MPX-32
5	task is demand page (DQE.DPG)
6	inhibit page out (DQE.NPGO)
7	reserved

Byte (Hex)	Symbol	Description
	DQE.MSPN	TSA maps required to span MIDLs and MEMLs; Field length = 1B.
	DQE.MST	Static memory type specification; Field length = 1B; Value — Memory Class 01 — E 02 — H 03 — S 04 — H1 05 — H2 06 — H3 This field is used to specify the type of memory required for in-swap.
60	DQE.PSSF	Pre-emptive system service head cell string forward linkage address; Standard head cell format; Field length = 1W; Contains address of next (top-to-bottom) entry in chain.
64	DQE.PSSB	Pre-emptive system service head cell string backward linkage address; Standard head cell format; Field length = 1W; Contains address of next (bottom-to-top) entry in chain.
68	DQE.PSPR	Pre-emptive system service head cell dummy priority (always zero); Standard head cell format; Field length = 1B.
	DQE.PSCT	Pre-emptive system service head cell number of entries in list; Standard head cell format; Field length = 1B.
	DQE.ILN	Interrupt level number; Field length = 1B; Identifies associated interrupt level for interrupt connected tasks.
	DQE.RESU	Reserved usage index; Field length = 1B.

Byte (Hex)	Symbol	Description
6C	DQE.TISF	Task interrupt head cell string forward linkage address; Standard head cell format; Field length = 1W; Contains address of next (top-to-bottom) entry in chain.
70	DQE.TISB	Task interrupt head cell string backward linkage address; Standard head cell format; Field length = 1W; Contains address of next (bottom-to-top) entry in chain.
74	DQE.TIPR	Task interrupt head cell dummy priority (always zero); Standard head cell format; Field length = 1B.
	DQE.TICT	Task interrupt head cell number of entries in list; Standard head cell format; Field length = 1B.
	DQE.SWIF	Swapping inhibit flags; Field length = 1B;

Bit	Task Meaning if Set
0	resident (DQE.RESP)
1	locked in memory (DQE.LKIM)
2	unbuffered I/O in progress (DQE.IO)
3	outswapped (DQE.OTSW)
4	leaving system (DQE.TLVS)
5	forced unswappable during terminal output (DQE.FCUS)
6	forced unswappable because swap file has not been allocated for it (DQE.FCRS)
7	imbedded in the operating system (DQE.INOS)

	DQE.UBIO	Number of unbuffered I/O requests currently outstanding; Field length = 1B.
78	DQE.RRSF	Run receiver head cell string forward linkage address; Standard head cell format; Field length = 1W; Contains address of next (top-to-bottom) entry in chain.
7C	DQE.RRSB	Run receiver head cell string backward linkage address; Standard head cell format; Field length = 1W; Contains address of next (bottom-to-top) entry in chain.
80	DQE.RRPR	Run receiver head cell dummy priority (always zero); Standard head cell format; Field length = 1B.

Byte (Hex)	Symbol	Description
	DQE.RRCT	Run receiver head cell number of entries in list; Standard head cell format; Field length = 1B.
	DQE.NSCT	Number of map blocks outswapped; Field length = 1H.
84	DQE.MRSF	Message receiver head cell string forward Linkage address; Standard head cell format; Field length = 1W; Contains address of next (top-to-bottom) entry in chain.
88	DQE.MRSB	Message receiver head cell string backward Linkage address; Standard head cell format; Field length = 1W; Contains address of next (bottom-to-top) entry in chain.
8C	DQE.MRPR	Message receiver head cell dummy priority (always zero); Standard head cell format; Field length = 1B.
	DQE.MRCT	Message receiver head cell number of entries in list; Standard head cell format; Field length = 1B.
	DQE.NWRR	Number of no-wait mode run requests outstanding; Field length = 1B.
	DQE.NWMR	Number of no-wait mode message requests outstanding; Field length = 1B.
90	DQE.RTI	Requested task interrupt flags; Field length = 1B;

Bit	Meaning if Set
0	reserved
1	priority one end action request. Used for pre-emptive system services. (DQE.EA1R)
2	debug break request (DQE.DBBR)
3	user break request (DQE.UBKR)
4	priority two end action request (DQE.EA2R)
5	message interrupt request (DQE.MSIR)
6-7	reserved

Byte (Hex)	Symbol	Description
	DQE.NWLM	No-wait run request limit. Field length = 1B.
	DQE.ATI	Active task interrupt flags; Field length = 1B; Bit — Meaning if Set 0 — reserved 1 — priority one active end action (DQE.AEA1) 2 — active debug break (DQE.ADM) 3 — active user break (DQE.AUB) 4 — priority two active end action (DQE.AEA) 5 — active message interrupt (DQE.AMI) 6-7 — reserved
	Reserved	Field length = 1B.
94	DQE.SAIR	System action task interrupt request; Bit — Meaning if Set 0 — request for delete of this task (DQE.DELR) 1 — reserved 2 — hold task request (DQE.HLDR) 3 — abort task request (DQE.ABTR) 4 — exit task request (DQE.EXTR) 5 — suspend task request (DQE.SUSR) 6 — run receiver mode request (DQE.RRRQ) 7 — reserved
	DQE.TAD	TSA address (logical); Field length = 1W; Byte zero contains DQE.SAIR.
98	DQE.ABC	Abort code; Field length = 3W.
A4	DQE.TSAP	Physical address of the TSA
A8-AC	DQE.SRID	If DQE.DPG is reset; Used swap space linked list; Field length = 2W.
	DQE.PGOL	If DQE.DPG is set; Page out list; Forward pointer to MPTL (MAP.SF); Field length = 1HW Backward pointer to MPTL (MAP.SB) Field length = 1HW.
	DQE.PGOC	Number of pages queued for pageout Field length = 1HW.
	Reserved	Field length = 1HW.

Byte (Hex)	Symbol	Description
B0	DQE.CDIR	Load module RID at activation; Field length = 8W.
	DQE.CVOL	Current working volume at activation; Field length = 8W.
D0	DQE.GID	Group swap identification; Field length = 1B.
D1	Reserved	1 Byte
D2	DQE.ASH	Number of shadow memory blocks currently allocated Field length = 1H.
D4	DQE.ACX2	Advance communication word; Field length = 1W.
D8	DQE.MRQ	Memory request doubleword; Reserved field length = 1B.
	DQE.MEM	Type of memory requested; Field length = 1B;

Value	Memory Class
01	E
02	H
03	S

	DQE.MEMR	Number of memory blocks required; Field length = 1H.
DC	DQE.MRT	Memory request type code; Field length = 1B;

Value	Meaning
00	in-swap only
01	preactivation request
02	activation request
03	memory expansion request
04	IOCS buffer request
05	shared memory request
06	system buffer request
07	release swap file space

If DQE.MRT equals 05, the next three bytes will contain the address of the shared memory table entry.

	Reserved	Field length = 1B.
	DQE.RMMR	Map register for requested memory; Field length = 1H.

Byte (Hex)	Symbol	Description
E0	DQE.MAPN	Inclusive span of maps in use; Field length = 1H.
	DQE.CME	Number of swappable class E map blocks currently allocated; For resident tasks, if not zero, reflects the total number of map blocks in use. Field length = 1H.
E4	DQE.CMH	Number of swappable class H map blocks currently allocated; For resident tasks, if not zero, reflects the total number of map blocks in use. Field length = 1H.
	DQE.CMS	Number of swappable class S map blocks currently allocated; For resident tasks, if not zero, reflects the total number of map blocks in use. Field length = 1H.
E8	Reserved	Reserved for MPX-32; Field length = 6W.

2.14 Dispatch Queue Address Table (DAT)

The dispatch queue address table (DAT) is a variable length table built by SYSGEN. It contains a maximum of 255 single word entries. It is accessed by the word adjusted DQE entry number, and contains the address of the associated DQE in the CPU dispatch queue area. The address of the DAT minus one word is contained in C.ADAT. The number of DAT entries is contained in C.NQUE and is equal to the number of DQEs.

2.15 File Assignment Table (FAT)

The file assignment table (FAT) is used to associate a logical file code (LFC) to a resource. It also coordinates access to the resource referenced by an LFC. The FAT is linked to the unit definition table (UDT) and the controller definition table (CDT) when the resource is allocated.

The FAT must contain information related to the requestor of the resource, such as position within the file (segment and byte within the segment) and current access mode. To increase efficiency, the FAT also contains information pertaining to allowable access modes, segmentation, and extendibility.

Word	0 7	8 15	16 23	24 31
0	Status bits (DFT.STB). See Note 1.	Access flags or system file code (DFT.ACF) See Note 2.	CDT index (DFT.CDTX)	
1	Flags (DFT.FLGS). See Note 3.	Number of FPTs assigned (DFT.NAS)	UDT index (DFT.UDTX)	
2	Segment definition area address (DFT.SEGA) or Volume name for dismount message (DFT.VNAM)			
3	Relative file block position (DFT.POS)			
4	Relative EOM block position (DFT.EOM). See Note 4.			
5	Relative EOF block number (DFT.EOF)			
6	Current segment position in file (DFT.CSEG) or device specification mask (DFT.MASK)		Number of segments (DFT.NSEG)	
7	Relative end block number of current segment (DFT.SEGE) or Unformatted medium identifier (MTF.REEL). See Note 5.		Append record pointer (DFT.AREC). See Note 5.	
8	File attributes field (DFT.ATTR). See Note 6.			
9	Append block number (DFT.ABLK) or volume number for multivolume media (MTF.VOL)			
10	Blocking buffer head cell address (DFT.BBA)			
11	Associated VAT index (DFT.VATX)	Number of opens on this FAT (DFT.OPCT)	Current access mode (DFT.CACM)	Resource type code (DFT.TYPE)
12	Address of parent directory resource descriptor (DFT.PDIR)			
13	Relative offset of parent directory entry (DFT.DOFF)			
14	Allocated resource table entry pointer (DFT.ARTA)			
15	Assigned access restrictions (DFT.ACCS). See Note 7.			

Notes:

1. Bits in DFT.STB are assigned as follows:

Bit	Meaning if Set
0	file open
1	file opened read/write
2	permanent file
3	blocking buffer output active
4	unformatted medium
5	volume resource
6	read only access
7	TSM associated FAT

2. Bits 0-4 in DFT.ACF are assigned as follows:

   Volume resource only:

Bit	Meaning if Set
0-1	reserved
2	$ read on SYC
3-4	reserved

   Unformatted medium only:

Bit	Meaning if Set
0	mount message has been inhibited or tape is shared
1	multivolume tape
2	mount message has been output
3	tape at EOT
4	tape at BOT

   Bits 5-7 in DFT.ACF apply only to volume usage and contain one of the following values:

Value	Meaning if Set
0	not a system file
1	SYC file
2	multivolume magnetic tape was generated by an MPX-32 revision 3.3 or later source system
3	SLO file
4	SBO file

3. Bits in DFT.FLGS are assigned as follows:

Bit	Meaning if Set
0	blocking buffer present
1	SMAP or DMAP assignment
2	multivolume magnetic tape was generated by an MPX-32 revision 3.3 or later source system
3	file has been assigned to the null device
4	this FAT entry is not in use
5	TSM I/O (task is swappable)
6	ANSI labeled tape assignment
7	reserved

4. Byte 3 of word four contains tape density for high speed tape (DFT.DENS) and EOM does not apply (DFT.EOM).
5. For an ANSI labeled tape assignment, this address contains the six-character Volume Identifier (VID).
6. Bits in DFT.ATTR are assigned as follows:

Bit	Meaning if Set
0	file is automatically extendable
1	file is implicitly shared
2	file data has been modified
3	unblocked specified at assignment
4	file opened for random access
5	file opened in blocked mode
6	expanded FCB
7	resource descriptor opened for modify
8	current access mode specified at assignment
9	resource to be marked blocked at close
10	queue inhibit
11	spool option requested
12	EOF update required
13	reserved for IOCS
14	file assigned to nonpublic volume
15	segmented file
16	task in resource queue when deleted
17	the date and time of last change field in the resource descriptor is not changed on a rewrite
18	data in file is blocked (DFT.BLKD)
19-31	reserved

7. Bytes in DFT.ACCS are assigned as follows:

Byte	Definition
0-1	Bit pattern from RR.ACCS if specified at assignment (see section 2.32 for details on RR.ACCS). If not specified, the bit pattern is from the appropriate access restriction field (RD.AOWNR, RD.AUGRP, RD.AOTHR) in the resource descriptor. See the Resource Descriptor (M.RDCOM) section in this chapter for details.
2	Bits are assigned as follows:

Bit	Meaning if Set
0	assigned for explicit shared use
1	assigned for exclusive use
2-7	reserved

3	Bit pattern specified in byte three of RD.SFLGS in the associated resource descriptor. See the Resource descriptor space definition (M.RDSPD) section in this chapter for details.

2.16 File Control Block (FCB)

The file control block (FCB) is used to convey information about requested I/O operations and to report their status to the requestor. The table entry is generally located in the task's address space. The task's FCB is linked to the file assignment table (FAT) when the resource is opened. This completes the logical connection from the task to the requested resource for subsequent use. The FCB is then linked to an I/O Queue (IOQ) entry when an operation for that logical connection is requested. When this is done, the status for the requested operation code is posted in the respective FCB.

Linkages among the FCB, FPT, and FAT are established at open time. To minimize I/O overhead, use the opened FCB for all I/O to a specific LFC. Using alternate FCBs is possible but not recommended because it changes these linkages. If alternate FCBs are used, an explicit close must be performed for each LFC used. Because the operating system relies on information in the FCB during I/O processing, the FCB must not be modified from the time the I/O operation issues until it completes end-action processing.

Word	0 7	8 12	13 31
0	Opcode (FCB.OPCD)	Logical file code (FCB.LFC)	
1	Reserved		
2	General control flags (FCB.GCFG)	Special flags (FCB.SCFG)	Reserved
3	Status flags (FCB.SFLG)		
4	Actual transfer quantity (FCB.RECL)		
5	Reserved	I/O queue address (FCB.IOQA)	
6	Special Status (FCB.SPST)	Wait I/O error return address (FCB.ERRT)	
7	Index to FPT (FCB.FPTI)	FAT address (FCB.FATA)	
8	Reserved	I/O buffer address (FCB.ERWA)	
9	Transfer quantity (bytes) (FCB.EQTY)		
10	Random access address (FCB.ERAA)		
11	Extended I/O status word one (FCB.IST1)		
12	Extended I/O status word two (FCB.IST2)		
13	Reserved	No-wait I/O normal end-action service address (FCB.NWOK)	
14	Reserved	No-wait I/O error end-action service address (FCB.NWER)	
15	Number of buffers (FCB.BBN)	Address of blocking buffer (FCB.BBA)	

Shaded areas are set by the system.

T1FCB

Word 0

Bit 0 Reserved

Bits 1-7 Operation code (FCB.OPCD) — type of function requested of the device handler. This field is set by IOCS as a function of the executed service.

Bits 8-31 Logical file code (FCB.LFC) — any combination of three ASCII characters is allowed. The LFC must match the previously assigned LFC of the I/O resource being accessed.

Word 1

Bits 0-31 Reserved

Word 2

Bits 0-7 General control flags (FCB.GCFG) — these eight bits enable the user to specify the manner in which an operation is to be performed by IOCS. The interpretation of these bits is shown below:

Bit	Meaning if Set	Definition
0	NWT	IOCS returns to the user immediately after the I/O operation is queued. If reset, IOCS exits to the calling program only when the requested operation has been completed.
1	NER	error processing is not performed by either the device handler or IOCS. An error return address is ignored and a normal return is taken to the caller; however, the device status is posted in the FCB unless bit 3 is set. If reset, normal error recovery is attempted. Normal error processing for disk and magnetic tape is automatic error retry. Error processing for unit record devices except the system console is accomplished by IOCS typing the message INOP to the console, which allows the operator to retry or abort the I/O operation. If the operator aborts the I/O operation, or if automatic error retry for disk or magnetic tape is unsuccessful, an error status message is typed to the console and the error return address is taken if provided. Otherwise, the task is aborted.
2	DFI	data formatting is inhibited. Otherwise, data formatting is performed by the appropriate device handler. See Table 2-1 for more explanation.
3	NST	device handlers perform no status checking and no status information is returned. All I/O appears to complete without error. Otherwise, status checking is performed and status information is returned as necessary.
4	RAN	file accessing occurs in the random mode. Otherwise, sequential accessing is performed.
5		reserved (M.FILE)
6	EXP	must be 1 for 16-word FCB.
7	IEC	this bit is reserved for internal IOCS use.

Bits 8-12 Special Control Specification (FCB.SCFG). — This field contains device control specifications unique to certain devices. Interpretation and processing of these specifications are performed by the device handlers. A bit setting is meaningful only when a particular type of device is assigned as indicated in Table 2-1.

Bits 13-31 reserved for extended control specifications

Bit	Meaning if Set	Definition
13	RXON	software read flow control required for 8-Line ACM (FCB.RXON)

Table 2-1
Special Control Flags

Device	Bit 2=0	Bit 2=1	Bit 8=0	Bit 8=1	Bit 9=0	Bit 9=1
Line Printer (LP)	Interpret first character as carriage control	Interpret first character as data See bit 8	Form control	No form control		
Discs, (DM,DF, FL)	Report EOF if X'0FE0FE0F' encountered in word 0 of 1st block during read of unblocked file	X'0FE0FE0F' in word 0 not recognized as EOF				
8-Line Asynchronous Communications Multiplexer (TY)	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**
	Perform special character formatting	No special character formatting	ASCII control passed as data	ASCII control character detect	Echo by controller	No echo by controller
	M.WRIT	**M.WRIT**	**SVC 1,X'3E'**	**SVC 1,X'3E'**	**M.WRIT**	**M.WRIT**
	Interpret first character as carriage control	Interpret first character as data	Stop transmitting break	Start transmitting break	Normal write	Initialize device (load UART parameters)

Device	Bit 10=0	Bit 10=1	Bit 11=0	Bit 11=1	Bit 12=0	Bit 12=1
Line Printer (LP)	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Discs, (DM,DF, FL)					Normal read	Read with byte granularity (word 2 bit 4 set)
8-Line Asynchronous Communications Multiplexer (TY)	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**
	(If bit 2=0) convert lower case character to upper case	Inhibit conversion	No special character detect	Special character detect	Do not purge type ahead buffer	Purge type ahead buffer
	M.WRIT	**M.WRIT**	**M.WRIT**	**M.WRIT**	**M.WRIT**	**M.WRIT**
			Normal write	Write with input sub-channel monitoring plus software flow control		

Continued on next page

Table 2-1
Special Control Flags (*Continued*)

Device	(Bit 2=0)	(Bit 2=1)	Bit 8	Bit 9		Bit 10	Bit 11	Bit 12
ALIM (Asynch-ronous Line Interface Module) Terminals (TY)	Read: receive data (bytes) defined for transfer count	**Bit 2**	**Bit 8**	**Bit 9**	**Read**	On Read:		
		0	1	0	=Blind mode reset	1= Inhibit conversion of lower case characters to upper case		
		0	0	1	=Echo on read			
		1	N/A	N/A	=Receive data			
		0	0	0	=Receive data			
					Write			
	Write: formatted	0	N/A	0	=Formatted write	0= Convert		
		0	N/A	1	=Initialize device			
		1	N/A	N/A	=Unformatted write			

Word 3

Bits 0-31 Status word (FCB.SFLG) — 32 indicator bits are set by IOCS to indicate the status, error, and abnormal conditions detected during the current or previous operation. The assignment of these bits is shown as follows:

Bits	Meaning if Set	Definition
0	OP	operation in progress. Request has been queued. (Note: Reset after post I/O processing complete.)
1	ERR	error condition found
2	BB	invalid blocking buffer control pointers have been encountered during file blocking or unblocking
3	PRO	write protect violation
4	INOP	device inoperable
5	BOM	beginning-of-medium (BOM) (load point) or illegal volume number (multivolume magnetic tape)
6	EOF	end-of-file
7	EOM	end-of-medium (end of tape, end of disk file)
8-9		reserved
10	TIME	last command exceeded time-out value and was terminated
11-15		reserved
16	ECHO	echo
17	INT	post program-controlled interrupt
18	LEN	incorrect length
19	PROG	channel program check
20	DATA	channel data check
21	CTRL	channel control check
22	INTF	interface check
23	CHAI	chaining check
24	BUSY	busy
25	ST	status modified
26	CTR	controller end
27	ATTN	attention
28	CHA	channel end
29	DEV	device end
30	CHK	unit check
31	EXC	unit exception

Word 4

Bits 0-31 Record length (FCB.RECL) — this field is set by IOCS to indicate the actual number of bytes transferred during read/write operations.

Word 5

Bits 0-7 Reserved

Bits 8-31 I/O queue address (FCB.IOQA) — this field is set by IOCS to point to the I/O queue for an I/O request initiated from this FCB

Word 6

Bits 0-7 Special status bits (FCB.SPST). The interpretation of these bits is shown below:

Bits	Definition
0	no-wait normal end action not taken
1	no-wait error end action not taken
2	request killed, I/O not issued
3	if set, exceptional condition has occurred in the I/O request
4	if set, software read flow control required
5-7	reserved

Bits 8-31 Wait I/O error return address (FCB.ERRT) — this field is set by the user and contains the address to which control is to be transferred in the case of an unrecoverable error when control bits 1 and 3 of word 2 are reset. If this field is not initialized and an unrecoverable error is detected under the above conditions, the requesting task is aborted.

Word 7

Bits 0-7 Index to FPT (FCB.FPTI) — this field is set by IOCS to index into the associated entry in the file pointer table (FPT)

Bits 8-31 FAT address (FCB.FATA) — this field is set by IOCS to point to the associated file assignment table (FAT) entry.

Word 8

Bits 0-7 Reserved

Bits 8-31 Data buffer address (FCB.ERWA) — start address of data area for read or write operations. (24 bit pure address)

Word 9

Bits 0-31 Quantity (FCB.EQTY)— number of bytes of data to be transferred

Word 10

Bits 0-31 Random access address (FCB.ERAA) — this field contains a block number (zero origin) relative to the beginning of the disk file. It is the start address for the current read or write operation with word 2 bit 4 set and word 2 bit 12 reset.

or

For disk read requests with word 2 bits 4 and 12 set (read with byte granularity), this word defines the byte offset relative to the beginning of the file where the current read will start.

Note: If word 9 is zero, the file retains its position prior to the call.

Word 11

Bits 0-31 Status word one (FCB.IST1) — these are the first 32 bits of status returned by the sense command

Word 12

Bits 0-31 Status word two (FCB.IST2) — these are the second 32 bits of status returned by the sense command

Word 13

Bits 0-7 Reserved

Bits 8-31 No-wait I/O (FCB.NWOK) — normal completion return address. This user routine must be exited by calling the M.XIEA service.

Word 14

Bits 0-7 Reserved

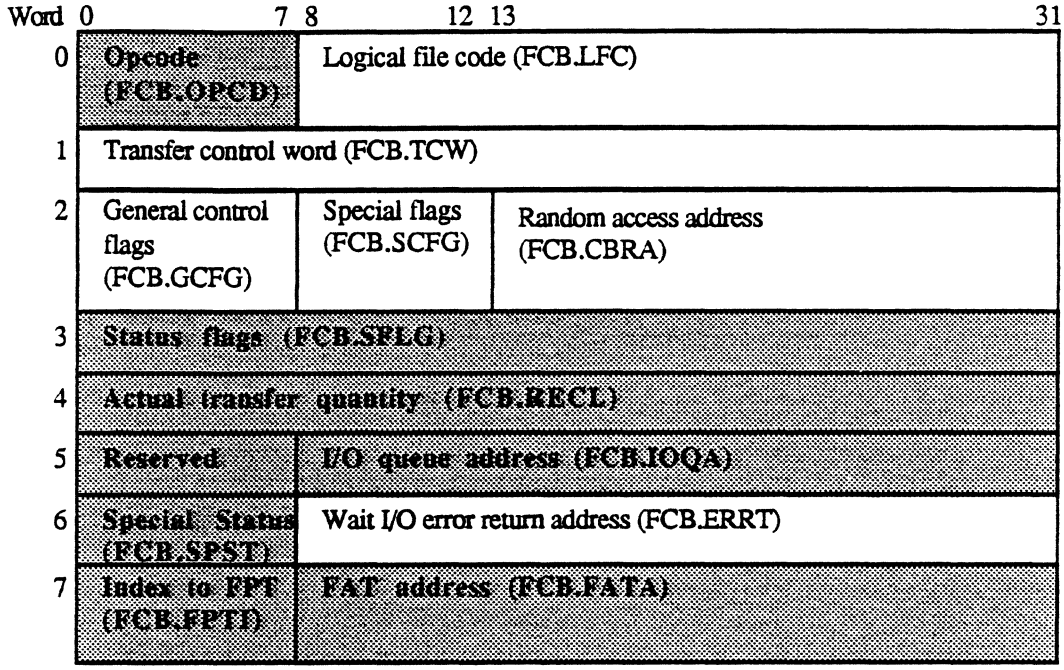
Bits 8-31 No-wait I/O (FCB.NWER) — error completion return address. This user routine must be exited by calling the M.XIEA service.

Word 15 (Applicable only to volume resource.)

Bits 0-7 (FCB.BBN) — Number of 192 word buffers for user supplied blocking buffers. A value of one or zero in this field specifies one blocking buffer.

Bits 8-31 Blocking buffer address (FCB.BBA) — starting address of a contiguous area of memory FCB.BBN buffers long

2.17 File Control Block (8 Word Compatible Mode)



Shaded areas are set by the system.

A.L8W.FCB

Word 0

Bit 0 Reserved

Bits 1-7 Operation code (FCB.OPCD) — type of function requested of the device handler. This field is set by IOCS as a function of the requested service. See Table 2-2 for allowable functions by device.

Bits 8-31 Logical file code (FCB.LFC) — any combination of three ASCII characters is allowed.

Word 1 (FCB.TCW)

This word supplies a transfer control word (TCW) that describes a data buffer and transfer quantity. If no TCW definition is supplied, the transfer buffer defaults to location zero of the task's logical address space and is 4096 words long.

Bits 0-11 Quantity — 12 bit field specifying the number of data items to be transferred. This quantity must include the carriage control character, if applicable. The transfer quantity is in units determined by the address in bits 12 to 31.

Bits 12-31 Format code and buffer address— bits 12, 30 and 31 specify byte, halfword, or word quantities for data transfers. They are interpreted as follows:

Type of Transfer	F (12)	C (30,31)	Address
Byte	1	xx	13-31
Halfword	0	x1	13-30
Word	0	00	13-29

Word 2

Bits 0-7 General control flags (FCB.GCFG) — these eight bits enable the user to specify the manner in which an operation is to be performed by IOCS. The interpretation of these bits is shown below:

Bit	Meaning if Set	Definition
0	NWT	IOCS returns to the user immediately after the I/O operation is queued. If reset, IOCS exits to the calling program only when the requested operation has been completed.
1	NER	error processing is not performed by either the device handler or IOCS. An error return address is ignored and a normal return is taken to the caller; however, the device status is posted in the FCB unless bit 3 is set. If reset, normal error recovery is attempted. Normal error processing for disk and magnetic tape is automatic error retry. Error processing for unit record devices except the system console is accomplished by IOCS typing the message INOP to the console, which allows the operator to retry or abort the I/O operation. If the operator aborts the I/O operation, or if automatic error retry for disk or magnetic tape is unsuccessful, an error status message is typed to the console and the error return address is taken if provided. Otherwise, the task is aborted.
2	DFI	data formatting is inhibited. Otherwise, data formatting is performed by the appropriate device handler. See Table 2-1 for more explanation.
3	NST	device handlers perform no status checking and no status information is returned. All I/O appears to complete without error. Otherwise, status checking is performed and status information is returned as necessary.
4	RAN	file accessing occurs in the random mode. Otherwise, sequential accessing is performed.
5		reserved (M.FILE)
6	EXP	must be 0 for 8 word FCB.
7	IEC	this bit is reserved for internal IOCS use.

Bits 8-12 Special Control Specification (FCB.SCFG). — This field contains device control specifications unique to certain devices. Interpretation and processing of these specifications are performed by the device handlers. A bit setting is meaningful only when a particular type of device is assigned as indicated in Table 2-2.

Bits 13-31 Random access address (FCB.CBRA) — This field contains a block number (zero origin) relative to the beginning of the disk file, and specifies the base address for read or write operations.

Table 2-2
Special Control Flags (8 Word FCB)

Device	Bit 2=0	Bit 2=1	Bit 8=0	Bit 8=1	Bit 9=0	Bit 9=1
Line Printer (LP)	Interpret first character as carriage control	Interpret first character as data See bit 8	Form control	No form control		
Discs, (DM,DF, FL)	Report EOF if X'0FE0FE0F' encountered in word 0 of 1st block during read of unblocked file	X'0FE0FE0F' in word 0 not recognized as EOF				
8-Line Asynchronous Communications Multiplexer (TY)	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**
	Perform special character formatting	No special character formatting	ASCII control passed as data	ASCII control character detect	Echo by controller	No echo by controller
	M.WRIT	**M.WRIT**	**SVC 1,X'3E'**	**SVC 1,X'3E'**	**M.WRIT**	**M.WRIT**
	Interpret first character as carriage control	Interpret first character as data	Stop transmitting break	Start transmitting break	Normal write	Initialize device (load UART parameters)

Device	Bit 10=0	Bit 10=1	Bit 11=0	Bit 11=1	Bit 12=0	Bit 12=1
Line Printer (LP)	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved
Discs, (DM,DF, FL)						
8-Line Asynchronous Communications Multiplexer (TY)	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**	**M.READ**
	(If bit 2=0) convert lower case character to upper case	Inhibit conversion	No special character detect	Special character detect	Do not purge type ahead buffer	Purge type ahead buffer
	M.WRIT	**M.WRIT**	**M.WRIT**	**M.WRIT**	**M.WRIT**	**M.WRIT**
			Normal write	Write with input sub-channel monitoring plus software flow control		

Continued on next page

Table 2-2
Special Control Flags (8 Word FCB) (*Continued*)

Device	(Bit 2=0)	(Bit 2=1)	Bit 8	Bit 9		Bit 10	Bit 11	Bit 12
ALIM (Asynch-	Read: receive	**Bit 2**	**Bit 8**	**Bit 9**	**Read**	On Read:		
ronous	data	0	1	0	=Blind mode reset	1= Inhibit		
Line	(bytes)	0	0	1	=Echo on read	conversion		
Interface	defined	1	N/A	N/A	=Receive data	of lower		
Module)	for	0	0	0	=Receive data	case		
Terminals	transfer					characters		
(TY)	count					to upper		
					Write	case		
	Write:	0	N/A	0	=Formatted write	0= Convert		
	formatted	0	N/A	1	=Initialize device			
		1	N/A	N/A	=Unformatted write			

Word 3

Bits 0-31 Status word (FCB.SFLG) — 32 indicator bits are set by IOCS to indicate the status, error, and abnormal conditions detected during the current or previous operation. The assignment of these bits is shown as follows:

Bits	Meaning if Set	Definition
0	OP	operation in progress. Request has been queued. (Note: Reset after post I/O processing complete.)
1	ERR	error condition found
2	BB	invalid blocking buffer control pointers have been encountered during file blocking or unblocking
3	PRO	write protect violation
4	INOP	device inoperable
5	BOM	beginning-of-medium (BOM) (load point) or illegal volume number (multivolume magnetic tape)
6	EOF	end-of-file
7	EOM	end-of-medium (end of tape, end of disk file)
8-9		reserved
10	TIME	last command exceeded time-out value and was terminated
11-15		reserved
16	ECHO	echo
17	INT	post program-controlled interrupt
18	LEN	incorrect length
19	PROG	channel program check
20	DATA	channel data check
21	CTRL	channel control check
22	INTF	interface check
23	CHAI	chaining check
24	BUSY	busy
25	ST	status modified
26	CTR	controller end
27	ATTN	attention
28	CHA	channel end
29	DEV	device end
30	CHK	unit check
31	EXC	unit exception

Word 4

Bits 0-31 Record length (FCB.RECL) — this field is set by IOCS to indicate the actual number of bytes transferred during read/write operations.

Word 5

Bits 0-7 Reserved

Bits 8-31 I/O queue address (FCB.IOQA) — this field is set by IOCS to point to the I/O queue for an I/O request initiated from this FCB

Word 6

Bits 0-7 Special status bits (FCB.SPST). The interpretation of these bits is shown below:

Bits	Definition
0	no-wait normal end action not taken
1	no-wait error end action not taken
2	kill command, I/O not issued
3	if set, exceptional condition has occurred in the I/O request
4	if set, software read flow control required
5-7	reserved

Bits 8-31 Wait I/O error return address (FCB.ERRT) — this field is set by the user and contains the address to which control is to be transferred in the case of an unrecoverable error when control bits 1 and 3 of word 2 are reset. If this field is not initialized and an unrecoverable error is detected under the above conditions, the user is aborted.

Word 7

Bits 0-7 Index to FPT (FCB.FPTI) — this field indexes into the appropriate entry in the file pointer table (FPT)

Bits 8-15 FAT address (FCB.FATA) — this field points to the file assignment table (FAT) entry associated with all I/O performed for this FCB. This field is supplied by IOCS.

2.18 File Pointer Table (FPT)

The file pointer table (FPT) provides the linkage between the file control block (FCB) and the file assignment table (FAT). It also allows for multiple logical file code assignments to be made equivalent to the same FAT. The linkage to the FAT is performed at assignment. The linkage to the FCB is performed at open and is re-established if necessary for every operation at opcode processing time. The FPT resides in the task's service area.

FPT entries one to six are reserved for the system as follows:

Entry 1 - System LFC *s*
Entry 2 - Load module LFC *LM
Entry 3 - H.VOMM resource descriptor LFC (1)
Entry 4 - H.VOMM directory LFC (2)
Entry 5 - H.VOMM DMAP/SMAP LFC (3)
Entry 6 - H.VOMM modify resource descriptor LFC X'FFFEE'

Each FPT entry has the following format:

	0 7	8 15	16 23	24 31
Word 0	Reserved	Logical file code (FPT.LFC)		
1	Flags (FPT.FLGS). See Note 1.	FCB address (FPT.FCBA)		
2	Reserved	FAT address (FPT.FATA)		

Notes:

1. Bits in FPT.FLGS are assigned as follows:

Bit	Meaning if Set
0	reserved
1	multiple FPT entries exist that point to the same FAT (i.e., $ASSIGN4 or $ASSIGN *lfc* TO LFC = *lfc* statements)
2	FPT busy flag
3	FPT open
4	this FPT entry is not in use
5	pseudo-SYC assignment (used by TSM)
6	pseudo-FPT for unassigned temporary file
7	reserved

2.19 I/O Queue (IOQ) Entry

The I/O queue (IOQ) entry is dynamically allocated from memory pool and contains information required to queue and process an I/O request. These entries are variable in length and support multiple device commands that are built starting at the end of the standard IOQ entry. The I/O queue consists of one or more I/O queue entries linked to either a controller definition table (CDT) or a unit definition table (UDT). See Figures 2-1 and 2-2.

I/O Queue (IOQ) Entry

	0 7	8 15	16 23	24 31
Word 0	String forward address (IOQ.SFA)			
1	String backward address (IOQ.SBA)			
2	Queue priority (IOQ.PRI)	I/O type (IOQ.TYPE)	Channel number (IOQ.CHNO)	Subaddress (IOQ.SUBA)
3	Reserved (IOQ.RTN)			
4	PSD1 of task interrupt routine (IOQ.PSD). See Note 1.			
5	PSD2 of task interrupt routine			
6	Status (IOQ.STAT) See Note 2.	FCB or TCPB address (IOQ.FCBA)		
7	Program number (IOQ.PRGN)	CDT address (IOQ.CDTA)		
8	Handler function word one (IOQ.FCT1)			
9	Handler function word two (IOQ.FCT2). See Note 3.			
10	Handler function word three (IOQ.FCT3). See Note 4.			
11	Handler function word four (IOQ.FCT4)			
12	32-bit flag word (IOQ.FLGS). See Note 5.			
13	FAT address (IOQ.FATA)			
14	Number of bytes transferred (IOQ.UTRN). See Note 6.		Number of words in OS buffer (IOQ.WOSB). See Note 6.	
15	OS buffer address (IOQ.FBUF)			
16	User's buffer address (IOQ.TBUF)			
17	I/O returned status word one (IOQ.IOST)			
18	I/O returned status word two (IOQ.IST1). See Note 7.			
19	I/O returned status word three (IOQ.IST2). See Note 8.			
20	UDT address (IOQ.UDTA)			
21	Control information from word two of FCB (IOQ.CONT)			
22	Address of context block (IOQ.CBLK) or device context area address (IOQ.DCAA)			
23	Mode bits (extended I/O) (IOQ.MODE). See Note 9. (or) Word address of set mode bits (IOQ.MOWD)	Queue priority temporary storage (IOQ.PSAV)	Number of extra words in this queue entry (IOQ.XTRA)	
24	Device inoperable buffer address (for I/O error processing) (IOQ.INOP)			
25	Address of first word of dynamic IOCD list (extended I/O) (IOQ.IOCD). See Note 10.			

Notes:

1. For no-wait I/O, this field is set to point to the I/O postprocessing routine (S.IOCS1). When I/O completes, control is passed to this service.

2. Bits in IOQ.STAT are assigned as follows:

Bit	Meaning if set
0	I/O queue is active. This bit is reset by the device handler when physical I/O transfer completes.
1	sense command was issued on behalf of this I/O request (extended I/O)
2	error retry was issued (rezero and retry entire IOCD list) (extended I/O)
3	operator intervention required. Do not restart I/O.
4	user's buffer is used for I/O
5	read ECC was issued (extended I/O)
6	error retry was issued (retry entire IOCD list) (extended I/O). Backspace write or read sequence performed for extended I/O tape.
7	reserved

3. For extended I/O devices, IOQ.FCT2 contains the 24-bit virtual address of the data or IOCL (bits zero to seven are zero).

4. For extended I/O devices, IOQ.FCT3 contains the adjusted byte transfer count in bits 0 to 31 (maximum is C.ADMASK plus one).

5. Bits in IOQ.FLGS are assigned as follows:

Bit	Meaning if set
0	multiplexed controller
1	OPCOM console request
2	TCW has been absolutized
3	IOQ will be linked to the UDT, not the CDT
4	deallocate OS buffer
5	extended I/O
6	error found
7	system console queue
8	data move required (OS to user buffer)
9	rewind command in IOCD list for magnetic tape or reserve command in IOCD list for disk (extended I/O)
10	nonexecute channel read command (extended I/O)
11	nonexecute channel write command (extended I/O)
12	special handler postprocessing required (handler EP6)
13	H.CT00 has been called with an FCB, not with a TCPB (i.e., not by H.IOCS,14)
14	floating IOQSIZE by 2 words
15	terminal input
16	terminal output

Bit	Meaning if set
17	task swappable during I/O
18	release command in IOCD list for disk (extended I/O)
19	no-wait I/O (not TSM)
20	I/O restart entry
21	nondevice access I/O performed
22	kill command issued for this I/O request
23	execute channel program (extended I/O)
24	user privileged
25	D-class controller (GPMC) only
26	physical I/O performed for a user requesting blocked I/O
27	static IOQ
28	reserved
29	EOF testing required for disk
30	movement in file is in negative direction
31	continuous EOF search (disk and floppy disk only)

6. For extended I/O devices, IOQ.UTRN is a full word (bits 0 to 31) and IOQ.WOSB is not applicable.

7. For extended I/O devices, IOQ.IST1 is initialized to the start address within the I/O queue for any dynamic IOCDs.

8. For extended I/O devices, IOQ.IST2 is initialized to the stop address within the I/O queue for any dynamic IOCDs.

9. Mode bits are peculiar to each device.

10. This field contains the absolute data or IOCL address associated with the I/O request.

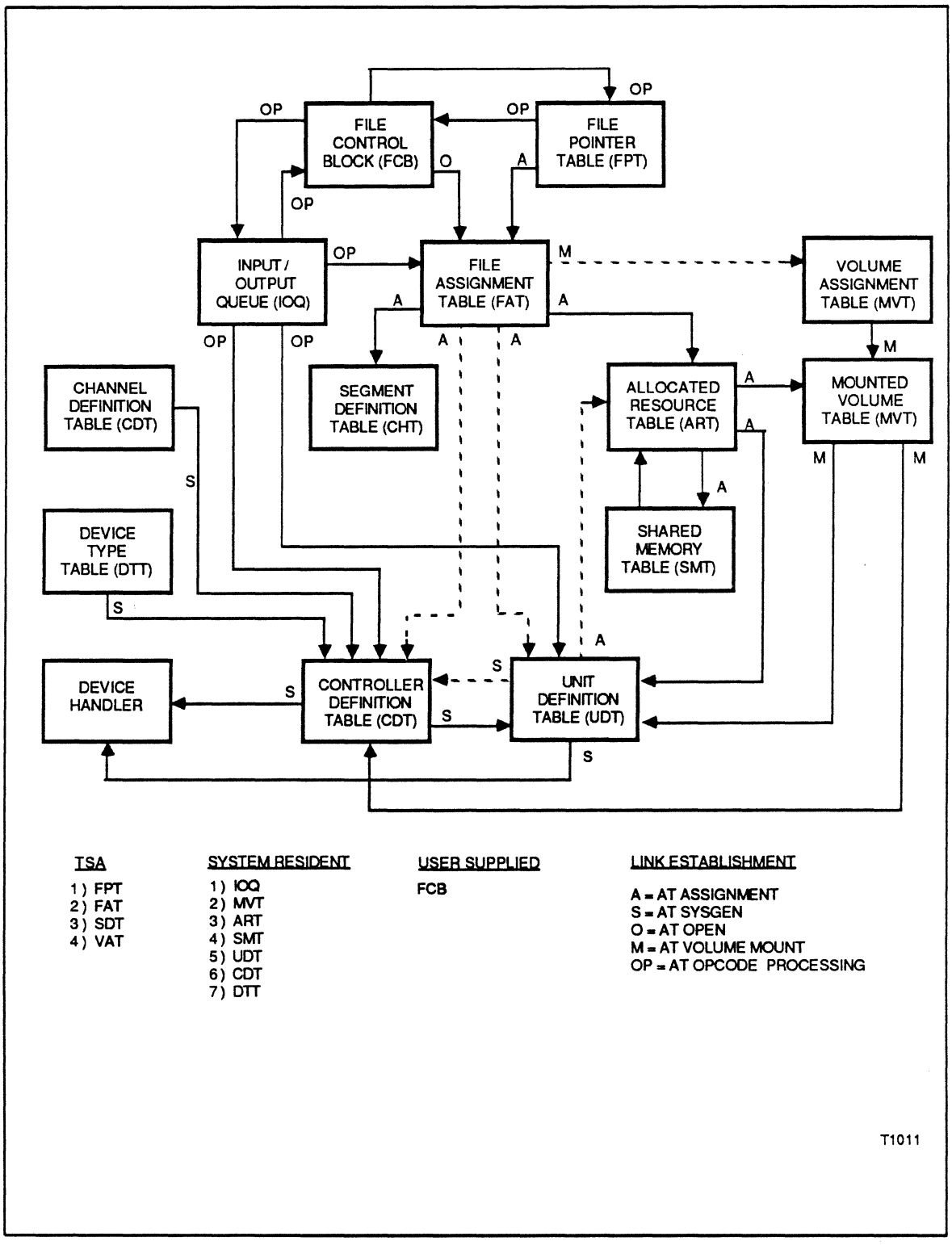


Figure 2-1
I/O Table Linkages

For F-class devices, the device specific tables in MPX-32 mirror the hardware configured at SYSGEN time as illustrated in the following figure. For each I/O channel configured, there is a corresponding channel definition table (CHT). For each controller on that channel, there is a controller definition table (CDT) linked to that CHT. For each device on each controller, there is a unit definition table (UDT) linked to the corresponding CDT. Handler specific device information is retained in the device context area (DCA) that is then linked to the corresponding UDT.

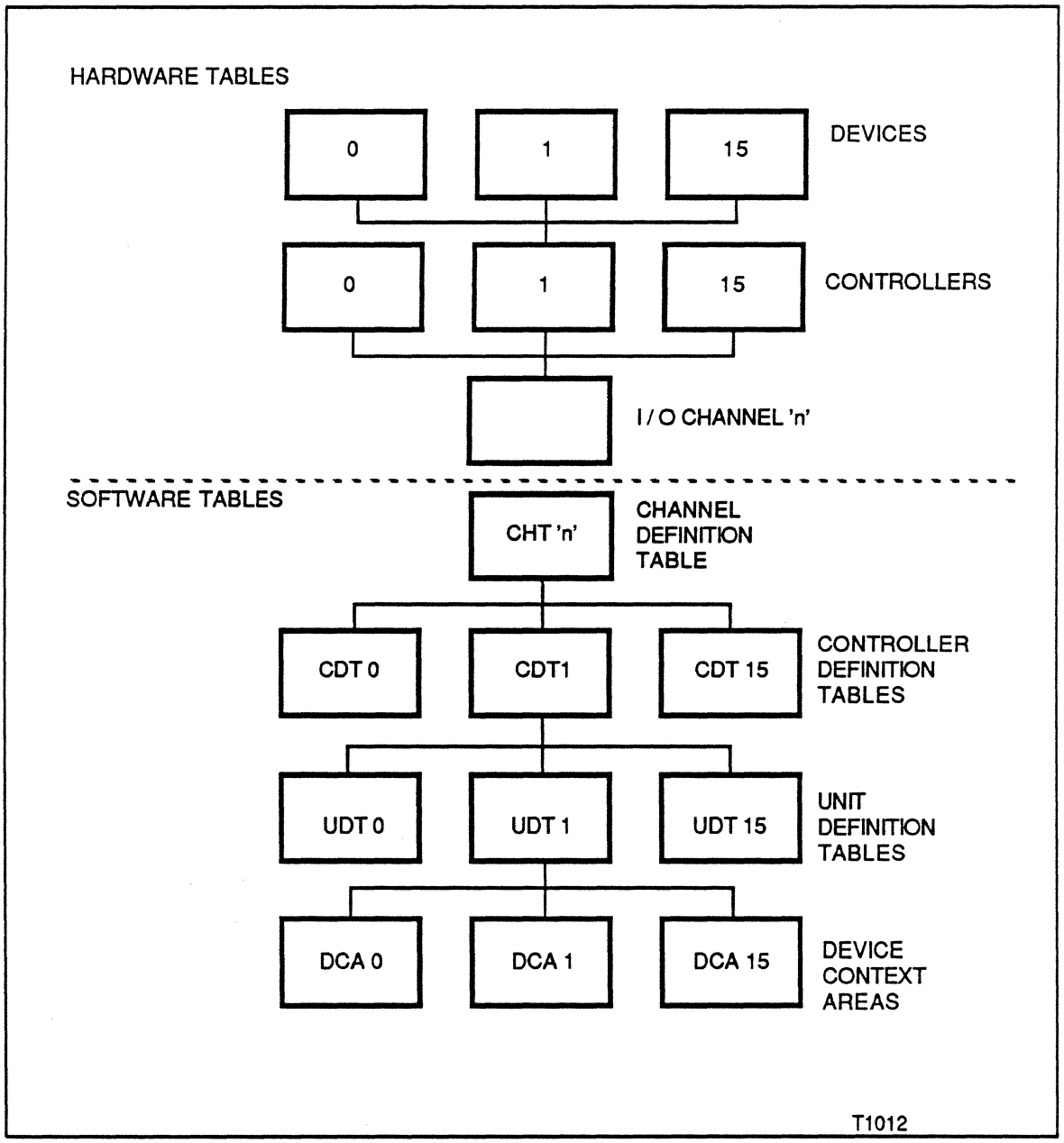


Figure 2-2
Handler Tables and Corresponding Hardware

2.20 M.KEY Entry Format

The M.KEY entry file is built by the KEY editor and interpreted by J.TSM. It is an unblocked, nonextendible file. Blank fields default to SYSTEM. All fields are in ASCII except the following:

- Access flags - See task service area (TSA) description of T.ACCESS for bit assignments.
- Tab settings - One byte per tab position. A zero indicates end of tabs. Maximum of eight tab positions.
- Key - Compressed key associated with owner name. Compression is done by H.FISE,8. A zero indicates no key.

	0 7	8 15	16 23	24 31
Word 0-1	Owner name			
2-3	Access flags			
4-5	Tab settings			
6	Reserved			
7	Owner key			
8-9	Project name			
10-11	Reserved			
12-15	Volume name			
16-19	Directory name			
20-23	Encrypted password			

2.21 M.PRJCT Format

The M.PRJCT file is built by J.PRJCT and is interpreted by J.TSM. It is an unblocked, nonextendible file. The project key is compressed by H.FISE,8. A zero indicates no key.

	0 7	8 15	16 23	24 31
Word 0-1	Project name			
2-6	Reserved			
7	Project key			
8-23	Reserved			

2.22 Map Image Descriptor List (MIDL)

The map image descriptor list (MIDL) entries are halfwords for all CONCEPT machines except the 32/2000 when running mapped out, which is a fullword entry. MIDL entries contain the physical map block numbers corresponding to a task's logical map blocks. The MIDL size varies depending on the type of task and the processor used.

The bit definitions in the MIDLs depend on whether MPX-32 is running mapped in or running mapped out, as on the CONCEPT 32/2000. If running mapped in, the old halfword definitions apply and if running mapped out, the new fullword definitions apply.

2.22.1 Halfword MIDL Entries

The contents of T.MIDLA point to the first MIDL entry.

For nonbase mode tasks, the maximum size of the MIDL is 256 maps minus the operating system size in maps.

If the TSM $SPACE command is used, the size of the MIDL is the logical address space, specified in maps, minus the operating system size in maps.

For base mode tasks, the size of the MIDL is the program's size, in maps, plus 16 maps if the task is greater than 2MB. If not, the MIDL size is 256 maps.

The memory attribute list (MEML) entries correspond one to one to the MIDL entries. The MEML describes the attributes associated with each map block.

Halfword MIDL entries have the following format:

0 4	5 15	0 4	5 15
Flags. See Note.	Physical map number	Flags	Physical map number

Notes:

Flag bits are assigned as follows:

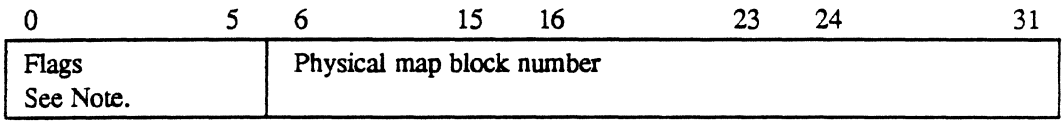
Bit	Meaning if Set
0	map number is valid (MIDL.VAL)
1	first protection granule is write protected (MIDL.PRO)
2	second protection granule is write protected (MIDL.PR2)
3	third protection granule is write protected (MIDL.PR3)
4	fourth protection granule is write protected (MIDL.PR4)

If bit 0 (MIDL.VAL) is set, the physical map number contains a valid map block number that represents an entry in the memory allocation table (MATA).

2.22.2 Fullword MIDL Entries

MIDL entries must be one fullword to support 256MB of physical memory, SRAM and/or DRAM. Fullword MIDL entries are used on any mapped out system image. The fullword MIDL is used by the CONCEPT 32/2000 processors only. All other processors use the halfword MIDL.

Fullword MIDL entries have the following format:



Notes:

Flag bits are assigned as follows:

Bit	Meaning if Set
0	map number is valid (MIDL.VAL)
1-2	map block has restrictions according to the following bit encoding:

PSD Priv.	P1	P2	
0	0	0	read/execute
0	1	0	read/write/execute
0	0	1	not used by MPX-32
0	1	1	not used by MPX-32
1	0	0	read/write/execute
1	1	0	read/write/execute
1	0	1	not used by MPX-32
1	1	1	not used by MPX-32

3	map block has been modified (MIDL.MOD)
4	map block has been referenced (MIDL.ACC)
5	indicates SelBUS (DRAM) memory (MIDL.SEL)

If bit 0 (MIDL.VAL) is set, the physical map number contains a valid map block number that represents an entry in the memory allocation table (MATA).

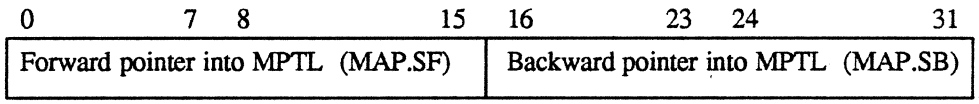
2.23 Memory Allocation Pointer List (MPTL)

The memory allocation pointer list contains pointers to head cells for free-page and page-out queues. Each entry has a forward pointer and a backward pointer. Each entry is one word in length. There is a one to one correspondence with the MATA, PST and PTE tables. The pointer value is a physical map block number which is used as an index into the MATA, PST and PTE tables. The table size is equal to one map block when physical memory is 16 MB. The address of the MPTL table is contained in C.MPTLA.

The MPTL is placed in a location which is unmapped from MPX-32 and is not included in the logical address space of a task. Therefore, MPX-32 must be executing unmapped when accessing the MPTL.

This table is created for all images.

The MPTL has the following format:



2.24 Memory Allocation Table (MATA)

The memory allocation table (MATA) contains the current status of each 2KW map block of main memory that is present in a configuration. The address of this table is contained in C.MATA.

Each MATA entry consists of a flag byte representing the status of a configured map block. There is one flag byte for every map block configured. The flag bytes are positional, relative to the first block of the configured class of memory.

0 7	8 15	16 23	24 31
Number of map blocks configured in system (MEM.CNT)		Starting map number for memory table (MEM.SMN)	
Flag bits (MEM.STAT). See Note.	MEM.STAT	MEM.STAT	MEM.STAT
MEM.STAT	MEM.STAT	etc.	etc.

Notes:

Flag bits in MEM.STAT are defined as follows:

Bit	Meaning if set
0	map block is allocated (MEM.ALL)
1	map block is shared (MEM.SHR)
2	map block is multiprocessor shared (MEM.PRO)
3	malfunction exists (MEM.MAL)
4	map block is nonpresent (MEM.CON)
5	map block is initialized (MEM.INIT)
6	defined below (MEM.CL1)
7	defined below (MEM.CL2)

Bit 6	Bit 7	Class
0	0	E
0	1	H
1	0	S
1	1	D*

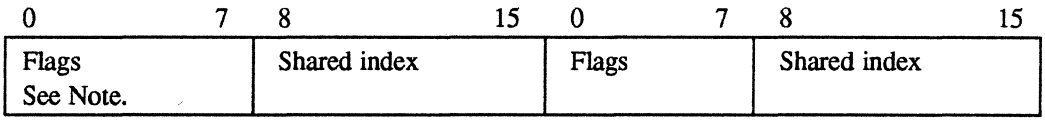
* CONCEPT 32/2000 only

2.25 Memory Attribute List (MEML)

The memory attribute list (MEML) entries correspond one-to-one to the map image descriptor list (MIDL) entries. The MEML describes the attributes associated with each map block. The maximum number of MEML entries is the same as MIDL entries (see section 2.22). The contents of T.MEMLA point to the first MEML entry.

2.25.1 Halfword MEML Format

Halfword MEML entries have the following format:



Notes:

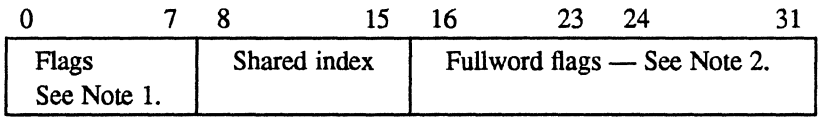
Flag bits are assigned as follows:

Bit	Meaning if set
0	E-type memory (MEML.TYE)
1	H-type memory (MEML.TYH)
2	S-type memory (MEML.TYS)
3	map is shared (MEML.SHR)
4	map is swappable (MEML.SWP)
5	map is valid (MEML.VAL)
6	map block used by system (MEML.SYS)
7	map is outswapped (MEML.OUT)

If bit 3 (MEML.SHR) is set, the shared index contains the index into the associated shared memory table where the map block has been allocated.

2.25.2 Fullword MEML Format

Fullword MEML entries have the following format:



Notes:

1. Flag bits are assigned as follows:

Bit	Meaning if set
0	E-type memory (MEML.TYE)
1	H-type memory (MEML.TYH)
2	S-type memory (MEML.TYS)
3	map is shared (MEML.SHR)
4	map is swappable (MEML.SWP)
5	map is valid (MEML.VAL)
6	map block used by system (MEML.SYS)
7	map is outswapped (MEML.OUT)

   If bit 3 (MEML.SHR) is set, the shared index contains the index into the associated shared memory table where the map block has been allocated.

   If bits 0-2 are not set, then DRAM has been specified (CONCEPT 32/2000 only).

2. The fullword flags are set as follows:

Bit	Meaning if set
16	page contents located in swap file (MEML.SPF)
17	page contents located in load module file (MEML.LDM)
18	page is in process of being paged in (MEML.PGI)
19	page is in process of being paged out (MEML.PGO)
20	map block is available for page out (MEML.PG)
21	zero the page contents on first page in (MEML.ZER)
22-31	reserved

2.26 Memory Pool Management

Memory pool is an area of main memory beginning at the high address end of resident MPX-32. Its size is specified at SYSGEN and it occupies an area up to the next map block boundary. It is used as temporary storage space by various system services. It contains, at any one time, line buffers, I/O queues, messages, IOCD lists, etc., that are dynamic and not predefined in size. C.SBUF contains the address of memory pool. C.SBUF+1W contains the number of words in memory pool.

If the memory pools, IOQPOOL and MSGPOOL, are specified at SYSGEN then IOQPOOL is the doubleword bounded area at the end of resident MPX-32 and MSGPOOL occupies the next doubleword bounded area. All other memory pool is then called miscellaneous pool, and occupies the next contiguous space. It is doubleword bounded and ends on a map block boundary. C.SBUFA and C.SBUFA+1W contain the address and the number of words, respectively, in IOQPOOL. C.SBUFB and C.SBUFB+1W contain the address and number of words, respectively, in MSGPOOL.

Areas within memory pool are allocated in multiples of two words (doubleword bounded). A free list and allocated list of buffer areas are maintained. They are in double-linked list format and are linked to head cells located in S.MEMM9 to control allocation and deallocation of memory pool areas. Entries are linked in ascending memory address order.

An allocation request for memory pool is obtained from the first free space large enough to satisfy the request. A four-word header is built and linked by address to the allocated head cell. The free area entry header is updated to reflect the size reduction caused by the requested allocation.

A deallocation request for memory pool causes the allocated list to be searched for the allocated entry. Verification is made on the buffer address and size provided by the caller. If valid, the entry is moved to the free list and agglomeration attempted with previous and next entries.

When the debugger and event trace are present in the system, the debugger prompt is displayed if an abnormal condition is detected when using the S.MEMM10 system subroutine to deallocate memory pool. The following register definitions are displayed in response to the debugger DR command:

Register	Contents
1	abort condition as follows:

Abort Condition	Description
1	buffer address in register three is not in memory pool
2	buffer address in register three is not allocated
3	invalid byte count is specified in the deallocation request

2	buffer header address if R1 is three
3	buffer address from the caller if R1 is one or two
4	free head cell address if R1 is one or two
7	invalid byte count from the caller if R1 is three

If the debugger TE command is entered to the debugger prompt and R1 is one or two, the request is ignored.

If the debugger TE command is entered to the debugger prompt and R1 is three, the request is continued using the corrected count obtained from the buffer header that was originally allocated (R2 address plus three words).

Buffer Header Entry

The address in R2 points to the following structure:

Forward link	Link address to next buffer header entry
Backward link	Link address to last buffer header entry
Address of this header	Address of this buffer header entry
Number of words in buffer	Number of words, excluding header, in this area
Memory pool buffer area	User buffer area

Free Head Cell

The address in R4 points to the following structure:

Forward link	Next free entry address
Backward link	Last free entry address
Address of this header	Address of this head cell
Number of entries in list	Number of entries in free list

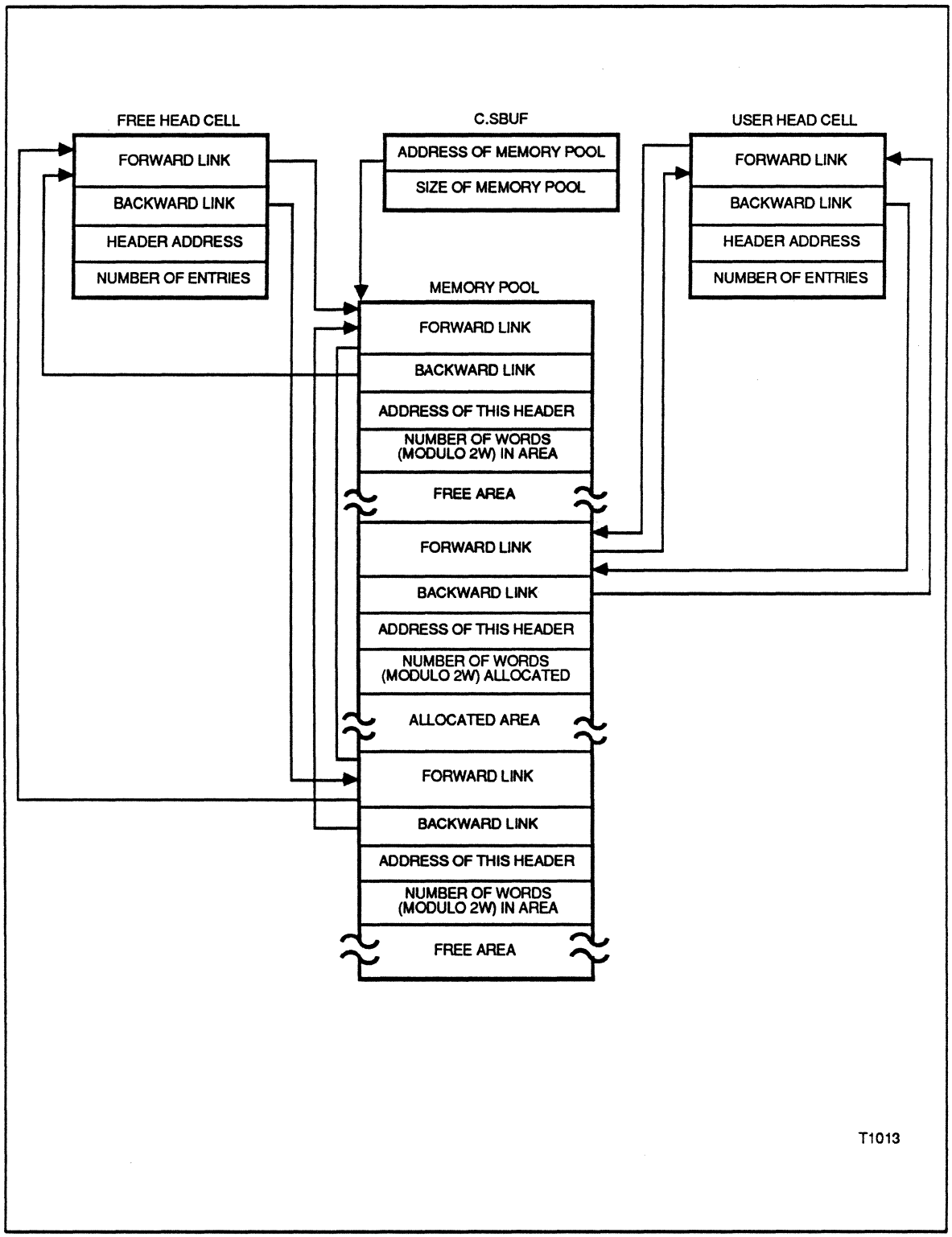


Figure 2-3
Memory Pool Diagram

2.27 Memory Resident Descriptor Table (MDT)

The memory resident descriptor table (MDT) is a table of resource descriptors that resides in main memory. The resource descriptors in the MDT are exact copies of the resource descriptors that reside on the disk.

	0 7 8 15 16 23 24 31
Word 0-191	First 192-word resource descriptor entry
192-384	Second 192-word resource descriptor entry
385	. . .
n	Last 192-word resource descriptor entry
n + 191	

2.28 Message or Run Request Queue (MRRQ)

H.EXEC creates this parameter block with a user-generated parameter send block (PSB). It is used for message and run request processing.

	0 7 8 15 16 23 24 31			
Word 0	String forward address (MQ.SF)			
1	String backward address (MQ.SB)			
2	Priority (MQ.PR)	Parameter send block address (MQ.PSBA)		
3	Task number of sending task (MQ.TNST)			
4	Sending task owner name word one			
5	Sending task owner name word two			
6	Passed parameter quantity (MQ.PPQ)		Sending task return buffer length (MQ.RBL)	
7	Completion status (MQ.CST). See Note 1.	Initial status (MQ.IST). See Note 2.	User status (MQ.UST)	Options (MQ.OPT). See Note 3.
8-9	End action PSD (MQ.EAPSD)			
10	Parameter area pointer (MQ.PPTR)			
11	Reserved			
n	Variable length storage area for passed and returned parameters			

Notes:

1. Bits in MQ.CST are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
0	operation in progress (PSB.IOP)
1	destination task aborted (PSB.DTA)
2	destination task deleted (PSB.DTD)
3	return parameters truncated (PSB.RPT)
4	send parameters truncated (PSB.SPT)
5	end action routine not processed (PSB.EANP)
6-7	reserved

2. Codes in MQ.IST are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
0	normal initial status
1	task number invalid (PSB.TSKE)
2	load module name error (PSB.LMNE)
3	reserved
4	load module format error (PSB.LMFE)
5	DQE space unavailable (PSB.DQEE)
6	I/O error reading directory (PSB.SMIO)
7	I/O error reading load module (PSB.LMIO)
8-9	reserved
10	invalid priority (PSB.PRIE)
11	invalid send buffer address (PSB.SBAE)
12	invalid return buffer address (PSB.RBAE)
13	invalid end action address (PSB.EAE)
14	memory pool unavailable (PSB.MPE)
15	destination task receiver queue full (PSB.DTQF)

3. Bits in MQ.OPT are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
24	no-wait mode (PSB.NWM)
25	no call-back mode (PSB.NCBM)
26-31	reserved

2.28.1 Remote Messaging Request Queue

The following structure is used in remote message request processing for Refective Memory System Software only.

	0 7	8 15	16 23	24 31
Word 0	String forward address (MQ.SF)			
1	String backward address (MQ.SB)			
2	Priority (MQ.PR)	Parameter send block address (MQ.PSBA)		
3	Zero			
4	Sending task owner name word 1			
5	Sending task owner name word 2			
6	Passed parameter quantity (MQ.PPQ)		Sending task return buffer length (MQ.RBL)	
7	Completion status (MQ.CST). See Note 1.	Initial status (MQ.IST). See Note 2.	User status (MQ.UST)	Options (MQ.OPT). See Note 3.
8-9	End action PSD (MQ.EAPSD)			
10	Parameter area pointer (MQ.PPTR)			
11	Reserved			
12	(MQ.RTNST) — Remote task number. See Note 4.			
13	(MQ.NIDST) — Remote node ID. See Note 5.			
n	Variable length storage area for passed and returned parameters			

Notes:

1. Bits in MQ.CST are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
0	operation in progress (PSB.IOP)
1	destination task aborted (PSB.DTA)
2	destination task deleted (PSB.DTD)
3	return parameters truncated (PSB.RPT)
4	send parameters truncated (PSB.SPT)
5	end action routine not processed (PSB.EANP)
6-7	reserved

2. Codes in MQ.IST are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
0	normal initial status
1	task number invalid (PSB.TSKE)
2	load module name error (PSB.LMNE)
3	reserved
4	load module format error (PSB.LMFE)
5	DQE space unavailable (PSB.DQEE)
6	I/O error reading directory (PSB.SMIO)
7	I/O error reading load module (PSB.LMIO)
8-9	reserved
10	invalid priority (PSB.PRIE)
11	invalid send buffer address (PSB.SBAE)
12	invalid return buffer address (PSB.RBAE)
13	invalid end action address (PSB.EAE)
14	memory pool unavailable (PSB.MPE)
15	destination task receiver queue full (PSB.DTQF)

3. Bits in MQ.OPT are the same as the parameter send block (PSB) format and are assigned as follows:

Bits	Meaning if Set
24	no-wait mode (PSB.NWM)
25	no call-back mode (PSB.NCBM)
26-31	reserved

4. Remote task number (MQ.RTNST) — this field is present as part of the fixed header length for messaging performed by RMSS 3.0 release.
5. Remote node ID (MQ.NIDST) — this field is present as part of the fixed header length for messaging performed by RMSS 3.0 release.

2.29 Module Address Table

The module address table contains the addresses of the resident system modules as they are referenced by the M.CALL macros. The address of this table is contained within C.MODD.

	0 7 8 15 16 23 24 31
Word 0	Reserved
1	Address of H.EXEC
2	Address of H.MONS
3	Address of H.IOCS
4	Address of H.FISE
5	Address of H.ALOC
6	Address of H.MEMM
7	Address of H.TSM
8	Address of H.TAMM
9	Address of H.REXS
10	Address of H.REMM
11	Address of H.VOMM
12	Reserved for ACX

2.30 Mounted Volume Table (MVT)

The mounted volume table (MVT) is a system resident table built at SYSGEN (its size is variable and determined by the SYSGEN process). The MVT is used to create and maintain an entry for each volume physically mounted on the system. Each entry is identified by the volume name and associated with the device the volume is mounted on. The entry contains a use count for the volume.

For nonpublic volumes, the use count in the MVT represents the number of tasks that are logically mounted to the volume. The count is decremented each time a logical dismount completes on behalf of a task that has the volume logically mounted but has no resources allocated on the volume. For the task J.TSM, a logical dismount of a nonpublic volume can only complete as a result of a logical dismount of the last TSM environment that has mounted the volume.

For public volumes, the use count is only maintained after a request for a physical dismount of that volume. While physical dismount is pending, the count represents the total resources allocated on the volume. The count increments or decrements as resources are allocated or deallocated on the volume.

When the MVT use count goes to zero and a physical dismount is pending, a physical dismount of the volume is performed. The MVT entry for the volume is cleared.

	0 7 8 15 16 23 24 31
Word 0-3	Volume name (MV.VOLNM)
4	CDT address of volume device (MV.CDTA)
5	UDT address of volume device (MV.UDTA)
6	Current number of users of this volume (MV.USERS)
7	Current number of temporary files allocated (MV.TFILS)
8	Space map start address (MV.SMAPS)
9	Space allocation map length in blocks (MV.SMAPL)
10	Number of allocation units reflected in space map (MV.SMAPU)
11	Number of allocation units currently available (MV.SMAPC). See Note 1.
12	Descriptor allocation map start address (MV.DMAPS)
13	Descriptor allocation map length in blocks (MV.DMAPL)
14	Number resource descriptors in descriptor map (MV.DMAPU)
15	Number resource descriptors currently available (MV.DMAPC)
16	Root directory segment definition (MV.ROOTS)
17	Root directory segment definition (MV.ROOTL)
18	Blocks per allocation unit (MV.BLKAU)
19	Number of blocks in system area (MV.SYSBL)
20-34	Dismount owner name (MV.DOWNR)
35	Volume flags (MV.FLAGS). See Note 2.
36	Associated SYSID, if multiprocessor volume (MV.SYSID)
37	Port (MV.MPID). See Note 3. / Reserved
38	Count of I/O errors since the last mount (MV.CAECT)
39	Reserved

Notes:

1. This number includes some blocks allocated by the operating system.
2. Bits in MV.FLAGS are assigned as follows:

Bits	Meaning if Set
0	system volume (MV.SYS)
1	volume is physically mounted (MV.OPER)
2	fixed media volume (MV.FIXED)
3	public volume (MV.PUBLC)
4	entry is active (in use) (MV.ACTV)
5	mount in progress (MV.MNT)
6	inhibit mount message (MV.NOMSG)

Bits	Meaning if Set
7	swap volume (MV.SWP)
8	volume device offline (MV.OFFLN)
9	volume not safe for use (MV.UNSAF)
10	physical dismount pending (no mounts allowed) (MV.DMNT)
11	DMAP locked (MV.DLOCK)
12	space map lock (MV.SLOCK)
13	root directory locked (MV.RLOCK)
14	multiprocessor volume flag (MV.DUALP)
15	lock all volume resources so physical dismount can occur (MV.PHDLK)
16	reserved for future development (MV.PRIM)
17-26	reserved
27	mounted for shared use (MV.SHRBL)
28-31	reserved

3. Port number under which multiport volume was mounted.

2.31 Physical Shared Memory Table (PSM)

The physical shared memory (PSM) table is a system resident structure used to define the reflective regions of memory to Reflective Memory Support Software (RMSS) tasks. It is created at SYSGEN and contains one entry for each Reflective Memory (RM) Bus.

	0 7	8 15	16 23	24 31
Word 0	PSM.STRT RM starting address		PSM.LEN RM length in map blocks	
1	PSM.CRAD (See Note 1)	PSM.NCNT (See Note 2)	PSM.INTI (See Note 3)	PSM.INTO (See Note 4)
2-5	PSM.BUSN Bus name as defined by M.BOOT (4 words)			
6	PSM.SMP Physical address of POOLPAR*n*			
7	PSM.RCAA Physical Address of Remote Context Area			
8	PSM.FLAG (Note 5)			
9	PSM.RI RI Instruction used by H.RMSS			

Notes:

1. PSM.CRAD contains the control register offset from location X'700' in memory. This value is set at SYSGEN.
2. PSM.NCNT contains the number of nodes on the bus. The number of nodes indicates the number of remote context areas (RCAs) in the POOLPAR*n* region.

   PSM.NCNT is initialized by J.BOOT.
3. PSM.INTI contains the input interrupt level for remote task activations.
4. PSM.INTO contains the output interrupt level for remote task activations.
5. PSM.FLAG contains one of the following bit settings:

Bit	Equate	Description
0	PSM.BERR	set if bus error during J.BOOT parsing of M.BOOT file. J.BOOT sets this flag.
1	PSM.PRIM	set if bus is primary bus for this node. J.BOOT sets this flag.
2	PSM.HOST	set if node is the host node for this bus. J.BOOT sets this flag.
3	PSM.OVL	set if an overlap occurs between BOOTPAR and COMSPAR regions.
4	PSM.VAL	set if the interrupt handlers have been initialized. When set, this flag indicates that this bus is valid for remote task activation (RTA).
5	PSM.OFLP	set if an offline is pending (SVC M.RMOFF in progress).
6	PSM.ONLP	set if an online is in progress (SVC M.RMON in progress).
7	PSM.OFFL	set if bus is currently offline.
8	PSM.BOOT	set if J.BOOT is currently active.

2.32 Resource Create Block (RCB)

The resource create block (RCB) is a doubleword bounded data structure which defines the attributes of a resource (permanent file, temporary file, memory partition, or directory) created by a Volume Management Module (H.VOMM) entry point.

	0 7 8 15 16 23 24 31
Word 0-1	Resource owner name (RCB.OWNR)
2-3	Resource project group name (RCB.USER)
4	Resource owner rights specifications (RCB.OWRI). See Note 1.
5	Resource project group rights specifications (RCB.UGRI). See Note 1.
6	Resource others rights specifications (RCB.OTRI). See Note 1.
7	File management flags (RCB.SFLG). See Note 2.
8	Maximum file extension increment (RCB.MXEI)
9	Minimum file extension increment (RCB.MNEI)
10	Maximum file size (RCB.MXSZ) (or) starting physical page (RCB.PPAG)
11	Original resource size (RCB.OSIZ)
12	Resource starting address (RCB.ADDR)
13	Resource RID buffer (RCB.FAST). See Note 3.
14	Option flags (RCB.OPTS). See Note 4.
15	Default override (RCB.FREE). See Note 5.

All RCB fields are optional. If the owner and project group names are not specified, the names default to those used by the calling task. All other fields not specified use the system defaults.

Notes:

1. Bits in RCB.OWRI, RCB.UGRI and RCB.OTRI are assigned as follows:

Bits	Meaning if Set
0	read access (RCB.READ)
1	write access (RCB.WRIT)
2	modify access (RCB.MODI)
3	update access (RCB.UPDA)
4	append access (RCB.APPN)
5-7	reserved
8	traverse directory access (RCB.TRAV)
9	delete resource access (RCB.DELE)
10	delete directory entry access (RCB.DEEN)
11	add directory entry access (RCB.ADEN)
12-31	reserved

Bits 2, 3, and 4 are not valid for use with memory partitions.

2. Bits in RCB.SFLG are assigned as follows (for any bit not set, system defaults apply):

Bits	Meaning if Set
0-7	resource type, equivalent to file type code, interpreted as two hexadecimal digits, 0 through FF (RCB.FTYP)
8-10	reserved
11	file EOF management required (RCB.EOFM)
12	resource fast access (RCB.FSTF)
13	resource not to be saved (RCB.NSAV)
14	reserved for MPX-32 usage
15	start block requested (RCB.SREQ)
16	file is executable (RCB.EXEC)
17	owner ID set on access (RCB.OWID)
18	project group ID set on access (RCB.UGID)
19	reserved
20	maximum file extension increment is zero. System default value not used. (RCB.MXEF)
21	minimum file extension increment is zero. System default value not used. (RCB.MNEF)
22	reserved
23	file zeroed on creation/expansion (RCB.ZERO)
24	file automatically extendible (RCB.AUTO)
25	file manually extendible (RCB.MANU)
26	file contiguity is desired (RCB.CONT)
27	resource is sharable (RCB.SHAR)
28	link access (RCB.LINK)
29-30	reserved
31	file data initially recorded as blocked (RCB.BLOK)

3. The resource RID buffer is the address within the resource creator's task where the eight word resource identifier (RID) is to be returned. If this parameter is not supplied (i.e. is zero), the RID for the created resource

4. Bits in RCB.OPTS are assigned as follows:

Bits	Meaning if Set
0	owner has no access rights (RCB.OWNA)
1	project group has no access rights (RCB.USNA)
2	others have no access rights (RCB.OTNA)
3-6	reserved
7	multi-segment create
8	spool file type (RCB.SPOO)
9	defines a static partition (RCB.STAT)
10-15	reserved
16-23	maximum segments at creation (RCB.SEGN)
24-31	defines memory class:

Value	Class
0	S (default)
1	E
2	H
3	S

5. Bits in RCB.FREE are assigned as follows (these bits override any corresponding bit set in RCB.SFLG and the system defaults):

Bits	Meaning if Set
0-7	must be zero
11	file EOF management not required
12	fast access not required
13	resource can be saved
23	do not zero file on creation/extension
24	file is not automatically extendible
25	file is not manually extendible
26	file contiguity is not desired
27	resource is not sharable
31	file data initially recorded as unblocked

2.33 Resource Inquiry Table (M.RIQ)

The resource inquiry table (M.RIQ) contains information specific to an allocated resource. The information is returned in the form of a series of pointers to various data structures within the system which describe the resource. For memory partitions only, words zero and five apply. For volume resources, words two through four apply to the device where the volume is mounted.

	0 7 8 15 16 23 24 31
Word 0	Address of allocated resource table entry (RIQ.ART)
1	Address of file assignment table entry (RIQ.FAT)
2	Address of unit definition table entry (RIQ.UDT)
3	Address of device type table entry (RIQ.DTT)
4	Address of controller definition table entry (RIQ.CDT)
5	Address of shared memory table entry (RIQ.SMT)
6	Address of file pointer table entry (RIQ.FPT)
7	Address of mounted volume table entry (RIQ.MVT)

2.34 Resource Logging Block (RLB)

The resource logging block (RLB) is a word-bounded data structure used to pass information between H.VOMM and the caller. The information is used to locate a directory entry and resource descriptor for a single resource or for all resources defined in a particular directory.

	0 7	8 15	16 23	24 31
Word 0	Pathname vector or RID address (RLB.TGT)			
1	Resource directory buffer address (192W) (RLB.BUFA). See Note 1.			
2	Associated mounted volume table entry address (RLB.MVTE)			
3	Parent directory RD block address (RLB.RDAD)			
4	Type (RLB.TYPE). See Note 2.	Buffer offset (RLB.BOFF)		
5	Length. See Note 3.	Return buffer address (RLB.DIRA)		
6	User FCB address (RLB.FCB)			
7	Flags. See Note 4.	Reserved (RLB.INT)		

Notes:

1. Optional. If not specified, a resource directory is not returned.
2. Bits in RLB.TYPE are assigned as follows:

Bits	Meaning if Set
0	indicates recall (RLB.RECA)
1-7	reserved

3. This word contains the address of a buffer and its length in words (the buffer can be up to 16 words long).
4. Bits in the flags byte are assigned as follows:

Bits	Meaning if Set
0-1	reserved
2	directory entry and resource descriptor for specified directory are returned
3	root directory
4	resource is located
5-7	reserved

2.35 Resource Requirement Summary (RRS) Entries

The resource requirement summary (RRS) is a doubleword bounded data structure used to identify the resources required by a task to the resource manager. Resources are statically allocated using the information in the RRS entry. The RRS is generally built by processors requiring static allocation of resources, such as TSM, cataloger, etc., or supplied as an argument for dynamic allocation.

For compatibility purposes, revision 1.*x* RRS formats can be used. The details of these formats can be found in Chapter 2 of a revision 1.*x* Technical Manual.

Type 1 - Assign by Pathname

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Plength (RR.PLEN)	Reserved. See Note 2.
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4-*n*	Pathname (variable length) (RR.NAME1)			

Type 2 - Assign to Temporary File

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Initial file size (RR.PLEN)	
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4-7	Volume name (16 characters; left-justified, blank-filled) (RR.NAME1) (Volume name is optional)			

Type 3 - Assign to Device

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Density (RR.DENS). See Note 5.	Zero
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4	Device type (RR.DT3). See Note 6.	Volume number (RR.VLNUM)	Channel number See Note 7. (RR.CHN3)	Subchannel number (RR.SCHN3)
5	Unformatted ID (1-4 characters) (RR.UNFID)			

Type 4 - Assign to LFC

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Zero	
2	Zero	Logical file code (RR.SFC)		
3	Options (RR.OPTS). See Note 4.			

Type 5 - Assign by Segment Definition

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	UDT index (RR.UDTI)	Reserved
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4	Starting block number (RR.STBLK)			
5	Number of blocks (RR.NBLKS)			

Type 6 - Assign by Resource ID

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Zero	Reserved
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4-7	Volume name (16 characters; left-justified, blank-filled) (RR.NAME1)			
8	Binary creation date (RR.DATE)			
9	Binary creation time (RR.TIME)			
10	Resource descriptor block address (RR.DOFF)			
11	Reserved		Resource type (RR.RTYPE)	

Type 7 - Reserved for Future Use

Type 8 - Reserved for Future Use

Type 9 - Mount by Device Mnemonic

	0 7	8 15	16 23	24 31
Word 0	Zero	System ID (RR.SYSID). See Note 11.		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Zero	
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4-7	Volume name (16 characters; left-justified, blank-filled) (RR.NAME1)			
8	Device type (RR.DT9). See Note 8.	Reserved	Channel number (RR.CHN9). See Note 9.	Subchannel number (RR.SCHN9)
9	Zero			

Type 10 - Assign to ANSI Tape

	0 7	8 15	16 23	24 31
Word 0	Zero	Logical file code (RR.LFC)		
1	Type (RR.TYPE). See Note 1.	Size (RR.SIZE)	Format (RR.FORM)	Protect (RR.PROT)
2	Access (RR.ACCS). See Note 3.			
3	Options (RR.OPTS). See Note 4.			
4	Record length (RR.RECL)		Block size (RR.BSIZE)	
5	Generation number (RR.GENN)			
6	Generation version number (RR.GENV)			
7	Absolute termination date (RR.EXPIA)			
8	Relative termination date (RR.EXPIR)		Logical volume identifier (RR.LVID)	
9	RR.LVID (cont.)			
10-13	17-character file identifier (RR.AFID)			
14	RR.AFID (cont.)	Reserved		
15	Reserved			

Type 11 - Assign to Shadow Memory

	0 7	8 15	16 23	24 31
Word 0	Zero			
1	Type (RR.TYPE) See Note 1.	Size (RR.SIZE)	Shadow flags (RR.SHAD). See Note 10.	
2	Start address (RR.SADD)			
3	End address (RR.EADD)			

Notes:

1. Bits in RR.TYPE are assigned as follows:

Value	Meaning
1	assign by pathname (RR.PATH)
2	assign to temporary file (RR.TEMP)
3	assign to device (RR.DEVC)
4	assign to secondary LFC (RR.LFC2)
5	assign to segment definition (RR.SPACE)
6	assign by resource ID (RR.RID)
7	reserved for future use
8	reserved for future use
9	mount by device mnemonic (RR.MTDEV)
10	assign to ANSI labeled tape (RR.ANS)
11	assign to shadow memory (RR.SHTYP)
12-255	reserved

2. Byte 3 is zero. This field is used by MPX-32 for big blocking buffers.
3. Bits in RR.ACCS are assigned as follows:

Bits	Meaning if Set
0	read access allowed (RR.READ)
1	write access allowed (RR.WRITE)
2	modify access allowed (RR.MODFY)(not valid for ANSI tapes)
3	update access allowed (RR.UPDAT)
4	append access allowed (RR.APPND)
5-15	reserved
16	explicit shared use requested (RR.SHAR)
17	exclusive use requested (RR.EXCL)
18	assign as volume mount device (RR.MNT)
19-31	reserved

4. Bits in RR.OPTS are assigned as follows:

Bits	Meaning if Set
0	treat as SYC file (RR.SYC) (TSM/JOB only)
1	treat as SGO file (RR.SGO) (TSM/JOB only)
2	treat as SLO file (RR.SLO)
3	treat as SBO file (RR.SBO)
4	explicit blocked option (RR.BLK)
5	explicit unblocked option (RR.UNBLK)
6	inhibit mount message (RR.NOMSG)
7	reserved for system use
8	automatic open requested (RR.OPEN)
9	user-supplied blocking buffer address in FCB (RR.BUFF)
10-11	reserved for system use
12	mount with no-wait (RR.NOWT)
13	mount as public volume (RR.PUBLC)
14	set by H.VOMM for special case handling of VOMM assignments (RR.VOMM)
15	file is spooled when deallocated (RR.SEP)
16	ANSI labeled tape on RRS type 3 (RR.ANSI)
17-31	reserved

5. RR.DENS contains the density specification for XIO high speed tape units. When specified, this field has the following bit significance:

Bits	Meaning if Set
0	indicates 800 bpi nonreturn to zero inverted (NRZI)
1	indicates 1600 bpi phase encoded (PE)
6	indicates 6250 bpi group coded recording (GCR)

If this field is zero, 6250 BPI is set by default.

6. RR.DT3 specifies whether or not a channel is present and specifies the device type:

Bits	Meaning if Set
0	channel present
1-7	device type

7. RR.CHN3 specifies whether or not a subchannel is present and specifies the channel number:

Bits	Meaning if Set
0	subchannel is present. Examined only if bit zero of RR.DT3 is set.
1-7	channel number

8. RR.DT9 specifies whether or not a channel is present and specifies the device type:

Bits	Meaning if Set
0	channel present
1-7	device type

9. RR.CHN9 specifies whether or not a subchannel is present and specifies the channel number:

Bits	Meaning if Set
0	subchannel is present. Examined only if RR.DT9 is set.
1-7	channel number

10. RR.SHAD contains the shadow flags that qualify the start and end addresses, or specify what portions of the task are to be shadowed:

Bits	Meaning if Set
0-7	reserved
8	shadow the task (RR.SHTSK)
9	shadow the TSA (RR.SHTSA)
10	shadow the stack (RR.SHST)
11	shadow memory is required (RR.SHRQ)
12	shadow the entire task (RR.SHALL)
13	absolute address (RR.ABS)
14	relative to the code section origin (RR.CREL)
15	relative to the data section origin (RR.DREL)

11. RR.SYSID is the ID for mounting a multiprocessor volume. Valid IDs are:

    Multiported (MP) 0 through F

    Dual Ported (DP) 0 or 1

For more information on mounting multiprocessor volumes see the MPX-32 Reference Manual Volume I, Chapter 4, Mounting Multiprocessor Volumes.

2.36 Shared Memory Table (SMT)

Each entry in the shared memory table (SMT) defines a shared memory area, such as CSECT, Global Common or Datapool. The number of entries in the SMT is established by the SYSGEN SHARE directive.

C.SMTA contains the address of the SMT; C.SMTN contains the number of entries in the SMT. Each entry is doubleword bounded.

	0 7	8 15	16 23	24 31
Word 0-7	Resource identifier (SMT.RID)			
8-9	Partition name (SMT.NAME)			
10-11	Owner name or task number associated with this partition inclusion (SMT.TNUM)			
12-13	Owner name of partition creator (SMT.OWNR)			
14-15	Project group of partition creator (SMT.PROJ)			
16-17	Swap file resource ID (SMT.SRID)			
18	Compatibility version level (SMT.COMP)			
19	Pathname identifier (SMT.PNID)			
20	SMT index (SMT.IND)	Address of associated allocated resource table entry (SMT.ARTA)		
21	Partition flags (SMT.FLAG). See Note 1.			
22	Starting 512-word page number (SMT.PAGE)		Total number of pages (SMT.PTOT)	
23	Starting map block (SMT.MAPS) See Note 2.		Number of map blocks (SMT.MAPN). See Note 2.	
24	Start of DSECT (SMT.DSES)		Number of blocks in DSECT (SMT.DSEN)	
25	Start of CSECT (SMT.CSES)		Number of blocks in CSECT (SMT.CSEN)	
26	Memory type (SMT.MTY)	Number of tasks not outswapped (SMT.UCNT)	Number of words of memory pool used for the SMT's MIDL (SMT.POOL)	
27	File offset of the write back read/write section within the shared image disk file (SMT.WBKS)		Reserved	
28	Address of the map image descriptor list (SMT.MIDL)			
29	Size in bytes of the read/write section to be written back (SMT.WBKN)			

	0 7	8 15	16 23	24 31
Word 30	Number of swappable E-class memory map blocks (SMT.CME)		Number of swappable H-class memory map blocks (SMT.CMH)	
31	Number of swappable S-class memory map blocks (SMT.CMS)		Number of outswapped map blocks (SMT.OTSW)	
32-33	Time when shared CSECT was cataloged or time when executable image was linked (SMT.TIME)			
34-35	Date when shared CSECT was cataloged or date when executable image was linked (SMT.DATE)			
36	Spare bytes (SMT.SPBT)			Swapper trial use count (SMT.TUC)
37-39	Reserved			

Notes:

1. Bits in SMT.FLAG are assigned as follows:

Bit	Meaning if Set
0	entry defines CSECT partition (SMT.CSCT)
1	entry defines static common (SMT.STCM)
2	entry defines dynamic common (SMT.DYCM)
3	entry defines shared image (SMT.SHIM)
4	partition is swappable (SMT.SWBL)
5	partition is currently outswapped (SMT.OUTS)
6	SMT is unstable (SMT.BLDG)
7	multicopy shared image (SMT.MULT)
8	previously outswapped (CSECT only) (SMT.PRSW)
9	no automatic dequeue (SMT.LOCK)
10	owner has read access (SMT.OW.R)
11	owner has write access (SMT.OW.W)
12	project group has read access (SMT.PJ.R)
13	project group has write access (SMT.PJ.W)
14	others have read access (SMT.OT.R)
15	others have write access (SMT.OT.W)
16	write back on last deallocation required (SMT.WRBK)
17	SMT active (SMT.ACTV)
18	partition established by OPCOM (SMT.OPCM). See Note 3.
19	partition requires shadow memory (SMT.SHAD)
20	shared image is sharing by owner group (SMT.SHBO)
21	entry defines memory disk (SMT.MD)
22	trial swap (SMT.TS)
23	CSECT is loaded (SMT.CSLD)
24	resource is remote (SMT.REM)
25	logical include on this SMT (SMT.LIN)
26	SMT describes shared memory that can be demand paged (SMT.DPG)

2. For a single-copy shared image partition, SMT.MAPN reflects the size of both the read only and read/write sections and SMT.MAPS reflects the start of the read-only section.

   For a multicopy shared image partition, SMT.MAPN reflects the size of the read-only section and SMT.MAPS reflects the start of the read only section. Data for the read/write section is obtained from disk.

3. An OPCOM-established dynamic partition or shared image remains in memory when its assign count goes to zero. A user task can establish a resident dynamic partition or shared image by including it, setting SMT.OPCM, and then excluding the partition or shared image.

2.37 Spooled File Data Structures

The organization of input and output for spooled files is shown in the following illustration. The remainder of this section describes the input and output spooling processes separately.

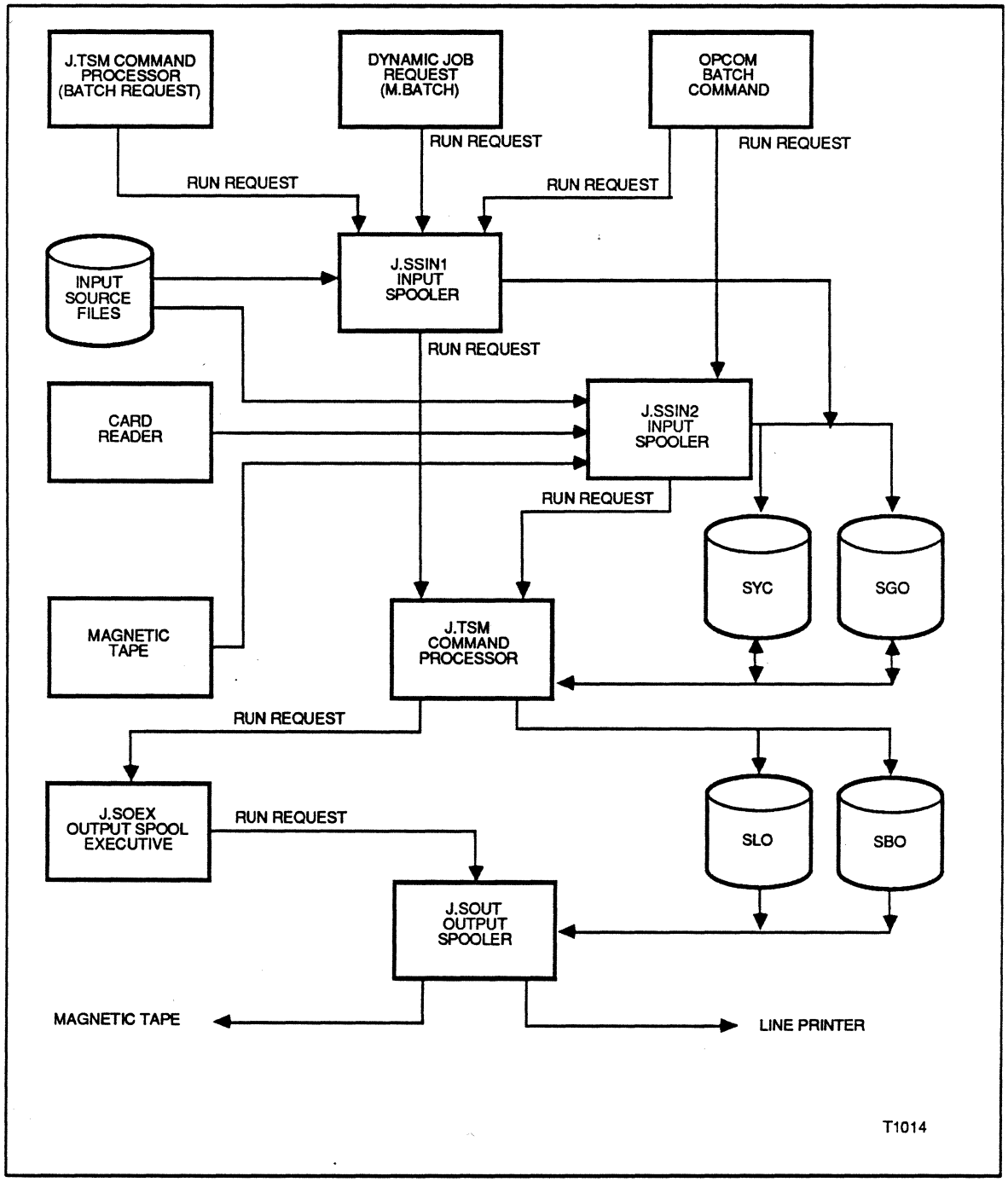


Figure 2-4
Spooled File Data Structures

2.37.1 J.SSIN Run Request

Input spooling is accomplished by run requesting a spool request to J.SSIN1 or J.SSIN2. Primary input source files must be blocked with either compressed or uncompressed data. The format of the J.SSIN run request is as follows:

	0 7	8 15	16 23	24 31
Word 0-7	Resource ID of the source file			
8	Line buffer address. See Note.			

Notes:

Line buffer address applies only to jobs batched by the command processor, J.TSM.

2.37.2 J.TSM Run Request

Each J.SSIN input spool request results in the creation of an SYC and SGO file and the initiation of a run request to J.TSM for batch processing. The format of the J.TSM run request is as follows:

	0 7	8 15	16 23	24 31
Word 0-7	Resource ID of SYC file			
8	Job number			
9	Flags and line buffer. See Note.			
10-11	Job name			
12-19	Resource ID of SGO file			

Notes:

Bits are assigned as follows:

Bits	Meaning if Set
0	sequential job
1	$DEFM specified in job stream
2	batch job
3-31	reserved

2.37.3 J.SOEX Run Request

J.SOEX is the first of two phases of output spooling. Spooled files which are to be output to a device for processing are queued to J.SOEX by a run request to determine device availability. The format of the J.SOEX run request is as follows:

	0 7 8 15 16 23 24 31
Word 0-7	MRRQ. See the MRRQ section.
8-15	Resource ID of SLO/SBO file
16-17	Device address (left-justified ASCII doubleword)
18-19	Task name/job name (left-justified ASCII doubleword)
20	Task number/job number (binary)
21	Flags. See Note 1.
22	Reprint or repunch copy count. See Note 2.
23	Reprint or repunch numbers. See Note 3.

Notes:

1. Flag bits are assigned as follows:

Bits	Meaning if Set
0	do not delete after deassignment
1	copy/reprint request
2	output in unformatted mode
3-23	reserved
24-31	value is as follows:

Value	Meaning
0	print output
1	punch output
2	plot output
3-255	reserved

2. This word contains the number of copies to be printed or punched in addition to the original.
3. The first halfword contains the beginning page number to be reprinted or repunched and the second halfword contains the ending page number to be reprinted or repunched.

Comments:

The actual J.SOEX run request is words 8 to 21. The run request is always appended to an MRRQ. See the Message or Run Request Queue section.

Temporary job-related SLO/SBO files are converted to permanent, non-SLO/SBO files prior to the run request of J.SOEX. Temporary job-related SLO/SBO files are output at end-of-job processing.

Permanent job-related SLO/SBO files are output by the M.DASN service as each is deassigned.

Nonjob-related SLO/SBO files are output by the M.DASN service. Temporary real-time SLO/SBO files are deleted after being output; permanent files are not. Real-time users may elect to use spooling directly by formatting and executing a run request to J.SOEX. The advantage is the specification of an output device.

2.37.4 J.SOUT Run Request

J.SOUT is the second phase of output spooling. When J.SOEX determines that a device is available for output processing, J.SOEX uses M.PTSK to activate J.SOUT. As part of the activation, a parameter buffer is passed to J.SOUT. The format of the J.SOUT parameter buffer is as follows:

	0 7 8 15 16 23 24 31
Word 0-7	Resource ID of SLO/SBO file
8	Reserved
9	Flags. See Note 1.
10	Device mnemonic (ASCII)
11	Task number/job number (binary)
12-13	Task name/job name (ASCII)
14	Reprint copy count. See Note 2.
15	Reprint page numbers. See Note 3.

Notes:

1. Bits are assigned as follows:

Bits	Meaning if Set
0-1	reserved
2	beginning a new job
3	job is batch origin
4	SLO output request
5	output device specified
6	permanent file - do not delete
7	reserved
8	inhibit banner page
9	display output - not formatted
10-31	reserved

2. This word contains the number of copies to be printed in addition to the original.
3. The first halfword contains the beginning page number to be reprinted and the second halfword contains the ending page number to be reprinted.

2.38 System Master Directory (SMD)

The system master directory (SMD) is not supported as of MPX-32 revision 2.0, but its format is retained for compatibility purposes. H.FISE compatible services and the File Manager utility are the only users of this structure.

Disk File SMD Entry

	0 7	8 15	16 23	24 31
Word 0-1	File name (SMD.AFN)			
2	File type (SMD.FTYP). See Note 1.	Start disk address (starting block number) (SMD.SBN)		
3	File indicators (SMD.FIN). See Note 2.	Length in 192-word blocks (SMD.BIF)		
4-5	User name (SMD.AUN)			
6	Compressed password (SMD.PWD)		UDT index (SMD.UDTX)	
7	Reserved (SMD.NU)			

Memory Partition SMD Entry

	0 7	8 15	16 23	24 31
Word 0-1	File Name (SMD.AFN)			
2	Starting logical page or number (SMD.SLP)		Starting physical page number or zero (SMD.SPP)	
3	File indicators (SMD.FIN). See Note 2.	Memory class (SMD.MC)	Length in pages (SMD.LIP)	
4-5	Reserved			
6	Compressed password (SMD.PWD)		Reserved	
7	Reserved (SMD.NU)			

Notes:

1. SMD.FTYP specifies a 2-character hexadecimal file type that is output by the Volume Manager in ASCII. Default is 00.

Value	Description
00-39	available for customer use
40-5F	reserved for system
60-9F	available for customer use
A0-AF	reserved for system
B0	base mode object file
BA	base mode shared image (or BASIC file)
BB	base mode object library file
BC	base mode macro library file
BE	base mode load module file
C0	spooled output file
CA	cataloged load module
CE	MPX-32/COFF executable image
CF	MPX-32/COFF shared image
D0	memory disk save task (J.MDSAVE) file
DB	symbolic debugger command file
ED	saved text editor file
EE	stored text editor file
FD	translated help file
FE	text editor work file
FF	SYSGEN generated file

2. Bits in SMD.FIN are assigned as follows:

Bits	Meaning if Set
0	permanent file is active
1	SYSGEN memory partition
2	no-save option is in effect
3	fast file
4	collision mapping
5	non-SYSGEN memory partition
6	password is required to write
7	password is required to read/write

2.39 Task Service Area (TSA)

The task service area (TSA) is a section of memory associated with each active task which is used by MPX-32 for storage of task-unique information. A TSA is allocated for each task when the task becomes active and is deallocated when the task terminates. The size of each task's TSA is fixed for the duration of the task's execution. However, the sizes of TSAs among tasks are variable and are dependent on the amount of space reserved for I/O activity.

As depicted in the following figure, the number of blocking buffers, file assignment table (FAT) entries and the file pointer table (FPT) entries is variable among tasks. For all tasks, the first six buffers' FAT and FPT entries are reserved for MPX-32 use and are present in every TSA.

The pushdown area in the TSA provides re-entrancy in calls to system modules. At each call to a system module entry point, T.REGP is incremented to the next 32-word pushdown level where the contents of the general purpose registers and program status doubleword (PSD) are saved. Within this 32-word level, 22 words are available for scratchpad storage by the module entry point being called. T.REGP is decremented to the previous pushdown level upon return to the entry point caller. When context switch away from a task occurs, the next pushdown level is used to preserve the contents of the task's registers and PSD. Ten words are used at the context switch level.

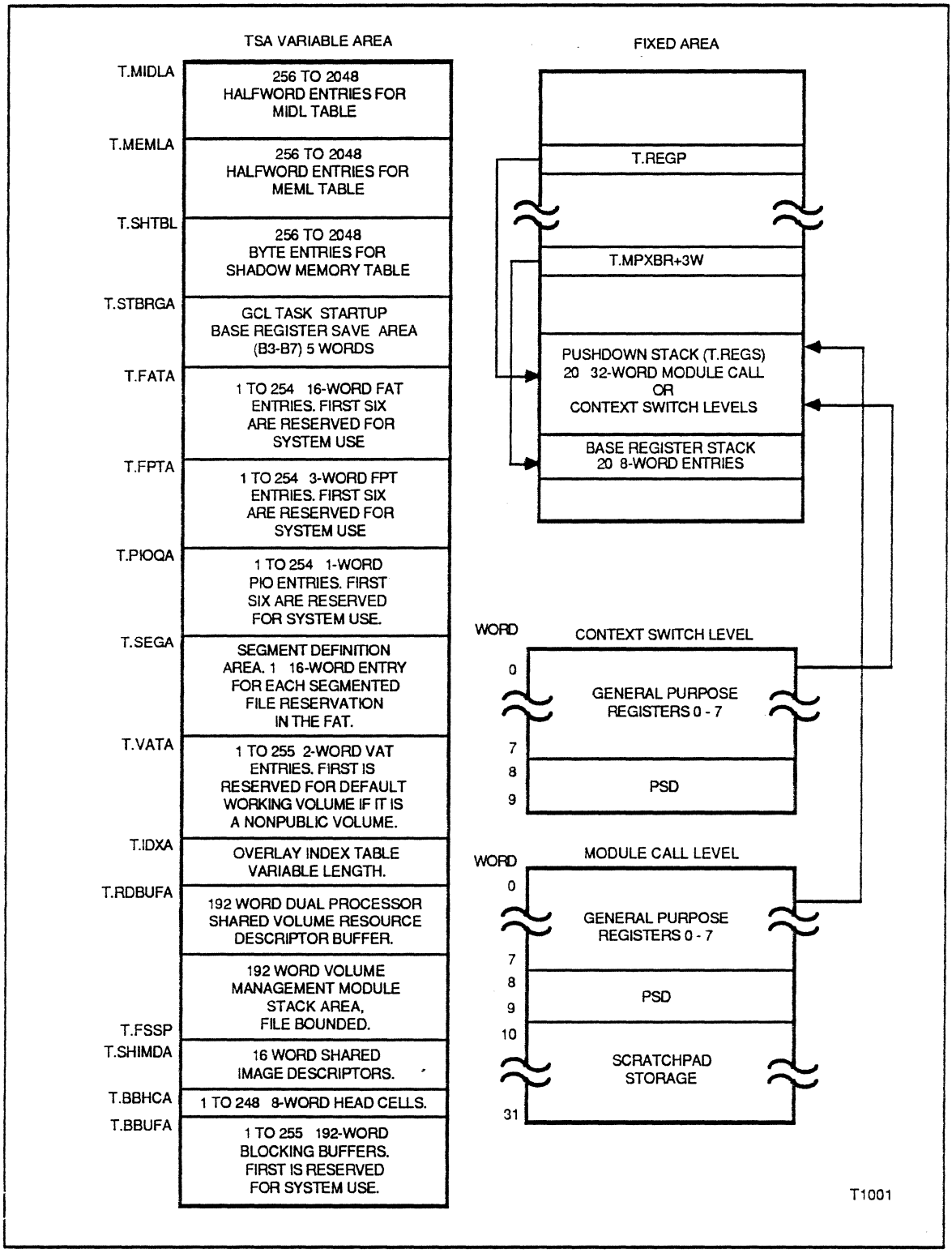


Figure 2-5
TSA Structure

TSA Structure

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31			
0-345	0	Reserved for MPX-32			
346-361	568	T.DFCB			
362-363	5A8	T.PRJCT			
364	5B0	T.PGOW			
365	5B4	T.PARENT			
366	5B8	T.REGP			
367	5BC	T.FSSP			
368-385	5C0	T.CONTXT			
386-389	608	T.FREE			
390-393	618	T.USED			
394	628	T.FIRST			
395	62C	T.LAST			
396	630	T.ITAC			
397	634	T.BASEP			
398-413	638	T.BFCB			
414-421	678	T.PROT			
422-429	698	T.BREGS			
430-437	6B8	T.MPP			
438	6D8	T.DBHAT			
439	6DC	T.PRNO			
440	6E0	T.ABRTA			
441	6E4	T.BBUFA			
442	6E8	T.VATA			
443	6EC	T.VATN	T.LMFPT	T.SEGN	T.BIT3
444	6F0	T.FATA			
445	6F4	T.FPTA			
446	6F8	T.SEGA			
447	6FC	T.BREAK			
448	700	T.MSGR			
449	704	T.BIAS			
450	708	T.TEND			
451	70C	T.END			
452	710	T.TRAD			
453	714	T.LINNO		T.UKEY	
454	718	T.BIT1	T.BIT2	T.BBUFN	T.FILES

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
455	71C	T.DSOR		T.DSSZ	
456	720	T.CSOR		T.CSSZ	
457	724	T.MEML1		T.MEML2	
458	728	Reserved			
459	72C	T.EAOR		T.EASZ	
460-461	730	T.ACCESS			
462-463	738	T.SYCS/T.LINBUF			
464-471	740	T.SGOS			
472-479	760	T.SLOS			
480-487	780	T.SBOS			
488-491	7A0	T.CDIR			
492-499	7B0	T.CVOL			
500	7D0	T.IPUAC			
501	7D4	T.CURH			
502	7D8	T.CRHX			
503	7DC	T.SYCF			
504	7E0	T.SGOF			
505	7E4	T.SLOF			
506	7E8	T.SBOF			
507	7EC	T.CPUSH	T.LDATTR	T.DBOR	
508	7F0	T.DPINFO		T.TASKK	T.BIT4
509	7F4	T.RDBUFA			
510	7F8	T.EXCPAD			
511	7FC	T.RORG			
512	800	T.RWORG			
513	804	T.DBSTAT			
514	808	T.MIDLA			
515	80C	T.MEMLA			
516	810	T.MEMLO			
517	814	T.STKSZ			
518-521	818	T.DBNAME			
522-523	828	T.EXPSD			
524	830	T.IDXA			
525	834	T.WORK		T.NSI	
526	838	T.SHIMDA			

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
527	83C	T.PREL			
528	840	T.DBSTW2			
529	844	T.BBHCA			
530	848	T.BIT5			
531	84C	T.SHTBL			
532	850	T.SMTMLT			
533	854	T.SIGSTK			
534-541	858-877	T.MPXBR			
542	878	T.MPXLM			
543	87C	T.NSTAT		T.REMIX	T.RTAIX
544	880	T.TSAOR		T.TSASZ	
545	884	T.ATBIAS		Reserved	
546-553	888-8A4	T.SPARES			
554	8A8	T.DSBSZ			
555	8AC	T.AGE			
556	8B0	T.CSECT			
557	8B4	T.DSECT			
558	8B8	T.CSBSZ			
559	8BC	T.LSTAGE			
560	8C0	T.SFPTA			
561	8C4	T.LASTP			
562	8C8	T.MIDL1		T.MIDL2	
563	8CC	Reserved			
564	8D0	T.TSAEND			
565	8D4	T.PGO2			
566	8D8	T.ADRTSA			
567	8DC	T.ATADDR			
568	8E0	T.AESTKP			
569	8E4	T.STBRGA			
570	8E8	T.PIOQA			
571	8EC	T.TIQA			
572	8F0	T.LDBSS			
573	8F4	T.PTRMP			
574	8F8	T.WKADR			
575	8FC	T.ABPSD			

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
576-1216	900	T.REGS
1217-2047	1300	Reserved for MPX-32
2048	2000	T.MIDL

Byte (Hex)	Symbol	Description
0		reserved for MPX-32
568	T.DFCB	demand page 16 word FCB
5A8	T.PRJCT	current project group name for file allocation (2 words)
5B0	T.PGOW	task option word one
5B4	T.PARENT	task sequence number of parent task (activator)
5B8	T.REGP	pointer/address of current level of pushdown in stack area
5BC	T.FSSP	address of the current pushdown level for file system (H.VOMM environment)
5C0	T.CONTXT	debug context area (18 words, must be on an 8 word boundary)
608	T.FREE	free list head cell (4 words)
618	T.USED	allocated list head cell (4 words)
628	T.FIRST	first logical memory address mapped into user task
62C	T.LAST	last logical memory address mapped into user task
630	T.ITAC	interval timer based accounting (CPU)
634	T.BASEP	address of file system buffer
638	T.BFCB	system service file control block (16 words)
678	T.PROT	reserved (8 words)
698	T.BREGS	task base register context area (8 words)
6B8	T.MPP	memory pool pointers (8 words)
6D8	T.DBHAT	address of debug halfword address table
6DC	T.PRNO	address of task's dispatch queue entry
6E0	T.ABRTA	address of task's abort receiver
6E4	T.BBUFA	address of task's blocking buffer
6E8	T.VATA	address of task's volume assignment table
6EC	T.VATN	number of entries in volume assignment table (1 byte)
	T.LMFPT	FPT index for load module file (1 byte)
	T.SEGN	number of dynamic segment definition areas (1 byte)

Byte (Hex)	Symbol	Description
	T.BIT3	TSA bit field assigned as follows:

Bit	Meaning when Set
0	PTASK RRS error encountered (T.RRERR)
1	static assignment in progress (T.SASSN)
2	M.GE or M.GD service in use (T.GEGD)
3	M.MEMB service in use (T.MEMB)
4	base mode save (T.BRSAVE)
5	modify descriptor in progress (T.MOD)
6	retry suspended (multiport only) (T.SUSP)
7	ACX-32 privileged task (T.ACXP)

6F0	T.FATA	address of task's file assignment table
6F4	T.FPTA	address of task's file pointer table
6F8	T.SEGA	address of segment definition area
6FC	T.BREAK	address of task's break receiver
700	T.MSGR	address of task's message receiver
704	T.BIAS	starting address of task's DSECT area
708	T.TEND	ending address of TSA when task is loaded
70C	T.END	last location loaded by the task loader within the DSECT plus one word
710	T.TRAD	transfer address of task's main segment
714	T.LINNO	number of lines on a TSM screen (1 halfword)
	T.UKEY	compressed original user key (1 halfword)
718	T.BIT1	bit variables (1 byte) assigned as follows:

Bit	Meaning when Set
0	first 4K of E-class I/O buffer in use (T.EIO1)
1	second 4K of E-class I/O buffer in use (T.EIO2)
2	arithmetic exception trap (T.EXCP)
3	reserved (T.EBUF)
4	suspend after activation (T.WAIT)
5	debugger required (T.DBG)
6	CSECT to share (T.SHR)
7	command file is active in TSM (T.COMFIL)

Byte (Hex)	Symbol	Description
	T.BIT2	bit variables (1 byte) assigned as follows:

Bit	Meaning when Set
0	E-class wait buffer (T.EBUF1)
1	E-class no-wait buffer (T.EBUF2)
2	user area of descriptor is being modified (T.MODU)
3	base mode debugger (T.BASE)
4	system administrator attribute (T.SAM)
5	all SLO output is directed to the terminal (T.UTSLO)
6	H.VOMM file control block reinitialization is required (T.FCBINT)
7	task cannot attach debugger (T.NODBG)

	T.BBUFN	number of blocking buffers associated with task (1 byte)
	T.FILES	number of FAT/FPT pairs associated with task (1 byte)
71C	T.DSOR	DSECT origin within T.MEML/T.MIDL (1 halfword); usually zero
	T.DSSZ	DSECT size in map blocks (1 halfword)
720	T.CSOR	CSECT origin within T.MEML/T.MIDL (1 halfword). If size is zero, T.CSOR is set equal to T.EAOR contents.
	T.CSSZ	CSECT size in map blocks (1 halfword)
724	T.MEML1	MEML overlay map one (1 halfword)
	T.MEML2	MEML overlay map two (1 halfword)
728		reserved for MEML overlay map two if fullword MIDLs
72C	T.EAOR	extended address origin in T.MEML/T.MIDL (1 halfword)
	T.EASZ	extended address size in map blocks (1 halfword)
730	T.ACCESS	privileged flags (2 words). Bit variables are assigned as follows:

Bit	Meaning when Set
0	EXIT directive disabled (T.EXIT)
1	ABORT directive disabled (T.ABORT)
2	ACTIVATE directive disabled (T.ACTVT)
3	BATCH directive disabled (T.BATCH)
4	BREAK directive disabled (T.BRK)
5	CONNECT directive disabled (T.CONNCT)
6	CONTINUE directive disabled (T.CONTNU)
7	DEPRINT directive disabled (T.DEPRNT)

Byte (Hex)	Symbol	Description

Bit	Meaning if Set
8	DEPUNCH directive disabled (T.DEPNCH)
9	DISABLE directive disabled (T.DISABL)
10	DISCONNECT directive disabled (T.DISCON)
11	DUMP directive disabled (T.DUMP)
12	ENABLE directive disabled (T.ENABLE)
13	reserved
14	HOLD directive disabled (T.HOLD)
15	KILL directive disabled (T.KILL)
16	LIST directive disabled (T.LIST)
17	MODE directive disabled (T.MODE)
18	MODIFY directive disabled (T.MODIFY)
19	OFFLINE directive disabled (T.OFFLNE)
20	ONLINE directive disabled (T.ONLINE)
21	PURGEAC directive disabled (T.PURGAC)
22	REDIRECT directive disabled (T.REDIR)
23	TSM REMOVE directive disabled (T.REMOVE)
24	REPRINT directive disabled (T.REPRNT)
25	REPUNCH directive disabled (T.REPNCH)
26	REQUEST directive disabled (T.REQEST)
27	DELETETIMER directive disabled (T.DTIMER)
28	reserved
29	SEARCH directive disabled (T.SEARCH)
30	SEND directive disabled (T.SEND)
31	SETTIMER directive disabled (T.STIMER)
32	SNAP directive disabled (T.SNAP)
33	START directive disabled (T.START)
34	STATUS directive disabled (T.STATUS)
35	SYSASSIGN directive disabled (T.SYSASN)
36	TIME directive disabled (T.TIME)
37	TSM URGENT directive disabled (T.URGENT)
38	RESUME directive disabled (T.RESUME)
39	ESTABLISH directive disabled (T.ESTAB)
40	access to other owners disabled (privileged) (T.OWNACC)
41	activation of privileged tasks disabled (privileged) (T.APRIV)
42	cataloging as privileged user disabled (privileged) (T.CPRIV)
43	TSM RESTART (privileged) disabled (T.RESTRT)
44	TSM URGENT directive disabled (privileged) (T.PRIOR)

Byte (Hex)	Symbol	Description

Bit	Meaning if Set
45	MOUNT directive disabled (T.MOUNT)
46	DISMOUNT directive disabled (T.DMOUNT)
47	system administrator attribute (privileged) enabled (T.SAA)
48	commands are not echoed (privileged) (T.NOCOMM)
49	cataloging tasks which set owner ID on access enabled (privileged) (T.OWNRID)
50	changing defaults enabled (T.CHANGD)
51	reserved for future use
52	TSM $SUBMIT jobs run sequentially
53	cannot use the J.MDTI Utility
54-55	reserved for future use
56-63	available for customer use

Byte (Hex)	Symbol	Description
738	T.SYCS	SYC definition (2 words)
	T.LINBUF	address of TSM's line buffer (1 word)
740	T.SGOS	SGO definition (8 words)
760	T.SLOS	SLO definition (8 words)
780	T.SBOS	SBO definition (8 words)
7A0	T.CDIR	name of current working directory (4 words)
7B0	T.CVOL	name of current working volume (8 words)
7D0	T.IPUAC	IPU real-time clock accounting
7D4	T.CURH	current high address in map
7D8	T.CRHX	current high address in extended space
7DC	T.SYCF	address of SYC dedicated FAT
7E0	T.SGOF	address of SGO dedicated FAT
7E4	T.SLOF	address of SLO dedicated FAT
7E8	T.SBOF	address of SBO dedicated FAT
7EC	T.CPUSH	number of push levels used for compatibility (1 byte)
	T.LDATTR	H.TAMM loader attributes
	T.DBOR	task debugger origin within T.MIDL/T.MEML (1 halfword). If debugger is not included with the task, T.DBOR is set equal to T.CSOR contents.
7F0	T.DPINFO	dual-processor information (1 halfword). Bit variables 0 to 15 indicate the base priority.
	T.TASKK	number of maps in the TSA and CALL stack (1 byte)

Byte (Hex)	Symbol	Description
	T.BIT4	bit variables (1 byte) assigned as follows:

Bit	Meaning if Set
0	inhibit exception handler (T.NOSET)
1	inhibit command line scan (T.INHSCN)
2	debugger load in progress (T.DBLIP)
3	arithmetic exception handling in progress (T.ARTECP)
4	base mode task with shared CSECT (T.BRSCS)
5	no checksum on load (T.NCKSM)
6	task is swappable (T.SWP)
7	shadow memory table is present (T.SHAD)

7F4	T.RDBUFA	address of dual-processor resource descriptor buffer
7F8	T.EXCPAD	arithmetic exception handler address
7FC	T.RORG	read-only section origin
800	T.RWORG	read/write section origin
804	T.DBSTAT	debugger status word
808	T.MIDLA	MIDL array address
80C	T.MEMLA	MEML array address
810	T.MEMLO	MIDL to MEML offset
814	T.STKSZ	number of bytes in CALL stack
818	T.DBNAME	base mode debugger name (4 words)
828	T.EXPSD	PSD at arithmetic exception (2 words)
830	T.IDXA	overlay count and index table address. Bytes are assigned as follows:

Bytes	Definition
0	number of entries in the overlay index table (each entry in the table is ten bytes in length)
1-3	address of the overlay index table

If this word is zero, there are no overlays in single file format. If this word is minus one, there are overlays in separate file format.

834	T.WORK	work space scratch area (1 halfword)
	T.NSI	number of shared image descriptors (1 halfword)
838	T.SHIMDA	shared image descriptor table address
83C	T.PREL	prelocation delta
840	T.DBSTW2	debugger status word 2
844	T.BBHCA	address of task's blocking buffer head cell

Byte (Hex)	Symbol	Description
848	T.BIT5	bits defined as follows:

Bit	Meaning if Set
0	task activation complete (T.TAC)
1	reserved
2	PTRACE debug activating task (T.PDBA)
3	no arithmetic exception handler (T.NOEXCP)
4	arithmetic exception occurred in CPU (T.CPUAE)
5	task is GCL load module (T.COFF)
6	task has included a multi-copied shared image (T.MCSHIM)
7	command line recall and edit (T.CLRE)
8	permanent IOQ in effect (T.PIOQE)
9	dispatch address to pseudo interrupt receivers requires 1 word offset (T.GCLOFF)
10	task is running with O.S. mapped out (T.MAPOUT)
11	task is running with TSA moved (T.MVTSA)
12	return to AID debugger or O.S. (T.RTRNOS)
13	task is running demand page mode (T.DPG)
14-31	reserved

84C	T.SHTBL	shadow memory table address
850	T.SMTMLT	SMT index for multicopied shared image with no read only section
854	T.SIGSTK	ADA stack size
858	T.MPXBR	8 word area containing pointers for EXTDMPX
	T.MPXBR+0W	the logical end of extended MPX-32
	T.MPXBR+1W	reserved
	T.MPXBR+2W	starting address of the base register stack
	T.MPXBR+3W	the current push level of the base register stack
	T.MPXBR+4W	reserved
	T.MPXBR+5W	extended MPX-32 logical starting address of segment three
	T.MPXBR+6W	extended MPX-32 logical starting address of segment two
	T.MPXBR+7W	extended MPX-32 logical starting address of segment one

Byte (Hex)	Symbol	Description
878	T.MPXLM	the first halfword contains the length of extended MPX-32 in units of 2KW maps. The second halfword contains EXTDMPX offset into the task's T.MIDL.
87C	T.NSTAT	number of static partition maps in use (1 halfword)
87E	T.REMIX	remote index to file control block
87F	T.RTAIX	remote task activation index
880	T.TSAOR	the starting MIDL number for the task's TSA within T.MIDL (1 halfword)
882	T.TSASZ	TSA size in map blocks (1 halfword)
884	T.ATBIAS	the number of MPX-32 maps included in the task's address space (1 halfword)
886-8A4	T.SPARES	reserved
8A8	T.DSBSZ	DSECT byte size
8AC	T.AGE	virtual time before page considered aged
8B0	T.CSECT	byte offset to CSECT code in load module
8B4	T.DSECT	byte offset to DSECT code in load module
8B8	T.CSBSZ	CSECT byte size
8BC	T.LSTAGE	virtual time of last age
8C0	T.SFPTA	shared image/FPT pair
8C4	T.LASTP	logical address of last stack frame in the TSA
8C8	T.MIDL1	MIDL overlay map one (1 halfword)
8CA	T.MIDL2	MIDL overlay map two (1 halfword)
8CC		reserved for MIDL overlay map two if fullword MIDLs
8D0	T.TSAEND	ending address of TSA when task is loaded. Exclude split portion of MPX when at MINADDR.
8D4	T.PGO2	second task option word
8D8	T.ADRTSA	TSA address from EXTDMPX specifications on non-split images
8DC	T.ATADDR	logical end of MPX-32 (+ 1 byte) for task
8E0	T.AESTKP	arithmetic exception segment list address save area
8E4	T.STBRGA	address of GCL task's startup base register save area
8E8	T.PIOQA	address of task's static IOQ table. This table provides indirect linkage between the FCB and the static IOQ address.
8EC	T.TIQA	terminal input queue entry address
8F0	T.LDBSS	GCL task's BSS section size in bytes
8F4	T.PTRMP	PTRACE memory pool use word
8F8	T.WKADR	address of working map block during task activation
8FC	T.ABPSD	abort PSD

Byte (Hex)	Symbol	Description
900	T.REGS	offset from the beginning of TSA to start of start of the pushdown stack
1300		reserved for MPX-32
2000	T.MIDL	halfword map image descriptor list (128 to 1024 words depending on the logical space requirement of task)

Bit	Meaning if Set
0	map number is valid (MIDL.VAL)
1	first protection granule is write protected (MIDL.PRO)
2	second protection granule is write protected (MIDL.PR2)
3	third protection granule is write protected (MIDL.PR3)
4	fourth protection granule is write protected (MIDL.PR4)
5-15	physical map number

or

Byte (Hex)	Symbol	Description
2000	T.MIDL	fullword map image descriptor list (256 to 2048 words on mapped out images depending on the logical space requirement of task)

Bit	Meaning if Set
0	map number is valid (MIDL.VAL)
1-2	map block has restrictions according to the following bit encoding:

PSD Priv.	P1	P2	
0	0	0	read/execute
0	1	0	read/write/execute
0	0	1	not used by MPX-32
0	1	1	not used by MPX-32
1	0	0	read/write/execute
1	1	0	read/write/execute
1	0	1	not used by MPX-32
1	1	1	not used by MPX-32

3	map block has been modified (MIDL.MOD)
4	map block has been referenced (MIDL.ACC)
5	indicates SelBUS (DRAM) memory (MIDL.SEL)

2.40 Terminal Line Buffer

The terminal line buffer, buffers terminal input and output. It is allocated from memory pool for each online task when the terminal is opened. The size of the buffer is determined by the contents of UDT.CHAR. The buffer is pointed to by T.LINBUF and always begins on a doubleword boundary. It is deallocated when the terminal is closed.

	0 7	8 15	16 23	24 31
Word 0-3	Last argument found by scanner. See Note 1.			
4	Buffer length. See Note 2.	Cursor index. See Note 3.	Field delimiter. See Note 4.	Field size. See Note 5.
5-*n*				

(length may vary)

Notes:

1. Two doublewords containing last argument found by syntax scanner. It is left justified and blank filled.
2. Number of words in line buffer (20 minimum, 59 maximum). This number does not reflect possible words required to maintain memory pool doubleword bounding.
3. Cursor index for next call to scanner. Relative to word 0 of line buffer.
4. The delimiting character previously found by scanner.
5. Number of significant characters in previous argument found by scanner.

2.41 Timer Table

The timer table contains all necessary information for the time scheduling of the functions provided in the create timer entry service. The functions include activating a program, resuming a program, setting a bit, resetting a bit, and requesting an interrupt.

The table also contains a variable number of five word timer entries that are specified at SYSGEN. Entries in the table are identified by a 2-character (ASCII) timer ID specified by the create timer entry service. Entries are deleted by the delete timer entry service. SYSGEN forces three entries for the exclusive use of J.SSIN1, J.SSIN2, and J.SOUT. The format of the timer table entries follows:

	0 3	4 7	8 15	16 31
Word 0	Timer Status. See Note 1.	Function code. See Note 2.	Reserved	Timer ID - two unique ASCII characters
1	Function parameter one. See Note 3.			
2	Function parameter two. See Note 3.			
3	Current time value in negative time units. See Note 4.			
4	Reset time value in negative time units. See Note 5.			

Notes:

1. The time status bits have the following meanings:

Bits	Meaning if Set
0	timer in hold state
1	timer entry is taken
2	re-issue activation request
3	reserved

2. The function code (4-bit numeric value) is assigned as follows:

Value	Meaning
1	activate program
2	resume program
3	set bit in static memory partition or operating system
4	reset bit in static memory partition or operating system
5	request interrupt between X'12' to X'7F'

3. The function code, bits 4 through 7 in word 0, determines the contents of the function parameters (words 1 and 2).

Function Code	Word	Contents
1	1	Dispatch queue address of task to be activated at time out. M.SETT preactivates the task, then acquires the dispatch queue address. The task remains suspended until time out.
	2	Reserved
2	1	Dispatch queue address of task to be resumed at time out. M.SETT acquires the dispatch queue address from the user-supplied task name or task number.
	2	Reserved
3	1	Address of word where the bit is to be set.
	2	The bit configuration to be ORed at time out with the data at the address specified in word one.
4	1	Address of word where the bit is to be reset.
	2	The bit configuration to be ANDed at time out with the data at the address specified in word one.
5	1	A request interrupt (RI) instruction, for the user-specified priority level, to be executed upon time out.
	2	Reserved

4. The current time value is the negative timer units to elapse before the selected function is completed. This value is incremented until it equals zero. At that time, the selected function is complete.

   If word 4 is zero when time out occurs, the entry is deleted. If word 4 is non-zero when time out occurs, the value in word 4 is loaded into word 3 and incremented.

5. The reset time value is the negative timer units to elapse before the selected function is repeated. When the timer expires, this value is loaded into word 3 and incremented.

   Word 4 is not changed until the timer is deleted or the system is rebooted.

2.42 Type Control Parameter Block (TCPB)

The type control parameter block (TCPB) allows I/O to and from the system console by setting up task buffer areas for messages output by a task and optional reads back from the console. If no input is desired, word one of the TCPB must be zero.

See the MPX-32 Reference Manual Volume I, Chapter 5 for further details on the TCPB.

Type Control Block (TCPB) using 19-bit address

	0 — 11	12	13 — 31
Word 0	Output quantity (TCP.OQ)	See Note 1.	Output data address (TCP.OTCW)
1	Input quantity (TCP.IQ)	See Note 1.	Input data address (TCP.ITCW)
2	Console Teletype Flags (TCP.FLGS). See Note 2.		

Notes:

1. Bit 12 is set to 1.
2. Bits in TCP.FLGS are assigned as follows:

Bits	Meaning if Set
0	no-wait I/O
31	operation in progress. This bit is reset after post-I/O processing completes.

Type Control Parameter Block (TCPB) using 24-bit address:

	0 7	8 15	16 23	24 31
Word 0	Output quantity (TCP.OQ)	Output data buffer address (TCP.OTCW)		
1	Input quantity (TCP.IQ)	Input data buffer address (TCP.ITCW)		
2	Console device flags (TCP.FLGS) See Note 1.			

Notes:

1. Bit interpretations for TCP.FLGS are:

Bits	Meaning if Set
0	no-wait I/O
1	data buffer addresses are 24-bit addresses (TCP.LAD) Note: This bit must be set.
31	operation in progress. This bit is reset after post-I/O processing completes.

2.43 Unit Definition Table (UDT)

The unit definition table (UDT) is a system resident structure that identifies device-dependent information required by a handler for a specific device. The UDT is built by the SYSGEN process, one for each device configured in the system. During SYSGEN, each UDT is linked to its corresponding controller definition table (CDT) and its associated controller and handler.

	0 7	8 15	16 23	24 31
Word 0	UDT index (UDT.UDTI)		CDT index (UDT.CDTI)	
1	Unit status (UDT.STAT). See Note 1.	Device type code (UDT.DTC). See Note 2.	Logical channel number (UDT.CHAN)	Logical subaddress (UDT.SUBA)
2	Reserved	Address of dispatch queue entry of task which has device allocated if device is not shared (UDT.DQEA)		
3	Physical channel number (UDT.PCHN)	Physical subaddress (UDT.PSUB)	Sectors per block (UDT.SPB) or number of characters per line (UDT.CHAR). See Note 3.	Sectors per allocation unit (UDT.SPAU) or number of lines per screen (UDT.LINE). See Note 4.
4	Flags (UDT.FLGS). See Note 5.	Number of sectors per track on disk or global line counter if a terminal (UDT.SPT)	Maximum byte transfer (UDT.MBX)	
5	Number of sectors on disk or tab setting if a terminal (UDT.SECS)			
6	Sector size, on disk or a tab setting if a terminal (UDT.SSIZ)		Number of heads on disk or a tab setting if a terminal (UDT.NHDS)	
7	Serial number if tape or removable disk (UDT.SERN). See Note 6.			
8	Peripheral time-out value (UDT.PTOV)			
9	Reserved	Address of device context area (UDT.DCAA) or handler name at initialization (UDT.HNAM)		
10	Bit flags (UDT.BIT2). See Note 7.		Associated allocated resource table index if assigned (UDT.ARTI)	
11	Service interrupt handler address (UDT.SIHA)			
12	Reserved (UDT.CXR) See Note 8.	Secondary flags (UDT.BIT3) See Note 9.	Reserved (UDT.SHFL) Udt. opt. or UDT.HIST. See Note 10	Reserved (UDT.DQEN)
13	Address of first IOQ linked to this device (UDT.FIOQ)			
14	Address of last IOQ linked to this device (UDT.BIOQ)			
15	Link Priority (UDT.LPR1)	Link Count (UDT.IOCT)	Unit Status byte 2 (UDT.STA2). See Note 11.	

Notes:

1. Bits in UDT.STAT are assigned as follows:

Bit	Meaning if Set
0	online (UDT.ONLI)
1	dual-portd XIO disk (UDT.DPDC)
2	allocated (UDT.ALOC)
3	terminal in use and not in wait (UDT.USE)
4	system output unable to allocate (UDT.NOAL)
5	shared device (UDT.SHR)
6	premounted (UDT.PREM)
7	terminal (TSM) device (UDT.TSM)

2. For example, 01 for any disk, 04 for any tape, etc. Valid device type codes are listed in Chapter 1 of this manual.
3. For disks, contains the number of sectors per block (UDT.SPB). For terminals, contains the number of characters per line (UDT.CHAR).
4. For disks, contains the number of sectors per allocation unit (UDT.SPAU). For SLO or terminals, contains the number of lines per page or screen (UDT.LINE).
5. Bits in UDT.FLGS are assigned as follows:

Bit	Meaning if Set
0	extended I/O device (UDT.FCLS)
1	I/O outstanding (UDT.IOUT)
2	removable disk pack (UDT.RMDV)
3	a break has been requested for this device (UDT.LOGO)
4	autoselectable for batch SLO (UDT.BSLO)
5	autoselectable for batch SBO (UDT.BSBO)
6	autoselectable for real-time SLO (UDT.RSLO)
7	autoselectable for real-time SBO (UDT.RSBO)

6. If the device is a terminal or console, the first halfword is the current terminal type for TERMDEF (UDT.CTDF) and the second halfword is the default terminal type (UDT.DTDF).
7. Bits in UDT.BIT2 are assigned as follows:

Bits	Meaning if Set
0	port is private; else switched (UDT.DIAL)
1	port is connected to modem (UDT.MODM)
2	port has graphic capability (UDT.GRFC) ← UDT, SCFF
3	port is full duplex (UDT.FDUX)
4	port is configured multidrop (UDT.MDRA)
5	volume mounted on device (UDT.VOL)
6	echo by computer (UDT.ECHO)
7	device has failed. Log off TSM (UDT.DEAD)
8	cache device (UDT.CAC)
9	inhibit automatic line wrap (UDT.NRAP)
10	spool device requires form feed after printing rather than before; initial form feed is inhibited (UDT.FEOP)

Bits	Meaning if Set
11	quarter inch cartridge tape drive (UDT.QITD)
12	software read flow control required (UDT.RXON)
13	software write flow control required (UDT.WXON)
14	hardware read flow control required (UDT.RHWF)
15	hardware write flow control required (UDT.WHWF)

8. For switched port, contains the value specified in the LOGONFLE CXR = option (UDT.CXR)
9. Bits in UDT.BIT3 are assigned as follows:

Bits	Meaning if Set
0	SCSI device (UDT.SCSI)
1-7	reserved

10. UDT.HIST is used as an address save area by pseudo device handlers, such as ON.IPXIO
11. Bits in UDT.STA2 are assigned as follows:

Bits	Meaning if Set
0	IOQ linked from UDT (UDT.IOQ)
1	IOP device (initialized by SYSGEN) (UDT.IOP)
2	device malfunction (UDT.MALF)
3	operator intervention applicable (UDT.INTV)
4	use standard XIO interface
5	floppy disk
6	cartridge module drive
7	moving head disk with fixed head option
8	if software read flow control enabled, use DTR line; otherwise, use RTS line. (UDT.RDTR)
9	memory disk (UDT.MD) or valid command line recall and edit device (UDT.CLRE)
10	memory allocated for memory disk (UDT.MDAL)
11	start address of memory disk specified at SYSGEN (UDT.MDST)
12	multiport device is shared with an MPX-32 Revision 3.2C or earlier version (UDT.PPV)
13	device is exclusive ANSI (UDT.ANSI)
14	serial printer (UDT.SLPR)
15	port is switched and CXR=N option has been specified (UDT.DCXR)

Device option word X'32' for SCSI
UDT.OPT = UDT.SHFL
UDT.WFMI 0 - write filemark immediate supported
UDT.QIC 1 - QIC tape drive
UDT.SEOF 2 - volmgr soft EOF mgmt request
UDT.NMVT 3 - multi volume tape processing invalid
UDT.DRTX 4 - SCSI tape coordinated by DRTX
UDT.8mm 5 - 8mm tape drive

2.44 Volume Assignment Table (VAT)

The volume assignment table (VAT) is used to identify a nonpublic volume associated with a particular task. The table is located in the task's service area (TSA). The VAT points to the information necessary to process access to the volume for a specific task and is required to be memory resident (frequently required information). The VAT contains a use count that represents the number of resources allocated on a non-public volume by the task.

A VAT entry is created for a task when a nonpublic volume is logically mounted by that task. A logical mount is necessary before a task is allowed to reference any resources on the nonpublic volume. A linkage in the VAT is then connected to a corresponding entry in the mounted volume table (MVT) for the requested volume. If an entry does not exist in the MVT for the requested volume, the volume must be physically mounted to establish an entry. The linkage in the VAT is then connected to the MVT.

	0 7	8 15	16 23	24 31
Word 0	Assign count (VA.ASSNS)	Mounted volume table address (VA.MVTA)		
1	Assigned access restrictions (VA.ACCS). See Note.			

Notes:

Bits in VA.ACCS are assigned as follows:

Bits	Meaning if Set
0-15	reserved
16	entry is available (VA.AVAIL)
17-31	reserved

2.45 Disk Resident Resource Descriptors (RD)

The following chart shows the MPX-32 disk resident resource descriptors most commonly used and their correlation to each other.

As shown in the chart, the first 86 words of each descriptor are identical. For example, words 0 through 63 are the common parameters and words 64 through 85 are the space parameters.

Unique descriptor types begin at word 86 and end at word 95, with the exception of the memory partition parameters that end at word 97 because of memory page parameters, and the volume descriptor parameters that end at word 159 because they do not contain a segment definition area.

The descriptors, with the exception of the two noted above, contain a segment definition area that begins at word 96 and ends at word 159. The segment definition area for memory partitions begins at word 98 and ends at word 159.

The descriptors contain a free usage area that begins at word 160 and ends at word 190.

Each descriptor is described following the chart.

	0 7 8 15 16 23 24 31
Words 0-63	Common parameters (M.RDCOM) - 29 spare words
64-85	Space parameters (M.RDSPD)
86-95	Bad block file descriptor parameters (M.BB.DEQ) - 8 spare words Descriptor allocation map parameters (M.DM.DEQ) - 4 spare words Descriptors descriptor parameter (M.DD.DEQ) - 9 spare words Descriptor map (DMAP) deallocation file descriptor (M.BD.DEQ) - 2 spare words Directory descriptor parameters (M.DI.DEQ) - 1 spare word File descriptor parameters (M.FI.DEQ) - no spare words Memory partition parameters (M.ME.DEQ) - 8 spare words Space allocation map parameters (M.SM.DEQ) - 3 spare words Space map (SMAP) deallocation file descriptor (M.BS.DEQ) - 3 spare words Volume descriptor parameters (M.VO.DEQ)
96-159	Memory partition parameters (M.ME.DEQ) (continued to word 97) Volume descriptor parameters (M.VO.DEQ) (continued) - 9 spare words Segment definitions (RD.SEGDF)
160-190	User area (RD.USER)
191	Reserved for MPX-32

2.45.1 Resource Descriptor (M.RDCOM)

The resource descriptor (M.RDCOM) defines the system common portion of a resource descriptor. The common area ends at 64W, immediately followed by the information specific to the resource descriptor type.

Words 0 to 7 correspond to M.RDID, words 8 to 25 correspond to M.RDACT, words 26 to 32 correspond to M.RDACC, and words 33 to 64 define the balance of the common area.

	0 7 8 15 16 23 24 31	
Word 0-3	Volume name (RD.IDNAM)	
4	Binary creation date (RD.DATE)	
5	Binary creation time (RD.TIME)	
6	Absolute block number of resource descriptor (RD.DOFF)	
7	Resource ID flags (RD.RDFLG). See Note 1.	Resource type (numeric value) (RD.RTYPE). See Note 2.
8	Binary date of creation/deletion (RD.CRDAT)	
9	Binary time of creation/deletion (RD.CRTIM)	
10	Binary date of expiration (RD.XPDAT)	
11	Binary time of expiration (RD.XPTIM)	
12	Binary date of last read access (RD.RDDAT)	
13	Binary time of last read access (RD.RDTIM)	
14	Binary date file last changed (RD.CHDAT)	
15	Binary time file last changed (RD.CHTIM)	
16	Binary date of last save (RD.SVDAT)	
17	Binary time of last save (RD.SVTIM)	
18	Binary date of last restore (RD.RSDAT)	
19	Binary time of last restore (RD.RSTIM)	
20-21	Owner name of last changer (RD.CHOWN)	
22-23	Owner name of creator (RD.CROWN)	
24	Count of opens in read mode (RD.RDCNT)	
25	Accounting flags (RD.AFLGS). See Note 3.	
26-27	Name of resource owner (RD.OWNR)	
28-29	Name of resource project group (RD.UGRP)	
30	Owner access/privileges (RD.AOWNR). See Note 4.	
31	Project group access/privileges (RD.AUGRP). See Note 4.	
32	Others access/privileges (RD.AOTHR). See Note 4.	

	0 7	8 15	16 23	24 31
Word 33	Reserved			
34	Resource link count (RD.LNKCT)			
35	Port number (1 through 16) of task that opened resource for write access (RD.WRID)	Port number (1 through 16) of task that opened resource for modify access (RD.MDID)	Port number (1 through 16) of task that opened resource for update access (RD.UPID)	Port number (1 through 16) of task that opened resource for append access (RD.APID)
36-43	Information to reconstruct spool files (RD.SUBMT)			
44	Number of MDT entries (RD.MDTC)		Number of hashes required to locate entry (RD.MDTH)	
45	Flag word (RD.MDTF). See Note 5.			
46-51	Reserved for MPX-32 Usage			
52-53	Owner name at last access (RD.RDOWN)			
54-57	Number of resource assigners by port number (RD.ASSN). See Note 6.			
58-61	Number of resource users by port number (RD.USER). See Note 7.			
62-63	Reserved			

Notes:

1. Internal flags reserved for MPX-32.
2. Values for RD.RTYPE are as follows:

Value	Meaning
1	volume type (RD.VOL)
2	resource descriptor description (RD.RESRC)
3	descriptor map descriptor (RD.DMAP)
4	space map descriptor (RD.SMAP)
5	root directory descriptor (RD.ROOT)
6	system image descriptor (RD.IMAGE)
7	bad block descriptor (RD.BDBLK)
8	value for spool file (RD.SPOOL)
9	reserved
10	permanent file or shared image (RD.FILE)
11	permanent directory (RD.DIR)
12	temporary file (RD.TFILE)
13	temporary directory (RD.TDIR)
14	static memory partition (RD.MEM)
15	dynamic memory partition (RD.TMEM)
16	device descriptor (RD.DEVC)
17	resource descriptor for the DMAP bad block deallocation file (RD.BDMAP)
18	resource descriptor for the SMAP bad block deallocation file (RD.BSMAP)

3. Bits in RD.AFLGS are assigned as follows:

Bits	Meaning if Set
0	record last read information (RD.AREAD)
1	RD is being deleted (RD.DEL)
2-31	reserved

4. Bits in RD.AOWNR, RD.AUGRP and RD.AOTHR are assigned as follows:

Bits	Meaning if Set
0	read access allowed (RD.READ)
1	write access allowed (RD.WRITE)
2	modify access allowed (RD.MODFY)
3	update access allowed (RD.UPDAT)
4	append access allowed (RD.APPND)
5-7	reserved
8	traverse directory access allowed (RD.TRAVR)
9	delete resource access allowed (RD.DELET)
10	delete directory entry access allowed (RD.DEENT)
11	add directory entry access allowed (RD.ADENT)
12-31	reserved

5. Bits in RD.MDTF are assigned as follows:

Bits	Meaning if Set
0	MDT entry is in use. This bit is never set in the disk copy of the resource descriptor.
1-31	reserved

6. Bytes in RD.ASSNS are as follows:

Byte	Meaning
0	number of assigners on port 0
1	number of assigners on port 1
2	number of assigners on port 2
3	number of assigners on port 3
4	number of assigners on port 4
5	number of assigners on port 5
6	number of assigners on port 6
7	number of assigners on port 7
8	number of assigners on port 8
9	number of assigners on port 9
10	number of assigners on port 10
11	number of assigners on port 11
12	number of assigners on port 12
13	number of assigners on port 13
14	number of assigners on port 14
15	number of assigners on port 15

7. Bytes in RD.USERS are as follows:

Byte	Meaning
0	number of users on port 0
1	number of users on port 1
2	number of users on port 2
3	number of users on port 3
4	number of users on port 4
5	number of users on port 5
6	number of users on port 6
7	number of users on port 7
8	number of users on port 8
9	number of users on port 9
10	number of users on port 10
11	number of users on port 11
12	number of users on port 12
13	number of users on port 13
14	number of users on port 14
15	number of users on port 15

2.45.2 Resource Descriptor Space Definition (M.RDSPD)

The resource descriptor space definition (M.RDSPD) descriptor defines a resource descriptor space definition area.

	0 7 8 15 16 23 24 31
Word 64	Space definition flags (RD.SFLGS). See Note.
65	Maximum file extension increment (RD.MXEXT)
66	Minimum file/directory extension increment (RD.MNEXT)
67	Maximum attainable file/directory size (RD.MXSIZ)
68	End-of-file relative block number (RD.EOFBL)
69	End-of-medium relative block number (RD.EOMBL)
70	Number of segments in file/directory (RD.NUMSG)
71	Absolute address (block number) of resource descriptor with extra segment definition (RD.XSABA)
72-75	Directory name (RD.DNAME)
76	Block address of parent directory descriptor (RD.PAREN)
77	Number of segments at creation time (RD.NUMCR)
78-79	Reserved
80-83	Directory entry value (RD.DIRP)
84	Resource directory address of parent directory (RD.DADD)
85	Directory entry index into parent (RD.DIDX)
86-95	Reserved for unique descriptor parameters

Notes:

Bits in RD.SFLGS are assigned as follows:

Bits	Meaning if Set
0-7	resource type, equivalent to file type code, interpreted as two hexadecimal digits, 0-FF (RD.FTYPE)
8-10	reserved
11	EOF management required (RD.EOFM)
12	fast access file (RD.FAST)
13	no-save file (RD.NSAVE)
14	file contains default image (RD.DEFI)
15	start segment requested (RD.SSREQ)
16	execute access (RD.EXEC)
17	set owner ID on access (RD.OWNID)
18	set project group ID on access (RD.USRID)
19	restoring this file (RD.PREFR)
20-22	reserved

Bits	Meaning if Set
23	zero file on creation/expansion (RD.ZERO)
24	automatically extendible (RD.AUTO)
25	manually extendible (RD.MANUL)
26	contiguity desired (RD.CONTG)
27	shareable access allowed (RD.SHRBL)
28	link access (RD.LINK)
29	file physically/logically contiguous (RD.NCNTG)
30	file written to DFI (RD.DFI)
31	file data is recorded as blocked (RD.BLOCK)

2.45.3 Bad Block Descriptor (M.BB.DEQ)

The bad block descriptor (M.BB.DEQ) defines the bad block file descriptor.

	0 7 8 15 16 23 24 31
Word 86	Number of bad blocks (BB.NUMBL)
87	Number of bad allocation units (BB.NUMAU)
88-95	Reserved

2.45.4 Descriptor Allocation Map Descriptor (M.DM.DEQ)

The descriptor allocation map descriptor (M.DM.DEQ) defines the descriptor map descriptor block.

	0 7 8 15 16 23 24 31
Word 86	Descriptor allocation map length in blocks (DM.DMAPL)
87	Descriptor allocation map length in words (DM.DMAPW)
88	Number of resource descriptors in descriptor map (DM.DMAPU)
89	Number of resource descriptors available (DM.DMAPC)
90	Number of words in last block (DM.LASTB)
91	Number of bits used in last word of map (DM.LASTW)
92-95	Reserved

2.45.5 Descriptors Descriptor (M.DD.DEQ)

The descriptors descriptor (M.DD.DEQ) defines the descriptor for the resource descriptors.

	0 7 8 15 16 23 24 31
Word 86	Maximum number of resource descriptors (DD.NUMRD)
87-95	Reserved

2.45.6 Descriptor Map (DMAP) Deallocation File Descriptor (M.BD.DEQ)

The DMAP deallocation file descriptor (M.BD.DEQ) defines the DMAP deallocation file descriptor. Each entry in the DMAP deallocation file is 4 bytes in length (BD.ESIZE) and contains the disk block number of the block (resource descriptor) to be deallocated.

	0 7	8 15	16 23	24 31
Word 86	Reserved			
87	Total number of entries that can be specified in the bad DMAP deallocation file (BD.NENTR)			
88	Total number of entries available in the bad DMAP deallocation file (BD.AVAIL)			
89-91	Reserved			
92	Number of resource assigners by the dual-port number (RD.ASSNS or RD.ART). See Note 1.		Number of resource users by the dual-port number (RD.USERS). See Note 2.	
93	Current multi-port access mode (RD.CACM)	Port number of resource lock owner (RD.MPID)	Reserved	
94	AR.FLAGS. See Note 3.		AR.XRL. See Note 3.	AR.SRL. See Note 3.
95	AR.ASSNS. See Note 3.	AR.USERS. See Note 3.	Reserved	AR.RDRS. See Note 3.

Notes:

1. Bytes in RD.ASSNS are as follows:

Byte	Description
0	number of assigners on port 0
1	number of assigners on port 1

   RD.ART is equivalent to RD.ASSNS, and indexes the ART information stored in words 94 and 95.

2. Bytes in RD.USERS are as follows:

Byte	Description
2	number of users on port 0
3	number of users on port 1

3. For detailed description of field contents, refer to the corresponding field within the allocated resource table (ART) in this chapter.

2.45.7 Directory Descriptor (M.DI.DEQ)

The directory descriptor (M.DI.DEQ) defines the directory descriptor block (also pertains to the root directory).

	0 7	8 15	16 23	24 31
Word 86	Number of entries per block (DI.ENTSG)			
87	Current number of active entries (DI.ACTIV)			
88	Total entry capacity (DI.TOTEN)			
89	Current number of available entries (DI.AVLEN)			
90	Directory descriptor flags (DI.FLAGS)			
91	Reserved			
92	Number of resource assigners by the dual-port number (RD.ASSNS or RD.ART). See Note 1.		Number of resource users by the dual-port number (RD.USERS). See Note 2.	
93	Current multiport access mode (RD.CACM)	Port number of resource lock owner (RD.MPID)	Reserved	
94	AR.FLAGS. See Note 3.		AR.XRL. See Note 3.	AR.SRL. See Note 3.
95	AR.ASSNS. See Note 3.	AR.USERS. See Note 3.	Reserved	AR.RDRS. See Note 3.

Notes:

1. Bytes in RD.ASSNS are as follows:

Byte	Description
0	number of assigners on port 0
1	number of assigners on port 1

   RD.ART is equivalent to RD.ASSNS, and indexes the ART information stored in words 94 and 95.

2. Bytes in RD.USERS are as follows:

Byte	Description
2	number of users on port 0
3	number of users on port 1

3. For detailed description of field contents, refer to the corresponding field within the allocated resource table (ART) in this chapter.

2.45.8 File Descriptor (M.FI.DEQ)

The file descriptor (M.FI.DEQ) defines the file descriptor block (also pertains to descriptors for the system image).

	0 7	8 15	16 23	24 31
Word 86-90	RD.STRUC. See Note 1.			
91	Assign count Port 0. See Note 2.	Assign count Port 1. See Note 2.	Use count Port 0. See Note 2	Use count Port 1. See Note 2.
92	Number of resource assigners by the dual-port number (RD.ASSNS or RD.ART). See Note 3.		Number of resource users by the dual-port number (RD.USERS). See Note 4.	
93	Current multi-port access mode (RD.CACM)	Port number of resource lock owner (RD.MPID)	Reserved	
94	AR.FLAGS. See Note 5.		AR.XRL. See Note 5.	AR.SRL. See Note 5.
95	AR.ASSNS. See Note 5.	AR.USERS. See Note 5.	Reserved	AR.RDRS. See Note 5.

Notes:

1. Reserved for use by utilities, runtime or related products. The value in byte zero of RD.STRUC indicates how the subsequent words 87 through 90 are interpreted. Byte zero of RD.STRUC is assigned as follows:

Value	Description
0	standard MPX-32 file structure
1	GCL file structure. GCLL last opened file unblocked. Word 87 contains byte count of file's EOF. Words 88-90 are unused.
2	GCL file structure. GCLL last opened file blocked. Words 87- 90 are unused.

2. Interpreted for explicit shared use only.
3. Bytes in RD.ASSNS are as follows:

Byte	Description
0	number of assigners on port 0
1	number of assigners on port 1

RD.ART is equivalent to RD.ASSNS, and indexes the ART information stored in words 94 and 95.

4. Bytes in RD.USERS are as follows:

Byte	Description
2	number of users on port 0
3	number of users on port 1

5. For detailed description of field contents, refer to the corresponding field within the allocated resource table (ART) in this chapter.

2.45.9 Memory Partition Descriptor (M.ME.DEQ)

The memory partition descriptor (M.ME.DEQ) defines the memory partition descriptor.

<table>
<tr><th></th><th>0 7 8 15 16 23 24 31</th></tr>
<tr><td>Word 86</td><td>Starting logical page number (ME.PPAGE)</td></tr>
<tr><td>87</td><td>Memory class (ME.MCLAS)</td></tr>
<tr><td>88</td><td>Starting logical page number (RD.LPAGE)</td></tr>
<tr><td>89</td><td>Total number of pages (RD.PGLEN)</td></tr>
<tr><td>90</td><td>Owner access rights (ME.AOWNR) | Project group access rights (ME.AUGRP). See Note. | Others access rights (ME.AOTHR). See Note. | Reserved</td></tr>
<tr><td>91-95</td><td>Reserved</td></tr>
</table>

Notes:

Access rights apply to the shared image loaded into memory. Bits in ME.AOWNR, ME.AUGRP, and ME.AOTHR are assigned as follows:

Bits	Meaning if Set
0	read access
1	write access
2-7	reserved

2.45.10 Space Allocation Map Descriptor (M.SM.DEQ)

The space allocation map descriptor (M.SM.DEQ) defines the space map descriptor block.

	0 7 8 15 16 23 24 31
Word 86	Space allocation map length in blocks (SM.SMAPL)
87	Space allocation map length in words (SM.SMAPW)
88	Number of allocation units in space map (SM.SMAPU)
89	Number of allocation units available (SM.SMAPC)
90	Number of words in last block (SM.LASTB)
91	Number of bits used in last word of map (SM.LASTW)
92	Number of blocks remaining as a fragment of allocation unit (SM.FRAG)
93-95	Reserved

2.45.11 Space Map (SMAP) Deallocation File Descriptor (M.BS.DEQ)

The SMAP deallocation file descriptor (M.BS.DEQ) defines the SMAP deallocation file descriptor. Each entry in the SMAP deallocation file is eight bytes in length (BS.ESIZE) and contains the segment definition of the disk space to be deallocated.

	0 7 8 15 16 23 24 31			
Word 86	Reserved			
87	Total number of entries that can be specified in the bad SMAP deallocation file (BS.NENTR)			
88	Total number of entries available in the bad SMAP deallocation file (BS.AVAIL)			
89-91	Reserved			
92	Number of resource assigners by the dual-port number (RD.ASSNS or RD.ART). See Note 1.		Number of resource users by the dual-port number (RD.USERS). See Note 2.	
93	Current multiport access mode (RD.CACM)	Port number of resource lock owner (RD.MPID)	Reserved	
94	AR.FLAGS. See Note 3.		AR.XRL. See Note 3.	AR.SRL. See Note 3.
95	AR.ASSNS. See Note 3.	AR.USERS. See Note 3.	Reserved	AR.RDRS. See Note 3.

Notes:

1. Bytes in RD.ASSNS are as follows:

Byte	Description
0	number of assigners on port 0
1	number of assigners on port 1

   RD.ART is equivalent to RD.ASSNS, and indexes the ART information stored in words 94 and 95.

2. Bytes in RD.USERS are as follows:

Byte	Description
2	number of users on port 0
3	number of users on port 1

3. For detailed description of field contents, refer to the corresponding field within the allocated resource table (ART) in this chapter.

2.45.12 Volume Descriptor (M.VO.DEQ)

The volume descriptor (M.VO.DEQ) defines the volume descriptor block.

	0 7 8 15 16 23 24 31		
Word 86	Binary date of last mount (VO.MTDAT)		
87	Binary time of last mount (VO.MTTIM)		
88	Binary date of last dismount (VO.DTDAT)		
89	Binary time of last dismount (VO.DTTIM)		
90	Total number of blocks on volume (VO.TOTBL)		
91	Total number of allocation units on volume (VO.TOTAU)		
92	Number of allocation units for user allocation (VO.USRAU)		
93	Number of blocks available for user allocation (VO.USRBL)		
94	Space map start address (VO.SMAPS)		
95	Space allocation map length in blocks (VO.SMAPL)		
96	Number of allocation units reflected in space map (VO.SMAPU)		
97	Number of allocation units currently available (VO.SMAPC). See Note 1.		
98	Descriptor allocation map start address (VO.DMAPS)		
99	Descriptor allocation map length in blocks (VO.DMAPL)		
100	Number of resource descriptors in descriptor map (VO.DMAPU)		
101	Number of resource descriptors currently available (VO.DMAPC)		
102	Root directory segment definition start (VO.ROOTS)		
103	Root directory segment definition size (VO.ROOTL)		
104	Blocks per allocation unit (VO.BLKAU)		
105	Number of blocks in system area (VO.SYSBL)		
106	Number of allocation units occupied by system (VO.SYSAU)		
107	Number of blocks of resource descriptors (VO.RESBL)		
108	Volume flags (VO.FLAGS). See Note 2.		
109	Number of critical I/O errors during mount sessions that ended with a proper dismount (VO.CAECT)		
110	Number of physical blocks not in logical volume (VO.FRAG)		
111	Reserved		
112-121	Number of IOCDs to boot maximum size system (VO.IOCD1 or VO.BTDSC)		
122	Number of blocks in restart code (VO.RNRST)		Number of blocks in restart image (VO.SGNOS)
123	Restart/bootstrap flags (VO.RFLGS). See Note 3.	Index to selected image for rest (VO.IMGSL)	Reserved

	0 7 8 15 16 23 24 31		
Word 124-125	SYSGEN (cold start) image name (VO.PRIMG)		
126	SYSGEN (cold start) starting block (VO.PRSTB)		
127	SYSGEN (cold start) number of blocks (VO.PRNBK)		
128	SYSGEN (cold start) image checks (VO.PRCKS)		
129	SYSGEN (cold start) number of bytes (VO.PRBCT)		
130-131	Current selected image name (VO.CUIMG)		
132	Current selected starting block number (VO.CUSTB)		
133	Current selected number of blocks (VO.CUNBK)		
134	Current selected image checksum (VO.CUCKS)		
135	Current selected number of bytes (VO.CUBCT)		
136	Sectors per block (VO.SPB)	Sectors per track (VO.SPT)	Sectors per cylinder (VO.SPC)
137	Device type (VO.DEVT). See Note 4.	Number of heads (VO.NHDS)	Sector size (VO.SSIZ)
138	Device class (VO.CLASS)	Reserved	Channel/subaddress for boot/restart (VO.CHNSA)
139	Boot device (VO.IPLDV)		
140-141	Reserved for future development use (VO.IOCD2)		
142-150	Drive attribute information (VO.DATR)		
151	Seek information for boot of primary image (VO.SEEK)		
152-159	Long resource identifier of current default system image (doubleword bounded) (VO.LRID)		

Notes:

1. This number includes some blocks allocated by the operating system.
2. Bits in VO.FLAGS are assigned as follows:

Bits	Meaning if Set
0	volume currently mounted (VO.MONTD)
1	volume is mounted under MP(DP)0 (VO.PORT0)
2	volume is mounted under MP(DP)1 (VO.PORT1)
3	volume is mounted under MP2
4	volume is mounted under MP3
5	volume is mounted under MP4
6	volume is mounted under MP5
7	volume is mounted under MP6
8	volume is mounted under MP7
9	volume is mounted under MP8

Bits	Meaning if Set
10	volume is mounted under MP9
11	volume is mounted under MPA
12	volume is mounted under MPB
13	volume is mounted under MPC
14	volume is mounted under MPD
15	volume is mounted under MPE
16	volume is mounted under MPF
17	critical access error occurred (VO.CAERR)
18-31	reserved

3. Bits in VO.RFLGS are assigned as follows:

Bits	Meaning if Set
0	program restart flag (VO.PGMRS)
1	default image established flag (VO.DEFLT)
2	bootstrap in retry mode (VO.BRTRY)
3	automatic default flag (VO.AUTO)
4	debugger requested flag (VO.DEBUG)
5-7	reserved

4. Bits in VO.DEVT are assigned as follows:

Bits	Meaning if Set
0	moving head disk (VO.MHDDT)
1	fixed head disk (VO.FHDDT)
2	cartridge module drive (VO.CMDDT)
3	reserved
4	device not present (VO.NPRES)
5	multi-/dual-port disk (VO.DUAL)
6-7	reserved

2.45.13 Segment Definitions (RD.SEGDF)

The segment definition (RD.SEGDF) total area covers words 96 to 159 of the M.RDSPD. Each entry in RD.SEGDF is a doubleword entry, with a maximum of 32 entries.

	0 7	8 15	16 23	24 31
Word 96	Absolute disk address (starting block number)			
97-159	Number of 192 word blocks			

2.45.14 User Area (RD.USER)

(RD.USER) covers words 160 to 191 of M.RDSPD and is reserved for application use.

	0 7	8 15	16 23	24 31
Word 160-190	Reserved for application use			
191	Reserved		Multiprocessor resource lock	

2.46 Disk Resident Structures

The following are disk resident structures in the order they appear in this section.

- Volume Format
- Load Module Structure
- Load Module Preamble
- Executable Image Structure
- Executable Image Preamble
- Shared Executable Image Structure
- Shared Executable Image Preamble
- Shared Image Descriptor
- COFF Load Module Structure
- COFF Executable Image Preamble
- COFF Shared Image Preamble

2.46.1 Volume Format

After a volume is formatted by J.VFMT (volume formatter), the volume has the following format:

Physical Block No.	
0-3	Bootstrap loader
4	Volume Descriptor
5	General allocatable resource descriptors' descriptor
6	Descriptors' allocation map (DMAP) descriptor
7	Space allocation map (SMAP) descriptor
8	Root directory descriptor
9	System image descriptor - if no image, reserved
10	Media deallocation file descriptor. See Note 1.
11	DMAP deallocation file descriptor
12	SMAP deallocation file descriptor
13-15	Reserved resource descriptors - for future use
16-*n*0	General allocatable resource descriptor area. See Note 2. The number of descriptors in this area is user specified with the J.VFMT format command, or the default of 1000 descriptors (blocks) is used.
*n*1-*n*2	DMAP - bit map for resource descriptors. The starting block, size, and other attributes are described by block 6. DMAP always begins on an allocation unit boundary.
*n*3-*n*4	SMAP - bit map for allocatable disk space. The starting block, size, and other attributes are described by block 7.
*n*5-*n*6	Fragment of allocation unit not used by DMAP and SMAP
*n*7-*n*8	Root directory - size varies for the number of entries specified by the user with the VFMT FORMAT command, or the default of 100 entries.
*n*9-*n*10	System image file
*n*11-*n*12	BDMAP deallocation file. The starting block, size, and other attributes are described by block 11.
*n*13-*n*14	BSMAP deallocation file. The starting block, size, and other attributes are described by block 12.
*n*15-*n*16	Free allocatable space (as described by SMAP)

Notes:

1. The cylinder closest to the spindle of all disk packs is reserved for the disk vendor's use.
2. This area is described by the DMAP area (blocks *n1* through *n*2). Block 16 is reserved for the system directory resource descriptor.

2.46.2 Load Module Structure

A load module is a permanent file of cataloged nonbase mode object code. The following is the load module structure.

Load Module Preamble
Resource Requirement Summary
CSECT Data
CSECT Relocation Matrix
DSECT Data
DSECT Relocation Matrix
Debugger Information
Module Information
Overlay

If either the CSECT or DSECT segments are empty, the empty segments are omitted.

2.46.3 Load Module Preamble

The load module preamble contains load module information in the following format:

Byte	Word	0 7 8 15 16 23 24 31			
0-4	0-1	PR.NAME — Load module name			
8-C	2-3	PR.USER — User name			
10	4	PR.MOUNT	Reserved	PR.KEY	
14	5	PR.TRAN — Module transfer address			
18	6	PR.CNTL	PR.FLAG	PR.NRRS	PR.PRIOR
1C	7	PR.PAGEC	PR.PAGED	PR.PGSIZ	PR.MEMS
20	8	PR.PAGEG	PR.FILE	PR.BUFR	PR.SEGS
24	9	PR.OPTN — Program option word			
28	10	PR.ORGC — Starting byte address of code section			
2C	11	PR.COMC	Reserved for big blocking buffers	Reserved for big blocking buffers	Reserved for MPX-32
30	12	PR.ENDC — Ending byte address of code section			
34	13	PR.SFAC — Relative file address of code section			
38	14	PR.BYTEC — Number of bytes in code section			
3C	15	PR.CHKC — Code section checksum			
40	16	PR.SFACR — Address of code section relocation matrix			
44	17	PR.BYTCR — Number of bytes in matrix			

Byte	Word	0 7 8 15 16 23 24 31
48	18	PR.CHKCR — Matrix checksum
4C	19	PR.ORGD — Starting byte address of data section
50	20	PR.COMD — Common delta
54	21	PR.ENDD — Ending byte address of data section
58	22	PR.SFAD — Relative file address of data section
5C	23	PR.BYTED — Number of bytes in data section
60	24	PR.CHKD — Data section checksum
64	25	PR.SFADR — Address of data section relocation matrix
68	26	PR.BYTDR — Number of bytes in matrix
6C	27	PR.CHKDR — Matrix checksum
70	28	PR.SYMG — Global symbol block number
74	29	PR.SYMP — Local symbol block number
78-7C	30-31	PR.DATE — Date load module was created
80	32	PR.INDEX — Block number of overlay
84	33	Reserved for MPX-32
8C	35	PR.LAS — logical address space required / Reserved for MPX-32
90	36-38	Reserved for MPX-32
9C	39	PR.SFAID — Block number of module information
A0-A4	40-41	PR.TIME — Time load module built
A8-AC	42-43	Reserved for MPX-32
B0	44	PR.MPXBR — EXTDMPX logical map address
B4	45	PR.GID / PR.FLAG3 / PR.FLAG4
B8	46	Reserved
BC-C0	47-48	Reserved for MPX-32
C4	49	PR.CATD
C8-100	50-64	Reserved for MPX-32
104	65	PR.PGO2 — Second task option word
108	66	PR.AGE — Virtual time before aging for demand page task
10C	67	PR.PRTGC / PR.PRTGD
110	68	PR.PRTGG / PR.PRTGE
114-27C	69-159	Reserved for MPX-32
280-2FC	160-191	Available for customer use

Byte (Hex)	Symbol	Description
0	PR.NAME	Load module name Field length = 2W; Left-justified and blank-filled eight-character ASCII; This name must match the file name.
8	PR.USER	User name; Field length = 2W; Left-justified and blank-filled eight-character ASCII; Zero if the USERNAME directive was not used.
10	PR.MOUNT	Mounts; Field length = 1B; Contains the number from the VOLUMES directive; Default value is zero.
	Reserved	Field length = 1B.
	PR.KEY	User key; Field length = 1H; Compressed ASCII; Zero if the USERNAME directive was not used.
14	PR.TRAN	Module transfer address; Field length = 1W; Module relative address where the module is to start execution when loaded; Bit 7 set indicates the transfer address is absolute.
18	PR.CNTL	Control; Field length = 1B;

Bit	Meaning When Set
0	overlay is in separate file format (PR.OVLS)
1	no overlays (PR.NOVL)
2	reserved
3	privileged task (PR.PRIV)
4	system administrator attribute (PR.SAM)
5-7	reserved

PR.FLAG — Flags;
Field length = 1B;

Bit	Description
0	absolute CSECT load addresses (PR.ABSC)
1	resident (program cannot be swapped out when in execution) (PR.RES)
2	shared (a single copy can be shared by two or more users) (PR.SHR)

Byte (Hex)	Symbol	Description
		Bit — Description 3 — absolute DSECT load addresses (PR.ABSD) 4 — do not attach debugger (PR.NODBG) 5 — multicopy (PR.MULTI) 6 — load base mode debugger 7 — MPX-32 Revision 2.0+ load module (PR.MPX20)
	PR.NRRS	RRS count; Field length = 1B; Number of entries in the resource requirement summary table.
	PR.PRIOR	Priority; Field length = 1B; Base execution priority of the load module.
1C	PR.PAGEC	CSECT pages; Field length = 1B; Number of 512-word pages in CSECT.
	PR.PAGED	DSECT pages; Field length = 1B; Number of 512-word pages in DSECT; Derived from the ending address of the DSECT (PR.ENDD).
	PR.PGSIZ	Block size; Field length = 1B; Defines the map block granularity required as specified in the environment directive. It is the number of 512-word protection granules in a map block (four hexadecimal).
	PR.MEMS	Memory class; Field length = 1B; The value indicates memory class as follows: one for E-class memory, two for H-class memory, and three for S-class memory. The default value is three.
20	PR.PAGEG	Common pages; Field length = 1B; Number of 512-word pages in Global Common/Datapool.
	PR.FILE	Files; Field length = 1B; Number from the FILES directive; Default value is five (the requirement for the debugger).

Byte (Hex)	Symbol	Description
	PR.BUFR	Buffers; Field length = 1B; Number from the BUFFERS directive; Default value is three (the requirement for the debugger).
	PR.SEGS	Segmented files; Number from the SEGFILES directive; Default is PR.FILE.
24	PR.OPTN	Program option word; Field length = 1W;
28	PR.ORGC	CSECT origin; Field length = 1W; Address at which to begin loading the CSECT; Bit 7 set indicates absolute origin.
2C	PR.COMC	CSECT common delta; Field length = 1B; Static buffer count; Field length = 1B. Head cell count; Field length = 1B.
	Reserved	For MPX-32; Field length = 1B.
30	PR.ENDC	CSECT ending address; Field length = 1W; Address of the first free word after the CSECT.
34	PR.SFAC	CSECT data block number; Field length = 1W; File relative block number of the CSECT data.
38	PR.BYTEC	CSECT data byte count; Field length = 1W; Number of bytes of CSECT data.
3C	PR.CHKC	CSECT data checksum; Field length = 1W; Checksum for the CSECT data; All halfwords of CSECT data are summed in a register.
40	PR.SFACR	CSECT relocation matrix block number; Field length = 1W; File relative block number of the CSECT relocation matrix.
44	PR.BYTCR	CSECT relocation matrix byte count; Field length = 1W; Number of bytes, rounded up to a multiple of four, of CSECT relocation matrix.

Byte (Hex)	Symbol	Description
48	PR.CHKCR	CSECT relocation matrix checksum; Field length = 1W; Checksum for the CSECT relocation matrix; All halfwords of CSECT relocation matrix are summed in a register.
4C	PR.ORGD	DSECT origin; Field length = 1W; Address at which to begin loading the DSECT; Bit 7 set indicates absolute origin.
50	PR.COMD	DSECT common delta; Field length = 1W; Not used, contains zero.
54	PR.ENDD	DSECT ending address; Field length = 1W; The address of the first free word after the DSECT.
58	PR.SFAD	DSECT data block number; Field length = 1W; File relative block number of the DSECT data.
5C	PR.BYTED	DSECT data byte count; Field length = 1W; Number of bytes of DSECT data.
60	PR.CHKD	DSECT data checksum; Field length = 1W; Checksum for the DSCECT data. All halfwords of DSECT data are summed in a register.
64	PR.SFADR	DSECT relocation matrix block number; Field length = 1W; File relative block number of the DSECT relocation matrix.
68	PR.BYTDR	DSECT relocation matrix byte count; Field length = 1W; Number of bytes, rounded up to a multiple of four, of DSECT relocation matrix.
6C	PR.CHKDR	DSECT relocation matrix checksum; Field length = 1W; Checksum for the DSECT relocation matrix. All halfwords of DSECT relocation matrix are summed in a register.
70	PR.SYMG	Global symbol block number; Field length = 1W; Relative sector number of the global symbol table; Zero indicates no symbols.

Byte (Hex)	Symbol	Description
74	PR.SYMP	Local symbol block number; Field length = 1W; Relative sector number of the local symbol table; Zero indicates no symbols.
78	PR.DATE	Date of creation; Field length = 2W; Date load module was created; Format identical to C.DATE.
80	PR.INDEX	Index block; Field length = 1W; Relative block number of the overlay index for single file load modules; Zero indicates no overlays.
84	Reserved	Field length = 2W.
8C	PR.LAS	Logical address size; Field length = 1H; The size in map blocks of the logical address space.
90	Reserved	For MPX-32; Field length = 3W.
9C	PR.SFAID	Block number of load module information; Field length = 1W.
A0	PR.TIME	Time load module built; Field length = 2W.
A8	Reserved	For MPX-32; Field length = 2W.
B0	PR.MPXBR	EXTDMPX logical map address; Field length = 1W.
B4	PR.GID	Task Group identification; Field length = 1B.
B5	PR.FLAG3	Flags; Field length = 1B;

Bits	Meaning if Set
0	reserved
1	enable realtime accounting (PR3.ONRA)
2	disable realtime accounting (PR3.OFRA)
3	task supplied blocking buffers (PR3.TSBB)
4	retain cataloged load module name (PR3.RCMN)
5	enable move TSA to extended (PR.ETSA)
6	request to move TSA to extended (PR.RTSA)
7	reserved

Byte (Hex)	Symbol	Description
B6	PR.FLAG4	Flags; Field length = 1HW; Bits — Meaning if Set 0 — enable mapout (PR.EMAP) 1 — request mapout (PR.RMAP) 2 — demand page this task (PR.DPG) 3 — do not demand page this task (PR.NDPG) 4 — segment mode task (PR.SEGT) 5-15 — reserved
B8	Reserved	Field length = 1W.
BC	Reserved	For MPX-32; Field length = 2W.
C4	PR.CATD	Pointer to Cataloger directives; Field length = 1W.
C8-100	Reserved	For MPX-32; Field length = 15W.
104	PR.PGO2	Second task option word; Field length = 1W.
108	PR.AGE	Virtual time before aging for demand page task Field length = 1W.
10C	PR.PRTGC	Protection granules in CSECT; Field length = 1HW.
10E	PR.PRTGD	Protection granules in DSECT; Field length = 1HW.
110	PR.PRTGG	Protection granules in GLOBAL; Field length = 1HW.
112	PR.PRTGE	Proctection granules in EXTENDED; Field length = 1HW.
114	Reserved	For MPX-32; Field length = 91W.
280	Reserved	For customer use; Field length = 32W.

2.46.4 Executable Image Structure

An executable image is a permanent file of base mode object code that was built by the Linker/X32. The following is the nonshared executable image structure.

Image Preamble
Resource Requirement Summary Table
Shared Image Descriptors
Read Only Image Section
Read/Write Image Section
Debugger Information
Relocation Lists

2.46.5 Executable Image Preamble

The executable image preamble contains image information in the following format:

Byte	Word	0 7 8 15 16 23 24 31
0	0	PR.LVER — Linker version number
4	1	PR.IVER — Image version number
8	2	PR.ROSIZ — Number of bytes in read only section
C	3	PR.RWSIZ — Number of bytes in read/write section
10	4	PR.MNT | PR.FLAG2 | PR.NSI
14	5	PR.TRAN — Transfer address
18	6	PR.CNTL | PR.FLAG | PR.NNRS | PR.BPRI
1C	7	PR.NOOL | PR.IPRI | PR.RRSSZ
20	8	PR.RRS | PR.FILE | PR.BUFR | PR.SEGS
24	9	PR.OPTN — Program option word
28	10	PR.RESVD | PR.SSIZ — Stack size in bytes
2C	11	PR.OLD — File address of overlay descriptors
30	12	PR.GST — File address of global symbol table
34	13	PR.DBG — File address of debugger information
38-4C	14-19	PR.ROSD — Read only image section descriptor
50-64	20-25	PR.RWSD — Read/write image section descriptor
68-74	26-29	PR.DBNAM — Debugger name
78-88	30-34	Reserved for MPX-32
8C	35	PR.LAS | Reserved for MPX-32
90	36	PR.REGD — Reserved for Linker
94	37	PR.OIT — Reserved for Linker
98	38	PR.SHIMG — File address of shared image descriptors
9C	39	Reserved
A0-A4	40-41	PR.TIME — Time image linked
A8-AC	42-43	PR.DATEB — Date image linked
B0	44	PR.MPXBR — EXTDMPX logical map address
B4	45	PR.GID | PR.FLAG3 | PR.FLAG4
B8	46	Reserved
BC-100	47-64	Reserved for MPX-32
104	65	PR.PGO2 — Second task option word
108	66	PR.AGE — Virtual time before aging for demand page task
10C-27C	67-159	Reserved for MPX-32
280-2FC	160-191	Reserved for customer use

Byte (Hex)	Symbol	Description
0	PR.LVER	Linker version number; Field length = 1W; Indicates which hash mechanism is used for hash accesses and the size of the global symbol table entries for use by the Symbolic Debugger/X32.
4	PR.IVER	Image version number; Field length = 1W; Task version number of the nonshared image.
8	PR.ROSIZ	Bytes in read only section; Field length = 1W; Number of bytes in the read only section.
C	PR.RWSIZ	Bytes in read/write section; Field length = 1W; Number of bytes in the read/write section.
10	PR.MNT	Dynamic mount count; Field length = 1B; Number of nonpublic volumes required for dynamic mounts; Default is zero.
	PR.FLAG2	Second flag; Field length = 1B;

Bit	Meaning if Set
0	debugger symbol support not present (PR2.NOSY)
1	debugger speed support present (PR2.DBSS)
2	task is an Ada task (PR2.ADA)
3	reserved for shared image use by owner (PR2.SHBO)
4	special arithmetic function (PR2.AF)
5	BSS section should be zeroed (PR2.ZBSS)
6	module is a COFF (PR2.COFF)
7	PTRACE debugger (PR2.PDBG)

	PR.NSI	Number of shared images included; Field length = 1H.
14	PR.TRAN	Transfer address; Field length = 1W.

Byte (Hex)	Symbol	Description
18	PR.CNTL	Control; Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	load module file status is segmented	load module status is continuous
1	link operation status (PR.LOS) is executable	link operation (PR.LOS) is nonexecutable
2	loader (PR.NOCKS) is no checksum	loader (PR.NOCKS) is checksum
3	task status (PR.PRIV) is privileged	task status (PR.PRIV) is unprivileged
4	system administrator (PR.SAM) attribute is on	system administrator (PR.SAM) attribute is off
5	overlay status is overlaid	overlay status is nonoverlaid
6	image status is shared	image status is nonshared
7	overwrite status is verified	overwrite status is not verified

PR.FLAG — Flags;
Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	address mode (PR.ABSC) is absolute	address mode (PR.ANSC) is relocatable
1	residency (PR.RES) is unswappable	residency (PR.RES) is swappable
2	read only section (PR.SHR) is shared	read only section (PR.SHR) is unshared
3	prelocation status (PR.PREL) is prelocated	prelocation status (PR.PREL) is not located
4	debugger (PR.NODBG) is not allowed. See Note.	debugger (PR.NODBG) is allowed
5	task configuration (PR.MULTI) is multicopied	task configuration (PR.MULTI) is unique
6	base mode executable image PR.BASE is always set	
7	PR.MPX20 always set	

Note: If bit 4 is set, the file and buffer assignment totals in PR.FILE and PR.BUFR are incremented for debugger support.

PR.NNRS — Resource requirement summary entry;
Field length = 1B;
Number of entries in the resource requirement summary table.

Byte (Hex)	Symbol	Description
	PR.BPRI	Execution priority; Field length = 1B; Base execution priority.
1C	PR.NOOL	Overlay; Field length = 1H; Number of overlays; Should be zeroed.
	PR.IPRI	I/O priority; Field length = 1B; Base I/O priority level; Default is equal to PR.BPRI.
	PR.RRSSZ	Resource requirement summary size; Field length = 1B; Size in blocks of the resource requirement summary table.
20	PR.RRS	Resource requirement summary address; Field length = 1B; File address of the resource requirement summary table.
	PR.FILE	Files; Field length = 1B; Total number of files that can be concurrently open during task execution. This number is the sum of the file allocations specified in the LINKER/X32 FILES directive plus the number of files specified in any shared images included in the task.
	PR.BUFR	Buffers; Field length = 1B; Total number of blocking buffers required by the task. This number is the sum of the buffers specified in the LINKER/X32 BUFFERS directive plus the number of buffers specified in any shared images included in the task.
	PR.SEGS	Segment definition; Field length = 1B; Segment definition area count. This number is the sum of the segmented files specified in the LINKER/X32 SEGFILES directive plus the number of files specified in any shared images included in the task.
24	PR.OPTN	Program option word; Field length = 1W;

Byte (Hex)	Symbol	Description
28	PR.RESVD	Reserved; Field length = 1B.
	PR.SSIZ	Stack size; Field length = 3B; Program stack size in bytes. This number is the sum of the STACKSIZE specified in the LINKER/X32 STACKSIZE directive plus the number of files specified in any shared images included in the task.
2C	PR.OLD	Overlay descriptors; Field length = 1W; Should be zeroed.
30	PR.GST	Global symbol table; Field length = 1W; File address of the Global Symbol Table.
34	PR.DBG	Debugger information; Field length = 1W; File address of debugger information.
38	PR.ROSD	Read only image section descriptor; Field length = 6W;

Word	Definition
0	load address (RO.LADD)
1	sector address (RO.FADD)
2	length in bytes (RO.SIZE)
3	checksum (RO.CKSM)
4	relocation list sector address (RO.RLAD)
5	number of relocation entries (RO.NRE)

50	PR.RWSD	Read/write image section descriptor; Field length = 6W;

Word	Definition
0	load address (RW.LADD)
1	sector address (RW.FADD)
2	length in bytes (RW.SIZE)
3	checksum (RW.CKSM)
4	relocation list sector address (RW.RLAD)
5	number of relocation entries (RW.NRE)

68	PR.DBGNM	Debugger name; Field length = 4W; Debugger name to be used if different from the default debugger name.
78-88	Reserved	For MPX-32

Byte (Hex)	Symbol	Description
8C	PR.LAS	Logical address size; Field length = 1H; The size in map blocks of the logical address space.
	Reserved	For MPX-32; Field length = 1H.
90	PR.REGD	Region descriptor; Field length = 1W; Should be zeroed.
94	PR.OIT	Overlay information table; Field length = 1W; Should be zeroed.
98	PR.SHIMG	Shared image descriptors; Field length = 1W; File address of shared image descriptors.
9C	Reserved	For MPX-32; Field length = 1W.
A0	PR.TIME	Time image linked or relinked; Field length = 2W.
A8	PR.DATEB	Date image linked or relinked; Field length = 2W.
B0	PR.MPXBR	EXTDMPX logical map address; Field length = 1W.
B4	PR.GID	Task group identification; Field length = 1W.
B5	PR.FLAG3	Flags; Field length = 1B; Bit — Meaning if Set 0 — reserved 1 — enable realtime accounting (PR3.ONRA) 2 — disable realtime accounting (PR3.OFRA) 3-4 — reserved
B6	PR.FLAG4	Flags; Field length = 1HW.
B8	Reserved	Field length = 1W.
BC	Reserved	For MPX-32; Field length = 18W.
104	PR.PGO2	Second task option word; Field length = 1W.
108	PR.AGE	Virtual time before aging for demand page task Field length = 1W.
10C	Reserved	For MPX-32 Field length = 93W.
280	Reserved	For customer use; Field length = 32W.

2.46.6 Shared Executable Image Structure

An executable image is a permanent file of base mode object code that was built by the Linker/X32. The following is the shared executable image structure.

Shared Image Preamble
Shared Image Descriptors
Read Only Image Section
Read/Write Image Section
Read/Write Writeback Image Section
Universal Symbol Table
Global Common Program Image Section Information

2.46.7 Shared Executable Image Preamble

The shared image preamble contains image information in the following format:

Byte	Word	0 7 8 15 16 23 24 31
0	0	PR.LVER — Linker version number
4	1	PR.IVER — Shared image version number
8	2	PR.COMP — Compatibility level
C	3	PR.PHADS — Physical load address
10	4	PR.MNT | PR.FLAG2 | PR.NSI
14	5	PR.TRAN — Should be zeroed
18	6	PR.CNTL | PR.FLAG | Reserved for Linker
1C	7	Reserved for Linker
20	8	Reserved | PR.FILE | PR.BUFR | PR.SEGS
24	9	Reserved for Linker
28	10	PR.SSIZE — Stack size in bytes (first byte reserved)
2C	11	PR.GCMFA — File address of global common definitions
30	12	PR.GCMNE — Number of global common definitions
34	13	PR.DBG — File address of debugger information
38-4C	14-19	PR.ROSD — Read only image section descriptor
50-64	20-25	PR.RWSD — Read/write image section descriptor
68-7C	26-31	PR.RWWB — Read/write writeback image section descriptor
80	32	PR.UST — File address of universal symbol table
84	33	Reserved for MPX-32
88	34	Reserved for MPX-32
8C	35	PR.USTSO — Sector offset for universal symbol table
90	36	PR.USHLN — Universal symbol hash table length in bytes
94	37	PR.USTLN — Universal symbol table length in bytes
98	38	PR.SHIMG — File address of shared image descriptors
9C	39	Reserved for MPX-32
A0-A4	40-41	PR.TIME — Time image linked
A8-AC	42-43	PR.DATE — Date image linked
B0-27C	44-159	Reserved for MPX-32
280-2FC	160-191	Reserved for customer use

Some of the above reserved areas are used by the Linker/X32. They are not used by the MPX-32 operating system.

Byte (Hex)	Symbol	Description
0	PR.LVER	Linker version number; Field length = 1W; Indicates that hash mechanism is used for hash accesses and the size of the global symbol table entries for use by the Symbolic Debugger/X32.
4	PR.IVER	Shared image version number; Field length = 1W; Task version number of the shared image.
8	PR.COMP	Compatibility level; Field length = 1W; The lowest version number that is compatible with this copy of the shared image.
C	PR.PHADS	Physical load address; Field length = 1W; The physical address where the shared image is loaded.
10	PR.MNT	Dynamic mount count; Field length = 1B; Number of nonpublic volumes required for dynamic mounts; Default is zero.
	PR.FLAG2	Second flag; Field length = 1B;

Bit	Meaning if Set
0	debugger symbol support not present (PR2.NOSY)
1	debugger speed support present (PR2.DBSS)
2	task is an ADA task (PR2.ADA)
3	share image by owner (PR2.SHBO)
4-7	reserved for executable image use

	PR.NSI	Number of shared images included; Field length = 1H.
14	PR.TRAN	Should be zeroed; Field length = 1W.

Byte (Hex)	Symbol	Description
18	PR.CNTL	Control; Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	load module file status is segmented	load module status is continuous
1	link operation status (PR.LOS) is executable	link operation status (PR.LOS) is nonexecutable
2	loader (PR.NOCKS) is no checksum	loader (PR.NOCKS) is checksum
3	task status (PR.PRIV) privileged	task status (PR.PRIV) is unprivileged
4	system administrator (PR.SAM) attribute is on	system administrator (PR.SAM) attribute is off
5	overlay status is overlaid	overlay status is nonoverlaid
6	image status is shared	image status is nonshared
7	overwrite status is verified	overwrite status is not verified

PR.FLAG Flags;
Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	address mode (PR.ABSC) is absolute	address mode (PR.ABSC) is relocatable
1	residency (PR.RES) is unswappable	residency (PR.RES) is swappable
2	read only section (PR.SHR) is shared	read only section (PR.SHR) is unshared
3	prelocation status (PR.PREL) is prelocated	prelocation status (PR.PREL) is not prelocated
4	debugger (PR.NODBG) is not allowed. See Note.	debugger (PR.NODBG) is allowed
5	task configuration (PR.MULTI) is multicopied	task configuration (PR.MULTI) is unique
6	base register load module (PR.BASE) is always set	
7	PR.MPX20 is always set	

Note: If bit 4 is set, the file and buffer assignment totals in PRE.FLE and PR.BUFR are incremented for debugger support.

Byte (Hex)	Symbol	Description
	Reserved	For Linker; Field length = 1H.
1C	Reserved	For Linker; Field length = 1W.
20	Reserved	For MPX-32; Field length = 1B.
	PR.FILE	Files; Field length = 1B; Number of dynamic file allocations specified in the Linker/X32 FILES directive.
	PR.BUFR	Buffers; Field length = 1B; Number of dynamic blocked file allocations specified in the Linker/X32 BUFFERS directive.
	PR.SEGS	Segment definition; Field length = 1B; Segment definition area counts specified in LINKER/X32 SEGFILES directive.
24	Reserved	For Linker; Field length = 1W.
28	PR.SSIZE	Reserved; Field length = 1B.
		Stack size; Field length = 3B; Program stack size in bytes specified in LINKER/X32 STACKSIZE directive.
2C	PR.GCMFA	File address of global common definitions; Field length = 1W.
30	PR.GCMNE	Number of global common definitions; Field length = 1W.
34	PR.DBG	Debugger information; Field length = 1W; Should be zeroed.
38	PR.ROSD	Real only image section descriptor; Field length = 6W;

Word	Definition
0	load address (RO.LADD)
1	sector address (RO.FADD)
2	length in bytes (RO.SIZE)
3	checksum (RO.CKSM)
4	relocation list sector address (RO.RLAD)
5	number of relocation entries (RO.NRE)

Byte (Hex)	Symbol	Description
50	PR.RWSD	Read/write image section descriptor; Field length = 6W; Word — Definition 0 load address (RW.LADD) 1 sector address (RW.FADD) 2 length in bytes (RW.SIZE) 3 checksum (RW.CKSM) 4 relocation list sector address (RW.RLAD) 5 Number of relocation entries (RW.NRE)
68	PR.RWWB	Read/write image section descriptor for writeback; Field length = 6W; Word — Definition 0 load address (WB.LADD) 1 sector address (WB.FADD) 2 length in bytes (WB.SIZE) 3 checksum (WB.CKSM) 4 relocation list sector address (WB.RLAD) 5 number of relocation entries (WB.NRE)
80	PR.UST	Universal symbol table; Field length = 1W; File address of the universal symbol table.
84	Reserved	For MPX-32; Field length = 1W.
88	Reserved	For MPX-32; Field length = 1W.
8C	PR.USTSO	Universal symbol table offset; Field length = 1W; Sector offset for the universal symbol table.
90	PR.USHLN	Universal symbol hash table; Field length = 1W; Length in bytes of the universal symbol hash table.
94	PR.USTLN	Universal symbol table; Field length = 1W; Length in bytes of the universal symbol table.
98	PR.SHIMG	File Address of shared image descriptors; Field length = 1W.

Byte (Hex)	Symbol	Description
9C	Reserved	Field length = 1W.
A0	PR.TIME	Time image linked or relinked; Field length = 2W.
A8	PR.DATE	Date image linked or relinked; Field length = 2W.
B0	Reserved	For MPX-32; Field length = 115W.
280	Reserved	For customer use; Field length = 32W.

2.46.8 Shared Image Descriptors

Every shared image has a shared image descriptor that is doubleword bounded. Shared image descriptors are used by the loader to verify that a task can access the required shared images.

Preassigned shared images are loaded at task activation. Other shared images are loaded by the task through run-time shared image calls.

	0 7	8 15	16 23	24 31
Word 0	Logical load address of shared image (SI.LOAD)			
1	Version number of shared image at link time (SI.VERS)			
2	SI.FLGS. See Note 1.	SI.PLEN See Note 2.	Reserved. See Note 3.	
3	Pathname identifier (SI.PNID)			
4-21	Pathname block of up to 72 bytes for shared images (SI.PNAME)			

Notes:

1. Bits in SI.FLGS are assigned as follows:

Bits	Meaning if Set
0	preassigned shared image
1	read/write access requested (SI.RWAC)
2	modified read/write image section request
3	position dependent shared image (SI.PDEP)
4	writeback mode requested (SI.WRBK)
5	writeback section present
6-7	reserved

2. Byte 1 is the shared image pathname length (SI.PLEN).
3. Byte 2 is reserved for LINKX32 (should be zeroed). Byte 3 is reserved for LINKX32-inclusion level of the shared image.

2.46.9 COFF Load Module Structure

The COFF load module enables loading and execution of tasks developed in the environment. The module has the following structure:

File Header (32 bytes) Unused by MPX-32
Preamble (736 bytes)
Resource Requirement Summary Table (multiple of 768 bytes)
Shared Image Descriptor Table (multiple of 768 bytes)
Section Headers
Read Only Section (Code)
Read/Write Section (data + bss)
Debugger Information

2.46.10 COFF Executable Image Preamble

The COFF Executable Image preamble follows the 32 byte file header and contains image information in the following format:

Byte	Word	0 7	8 15	16 23	24 31
0	0	PR.LVER — Linker version number			
4	1	PR.IVER — Image version number			
8	2	PR.ROSIZ — Number of bytes in read only section			
C	3	PR.RWSIZ — Number of bytes in read/write section			
10	4	PR.MNT	PR.FLAG2	PR.NSI	
14	5	PR.TRAN — Transfer address			
18	6	PR.CNTL	PR.FLAG	PR.NNRS	PR.BPRI
1C	7	PR.NOOL		PR.IPRI	PR.RRSSZ
20	8	PR.RRS	PR.FILE	PR.BUFR	PR.SEGS
24	9	PR.OPTN — Program option word			
28	10	PR.RESVD	PR.SSIZ — Stack size in bytes		
2C-34	11-13	Reserved for MPX-32			
38-4C	14-19	PR.ROSD — Read only image section descriptor			
50-64	20-25	PR.RWSD — Read/write image section descriptor			
68-74	26-29	PR.DBNAM — Debugger name			
78-88	30-34	Reserved			
8C	35	PR.LAS		Reserved	
90	36	PR.REGD — Reserved for Linker			
94	37	PR.OIT — Reserved for Linker			
98	38	PR.SHIMG — File address of shared image descriptors			
9C	39	Reserved			
A0-A4	40-41	PR.TIME — Time image linked			
A8-AC	42-43	PR.DATEB — Data image linked			
B0	44	Reserved for PR.MPXBR			
B4	45	Reserved for PR.GID	PR.FLAG3	PR.FLAG4	
B8	46	PR.BSS — Size of COFF — type BSS section			
BC-EC	47-59	Reserved for MPX-32			
F0-100	60-64	PR.STBRG			
104	65	PR.PGO2 — Second task option word			
108	66	PR.AGE — Virtual time before aging for demand page task			
10C-27C	67-159	Reserved for MPX-32			
280-2DC	160-183	Reserved for customer use			

Byte (Hex)	Symbol	Description
0	PR.LVER	Linker version number; Field length = 1W; Indicates which hash mechanism is used for hash accesses and the size of the global symbol table entries for use by the symbolic debugger.
4	PR.IVER	Image version number; Field length = 1W; Task version number of the nonshared image.
8	PR.ROSIZ	Bytes in read only section; Field length = 1W; Number of bytes in the read only section.
C	PR.RWSIZ	Bytes in read/write section; Field length = 1W; Number of bytes in the read/write section.
10	PR.MNT	Dynamic mount count; Field length = 1B; Number of nonpublic volumes required for dynamic mounts; Default is zero.
	PR.FLAG2	Second flag; Field length = 1B;

Bits	Meaning if Set
0	debugger symbol support not present (PR2.NOSY)
1	debugger speed support present (PR2.DBSS)
2	task is an Ada task (PR2.ADA)
3	reserved for shared image use by owner (PR2.SHBO)
4	special arithmetic function (PR2.AF)
5	BSS section should be zeroed (PR2.ZBSS)
6	module is a COFF load module (PR2.COFF)
7	PTRACE Debugger (PR2.PDBG)

	PR.NSI	Number of shared images included; Field length = 1H.
14	PR.TRAN	Transfer address; Field length = 1W.

Byte (Hex)	Symbol	Description
18	PR.CNTL	Control; Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	load module file status is segmented	load module status is continuous
1	link operation status (PR.LOS) is executable	link operation (PR.LOS) is nonexecutable
2	loader (PR.NOCKS) is no checksum	loader (PR.NOCKS) is checksum
3	task status (PR.PRIV) is privileged	task status (PR.PRIV) is unprivileged
4	system administrator (PR.SAM) attribute is on	system administrator (PR.SAM) attribute is off
5	overlay status is overlaid	overlay status is nonoverlaid
6	image status is shared	image status is nonshared
7	overwrite status is verified	overwrite status is not verified

PR.FLAG Flags:
Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	address mode (PR.ABSC) is absolute. Always set.	n/a
1	residency (PR.RES) is unswappable	residency (PR.RES) is swappable
2	read only section (PR.SHR) is shared	read only section (PR.SHR) is unshared
3	prelocation status (PR.PREL) is prelocated	prelocation status (PR.PREL) is not prelocated
4	debugger (PR.NODBG) is not allowed. See Note.	debugger (PR.NODBG) is allowed
5	task configuration (PR.MULTI) is multicopied	task configuration (PR.MULTI) is unique
6	base mode executable image PR.BASE is always set	
7	PR.MPX20 always set	

Note: If bit 4 is set, the file and buffer assignment totals in PR.FILE and PR.BUFR are incremented for debugger support.

PR.NNRS Resource requirement summary entry;
Field length = 1B;
Number of entries in the resource requirement summary table.

Byte (Hex)	Symbol	Description
	PR.BPRI	Execution priority; Field length = 1B; Base execution priority.
1C	PR.NOOL	Overlay; Field length = 1H; Number of overlays; Should be zeroed.
	PR.IPRI	I/O priority; Field length = 1B; Base I/O priority level; Default is equal to PR.BPRI.
	PR.RRSSZ	Resource requirement summary size; Field length = 1B; Size in blocks of the resource requirement summary table.
20	PR.RRS	Resource requirement summary address; Field length = 1B; File address of the resource requirement summary table.
	PR.FILE	Files; Field length = 1B; Total number of files that can be concurrently open during task execution. This number is the sum of the file allocations specified in the Linker FILES directive plus the number of files specified in any shared images included in the task.
	PR.BUFR	Buffers; Field length = 1B; Total number of blocking buffers required by the task. This number is the sum of the buffers specified in the Linker BUFFERS directive plus the number of buffers specified in any shared images included in the task.
	PR.SEGS	Segment definition; Field length = 1B; Segment definition area count. This number is the sum of the segmented files specified in the Linker SEGFILES directive plus the number of files specified in any shared images included in the task.
24	PR.OPTN	Program option word; Field length = 1W.
28	PR.RESVD	Reserved; Field length = 1B.

Byte (Hex)	Symbol	Description
	PR.SSIZ	Stack size; Field length = 3B; Program stack size in bytes. This number is the sum of the STACKSIZE specified in the Linker STACKSIZE directive plus the number of files specified in any shared images included in the task.
2C		Reserved for MPX-32; Field length = 3W.
38	PR.ROSD	Read only image section descriptor; Field length = 6W;

Word	Definition
0	load address (RO.LADD)
1	byte offset from beginning of file (RO.FADD)
2	length in bytes (RO.SIZE)
3	checksum (RO.CKSM)
4	byte offset of relocation list from beginning of file (RO.LAD)
5	number of relocation entries (RO.NRE)

50	PR.RWSD	Read/write image section descriptor; Field length = 6W;

Word	Definition
0	load address (RW.LADD)
1	byte offset from beginning of file (RW.FADD)
2	length in bytes (RW.SIZE)
3	checksum (RW.CKSM)
4	byte offset of relocation list from beginning of file (RW.RLAD)
5	number of relocation entries (RW.NRE)

68	PR.DBNAM	Debugger name; Field length = 4W; Debugger name to be used if different from the default debugger name.
78-88	Reserved	Field length = 5W.
8C	PR.LAS	Logical address size; Field length = 1H; The size in map blocks of the logical address space
	Reserved	For MPX-32; Field length = 1H.

Byte (Hex)	Symbol	Description
90	PR.REGD	Region descriptor; Field length = 1W; Should be zeroed.
94	PR.OIT	Reserved Field length = 1W; Should be zeroed.
98	PR.SHIMG	Shared image descriptors; Field length = 1W; File address of shared image descriptors.
9C	Reserved	For MPX-32; Field length = 1W.
A0	PR.TIME	Time image linked or relinked; Field length = 2W.
A8	PR.DATEB	Date image linked or relinked; Field length = 2W.
B0	Reserved	For PR.MPXBR; Field length = 1W.
B4	Reserved	For PR.GID; Field length = 1B.
B5	PR.FLAG3	Flags; Field length = 1B;

Bit	Meaning if Set
0	pseudo interrupt receiver address requires 1 word offset (PR3.OFF)
1	enable realtime accounting (PR3.ONRA)
2	disable realtime accounting (PR3.OFRA)
3-4	reserved

B6	PR.FLAG4	Flags; Field length = 1H.
B8	PR.BSS	Size of COFF-type BSS section; Field length = 1W; The size in bytes needed to load the COFF BSS section.
BC-EC	Reserved	For MPX-32; Field length = 13W.
F0-100	PR.STBRG	LD assigned base register values to be loaded at task startup (B3-B7); Field length = 5W.

Byte (Hex)	Symbol	Description
104	PR.PGO2	Second task option word; Field length = 1W.
108	PR.AGE	Virtual time before aging for demand page task; Field length = 1W.
10C-27C	Reserved	For MPX-32; Field length = 93W.
280-2DC	Reserved	For customer use; Field length = 24W.

2.46.11 COFF Shared Image Preamble

The COFF shared image preamble follows the 32 byte file header and contains image information in the following format:

Byte	Word	0 7 8 15 16 23 24 31
0	0	PR.LVER — Linker version number
4	1	PR.IVER — Shared image version number
8	2	PR.COMP — Compatibility level
C	3	PR.PHADS — Physical load address
10	4	PR.MNT | PR.FLAG2 | PR.NSI
14	5	PR.TRAN — Should be zeroed
18	6	PR.CNTL | PR.FLAG | Reserved for Linker
1C	7	Reserved for Linker
20	8	Reserved | PR.FILE | PR.BUFR | PR.SEGS
24	9	Reserved for Linker
28	10	PR.SSIZE — Stack size in bytes (first byte reserved)
2C-34	11-13	Reserved for MPX-32
38-4C	14-19	PR.ROSD — Read only image section descriptor
50-64	20-25	PR.RWSD — Read/write image section descriptor
68-7C	26-31	PR.RWWB — Read/write writeback image section descriptor
80	32	PR.UST — File address of universal symbol table
84	33	Reserved for MPX-32
88	34	Reserved for MPX-32
8C	35	PR.USTSO — Sector offset for universal symbol table
90	36	PR.USHLN — Universal symbol hash table length in bytes
94	37	PR.USTLN — Universal symbol table length in bytes
98	38	PR.SHIMG — File address of shared image descriptors
9C	39	Reserved for MPX-32
A0-A4	40-41	PR.TIME — Time image linked
A8-AC	42-43	PR.DATE — Date image linked
B0-25C	44-151	Reserved for MPX-32
260-28C	152-183	Reserved for customer use

Some of the above reserved areas are used by the Linker. They are not used by the MPX-32 operating system.

Byte (Hex)	Symbol	Description
0	PR.LVER	Linker version number; Field length = 1W; Indicates that hash mechanism is used for hash accesses and the size of the global symbol table entries for use by the symbolic debugger.
4	PR.IVER	Shared image version number; Field length = 1W; Task version number of the shared image.
8	PR.COMP	Compatibility level; Field length = 1W; The lowest version number that is compatible with this copy of the shared image.
C	PR.PHADS	Physical load address; Field length = 1W; The physical address where the shared image is loaded.
10	PR.MNT	Dynamic mount count; Field length = 1B; Number of nonpublic volumes required for dynamic mounts; Default is zero.
	PR.FLAG2	Second flag; Field length = 1B;

Bit	Meaning if Set
0	debugger symbol support not present (PR2.NOSY)
1	debugger speed support present (PR2.DBSS)
2	task is an ADA task (PR2.ADA)
3	share image by owner (PR2.SHBO)
4	reserved for executable image use
5	reserved
6	module is a COFF load module (PR.COFF)
7	reserved

	PR.NSI	Number of shared images included; Field length = 1H.
14	PR.TRAN	Should be zeroed; Field length = 1W.

Byte (Hex)	Symbol	Description
18	PR.CNTL	Control; Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	load module file status is segmented	load module status is continuous
1	link operation status (PR.LOS) is executable	link operation status (PR.LOS) is nonexecutable
2	loader (PR.NOCKS) is no checksum	loader (PR.NOCKS) is checksum
3	task status (PR.PRIV) privileged	task status (PR.PRIV) is unprivileged
4	system administrator (PR.SAM) attribute is on	system administrator (PR.SAM) attribute is off
5	overlay status is overlaid	overlay status is nonoverlaid
6	image status is shared	image status is nonshared
7	overwrite status is verified	overwrite status is not verified

PR.FLAG Flags;
Field length = 1B;

Bit	Meaning if Set	Meaning if Reset
0	address mode (PR.ABSC) is absolute	address mode (PR.ABSC) is relocatable
1	residency (PR.RES) is unswappable	residency (PR.RES) is swappable
2	read only section (PR.SHR) is shared	read only section (PR.SHR) is unshared
3	prelocation status (PR.PREL) is prelocated	prelocation status (PR.PREL) is not prelocated
4	Debugger (PR.NODBG) is not allowed. See Note.	Debugger (PR.NODBG) is allowed
5	task configuration (PR.MULTI) is multicopied	task configuration (PR.MULTI) is unique
6	base register load module (PR.BASE) is always set	
7	PR.MPX20 is always set	

Note: If bit 4 is set, the file and buffer assignment totals in PRE.FLE and PR.BUFR are incremented for debugger support.

Byte (Hex)	Symbol	Description
	Reserved	For Linker; Field length = 1H.
1C	Reserved	For Linker; Field length = 1W.
20	Reserved	For MPX-32; Field length = 1B.
	PR.FILE	Files; Field length = 1B; Number of dynamic file allocations specified in the Linker FILES directive.
	PR.BUFR	Buffers; Field length = 1B; Number of dynamic blocked file allocations specified in the Linker BUFFERS directive.
	PR.SEGS	Segment definition; Field length = 1B; Segment definition area counts specified in Linker SEGFILES directive.
24	Reserved	For Linker; Field length = 1W.
28	PR.SSIZE	Reserved; Field length = 1B.
		Stack size; Field length = 3B; Program stack size in bytes specified in Linker STACKSIZE directive.
2C	Reserved	For MPX-32; Field length = 3W.
38	PR.ROSD	Read only image section descriptor; Field length = 6W;

Word	Definition
0	load address (RO.LADD)
1	sector address (RO.FADD)
2	length in bytes (RO.SIZE)
3	checksum (RO.CKSM)
4	relocation list sector address (RO.RLAD)
5	number of relocation entries (RO.NRE)

Byte (Hex)	Symbol	Description
50	PR.RWSD	Read/write image section descriptor; Field length = 6W; Word — Definition 0 — load address (RW.LADD) 1 — sector address (RW.FADD) 2 — length in bytes (RW.SIZE) 3 — checksum (RW.CKSM) 4 — relocation list sector address (RW.RLAD) 5 — number of relocation entries (RW.NRE)
68	PR.RWWB	Read/write image section descriptor for writeback; Field length = 6W; Word — Definition 0 — load address (WB.LADD) 1 — sector address (WB.FADD) 2 — length in bytes (WB.SIZE) 3 — checksum (WB.CKSM) 4 — relocation list sector address (WB.RLAD) 5 — number of relocation entries (WB.NRE)
80	PR.UST	Universal symbol table; Field length = 1W; File address of the universal symbol table.
84	Reserved	For MPX-32; Field length = 1W.
88	Reserved	For MPX-32; Field length = 1W..
8C	PR.USTSO	Universal symbol table offset; Field length = 1W; Sector offset for the universal symbol table.
90	PR.USHLN	Universal symbol hash table; Field length = 1W; Length in bytes of the universal symbol hash table.
94	PR.USTLN	Universal symbol table; Field length = 1W; Length in bytes of the universal symbol table.
98	PR.SHIMG	File Address of shared image descriptors; Field length = 1W.
9C	Reserved	Field length = 1W.

Byte (Hex)	Symbol	Description
A0	PR.TIME	Time image linked or relinked; Field length = 2W.
A8	PR.DATE	Date image linked or relinked; Field length = 2W.
B0	Reserved	For MPX-32; Field length = 107W.
260	Reserved	For customer use; Field length = 22W.

2.47 Internal J.VFMT Structures

2.47.1 Newboot Macro Offsets (M.BO.EQU)

M.BO.EQU is part of the MPX-32 Macro Library providing symbolic access to the new disk bootstrap program for J.VFMT and RESTART. This macro's equates are offsets into the new disk bootstrap code which is contained within J.VFMT and written to the disk.

Using this macro with the new bootstrap program removes the requirement for RESTART to reference absolute memory locations. Additionally, J.VFMT's code verifies that the bootstrap code generated during assembly is consistent with the equate values within this macro. These two features simplify changing the bootstrap and reduce unexpected coding errors among bootstrap, J.VFMT, and RESTART.

The macro contains the following definitions (offsets):

IOCD.WD0	EQU	X'00'	class F IOCD word 0
IOCD.OP	EQU	X'00'	class F IOCD op code byte
IOCD.XFR	EQU	X'00'	class F IOCD transfer address
IOCD.WD1	EQU	X'04'	class F IOCD word 1
IOCD.FLG	EQU	X'04'	class F IOCD flag byte
IOCD.CNT	EQU	X'06'	class F IOCD transfer count
IOCD.LEN	EQU	X'08'	class F IOCD length
BO.STRT	EQU	X'00'	bootstrap code start
BO.PSD1	EQU	X'00'	new PSD word 1 after IPL
BO.PSW	EQU	X'00'	another name for BO.PSD1
BO.XFER	EQU	X'00'	bootstrap transfer address (PSW PC)
BO.PSD2	EQU	X'04'	new PSD word 2 after IPL
BO.REST	EQU	X'04'	memory location stuffed by RESTART
BO.IOCL	EQU	X'08'	location for chained IOCDs to sustain IPL
BO.IOCLN	EQU	4	number of chained-to-IPL IOCDs before TIC
BO.TIC	EQU	X'28'	location of TIC IOCD (TIC to image load IOCDs)
BO.MODAT	EQU	X'30'	contains Load Mode data byte
BO.DUMMY	EQU	X'31'	reserved (dummy) byte
BO.NIOCD	EQU	X'32'	contains number of read IOCDs space reserved for
BO.RSTPC	EQU	X'34'	location of restart's PC
BO.REV	EQU	X'38'	revision number of this bootstrap
BO.INCH	EQU	X'40'	location of INCH IOCD
BO.NOP	EQU	X'48'	location of No Op/Init Cont IOCD
BO.INCC	EQU	X'48'	another name for BO.NOP
BO.LMOD	EQU	X'50'	location of Load Mode IOCD
unnamed	EQU	X'58'	normal location for target address of TIC IOCD
BO.SEEK	EQU	X'00'	offset from TIC target to image Seek IOCD
BO.READ	EQU	X'08'	offset from TIC target to first image Read IOCD
	(Number of Read IOCD prototypes is in BO.NIOCD)		
VD.PBUFF	EQU	X'200'	buffer address in bootstrap where partial volume descriptor read in from IPL disk
VD.DATAL	EQU	X'3C'	length of partial volume descriptor buffer

2.47.2 Disk Parameter Table Structures

2.47.2.1 Disk Parameter Table Offsets (M.DPT)

The macro M.DPT is part of the MPX-32 Macro Library. It is used to interface to the new disk parameter table in J.VFMT.

The macro contains the following definitions (offsets):

```
DPT.DDTC    EQU   X'00'   disk device type code
DPT.SPB     EQU   X'08'   number sectors per block
DPT.SPAU    EQU   X'09'   number sectors per allocation unit
DPT.SPT     EQU   X'0A'   number sectors per track
DPT.NHDS    EQU   X'0B'   number read/write heads
DPT.SSIZ    EQU   X'0C'   formatted sector size (number words)
DPT.RAWS    EQU   X'0E'   raw (unforamtted) sector size (bytes)
DPT.RAWF    EQU   X'10'   raw sector size fragment (bits)
DPT.RSB1    EQU   X'11'   reserved
DPT.NCYL    EQU   X'12'   total number cylinders on disk
DPT.SECS    EQU   X'14'   total number sectors on disk
DPT.RAWT    EQU   X'18'   raw track size (capacity) in bytes
DPT.RSW1    EQU   X'1C'   reserved
DPT.MODL    EQU   X'20'   Encore Disk Model Number
DPT.SIZE    EQU   X'28'   size of each DPT entry
```

2.47.2.2 Disk Parameter Table Format (SJ.VFDPT)

The following diagram shows the format for each DPT entry in the disk parameter table (SJ.VFDPT).

```
MODEL NUM     (DPT.MODL).....................................................
RESERVED      ..............................................................  :
RAW TRK SIZ   (DPT.RAWT)..................................................  :  :
TOT SECTS     (DPT.SECS)..............................................     :  :  :
NUM CYLS      (DPT.NCYL)..........................                :     :  :  :
RESERVED      ..................................    :     :        :     :  :  :
R SECT FRAG   (DPT.RAWF)...................... :    :     :        :     :  :  :
RAW SECT SZ   (DPT.RAWS)...................... : :    :     :        :     :  :  :
FMT SECT SZ   (DPT.SSIZ).................    : : :    :     :        :     :  :  :
NUM HEADS     (DPT.NHDS).............   :    : : :    :     :        :     :  :  :
SECT/TRACK    (DPT.SPT)...........  :   :    : : :    :     :        :     :  :  :
SECT/ALC UN   (DPT.SPAU).......  :   :   :    : : :    :     :        :     :  :  :
SECT/BLOCK    (DPT.SPB).....  :   :   :   :    : : :    :     :        :     :  :  :
DISK TYPE     (DPT.DDTC).  :   :   :   :   :    : : :    :     :        :     :  :  :
                       :   :   :   :   :   :    : : :    :     :        :     :  :  :
                       :   :   :   :   :   :    : : :    :     :        :     :  :  :
                       :   :   :   :   :   :    : : :    :     :        :     :  :  :
                       :   :   :   :   :   :    : : :    :     :        :     :  :  :
DPT.E FORM           64, 8, 8, 8, 8, 16,  16,8,8,  16,     32,    32,32,     64
DPT.E          C'MMNNNN', N,NN,NN,NN,NNN,NNNN,N,N,NNNN,NNNNNNN,NNNNN,NN,C'NNNN'

DPT.E          C'DP0080', 1,02,22,05,192,0912,0,0,0823,0090530,20160,00,C'8138'
DPT.E          C'DP0337', 1,05,45,10,192,0908,0,0,0823,0370350,40960,00,C'8887'
DPT.E          C'DP0474', 1,05,30,20,192,0934,0,0,0842,0505200,28160,00,C'8884'
DPT.E          C'DP0500', 1,05,45,10,192,0909,0,0,1217,0547650,41088,00,C'8812'
DPT.E          C'DP0689', 1,09,45,20,192,0908,0,0,0842,0757800,40960,00,C'8889'
DPT.E          C'DP0800', 1,09,45,23,192,0892,0,0,0850,0879750,40960,00,C'8881'
DPT.E          C'DP0850', 1,09,45,15,192,0909,0,0,1381,0932175,41088,00,C'8813'
DPT.E          C'DP0858', 1,09,54,16,192,0927,0,0,1064,0919296,50400,00,C'8888'
DPT.E          C'DP1230', 1,11,55,15,192,0909,0,0,1635,1348875,50400,00,C'8814'
DPT.E          C'DC0080', 1,02,20,05,192,0000,0,0,0823,0082300,20160,00,C'VAR.'
DPT.E          C'MH080 ', 1,02,20,05,192,0000,0,0,0823,0082300,20160,00,C'VAR.'
DPT.E          C'DC0160', 1,04,20,10,192,0000,0,0,0823,0164600,20160,00,C'8127'
DPT.E          C'MH160 ', 1,04,20,10,192,0000,0,0,0823,0164600,20160,00,C'8127'
DPT.E          C'DC0300', 1,04,20,19,192,0000,0,0,0823,0312740,20160,00,C'9346'
DPT.E          C'MH300 ', 1,04,20,19,192,0000,0,0,0823,0312740,20160,00,C'9346'
DPT.E          C'DC0340', 1,04,20,24,192,0000,0,0,0711,0341280,20160,00,C'8858'
DPT.E          C'MH340 ', 1,04,20,24,192,0000,0,0,0711,0341280,20160,00,C'8858'
DPT.E          C'DC0600', 1,10,20,40,192,0000,0,0,0843,0674400,20160,00,C'8155'
DPT.E          C'MH600 ', 1,10,20,40,192,0000,0,0,0843,0674400,20160,00,C'8155'
DPT.E          C'SD0150', 1,04,24,09,192,0000,0,0,0967,0208872,00000,00,C'8820'
DPT.E          C'SD0300', 1,04,32,09,192,0000,0,0,1362,0392256,00000,00,C'8828'
DPT.E          C'SD0700', 1,07,35,15,192,0000,0,0,1546,0811650,00000,00,C'8833'
```

3 System Task Descriptions

3.1 Swap Scheduler Task (J.SWAPR)

The swap scheduler (J.SWAPR) is a nonswappable memory management task. J.SWAPR provides memory allocation for tasks that require memory, and services memory requests when memory is not available. J.SWAPR is not embedded in the operating system, and does not occupy logical address space. J.SWAPR is not mapped into the address space of every task in the system. J.SWAPR is a memory resident, privileged task. See Figure 3-1.

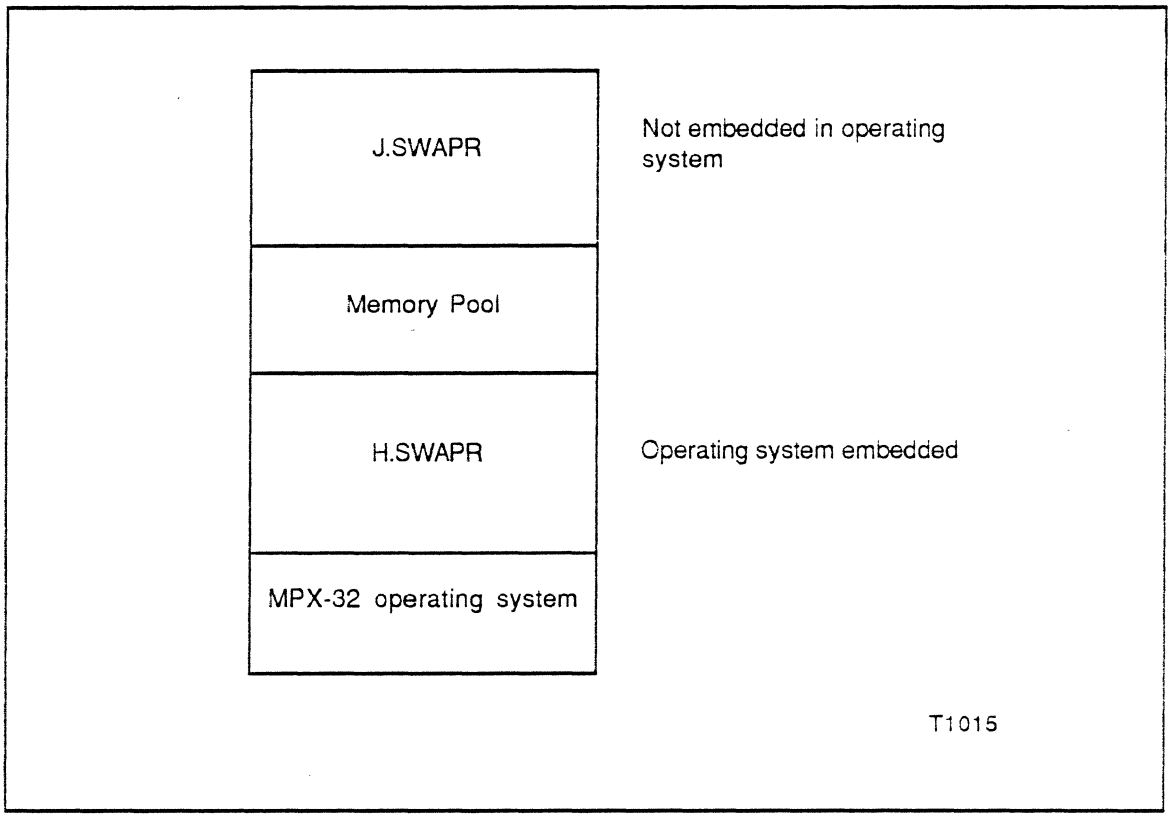


Figure 3-1
System Swap Scheduler

J.SWAPR is activated by the system initializer task (SYSINIT). Once activated, J.SWAPR remains in a suspended state until it is needed. When resumed, J.SWAPR executes at the priority of the highest priority task in the memory request queue (MRQ). When all swap activity is completed, J.SWAPR returns to the suspended state. J.SWAPR remains suspended until it is resumed by H.EXEC,9 in response to a memory scheduler event.

To free memory, J.SWAPR selects a task for outswap and proceeds to write all or parts of the task to a secondary storage area called the swap file. When memory becomes available, outswapped tasks that are requesting memory are inswapped in priority order, and are allowed to execute normally.

J.SWAPR uses two sets of MIDL and MEML arrays for task map manipulation. J.SWAPR's TSA MIDL/MEML arrays are used to map the candidate task's TSA on top of J.SWAPR. See Figure 3-2. If the task is to be outswapped, the MIDL entries for maps to be rolled out are copied to J.SWAPR's working MIDL. See Figure 3-3. J.SWAPR then calls internal subroutine S.SWAP1A to remap to the working MIDL. This gives J.SWAPR addressability to the outswap candidate's memory in order to write the maps to the swap file.

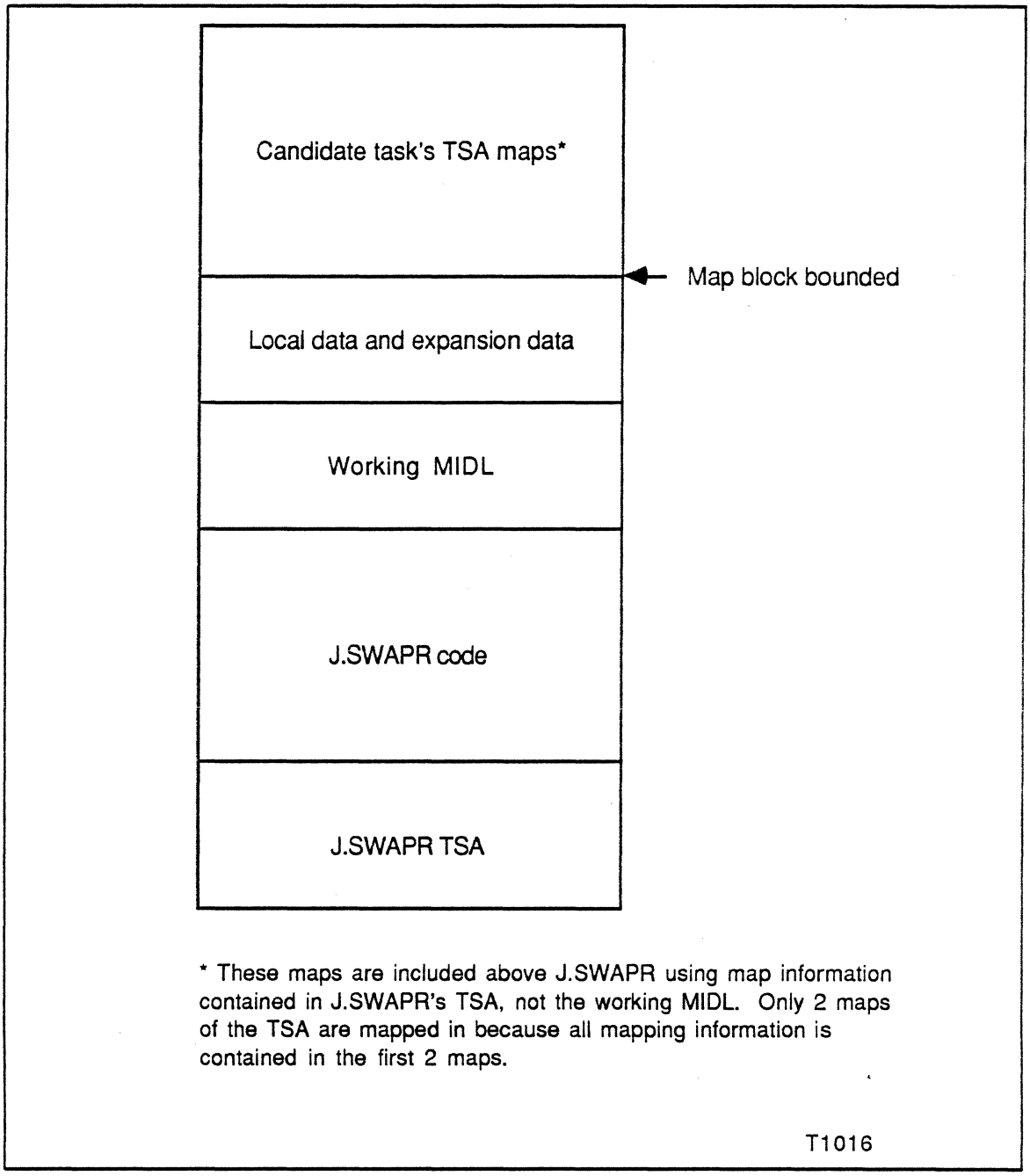


Figure 3-2
Mapping of Candidate Task's TSA (an overview)

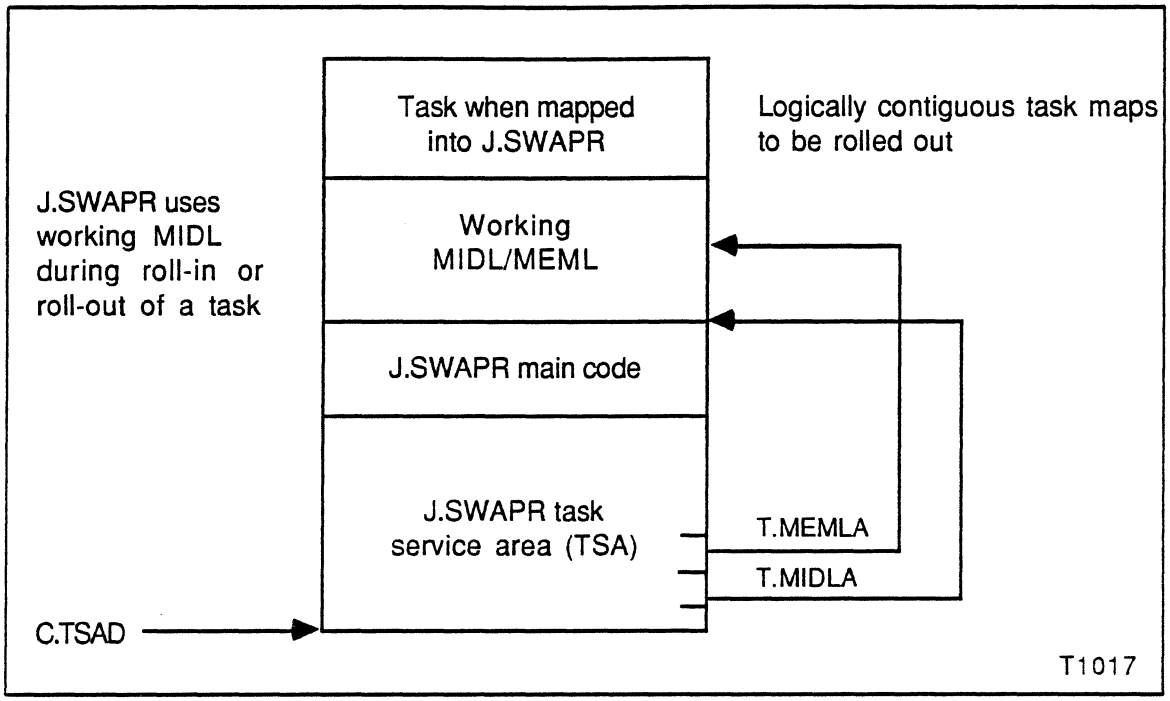


Figure 3-3
Mapping of a Candidate Task During Roll-out

J.SWAPR has an operating system resident counterpart called H.SWAPR. H.SWAPR performs MIDL entry copying, and holds data structures that are required by other areas of the operating system. This ensures that the system trace and system debugger function normally. Base R in the system debugger references H.SWAPR, not J.SWAPR.

J.SWAPR also provides for selective partial outswapping. Partial outswapping allows a small portion of a large task to be outswapped when only a small portion of memory is required. Partial outswapping saves the time it would take to outswap and eventually inswap the un-needed memory.

The SYSGEN SWAPLIM directive allows the user to specify a minimum partial outswap quantum. When heavy swapping is anticipated, a value between 7 and 15 is recommended. A SWAPLIM within this range reduces the number of swaps necessary to obtain the required memory. A SWAPLIM value of 16 or greater can reduce the effectiveness of partial swapping.

When memory is requested by a task or tasks, and there is insufficient memory to satisfy the request, J.SWAPR is resumed by using H.EXEC,37. J.SWAPR allocates and deallocates memory for the memory requests listed in sections 3.1.2 and 3.1.3.

The following illustrates J.SWAPR's task layout.

Remap area: used to map task's TSA or memory that is being swapped
J.SWAPR swap file entries pool (Q-entries): expands as the system runs to meet the system's requirements. See Note 1.
Dedicated system buffer space
J.SWAPR's shadow tables: used to allocate shadow memory. See Note 2.
J.SWAPR's outswap shadow tables: one for each DQE. See Note 2.
J.SWAPR's working MIDL/MEML arrays (WORKMIDL and WORKMEML): used for managing memory that is being swapped. See Note 3.
Swap context area: one entry for each DQE
J.SWAPR's DSECT
J.SWAPR's TSA (SWPMIDL and SWPMEML)
Memory pool
H.SWAPR
MPX-32

Notes:

1. Q-entries are also built from the initialization code after it is executed.
2. Shadow tables are present only if shadow memory is configured.
3. WORKMIDL and WORKMEML point to the mapping information for J.SWAPR while MIDLA and MEMLA point to the mapping information for the logical address space called the remap area.

3.1.1 J.SWAPR Processing

J.SWAPR can perform the following types of processing:

- dispatch processing
- inswap processing
- shared memory request processing
- no memory available processing
- outswap processing
- I/O error handling processing
- initialization processing

When sufficient memory is unavailable, inswap and outswap are both serial processes and are completed before the MRQ is re-examined. When sufficient memory is available to load an outswapped task into memory from the swap volume, the inswap task is initiated. The difference between memory requests and inswap requests is that there is no associated disk file to read. Tasks linked to the MRQ can be queued for both expansion and inswap.

3.1.1.1 Dispatch Processing

Dispatch processing begins when swap activity is resumed. Dispatch processing performs the following functions:

- calls the inswap routine if the task is outswapped
- allocates memory as required by calling H.MEMM,1
- outswaps a task if no memory is available
- links the task's DQE to the ready queue
- suspends itself if the MRQ (memory request queue) is empty
- redispatches if entries still exist in the MRQ
- calls H.EXEC,7 to report memory requests event complete
- changes J.SWAPR's priority to that of the requesting task

The dispatcher examines the memory request queue (MRQ). If entries are not present, J.SWAPR suspends until another memory request event occurs. If entries are present, dispatch processes memory requests, and counts unprocessed memory (MRQ.CNT) and memory release events (C.RRUN). J.SWAPR attempts to allocate memory to tasks waiting for memory.

3.1.1.2 Inswap Processing

When sufficient memory is available, J.SWAPR allocates the memory to the highest priority task on the MRQ. If the request is for inswap, J.SWAPR reads the swapped image into the newly allocated memory. This process is completed in two passes.

On the first pass, the outswapped TSA and DSECT are read off the swap space, and the swap space entries are released.

On the second pass, the outswapped shared regions are interrogated to find out if they are outswapped. If they are outswapped, memory must be allocated before they can be loaded into the task's TSA tables. If sufficient memory cannot be allocated, the task is put on the MRQ requesting dynamic expansion to include a shared region. After the shared memory is inswapped, the used swap space entries are released if the shared memory is not a CSECT.

3.1.1.3 Shared Memory Request (SISHR) Processing

Shared memory request (SISHR) process begins when there is currently outswapped shared memory that needs to be inswapped, or when a specified memory partition needs to be included into the address space of a task.

3.1.1.4 No Memory Available (NOMEM) Processing

No memory available (NOMEM) processing begins when there is insufficient memory to fulfill the memory requestor's requirements. NOMEM determines if enough memory can be freed by outswapping. If so, NOMEM calls the outswap process to free the required memory, then calls the inswap or SISHR process to allow the memory requestor to get the free memory.

3.1.1.5 Outswap Processing

Outswap processing occurs when the outswapping of a particular task or tasks will free sufficient memory for the memory requestor.

In order to complete the outswap process, the TSA of the outswap candidate is mapped into J.SWAPR. The protocol of outswap requires two (2) complete passes of the user's TSA tables (MIDL and MEML).

On the first pass, only nonshared maps are logically built into J.SWAPR's work area in their original TSA format. A second swap space is allocated and the outswap I/O process is performed. A used swap space entry is then added to DQE.SRID.

On the second pass, all swappable regions (if any) are built into a work area inside J.SWAPR in logically contiguous format. A swap space is allocated and the outswap I/O process is performed. A used swap entry is then added to SMT.SRID. If there are no shared regions, I/O is not performed in this pass.

At the end of each pass, memory is returned to the free list. When both passes of the outswap are complete, the MRQ is re-examined to find the highest priority candidate to receive the memory from the free list.

Outswap candidates are first chosen from the wait states, then from the run state. The default order is described in the next section.

Selection of Outswap Candidates — The swapper initially attempts to process the memory requirement for the highest priority task on the MRQ. If insufficient memory is available, the swapper examines the state queues on a priority basis searching for the memory class and number of map blocks required by the requesting task. The first task found with resources satisfying any of the requirements of the requester is partially or totally outswapped. When the outswap process is complete, the swapper re-examines the MRQ and continues to process memory requests.

Outswap candidates are first chosen from wait states and then from run states. The order is shown below:

```
NWS       DATAW   15        NUMBER OF WAIT STATES
WAITSTAT  DATAW   C.HOLD    WAIT STATE Q POINTERS IN OUTSWAP ORDER
          DATAW   C.SUSP
          DATAW   C.RUNW
          DATAW   C.SWDV
          DATAW   C.SWDC
          DATAW   C.SWSR
          DATAW   C.SWSM
          DATAW   C.SWLO
          DATAW   C.SWFI
          DATAW   C.MRQ
          DATAW   C.ANYW
          DATAW   C.SWGQ
          DATAW   C.SWTI
          DATAW   C.SWIO
          DATAW   C.SWMP

NRS       DATAW   12        NUMBER OF RUN STATE Q ENTRIES
RUNSTAT   DATAW   C.SQ64    RUN STATE Q POINTERS IN OUTSWAP ORDER
          DATAW   C.SQ63
          DATAW   C.SQ62
          DATAW   C.SQ61
          DATAW   C.SQ60
          DATAW   C.SQ59
          DATAW   C.SQ58
          DATAW   C.SQ57
          DATAW   C.SQ56
          DATAW   C.SQ55
          DATAW   C.RIPU
          DATAW   C.SQRT
```

Once an outswap candidate is selected by J.SWAPR, DQE.SOPO (swap-on priority only) and DQE.SWIF are examined. Both locations are one byte variables containing several swap limitations.

Swap inhibit conditions are checked first. If any DQE.SWIF bit other than DQE.TLVS (task leaving system) is set, the task is unswappable. If DQE.SOPO bits DQE.BMAP or DQE.MDTA are set, the task is unswappable. J.SWAPR then searches for a new outswap candidate.

DQE.SOPO is examined only if DQE.SWIF equals zero or if DQE.TLVS is set. If the DQE.USPO bit in DQE.SOPO is set, the priority of the memory requestor is compared to the outswap task priority. If the user settable swap-on priority only feature is enabled, and if the DQE.USPO bit in DQE.SOPO is set, the priority of the memory requestor is compared to the outswap task priority. J.SWAPR searches for another outswap candidate if the memory requestor does not have a higher priority if the outswap task is ready to run, if the memory requestor does not have a higher or equal priority or if the outswap task is in a wait state. If DQE.SOPO equals zero, the outswap candidate is outswapped.

3.1.1.6 I/O Error Handling Processing

The I/O error handling portion of J.SWAPRs code helps the system recover from I/O errors. If a write error is encountered during the outswap of a task, J.SWAPR assumes that the error is a bad block on the disk. J.SWAPR discards the current Q-entry that is defining the bad swap space, allocates another Q-entry for new swap space, and retries the write request. If succeeding writes fail and the end of the swap file is reached, J.SWAPR releases the bad Q-entries and suspends for ten seconds. After the suspension, J.SWAPR retries the write. J.SWAPR repeats the process until the write is successfully completed.

On a read error during the inswap of an outswapped task, J.SWAPR places the inswap candidate in the hold state, outputs a message to the console, and continues the swapping activities. In order to resume the task in the hold state, the operator must issue the OPCOM CONTINUE directive for that task.

3.1.1.7 Initialization Processing

The system initialization program (SYSINIT) creates a swap file and activates J.SWAPR. J.SWAPR then activiates its secondary initialization routine S.SWAP99. S.SWAP99 then does the following:

- initializes the swap activity table
- allocates memory for Q-entries dynamically
- allocates the working MIDL/MEML dynamically
- allocates the shadow memory tables dynamically, if applicable
- allocates the swap files space dynamically
- assigns and opens the swap file
- sets the MIDL/MEML pointers for use by the remap routines
- computes the location where task's TSA will reside in J.SWAPR
- sets up swap inhibit and swap-on priority only (SOPO) masks
- builds Q-entries from J.SWAPR's secondary initialization code space

After the initialization code is processed, its space is used as a buffer area for I/O operations or as an extension of linked list structures. This ensures a compact J.SWAPR.

3.1.2 J.SWAPR Internal Subroutines

The following are internal subroutines of J.SWAPR:

Subroutine	Description
S.SWAP1	remaps task into J.SWAPR not using the working array
S.SWAP1A	remaps J.SWAPR using the working MIDL/MEML array
S.SWAP2	updates the SMT use count when a task using shared image is rolled back into memory. S.SWAP2 also updates tasks MIDL/MEML to correctly reference the shared image.
S.SWAP3	determines the outswap candidate. S.SWAP3 also checks for sufficient available outswap maps. For example, do the requested maps equal the maps available if the candidate is outswapped.
S.SWAP4	reads in a task from the swap file with the initial 128KB read. It is followed by a single disk sector read until the task is restored to memory. Memory roll-in is a random access operation.
S.SWAP5	writes a task to the swap file with the initial 128KB write. It is followed by single disk sector writes until the task is rolled out of memory. Memory roll-out is a random access operation.
S.SWAP6	allocates the swap space on the swap file; if none is available, it aborts the task.
S.SWAP7	releases swap spaces. Where applicable, it will coalesce two small free swap spaces into one large free swap space. This is a garbage collection routine.
S.SWAP8	allocates a free space entry
S.SWAP9	releases a free space entry
S.SWAP10	builds a linked list of free entries
S.SWAP11	links an entry to a given queue by priority
S.SWAP12	unlinks an entry from a given queue
S.SWAP13	increases the priority of a target task. The target task's DQE address and the amount of increase are supplied by the caller.
S.SWAP14	stores information needed to inswap a task requesting shadow memory
S.SWAP15	is a swap file extension subroutine
S.SWAP16	adds an entry into the shared memory include list (SMIL)
S.SWAP17	removes an entry from the shared memory include list (SMIL)

Subroutine	Description
S.SWAP18	allocates a map block for more queue entry space
S.SWAP19	determines the total number of map blocks needed to inswap an outswapped task
S.SWAP20	counts the number of swappable map blocks belonging to an outswap candidate
S.SWAP21	determines if an inswap of a task requiring multiple memory types is possible
S.SWAP22	determines how many map blocks of a shared memory partition to outswap
S.SWAP23	determines if a task is swap-on priority only (SOPO)
S.SWAP24	is the thrash control subroutine
S.SWAP99	performs J.SWAPR self initialization. S.SWAP99 is executed once when SYSINIT activates J.SWAPR. The space occupied is then reclaimed for swap space entries by S.SWAP10 if the initial allocation of swap entries is used up.

3.1.3 J.SWAPR Memory Request Functions

Memory request functions performed by J.SWAPR are:

- memory expansion request
- memory deallocation request
- inswap request
- change in task status request
- shared memory include request
- exit conditions

See the following sections for descriptions of the memory request functions.

There are memory request function codes for J.SWAPR. These function codes determine the action that J.SWAPR performs for a requesting task. See Table 3-1.

Table 3-1
Memory Request Function Codes for J.SWAPR

Function Code	Description
0	inswap request
1	pre-activation request
2	task activation request
3	memory expansion request
4	IOCS buffer request
5	shared memory include request
6	system buffer space request
7	release swap space request
8	exclude shared memory request
9	SW.MON activation request

3.1.3.1 Memory Expansion Request

A memory expansion request occurs when there is insufficient memory to satisfy a task's dynamic memory request. The task is linked to the memory request queue (MRQ) with the number of maps needed, the type of memory needed, the address where the memory is loaded, and the expansion request code. J.SWAPR is then resumed to process the request.

3.1.3.2 Memory Deallocation Request

When a task deallocates all or some of its memory and there are tasks linked to the MRQ, J.SWAPR is resumed so it can reallocate the memory to a task in the MRQ.

3.1.3.3 Inswap Request (Memory Roll-in)

When a currently outswapped task is ready for execution, J.SWAPR is resumed. The task is located in the MRQ, and the inswap request is processed.

3.1.3.4 Change in Task Status Request

When a task which has previously been ineligible for swapping due to unbuffered I/O in progress, release of a lock in memory flag, expiration of a stage 1 time quantum, etc., J.SWAPR is resumed.

3.1.3.5 Shared Memory Include Request

When there is sufficient memory to satisfy the inclusion of a task's dynamic shared memory partitions, J.SWAPR, having been resumed prior to receiving the request, links the tasks to the MRQ to process the request.

3.1.3.6 Exit Conditions

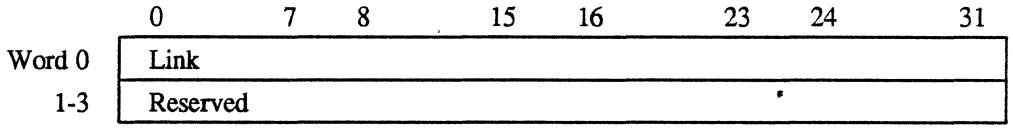
When J.SWAPR finds the MRQ empty, or when there are no outstanding requests to process, J.SWAPR suspends itself by unlinking from the ready-to-run queue and relinking to the wait-for-memory-event queue. This is accomplished by using H.EXEC,8.

3.1.4 Managing Swap Space Entries

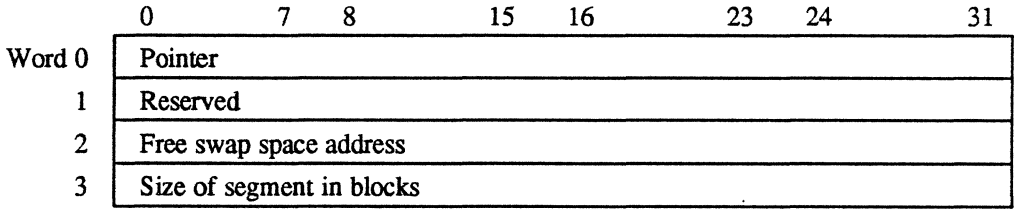
J.SWAPR manages swap space entries by using four one-way linked entry lists:

List	Description
H.ENTRY	points to free queue entry lists
Q.SWP	lists all free queue entries
DQE.SRID	contains queue entries for DSECT memory
SMT.SRID	contains queue entries for outswapped shared memory

H.ENTRY is an internal linked head cell list that monitors the free swap file entries. H.ENTRY is located in H.SWAPR. The first word is the link, and the last three words are reserved. The addresses of the swap file free entries are maintained by Q.SWP.



Q.SWP is an internal linked list head cell that maintains the addresses of all swap file free entries. Q.SWP has a four word entry for each linked list entry. The swap space entry, Q.SWP, is as follows:



DQE.SRID is a two-word linked list headcell. DQE.SRID functions as a queue and is located in the task's DQE. Each swap space entry is four words long. The first word points to the first used swap space entry. The second word points to the last entry. All entries except the headcell are located internal to J.SWAPR. When a task's nonshared memory is outswapped, the swap space is allocated and DQE.SRID is updated. When the memory is inswapped, the swap space is freed, and DQE.SRID is emptied. The swap space entry (headcell = DQE.SRID) is as follows:

	0 7	8 15	16 23	24 31
Word 0	Pointer			
1	Number of outswapped maps		Number of outswapped TSA maps	
2	Used swap space address			
3	Size of segment in blocks			

SMT.SRID is a linked list headcell located in the SMT. When memory is a CSECT, the swap space is not released, and the SMT.SRID remains the same as before the inswap. SMT.SRID empties when the task exists and the user count is zero. A SMT.SRID entry has the same format as a DQE.SRID except the right half of the second word is always zero. The swap space entry (headcell = DMT.SRID) is as follows:

	0 7	8 15	16 23	24 31
Word 0	Pointer			
1	Number of outswapped maps		Zero	
2	Used swap space address			
3	Size of segment in blocks			

3.1.5 Swap Context Area

The swap context area (SCA) is a table of fixed size entries. There is one entry for each DQE in the system. The SCA entries are indexed by the DQE index number which is the first byte of the task activation number (DQE.TAN).

	0 7	8 15	16 23	24 31
Word 0	SCA.TAN. See Note 1.			
1	SCA.SMIL. See Note 2.			
2	SCA.MIDL. See Note 3.			
3	SCA.MAPN. See Note 4.		SCA.FLGS. See Note 5.	

Notes:

1. SCA.TAN contains the task activation number most recently associated with the entry.
2. SCA.SMIL contains the address of the first shared memory include (SMI) Q-entry of the shared memory include list (SMIL).
3. SCA.MIDL can contain the beginning address with J.SWAPR's address space where the last partial outswap MIDL scan started.
4. SCA.MAPN is used with SCA.MIDL. SCA.MAPN is the negative number of MIDs left to scan when outswapping a partially swapped task.
5. SCA.FLGS indicates the swap state of a task. The bit settings are as follows:

Bits	Meaning if Set
0	DSECT is partially outswapped
1	DSECT is completely outswapped
2-15	reserved

3.1.6 Swap Activity Table

The swap activity table is maintained and updated by the J.SWAPR subroutine S.SWAP24. This subroutine determines how active J.SWAPR is and sets the global swap-on priority only (SOPO) flag if the actual swap activity exceeds the desired maximum swap activity. S.SWAP24 is called at the end of an inswap, at the end of an outswap, and just prior to searching for an outswap candidate.

3.1.7 Shadow Memory Outswap Tables

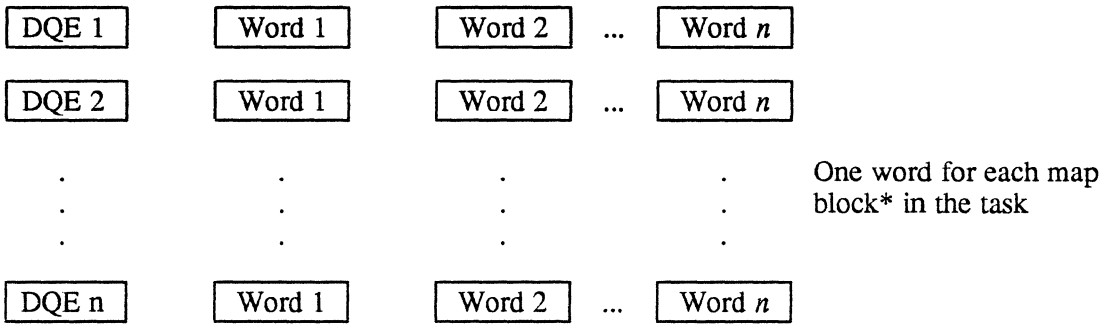
J.SWAPR initializes outswap shadow tables within its logical address space. If needed, J.SWAPR dynamically allocates more memory for these tables. The number of words of memory needed for the outswap tables is equal to the number of dispatch queue entries in the system times the sum of the greater number of physical map blocks shadowed by either processor 1 or processor 2 plus one. For example:

1. 48 dispatch queue entries
2. one 128KB shadow memory board on processor 1 (16 map blocks)
3. two 128 KB shadow memory board on processor 2 (32 map blocks)

Outswap shadow tables needed equals 48 X (32 + 1) words.

J.SWAPR uses the outswap shadow memory tables to remember which map blocks an outswapped task may need to be shadowed when inswapped.

Outswap Shadow Table



* The bits in each word have the following meaning:

Bit	Label	Meaning if Set
0	SH.REQST	shadow memory requested
1	SH.REQRD	shadow memory required
2	SH.IPU	IPU shadow memory
3	SH.BTH	IPU/CPU common shadow memory
4-31	N/A	contains the MIDL of the map image descriptor. This points to the map block number to be shadowed.

3.2 Terminal Services Manager Task (J.TSM)

3.2.1 Functional Description

The terminal services manager (J.TSM) is a nonresident privileged system task which provides job control processing in both the batch and interactive environments. Its functions include validating logon requests, processing command files, initiating batch jobs, and activating tasks.

3.2.2 Operational Design

J.TSM is designed to operate in five logical interrupt levels. An activation request at a given level may interrupt processing of any lower level. The context of an interrupted process is saved in the TSA stack to enable resumption after completion of the interrupting level. The following are the five logical interrupt levels in order from lowest to highest:

- Base (initial task activation level)
- Message
- End Action
- Break
- Abort

3.2.2.1 Base Level

J.TSM is activated at the base level by SYSINIT following a system restart. SYSINIT computes the maximum number of files, segment tables, and mounted volume assignments that J.TSM might need based on communication region variables. These parameters are used to construct J.TSM's TSA via the M.PTSK system service.

At J.TSM's initial activation, a one time initialization is performed by J.TSM, at which time the abort, break, and message level entry points are established. The base level then enters a scan, all terminal and batch context searching for any outstanding base level service requests. After all base level services are performed, the base level requests an indefinite suspension. The base level will be removed from the any wait state whenever J.TSM exits from a break, message or end action level.

Base level service requests include:

- context clean-up if device failure
- logon end action chain initiation
- task status request
- message sending
- batch job initiation

3.2.2.2 Message Level

All messages sent to J.TSM have a message type passed in a byte field of the message. The message type determines the type of process to be performed by J.TSM. If the message level is active, further message requests are queued from the message head cell in J.TSM's DQE. The message types recognized by J.TSM are as follows:

Type	Processing
0	echo messages to screen
1	set SYC record
2	task exit processing
3	validate owner name
4	task hold processing
5	purge account file
6	validate project group
7	update key file
8	update project file
9	output message on system console
10	invoke online help
11	perform TSM procedure call
12	shutdown TSM

3.2.2.3 End Action Level

End action is the transfer of control upon completion of no-wait I/O to the end action address specified in the FCB. No-wait I/O with end action is used by J.TSM to read/write terminals and batch SYC. Therefore, each terminal session is a chain of end action activations as illustrated below:

```
Base level scan
Write----------ENTER YOUR OWNERNAME:
Request indefinite suspension
          .
          .
Enter end action from write
Read----------SYSTEM <ret>
Process owner name entered
Report end action complete
          .
          .
Enter end action from read
Write----------TSM>
Report end action complete
          .
          .
Enter end action from write
Read----------SHOW <ret>
Report end action complete
          .
          .
Enter end action from read
Process show command at end action level
Write----------(show output to TTY)
Report end action complete
          .
          .
         etc.
```

Services performed at the end action level are as follows:

- terminal I/O
- SYC I/O for interactive and batch
- interactive command processing
- interactive and batch command file processing
- TSM and batch accounting
- abort and error message output

Since the bulk of command processing is performed at the end action level, J.TSM performs only one command at a time. If the end action level is active, further end action requests are queued from the task interrupt head cell in J.TSM's DQE.

3.2.2.4 Break Level

Break level processing sets the break received flag in J.TSM and searches the TSM input request queue. If the queue is empty, break level processing exits. If the queue is not empty, break level processing first processes all input requests and then exits. The break level exit removes J.TSM base level from the any wait state to resume the scan for base level service requests. This level is entered when a user presses the wake up character (not the break key) on their terminal.

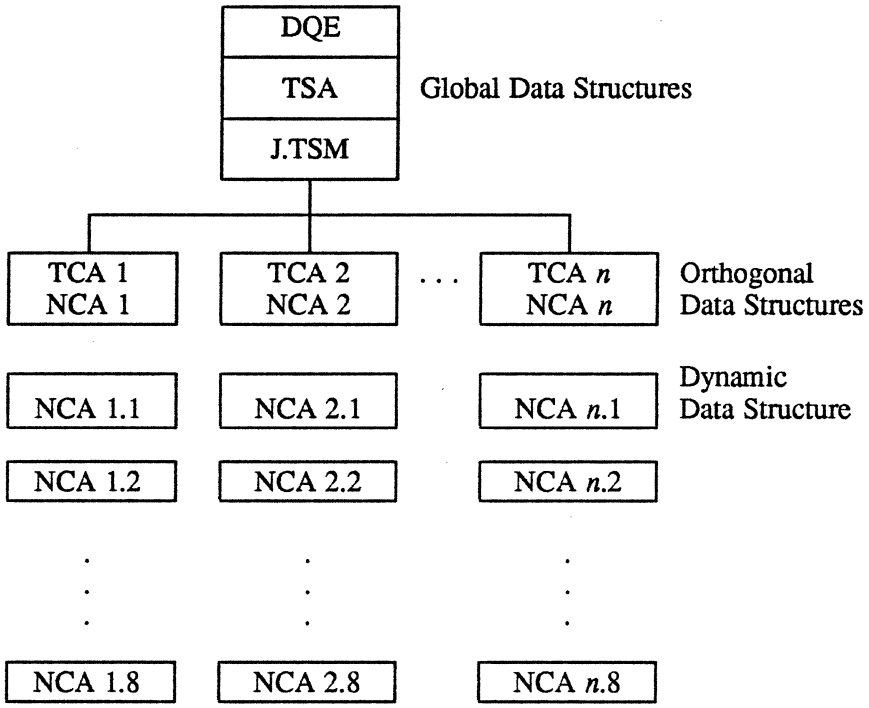
3.2.2.5 Abort Level

J.TSM should never be entered at the abort level. If it is, a fatal system error message is sent to the system console requesting a system reboot. J.TSM then attempts to run by restarting the base level scan.

3.2.3 Data Structures

J.TSM retains a terminal context area (TCA)/nested context area (NCA) for each terminal and batch job configured at SYSGEN time. Each $CALL command dynamically acquires a new NCA. Each end action level activation of J.TSM generally operates on one TCA/NCA. In addition to the TCA/NCA, some TSM commands use global data structures such as flags local to J.TSM, the project group in the TSA, or the owner name in the DQE. The global data structures are redefined by J.TSM upon entry to each end action message, and certain base level operations. Processes which depend on these global data structures, but can be logically interrupted by some higher logical activation of J.TSM, must logically block higher requests by setting the synchronous task interrupt bit in J.TSM's DQE.

A symbolic diagram of J.TSM's data structures is illustrated below:



3.2.3.1 Terminal Context Area (TCA) Table

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
0	000	TCA.FCB
16	040	TCA.EFCB
32	080	TCA.PSB and TCA.TDFB
38	098	TCA.TDFC
40	0A0	TCA.SPAD and TCA.PFCB
50	0C8	TCA.CNP
56	0E0	TCA.MFCB
72	120	TCA.MHDR
77	134	TCA.MBUF
97	184	TCA.TWA1
98	188	TCA.CPU
99	18C	TCA.IPU
100	190	TCA.ACES
102	198	TCA.SGOS
110	1B8	TCA.SLOS
118	1D8	TCA.SLOD
120	1E0	TCA.NAME
122	1E8	TCA.JOB#
123	1EC	TCA.LFLG
124	1F0	TCA.SBOS
132	210	TCA.SBOD
134	218	TCA.NAMB
136	220	TCA.JBB#
137	224	TCA.BFLG
138	228	TCA.HDR1 and TCA.MSGB
142	238	TCA.ONR1
145	244	TCA.DAT1
148	250	TCA.TIM1
150	258	TCA.MBX1
171	2AC	TCA.HDR2
175	2BC	TCA.ONR2
178	2C8	TCA.DAT2
181	2D4	TCA.TIM2
183	2DC	TCA.MBX2

Word No. (Decimal)	Byte (Hex)	0 7 8 15 16 23 24 31
204	330	TCA.JFLG
205	334	TCA.MCEA
206	338	TCA.LABL
207	33C	TCA.STRA
208	340	TCA.AVIN
209	344	TCA.ACCU
210	348	TCA.LCSF
211	34C	TCA.LCSB
212	350	TCA.LOGN
213	354	TCA.TERM | TCA.MPLC
214	358	TCA.LBFA
215	35C	TCA.UDTA
216	360	TCA.TNUM
217	364	TCA.ERR
218	368	TCA.SP01
219	36C	TCA.SP02
220	370	TCA.SP03
221	374	TCA.SP04
222	378	TCA.SP05
223	37C	TCA.SP06
224	380	TCA.SP07
225	384	TCA.SP08
226	388	TCA.SP09
227	38C	TCA.OLBA
228	390	TCA.TIQA
229	394	TCA.WBA
230	398	TCA.WBAC
231	39C	TCA.RLBF
232	3A0	TCA.RLBL
233	3A4	TCA.RLBC
234	3A8	TCA.GMAT
235	3AC	TCA.SCTA
236	3B0	TCA.LBSP
237	3B4	TCA.PFMA

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
238	3B8	TCA.LBCC		TCA.LBCI	
239	3BC	TCA.ORCT		TCA.LBEX	
240	3C0	TCA.OLCI		TCA.EOS	
241	3C4	TCA.LC	TCA.CHSA	TCA.GMET	TCA.RCNT
242	3C8	TCA.RCDC	TCA.RCBF	TCA.NRCB	TCA.CCNT
243	3CC	TCA.LBSC	TCA.SPB3		
244	3D0	TCA.BLDN			
245	3D4	TCA.RCN1	TCA.RCN2	TCA.RNUM	TCA.RUPN
246	3D8	TCA.PCLB			
247	3DC	TCA.SK00			
248	3E0	TCA.SK01			
249	3E4	TCA.SK02			
250	3E8	TCA.SK03			
251	3EC	TCA.SK04			
252	3F0	TCA.NCAA			
253	3F4	TCA.FNCA			
254	3F8	TCA.RIDL			
255	3FC	TCA.TPCB			
256	400	TCA.ROPC			
257	404	TCA.KEYA			
258	408	TCA.PLFC			
259	40C	TCA.PSR0			
260	410	TCA.PSWD			
264	420	TCA.SEC			
264	420	TCA.SFLG	TCA.LCNT	Reserved	
265	424	TCA.STAT			
266	428	TCA.STS2			
267	42C	TCA.STS3			
268	430	TCA.RRSF			
269	434	TCA.RRSN			
270	438	TCA.KTAB			
272	440	TCA.KPRJ			

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
274	448	TCA.KVOL			
278	458	TCA.KDIR			
282	468	TCA.KTNO	TCA.KPNO	TCA.KVNO	TCA.KDNO
283	46C	TCA.KRSV			
284	470	TCA.MODE	TCA.NRRS	TCA.ALLO	TCA.SPCE
285	474	TCA.NBUF	TCA.NFIL	TCA.PRIO	TCA.IOER
286	478	TCA.LMN			
288	480	TCA.SUDO			
290	488	TCA.ONRN			
292	490	TCA.PROJ			
294	498	TCA.USRK			
294	498	TCA.VAT	TCA.FLG2	TCA.EXTD	
295	49C	TCA.PGOW			
296	4A0	TCA.USW			
297	4A4	TCA.RPTR			
298	4A8	TCA.PGO2			
299	4AC	TCA.FSIZ		TCA.RSIZ	
305	4C4	TCA.FRRS			

Byte (Hex)	Symbol	Description
000	TCA.FCB	terminal I/O FCB (16 words)
040	TCA.EFCB	command file I/O FCB (16 words)
080	TCA.PSB	mount request parameter send block (8 words)
080	TCA.TDFB	TERMDEF block for $INIT PRO (6 words)
098	TCA.TDFC	TERMDEF return address (1 word)
0A0	TCA.SPAD	scratch pad area (10 words)
0A0	TCA.PFCB	MPX.PRO FCB (16 words)
0C8	TCA.CNP	CNP for mount requests (6 words)
0E0	TCA.MFCB	message ($SIGNAL) FCB (16 words)
120	TCA.MHDR	command file buffer header (5 words)
134	TCA.MBUF	command file buffer (20 words)
184	TCA.TWA1	temporary word area (1 word)
188	TCA.CPU	accumulated CPU time/session (1 word)
18C	TCA.IPU	accumulated IPU time/session (1 word)

Byte (Hex)	Symbol	Description
190	TCA.ACES	privilege access bits defined in M.KEY (1 doubleword)
198	TCA.SGOS	SGO long RID (8 words)
1B8	TCA.SLOS	SLO long RID (8 words)
1D8	TCA.SLOD	SLO device (2 words)
1E0	TCA.NAME	SLO project/job name (2 words)
1E8	TCA.JOB#	SLO binary job number (1 word)
1EC	TCA.LFLG	SLO listed output flags (1 word) The flag bits have the following meanings:

Bit	Meaning When Set
0	do not delete file on deassignment
1	copy/reprint request
2	output is unformatted
3-23	reserved
24-31	byte values are as follows:

Value	Description
0	print
1	punch
2	plot
3-255	reserved

1F0	TCA.SBOS	SBO long RID (8 words)
210	TCA.SBOD	SBO device (2 words)
218	TCA.NAMB	SBO project/job name (2 words)
220	TCA.JBB#	SBO binary job number (1 word)
224	TCA.BFLG	SBO flags (1 word)

Bit	Meaning When Set
0	do not delete file on deassignment
1	copy/reprint request
2	output is unformatted
3-23	reserved
24-31	byte values are as follows:

Value	Description
0	print
1	punch
2	plot
3-255	reserved

Byte (Hex)	Symbol	Description
228	TCA.MSGB	message buffer (66 words. Overlays area from TCA.HDR1 to TCA.MBX2)
228	TCA.HDR1	header for first message (2 doublewords)
238	TCA.ONR1	ownername of first message (3 words)
244	TCA.DAT1	date of first message (3 words)
250	TCA.TIM1	time of first message (2 words)
258	TCA.MBX1	first message mailbox (21 words)
2AC	TCA.HDR2	header for second message (2 doublewords)
2BC	TCA.ONR2	ownername of second message (3 words)
2C8	TCA.DAT2	date of second message (3 words)
2D4	TCA.TIM2	time of second message (2 words)
2DC	TCA.MBX2	second message mailbox (21 words)
330	TCA.JFLG	conditional flags set by $SETF command (1 word)
334	TCA.MCEA	command file end action address (1 word)
338	TCA.LABL	command file target label (1 word)
33C	TCA.STRA	string address for $SET command (1 word)
340	TCA.AVIN	NCA.ALST/NCA.VPAR index save area (1 word)
344	TCA.ACCU	accumulator for $SETI command (1 word)
348	TCA.LCSF	string forward for looping contexts (1 word)
34C	TCA.LCSB	string back for looping contexts (1 word)
350	TCA.LOGN	binary logon time (1 word)
354	TCA.TERM	original line and page sizes (1 halfword)
356	TCA.MPLC	MPX.PRO file line counter
358	TCA.LBFA	line buffer address in memory pool (1 word)
35C	TCA.UDTA	address of terminal UDT (1 word)
360	TCA.TNUM	task number of currently active task (1 word)
364	TCA.ERR	last abort code (1 word)
368	TCA.SP01	gate/ungate R3 save area (1 word)
36C	TCA.SP02	gate/ungate R1 save area (1 word)
370	TCA.SP03	gate/ungate call from subroutine R0 save area (1 word)

Byte (Hex)	Symbol	Description
374	TCA.SP04	GETMEM denial address save area (1 word)
378	TCA.SP05	CL.MACRO R0 save area (1 word)
37C	TCA.SP06	temporary line buffer size and address (1 word)
380	TCA.SP07	CL.TLB R0 save area (1 word)
384	TCA.SP08	CL.TLB R3 save area (1 word)
388	TCA.SP09	SEOJ error type and CNP address (1 word)
38C	TCA.OLBA	original line buffer address (1 word)
390	TCA.TIQA	TIQ address (1 word)
394	TCA.WBA	write buffer address (1 word)
398	TCA.WBAC	write transfer count (1 word)
39C	TCA.RLBF	recall buffer first pointer (1 word)
3A0	TCA.RLBL	recall buffer last pointer (1 word)
3A4	TCA.RLBC	recall current pointer (1 word)
3A8	TCA.GMAT	GETMEM address table (1 word)
3AC	TCA.SCTA	special character function table (1 word)
3B0	TCA.LBSP	special line buffer printer (1 word)
3B4	TCA.PFMA	possible function match pointer (1 word)
3B8	TCA.LBCC	line buffer character count (1 halfword)
3BA	TCA.LBCI	line buffer cursor index (1 halfword)
3BC	TCA.ORCT	original requested read count (1 halfword)
3BE	TCA.LBEX	extra character blank length (1 halfword)
3C0	TCA.OLCI	old cursor index (1 halfword)
3C2	TCA.EOS	end of screen count (1 halfword)
3C4	TCA.LC	left cursor character (1 byte)
3C5	TCA.CHSA	character save area (1 byte)
3C6	TCA.GMET	GETMEM number of entries (1 byte)
3C7	TCA.RCNT	recall nesting count (1 byte)
3C8	TCA.RCDC	recall display count (1 byte)
3C9	TCA.RCBF	recall buffers full (1 byte)
3CA	TCA.NRCB	number of recall buffers (1 byte)
3CB	TCA.CCNT	current nesting level (1 byte)

Byte (Hex)	Symbol	Description
3CC	TCA.LBSC	special line buffer character count (1 byte)
3CD	TCA.SPB3	reserved (3 bytes)
3D0	TCA.BLDN	area to build $RECALL number (1 word)
3D4	TCA.RCN1	$RECALL range — first number (1 byte)
3D5	TCA.RCN2	$RECALL range — second number (1 byte)
3D6	TCA.RNUM	number of commands to recall (1 byte)
3D7	TCA.RUPN	number of up arrows to do for $RECALL (1 byte)
3D8	TCA.PCLB	procedure call line buffer save area (1 word)
3DC	TCA.SK00	save word 0 for seek routine (1 word)
3E0	TCA.SK01	save word 1 for seek routine (1 word)
3E4	TCA.SK02	save word 2 for seek routine (1 word)
3E8	TCA.SK03	save word 3 for seek routine (1 word)
3EC	TCA.SK04	save word 4 for seek routine (1 word)
3F0	TCA.NCAA	current nested context area (1 word)
3F4	TCA.FNCA	first nested context area (1 word)
3F8	TCA.RIDL	nested command file RID list (1 word)
3FC	TCA.TPCB	procedure call buffer address (1 word)
400	TCA.ROPC	$ERR - residual output count (1 word)
404	TCA.KEYA	key file record address (1 word)
408	TCA.PLFC	MPX.PRO file LFC (1 word)
40C	TCA.PSR0	$INIT PRO R0 save area (1 word)
410	TCA.PSWD	temporary area for encrypted password (2 doublewords)
420	TCA.SEC	security word — flags and counter (1 word)
	TCA.SFLG	flag portion of TCA.SEC (1 byte). The flag bits have the following meanings:

Bit	Meaning When Set
0	valid ownername (TCA.VNAM)
1	valid password/key (TCA.PASS)
2-6	reserved
7	overflow bit for counter
8-15	failed logon attempts (TCA.LCNT)

Byte (Hex)	Symbol	Description
424	TCA.STAT	status flags (1 word). The flag bits have the following meanings:

Bit	Meaning When Set
0	terminal in task mode (TCA.TASK)
1	terminal in command mode (TCA.COMM)
2	first mailbox has message
3	second mailbox has message
4	next mailbox toggle
5	terminal has valid ownername (TCA.OWNR)
6	context borrowed for message transfer
7	M.CNTRL command file in process
8	option noerror
9	inhibit signals
10	batch job
11	$JOB card required
12	spool file requires deassignment
13	sequential job
14	SLO assigned to UT ($SYSOUT=UT)
15	job in effect
16	remove job requested
17	option nocommand
18	previous task aborted
19	self identifier for $SHOW command
20	waiting for memory pool
21	enable asynch messages
22	option noabort
23	option quiet
24	option unquiet
25	exiting deferred by busy FCB
26	dead modem wind down in progress
27	terminal force online (TCA.FONL)
28	submit job
29	SLO is full (IO98 on SLO)
30	gate entered in synch mode
31	waiting for SYC read complete

Byte (Hex)	Symbol	Description
428	TCA.STS2	status flags (1 word). The flag bits have the following meanings:

Bit	Meaning When Set
0	reading first line of a command file
1	processing a command file
2	scanning for label
3	unrecoverable I/O error
4	next record already read
5	reading from message receiver
6	dollar sign ($) is required
7	command file input mode
8	command file end action requested
9	current line must be a $JOB
10	"no memory pool" message sent
11	command is one shot
12	limit command to "HOLD" set
13	send "memory pool available" message
14	unable to assign terminal
15	special CNP for convert PN to PNB
16	multiple logon message flag
17	disable terminal flag
18	found equals sign in $CHAN DIR= (TCA.CDEQ)
19	processing $RECALL command (TCA.RCLL)
20	special check for "#" (TCA.SCPS)
21	parsing first $RECALL number (TCA.RN01)
22	parsing second $RECALL number (TCA.RN02)
23	found a ":" in $RECALL command
24-31	reserved

Byte (Hex)	Symbol	Description
42C	TCA.STS3	status flags (1 word).

Bit	Meaning When Set
0	file name exceeds 8 characters (TCA.FNGE)
1	modem in logon sequence (TCA.MLOG)
2	modem in two second delay loop (TCA.MDWT)
3	no delay for modem in logon (TCA.NODL)
4	flag $EXECUTE command for parse (TCA.EXEC)
5	single channel mode (TCA.SCHM)
6	GOBACK in effect (TCA.BACK)
7	addition/subtraction toggle (TCA.OPER)
8	suspend context (TCA.SUSP)
9	context processing in progress (TCA.PROG)
10	edit mode test flag (TCA.EDMD)
11	Command Line Recall/Edit in use (TCA.CLRE)
12	Command Line Recall/Edit deferred read (TCA.DERD)
13	recall mode enabled (TCA.RCMD)
14	SYC closed (TCA.CSYC)
15	MPX.PRO file syntax error (TCA.PSER)
16	MPX.PRO TERMDEF error (TCA.PTER)
17	MPX.PRO HIST.BUF = 0 (TCA.PHB0)
18	SLO closed (TCA.CSLO)
19	context in procedure call mode (TCA.TPCM)
20	procedure call $RRS command (TCA.SRRS)
21	procedure call error flag (TCA.TPCE)
22	procedure call text output (TCA.TEXT)
23	procedure call output data not formatted (TCA.ODNF)
24	MPX.PRO reserved control string error (TCA.RCSE)
25	TCA.SPAD contains TCA.LBFA (TCA.LBFS)
26	terminal in wait mode (TCA.WAIT)
27	reserved
28	auto logical mount at logon (TCA.AMNT)
29	lower case enabled (TCA.LCEN)
30	MPX.PRO duplicate key error (TCA.DUPK)
31	remote task activation load module found (TCA.RTAF)

Byte (Hex)	Symbol	Description
430	TCA.RRSF	number of free words for RRSs (1 word)
434	TCA.RRSN	address of next RRS entry (1 word)
438	TCA.KTAB	key file tab settings (2 words)
440	TCA.KPRJ	key file project name (2 words)
448	TCA.KVOL	key file volume name (4 words)
458	TCA.KDIR	key file directory name (4 words)
468	TCA.KTNO	key file tab count (1 byte)
469	TCA.KPNO	key file project length (1 byte)
46A	TCA.KVNO	key file volume length (1 byte)
46B	TCA.KDNO	key file directory length (1 byte)
46C	TCA.KRSV	reserved for bounding (1 word)
470	TCA.MODE	execution mode flags (1 byte). The flag bits have the following meanings:

Bit	Meaning When Set
0	reserved
1	job oriented (TCA.JOB)
2	terminal task (TCA.TRMT)
3	batch task (TCA.BTCH)
4	load debugger with task (TCA.DOLY)
5	RTM resident established (TCA.RESD)
6	command file active (TCA.DFIL)
7	SLO assigned to SYC (TCA.SLO)

471	TCA.NRRS	number of RRS entries (1 byte)
472	TCA.ALLO	number of pages to allocate (1 byte)
473	TCA.SPCE	task's logical address space (1 byte)
474	TCA.NBUF	number of blocking buffers (1 byte)
475	TCA.NFIL	number of FATs/FPTs to be reserved (1 byte)
476	TCA.PRIO	priority level of task (1 byte)
477	TCA.IOER	I/O error counter (1 byte)
478	TCA.LMN	load module name (1 doubleword)
480	TCA.SUDO	pseudonym of task (1 doubleword)
488	TCA.ONRN	ownername of task (1 doubleword)
490	TCA.PROJ	project name of task (1 doubleword)

Byte (Hex)	Symbol	Description
498	TCA.USRK	user key
498	TCA.VAT	number of VAT entries for dynamic mounts
499	TCA.FLG2	PTA flags (1 byte). The flag bits are defined as follows:

Bit	Meaning When Set
0	debug activating task (TCA.DEBUG)
1	Command Line Recall/Edit in effect (TCA.CLRE)
2-3	reserved
4	expanded PTASK block (TCA.EBLK)
5-7	reserved

49A	TCA.EXTD	map block number for extended MPX or -1 for MAXADDR or -2 for MINADDR (TCA.EXTD) (1 halfword)
49C	TCA.PGOW	option word for activation (1 word)
4A0	TCA.USW	user status word (1 word)
4A4	TCA.RPTR	pointer to RRS list
4A8	TCA.PGO2	second task option word (1 word)
4AC	TCA.FSIZ	size of fixed area of parameter task block (one halfword)
4AE	TCA.RSIZ	size of resource requirement summary (one halfword)
4C4	TCA.FRRS	address of first RRS entry (1 word)

3.2.3.2 Nested Context Area (NCA) Table

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
0	000	NCA.MPAR			
64	100	NCA.MSIZ			
68	110	NCA.ALST			
84	150	NCA.FLST			
88	160	NCA.CPAR			
152	260	NCA.VPAR			
168	2A0	NCA.PPOS			
172	2B0	NCA.CDIR			
176	2C0	NCA.CVOL			
184	2E0	NCA.PROJ			
186	2E8	NCA.MNUM	NCA.CNUM	NCA.STAT	
187	2EC	NCA.CNCA			
188	2F0	NCA.DALW			
189	2F4	NCA.FREE			

Byte (Hex)	Symbol	Description
000	NCA.MPAR	command file parameter area (32 doublewords).
100	NCA.MSIZ	command file parameter size area (16 bytes).
110	NCA.ALST	command file allocated space list (16 words).
150	NCA.FLST	command file free space list (4 words).
160	NCA.CPAR	command file argument area (256 bytes).
260	NCA.VPAR	command file binary parameter area (16 words).
2A0	NCA.PPOS	command file caller's parameter position (16 bytes).
2B0	NCA.CDIR	current working directory (4 words).
2C0	NCA.CVOL	long RID for current directory (8 words).
2E0	NCA.PROJ	current project name (1 doubleword).
2E8	NCA.MNUM	command file number of parameters (1 byte).
2E9	NCA.CNUM	command file number of arguments (1 byte).
2EA	NCA.STAT	halfword for status flags (1 halfword). The flag bits have the following meanings:

Bit	Meaning When Set
0	nested context flag (NCA.NEST)
1	nested command file error (NCA.NERR)
2-15	reserved

2EC	NCA.CNCA	caller's NCA address (1 word).
2F0	NCA.DALW	direct access location word (1 word).
2F4	NCA.FREE	reserved (3 words).

3.2.3.3 TSM Procedure Call Buffer (TPCB)

The TSM procedure call buffer (TPCB) is used by H.TSM to transfer data to and from J.TSM for TSM procedure calls. The TPCB is a 26 word buffer allocated from the miscellaneous memory pool area.

	0 31
Word 0	TPC.STAT. See Note 1.
1	TPC.DQEA. See Note 2.
2	TPC.TQUA. See Note 3.
3-26	TPC.BUFF. See Note 4.

Notes:

1. Bits in TPC.STAT are assigned as follows:

Bit	Meaning if Set
0	exit TSM procedure call mode (TPC.EXIT)
1	error occurred in TSM procedure call (TPC.ERR)
2	return data is ASCII text (TPC.TEXT)
3	invalid return buffer was supplied (TPC.NRB)
4	return buffer size is zero (TPC.RBZ)
5	return data was truncated (TPC.RDT)

2. This word contains the DQE address of the task performing the TSM procedure call.
3. This word contains the number of bytes in the data transfer buffer.
4. This word is the first word of the variable length data transfer buffer.

3.2.4 Intertask Communications

J.TSM interacts with two system tasks: J.SSIN and J.SOEX. J.SSIN spools input job streams and queues the job requests to the run request head cell in J.TSM's DQE. J.SOEX is sent run requests by J.TSM to service the PRINT command.

3.2.5 TSM Command Line Recall and Edit (CLRE) Processing

If CLRE is enabled for a terminal, when a formatted read is requested for that terminal, H.TSM initializes a TIQ for that read. H.TSM then links the TIQ to the input request queue (C.TSMIR) and sends a break request to J.TSM to process the read. Following is the TIQ data structure:

	0 7	8 15	16 23	24 31
Word 0	String Forward			
1	String Backward			
2	DQE address (TIQ.DQEA)			
3	Line buffer address (TIQ.LBFA)			
4	Read Option (TIQ.RDOP). See Note 1.	Read request byte count (TIQ.RDRQ)	UDT index (TIQ.UDTI)	
5	Read completion status (TIQ.RDCP) See Note 2.	Actual read byte count (TIQ.RDAC)	Reserved	
6	Reserved			
7	Reserved			

Notes:

1. TIQ.RDOP bits are defined as follows:

Bit	Meaning When Set
0	lowercase read (TIQ.LC)
1	no echo read (TIQ.NE)
2	option prompt (TIQ.OPP)
3	input requested (TIQ.IR)
4	input active (TIQ.IA)
5	associated task deleted (TIQ.DELR)
6-7	reserved

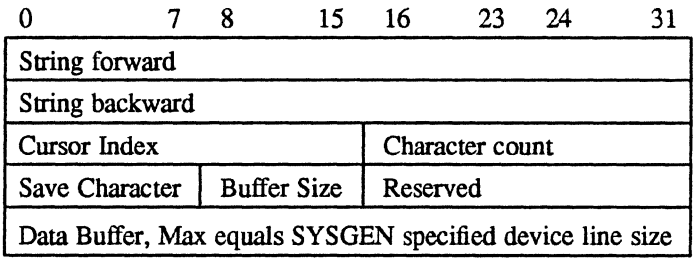
2. TIQ.RDCP bits are defined as follows:

Bit	Meaning When Set
0	I/O error (TIQ.ER)
1	EOF encountered (TIQ.EOF)
2-7	reserved

When the J.TSM break receiver is entered, the TSM input request queue is searched. If the queue is empty, J.TSM exits break receiver processing. If the queue is not empty, J.TSM unlinks the TIQ entry from the input ready queue, links the TIQ to the input active queue, and issues a no-wait unformatted, ASCII control character detect read to the appropriate terminal. J.TSM does not exit break processing until the input request queue is empty.

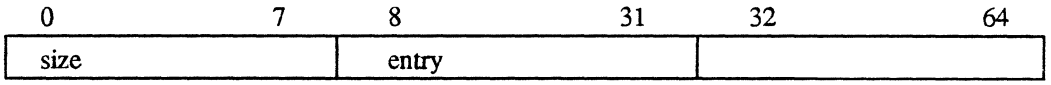
When J.TSM enters CLRE input end action, the recall or edit function requested is determined and executed. Each terminal has a control string table and a recall buffer consisting of a 1- to 23-line buffers. Each terminal also has a current recall line buffer and a CLRE write buffer associated with it. These structures are maintained with pointers in each terminal's TCA.

Following is the recall line buffer data structure.



The memory necessary for the following structures is allocated in 192-word blocks and is not necessarily contiguous.

TCA.SCTA is a pointer to the control string function table. There is one entry in this table for each function defined in the MPX.PRO file. Each entry is two words in size and has the following format:



size is the size of the control string (1 byte)
entry is the control string, maximum of 7 bytes.

TCA.GMAT is a pointer to the memory address table. Each entry is one word in size and contains the address of a 192 word memory block obtained by the M.MEMB service.

TCA.WBA is a pointer to the CLRE write buffer. This buffer is two times the terminal line size in bytes.

TCA.2LBF is a pointer to the first recall line buffer allocated for this context.

TCA.RLBL is a pointer to the last recall line buffer allocated for this context.

3.3 System Mount Task (J.MOUNT)

J.MOUNT, the system nonresident media mounting task, is executed with a run request from the Resource Management Module (H.REMM). J.MOUNT performs the following functions:

- mounts a formatted volume; for example, disk or floppy disk
- mounts an unformatted medium; for example, tape, disk or floppy disk
- dismounts a formatted volume
- dismounts an unformatted medium

Information in the run request from H.REMM determines which function J.MOUNT performs.

When a volume is being mounted, J.MOUNT initially executes the AUTODISK subroutine on the volume to be mounted. For more information on AUTODISK refer to the Autodisk Subroutine section in Chapter 7 of this volume.

3.3.1 Run Request Interface

H.REMM passes a block of information with each run request to J.MOUNT. This block can be two words or 16 words. If it is two words, a formatted mount is being requested. If it is 16 words, one of the other three functions is being requested. Information in the block identifies the specific function requested.

3.3.1.1 Formatted Mount Requests

The formatted mount request from H.REMM carries with it a two word message. This message contains the following information:

Word 0	Mount device specification
Word 1	Flag field and mounted volume table address

Word 0: The mount device specification contains the device type in byte zero and the device channel address, if present, in byte two with bit zero set to indicate its presence. The device subchannel address, if present, is supplied in byte three with bit 16 set to indicate its presence.

Word 1: The flag field has the following bit significance when set:

Bit	Meaning if set
0	mount message inhibited
1-7	reserved

The mounted volume table address is the address of the mounted volume table (MVT) entry to be completed for this volume.

3.3.1.2 Unformatted Mount Requests

When requesting an unformatted mount, H.REMM passes a 16 word message. This message contains the file assignment table (FAT) entry for the device on which the medium is to be mounted.

3.3.1.3 Formatted and Unformatted Dismount Requests

H.REMM passes a 16 word message with both formatted and unformatted dismount requests. This message, like the unformatted mount message, contains the FAT entry for the device requiring the dismount. J.MOUNT distinguishes between mount and dismount requests by testing bit two of DFT.ACF (mount message output) in the H.REMM supplied FAT. If this bit is set, a dismount is being requested. J.MOUNT distinguishes between a formatted or an unformatted dismount by testing bit four of DFT.STB (unformatted medium). If this bit is set, unformatted is indicated.

3.3.2 Mount Messages and Requests

For formatted volumes, the mount message that displays to the system console is as follows:

```
MOUNT VOLUME volname ON ddccss
REPLY R,H,A OR DEVICE:
```

volname is the 1- to 16-character volume name
ddccss is the device mnemonic, channel, and subaddress

For unformatted media, the mount message to the system console is:

```
MOUNT name VOLnnn ON ddccss
TASK taskname,taskno REPLY R,H,A OR DEVICE:
```

name is the 1- to 4-character reel ID
nnn is the volume sequence number if a multivolume tape
ddccss is the device mnemonic, channel, and subaddress
taskname is the 8-character task name
taskno is the 8-digit hexadecimal task number

Respond to the mount message with one of the following:

- Reply `R` to indicate the volume is ready and to proceed with processing.
- Reply `A` to abort the requesting task.
- Reply `H` to hold the requesting task and reprocess the mount at a later time.
- If desired, specify a device other than the one requested by the user for the volume to be mounted on. To do so, use a format similar to the one displayed in the mount message (e.g., DM0804, M91001). If this option is specified, another mount message issues to confirm the user's choice. The device for a formatted mount can be changed several times before replying `R`, `H`, or `A`. The device specification for an unformatted mount can only be given once. If a second device specification is given, a warning message and a repeat of the mount message display.

An additional option, available only on unformatted mount requests, is to specify recording density. If the requested device allows software control of the density, the user can specify the recording density in the `R` reply to the mount message. The available density specifications are listed below:

Reply	Meaning
`R`	use default density specification
`RN`	NRZI mode, 800 bpi
`RP`	PE mode, 1600 bpi
`RG`	GCR mode, 6250 bpi
`R800`	NRZI mode, 800 bpi
`R1600`	PE mode, 1600 bpi
`R6250`	GCR mode, 6250 bpi

The density specification, if given, must immediately follow the `R`, as shown above in the Reply column.

3.3.3 Checks on Mounted Volumes

J.MOUNT verifies the volume name for the current volume by comparing the name in the MVT entry supplied by H.REMM with the name in the volume descriptor of the volume currently on the requested device. If the names do not match, J.MOUNT informs the system console and provides the actual name of the volume. The exception to this is when the system is being initialized by SYSINIT. Because SYSINIT does not know the name of the system volume or the swap volume, J.MOUNT cannot verify it. Instead, J.MOUNT uses the volume name in the volume descriptor. When SYSINIT is running, normally no mount message displays. However, if a user modification to SYSINIT turns on the mount message for the system or swap volume, then the SYSTEM VOLUME name displays.

In addition to checking the volume name, J.MOUNT checks the volume's last dismount date and time against the current date and time. If the dismount values are more recent than the current date and time, a warning message displays and the user can continue or abort. This check helps detect incorrect entry of date and time, which could cause problems in the operating system that uses date and time marking on files.

When mounting a formatted volume, J.MOUNT also checks whether the volume was previously dismounted. If so, volume clean-up is not needed since H.REMM updates the volume descriptor's values for available and allocated space when a volume dismounts. Additionally, because H.VOMM maintains the space and descriptor maps during normal system operation, the maps reflect the current condition of the disk and need no updating.

The exception to this occurs when an I/O error occurs during the last mount session while updating critical disk file system structures (RDs, directories, SMAP or DMAP). In this case, the state of SMAP and DMAP is not reliable. Therefore, J.MOUNT performs volume cleanup while mounting a formatted volume. This indicates that critical I/O errors have occurred on the volume during the last mount session.

If the volume was not previously dismounted, J.MOUNT performs volume clean-up. Volume clean-up is necessary because the volume descriptor and the maps can contain invalid information, and temporary files and unprocessed spool files can exist (this is possible following a system crash). For multiport disk mount, volume cleanup is an option for the user.

During volume clean-up, J.MOUNT reads all resource descriptors on the disk being mounted, starting with the volume descriptor. J.MOUNT rebuilds the space map, the descriptor map, and the pertinent counts contained in the volume descriptor. Each type of resource descriptor is processed according to its particular needs. For example, temporary files are deleted, permanent files are marked as allocated in the SMAP and DMAP, and spool files are resubmitted.

If an invalid resource descriptor is detected, J.MOUNT zeroes words 7 and 22 of the resource descriptor (the resource descriptor type and the link count), and stores a problem code, in ASCII, in the free section of the resource descriptor. Currently, an invalid resource descriptor type field or a bad segment definition are the only reasons for marking a resource descriptor invalid. If the system debugger is configured into the current system and control switch six is set during mount, J.MOUNT enters the debugger when it encounters an invalid resource descriptor, thereby allowing the user to discover the nature of the problem (R2 contains the address of the invalid resource descriptor). For multiport resources, all access information and resource descriptor locks are reinitialized.

If a disk is inserted into a disk drive and mounted, that has a defect in cylinder 0, unpredictable errors may occur. Before continuing to use the disk, verify that cylinder 0 is good or data may be lost from disk.

If file overlap is detected, the following messages display on the system console and the volume is not mounted:

```
FILE OVERLAP HAS OCCURRED IN RD num
RD TYPE num
FILENAME is name
SECTORS num THROUGH num
```

num is a hexadecimal number
name is the 1- to 16-character file name

If file overlap is detected in the DMAP/SMAP deallocation file descriptor area, the following message is displayed on the system console and the volume is not mounted.

```
J.MOUNT - ERROR - FILE OVERLAP HAS OCCURRED IN THE BAD SMAP
```

or

```
J.MOUNT - ERROR - FILE OVERLAP HAS OCCURRED IN THE BAD DMAP
```

To mount the disk so data can be recovered, set control switch zero or seven. The following control switches apply to the clean-up of formatted volumes:

Switch	Function if Set
0	inhibits volume clean-up by J.MOUNT
6	if J.MOUNT encounters an invalid resource descriptor due to an invalid resource descriptor type field or space definition, it branches and links to the system debugger (if present) with register two pointing to the resource descriptor
7	J.MOUNT prereads the file space bit map (SMAP) or the resource descriptor allocation bit map (DMAP); J.MOUNT does not perform file overlap detection
8	delete spooled output files instead of resubmitting them for processing

3.3.4 Dismount Messages and Requests

J.MOUNT is requested by H.REMM to perform a physical dismount of a formatted volume or to display messages about the dismount of an unformatted medium.

For formatted dismounts, J.MOUNT verifies the presence of the requested volume name on the assigned device. If the volume name on the device matches the name in the MVT entry for the requested volume, then J.MOUNT updates the volume descriptor of the mounted volume using current information in the MVT for that volume. J.MOUNT then deallocates the mount device and clears the corresponding MVT entry. Formatted dismounts require operator interaction through the system console to complete, unless operator interaction has been inhibited by the NOMSG option in the original mount or dismount request, or through the OPCOM MODE/SIMM or SNOP with the system-wide SNOP with option.

For formatted volumes, the following dismount message displays at the system console:

```
CONFIRM PHYSICAL DISMOUNT OF VOLUME volname
FROM ddccss
REPLY R TO RESUME:
```

volname is the 1- to 16-character volume name
ddccss is the device mnemonic, channel, and subaddress

The operator must enter a character response to continue.

For unformatted media, the dismount message that displays at the console is:

DISMOUNT *reel* VOL *nnn* FROM *ddccss*

reel is the 1- to 4-character reel ID
nnn is the volume sequence number if multivolume tape
ddccss is the device mnemonic, channel, and subaddress

After a formatted volume or unformatted media is dismounted, the following message displays at the requestor's terminal:

PHYSICAL DISMOUNT OF VOLUME *volname* FROM DEVICE *ddccss* COMPLETE

3.3.5 Error Status Return

J.MOUNT returns error status through the user status field of the sender's parameter send block (PSB). A summary of these codes and their meaning follows:

Code	Explanation
0	normal return, no error
1	requested volume name does not match name in the disk's volume descriptor
8	unrecoverable I/O error on requested volume
20	unable to initialize the requested volume
37	invalid request parameter length
43	user requested hold
42	user requested abort

3.4 Multiprocessor Recovery Task (J.UNLOCK)

J.UNLOCK is the multiprocessor recovery task. In a multiprocessor system, J.UNLOCK allows any processor to recover and continue processing when one of the port processors goes offline. Resource locks, assign counts, and user counts owned by the offline processor are removed from the shared volumes by an online processor.

3.4.1 Structure

J.UNLOCK is a resident, privileged task which resides in the any wait queue. It is activated by an OPCOM UNLOCK directive or by J.MOUNT when a multiprocessor shared volume is mounted. When the last multiprocessor shared volume is dismounted from the system, J.UNLOCK deactivates.

3.4.2 Entry Conditions

When activated by J.MOUNT, J.UNLOCK is linked to the any wait queue. J.UNLOCK is resumed when a run request is issued from J.MOUNT or when an OPCOM UNLOCK directive is issued. J.UNLOCK executes at priority 58.

3.4.3 Exit Conditions

Each time a multiprocessor shared volume is dismounted, J.MOUNT sends J.UNLOCK a run request. If no shared volumes are mounted when the request is received, J.UNLOCK exits.

J.UNLOCK has an abort receiver, and can be aborted by the OPCOM ABORT directive. When aborted, J.UNLOCK completes its current activity before exiting.

3.4.4 Multiprocessor Recovery

When one processor goes offline, J.UNLOCK scans the resource descriptors (RDs) on the multiprocessor shared volumes. The resource allocation status of each RD is changed to reflect the online processors.

After the status is changed, each RD is allocated by space definition. Space definition allows access to a resource which was locked by an offline processor. A time-out value prevents contention between RDs and other tasks in the same environment. The RDs are then recorded and processed.

J.UNLOCK compares the allocation information in each RD with the allocation information in the memory resident allocated resource table (ART) for the RD. If an ART does not exist for an RD, the RD's allocation information is reinitialized. If an RD contains status for an offline processor, the RD's information is reinitialized.

The allocation information in each RD is documented in Chapter 2.

Words 93, 94, and 95 correspond to words 1, 2, and 3 of the memory resident ART entry.

The processing of each resource descriptor (RD) contains four steps:

1. multiprocessor RD lock processing
2. assign and user count processing
3. resource exclusive and inclusive lock processing
4. reader count and access processing

Multiprocessor lock processing stores the RD locks in bytes 2 and 3 of word 191. If a lock in effect belongs to the port to be unlocked, word 191 is reinitialized.

Assign and user count processing reinitializes the RD's allocation information if the total assign count is zero. If the count is greater than zero, the specified processor's assign and user counts are reinitialized. The memory resident ART counts for the specified port processor are modified to reflect the new assign and user counts.

Resource exclusive and inclusive lock processing removes locks owned by an offline processor. The locks are removed from AR.PORT within AR.FLAGS in the ART.

Reader count and access processing corrects the reader count and access modes.

There are eight combinations of resource descriptor (RD) access modes:

- readers
- readers plus one writer
- readers plus one appender
- one writer
- one modifier
- one updater
- one appender
- one modifier plus one appender

Only one combination is allowed at a time.

The resource allocation flags field of the allocated resource table (ART) is checked for set flags. To determine the access mode, J.UNLOCK uses the RD's access mode plus AR.WOWN and AR.WOWN2 from the online processor's ART.

For the readers group (combination one), the read access mode is set and the new reader count is set to equal the online processor's assign count. The access field is initialized if the reader count is zero.

For the reader plus one writer or appender groups, combinations two and three, if AR.WOWN or AR.WOWN2 is set, the read count is set to be one less than the online processor's assign count. If the resulting read count is zero, read access is removed. The read count plus the writer or appender equals the online processor's assign count.

For one writer, modifier, updater, or appender, combinations four, five, six, and seven, the reader count is zero. If the writer is not from this environment, the access field is initialized.

For the one modifier plus one appender group (combination eight), the reader count is zero. J.UNLOCK uses A.WOWN and A.WOWN2 to determine if the processor has modify, append, or both access modes set. The access field is set accordingly.

3.4.5 Error Status Return

J.UNLOCK sends a message to the operator console if an error occurs.

3.5 System Spooled Output Tasks (J.SOUT and J.SOEX)

3.5.1 Functional Description

The spooled output executive (J.SOEX) schedules activations of the spooled output task, J.SOUT. The activations are based on device availability. If spooled output is requested on a device in use, the request is queued by J.SOEX on the SOEX run request queue (SRRQ) until the device becomes available.

Spooled output—printing and punching—is controlled by J.SOUT. An activation of J.SOUT can exist for every output device configured into the system.

Upon completion of output spooling, J.SOUT sends a break to J.SOEX to indicate the end of spooling and exits the system.

3.5.2 Operational Design

J.SOEX is activated by a run request from a user task or one of the following tasks or services:

- J.TSM
- J.MOUNT
- M.DASN
- SYSINIT

When J.SOUT scheduling is complete, J.SOEX goes into a wait state. J.SOEX is resumed by new run requests or by devices becoming available. Once activated, J.SOEX never exits the system.

3.5.2.1 J.SOEX Message Receiver

The J.SOEX message receiver handles messages sent to it by OPCOM. All messages are sent in the wait mode with call back enabled. The messages have the following format:

	0 7	8 15	16 23	24 31
Word 0	Type. See Note 1.	Subtype. See Note 2.	Reserved	
1	Variable message information. See Note 3.			
2	Variable message information. See Note 4.			
3	Variable message information. See Note 5.			
4	Job or task number			
5-6	Job or task name			
7-8	Device mnemonic			

Notes:

1. This byte indicates the type of message as follows:

Type	Description
0	list print directive
1	list punch directive
2	deprint directive
3	depunch directive
4	reprint directive
5	repunch directive
6	redirect directive

2. This byte indicates the message subtype to be used by J.SOEX as follows:

Type	Description
0	default subtype for particular directive
1	job/task
2	task name
3	device mnemonic

3. This word contains variable message information depending on the message type as follows:

Message Type	Message Information
0	current SRRQ entry address
1	current SRRQ entry address
2	not used
3	not used
4	reprint/repunch count
5	reprint/repunch count
6	not used

4. This word contains variable message information depending on the message type as follows:

Message Type	Message Information
0	string forward address of current SRRQ entry
1	string forward address of current SRRQ entry
2	not used
3	not used
4	reprint/repunch starting/stopping pages
5	reprint/repunch starting/stopping pages
6	first word of the REDIRECT destination device mnemonic. This is zero if the default output device is used.

5. This word contains variable message information depending on the message type as follows:

Message Type	Message Information
0	string backward address of current SRRQ entry
1	string backward address of current SRRQ entry
2-5	not used
6	second word of the REDIRECT destination device mnemonic. This is zero if the default output device is used.

Message types 0 and 1 retrieve information from the SRRQ for the LIST PRINT and LIST PUNCH directives. The information is returned to OPCOM as part of the call back.

Message types 2 and 3 initiate SRRQ processing by J.SOEX for DEPRINT and DEPUNCH directives. No information is returned from these message types.

Message types 4 and 5 initiate SRRQ modifications by J.SOEX for REPRINT and REPUNCH directives. No information is returned from these message types.

Message type 6 initiates SRRQ modifications by J.SOEX for the REDIRECT directive. No information is returned from this message type.

3.5.2.2 Call Back Information

The call back can contain information on up to ten SRRQ entries. This information is formatted and displayed by OPCOM. The call back information is contained in a buffer with the following format:

	0 7	8 15	16 23	24 31
Word 0	Status. See Notes 1 and 2.	Number of returned entries		
1	Current SRRQ entry address			
2	Current string forward pointer			
3	Current string backward pointer			
4	Job or task number. See Note 3.			
5-6	Job or task name			
7-8	Owner name			
9	Priority			
10-11	Reserved			
12-84				

Notes:

1. The first four words contain a header.
2. This halfword indicates the status as follows:

Status	Description
0	SRRQ not completely processed
1	SRRQ processed

   If the status is zero, another message is sent to J.SOEX. J.SOEX then completes the SRRQ processing.
3. Words 4 through 11 make up one SRRQ entry description. These eight words can be repeated until the call back buffer contains ten SRRQ entries.

   The initial message for J.SOEX to process a LIST PRINT or LIST PUNCH directive contains zero values in words 1 through 3. If another message is sent to J.SOEX, words 1 through 3 in the return buffer are used as words 1 through 3 in the new message. This lets J.SOEX continue directive processing where it left off in the SRRQ.

3.5.2.3 Return Status

When the message receiver completes processing and exits the message receiver level, return status is posted in the user status byte of the parameter send block. Returned status is defined as follows:

Status	Description
0	operation successful
1	no entries in SRRQ
2	SRRQ pointers incorrectly linked
3	error condition encountered
4	invalid message type
5	attempting to deprint SBO output
6	attempting to depunch SLO output
7	error encountered while attempting to delete resource
8	invalid or missing default SLO device
9	invalid or missing default POD device
10-255	reserved

3.5.2.4 Break Receiver

J.SOEX has a break receiver that is entered when an interrupt is sent by a user task or any of the following tasks or services:

- J.SOUT
- J.TSM
- J.MOUNT
- M.DASN
- SYSINIT

J.TSM, M.DASN, and real-time tasks send interrupts to J.SOEX after a run request is sent. If a task does not send an interrupt, J.SOEX checks for queued run requests. If there are any, control is transferred to the break receiver that processes the run request.

When the break receiver is entered, information is transferred from the memory run request queue (MRRQ) to the J.SOEX run request queue (SRRQ). Memory for the SRRQ entries is allocated from the logical address space of J.SOEX. The format of an SRRQ is the same as the format of a J.SOEX run request. See the J.SOEX Run Request section in Chapter 2. After the transfer, the MRRQ entry is unlinked from the run request and the associated memory pool is deallocated.

3.6 Online Help Facility

The online help facility enables users to display help information on their terminals and to compose additional help information. This section describes the tasks and data structures that make up the online help facility. For information about how to use online help, refer to the Online Help chapter in Volume II of the MPX-32 Reference Manual.

3.6.1 Online Help Tasks

Online help contains the following tasks:

HELP	sends message requests to J.HLP
J.HLP	displays help information on the terminal
HELPT	translates the information files into a usable format for J.HLP

The following sections describe these tasks in more detail.

3.6.1.1 HELP Task

The HELP task receives requests for information from users and, in turn, sends message requests to J.HLP. HELP is a multicopied task and can be simultaneously invoked by more than one user.

Once HELP sends the message request to J.HLP, it waits for J.HLP to return a message request informing HELP that J.HLP has completed servicing the terminal. HELP then returns control to TSM.

3.6.1.2 J.HLP Task

The J.HLP task receives message requests for help information from the HELP task, J.TSM, or user tasks. It then takes the help information requested from the help files and displays this information on the screen. J.HLP is always active (ANYW) and waiting for message requests from the tasks via J.TSM. J.HLP performs I/O to a terminal in no-wait mode, and functions at end-action interrupt level. This enables J.HLP to concurrently service multiple terminals.

The following is a list of the routines in J.HLP:

Routine	Description
INIT	allocates terminals and builds a terminal context area (TCA) per terminal
GETFILES	prepares a list of the information files
OPNFILES	opens all information files and builds an FCB for each
BLDTNTAB	builds a topic name table in alphabetical order
SEEK	manipulates blocking buffers to read certain locations in files
LOCATE	gets location of the next read/write to the information file
MD_ME	puts highlight on/off codes in strings being written to screen
CURMOV	get codes to place cursor at a certain location on the screen
ARR.STR	gets appropriate arrow key strings as defined in MPX.PRO
GETARROW	detects if an arrow key has been pressed
BACKWARD	backtracks a screen of information from a file
ABRTRCVR	abort receiver
RCVR	message receiver; accepts message requests from other tasks.
TYIOEA	terminal I/O end-action receiver, next action routines
FILERR	marks files with errors as offline
BADFILE	displays a message on the terminal if an offline file is encountered
TOPISRCH	searches topic name table for a topic name
FINDTOPI	finds the specified topic in the topic name table
DISPTEXT	displays text for the specified topic
DISPKEYW	displays keywords for the specified topic
MENU	builds and displays the menu mode prompt
KEYWRESP	prompts for and processes a keyword input response
TOPIRESP	prompts for and processes a topic name input response
KEYSEL	processes a keyword selection via the arrow support
QUIT.HLP	quits help
ATLINK	links an entry into audit trail
ATUNLINK	unlinks (i.e. returns) all audit trail memory
PRT_SCRN	builds, outputs and processes the print screen menu
GET_SNUM	prompts for and processes the number of screens to print
CHK_SNUM	checks entered number against audit trail total
NEXT_ATS	gets address of next screen to print
GET_PATH	prompts for the name of a permanent file
CHK_PATH	checks if the entered file is usable
PRT_OPEN	assigns and opens the output file
PRT_CLSE	closes and deassigns the output file
LINE_NUM	prefixes the line number to the text to be printed
CHG_DEFS	changes defaults to user's current volume and directory

The following is a list of the next action modes and routines that are determined by the TYIOEA routine listed above:

Next Action Mode	Routine
Find topic mode (FTM)	FINDTOPI
Display text mode (DTM) DTCSSM - Clear screen submode DTTLSM - Title line submode of display text DTPISM - Positional information submode DTDTSM - Text line submode of display text	DISPTEXT
Display keywords mode (DKM) DKRKSM - Read keywords submode DKKMSM - Keyword prompt message submode DKKRSM - Keyword prompt read DKDLSM - Display keyword list submode	DISPKEYW
Menu mode (MEM) MEMDSM - Menu display MEMRSM - Menu prompt read MEMPSM - Menu prompt response processing	MENU
Process keyword menu selection mode (KRM) KRDPSM - Display keyword prompt KRRKSM - Read keyword KREASM - Read keyword end action KRFASM - Keyword search failure acknowledge	KEYWRESP
Process topic menu response mode (TRM) TRDPSM - Display topic prompt TRRTSM - Read topic name TREASM - Read topic name end action	TOPIRESP
Keyword selection via arrow mode (KSM) KSINSM - Choose first keyword to put cursor on KCWCSM - Write cursor on screen location KSRRSM - Read 1 character response KSPRSM - Process 1 character response	KEYSEL
Quit help mode (QHM)	QUIT.HLP
Print screen mode (PSM)	PRT_SCRN

The memory requirement for J.HLP in its own logical address space is dynamic. Given the maximum logical task space of 128KW minus the size of the resident OS and TSA area, the memory requirement for J.HLP at any given time is:

No. words + 3000 + 6**tn* + 40**tc* + 392**cu* + 20**hf* + 17**ck*

tn total number of topic names in all help files

tc number of configured terminals

cu number of users currently using J.HLP

hf number of help files

ck number of keywords currently being displayed by J.HLP

3.6.1.3 HELPT Task

The HELPT task translates information files that a user creates with an ASCII editor into a usable format for J.HLP. These files remain as ASCII files so that the user can edit the files as needed. Each time a file is edited, it must be translated with the HELPT task. The following is a list of the HELPT states:

State	Description
INST	initial program state; wait to transfer to topic state
TNST	topic name state; processes and writes topic name record
LKST	sets up linked lists for text, keywords, and cursor position information
TXST	puts text in linked lists
KWST	finds keyword in text, puts .KW and .PI's in linked list
PIST	sorts positional information and writes to file
WRST	write text, keywords and deallocates memory from all lists
EFST	sorts topic name list, writes, and updates .TL records
SQST	error in state transition, report it and abort

3.6.2 Data Structures

Online help contains the following data structures:

Terminal Context Area (TCA)
Topic Name Table List (TNTL)
Keyword List (KWLI)
Positional Information List (PILI)
Print Screen Audit Trail (PSAT)

The following text describes those data structures.

3.6.2.1 Terminal Context Area (TCA)

The TCA contains information about the terminal from which the user is accessing the Online Help facility. The following table shows the fields of the Online Help facility TCA:

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
0-15	000	TCA.FCB			
16-31	040	TCA.PFCB			
32	080	TCA.UDTA			
33	084	TCA.STK			
34	088	TCA.STKL	TCA.NACT	TCA.SACT	TCA.LINE
35	08C	TCA.CLIN	TCA.CROW	TCA.ARNO	TCA.RESB
36	090	TCA.STAT			
37	094	TCA.TYID			
38	098	TCA.TNO			
39	09C	TCA.STCA			
40	0A0	TCA.LBFA			
41	0A4	TCA.FCBA			
42	0A8	TCA.LOCW			
43	0AC	TCA.KWLF			
44	0B0	TCA.KWLL			
45-46	0B4	TCA.PILF			
47	0BC	TCA.PILL			
48	0C0	TCA.CKWA			
49	0C4	TCA.ARRO/TCA.LEFT			
50	0C8	TCA.RGHT			
51	0CC	TCA.UP			
52	0D0	TCA.DOWN			
53	0D4	TCA.TXLN			
54	0D8	TCA.CPIA			
55	0DC	TCA.RTRN			
56	0E0	TCA.KWLN			
57-59	0E4	TCA.ATHC			

Word No. (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
60	0F0	TCA.PSMA			
61-73	0F4	TCA.PATH			
74	128	TCA.AIDX			
75	12C	TCA.CEAD			
76-79	130	TCA.CVOL			
80-83	140	TCA.CDIR			
84-85	150	TCA.OWNR			
86-87	158	TCA.PROJ			
88	160	TCA.PKEY	TCA.RSVB		
89	164	TCA.RESW			

Byte (Hex)	Symbol	Description
000	TCA.FCB	a 16-word FCB assigned to a user terminal. All terminal I/O is done in no-wait through this FCB
040	TCA.PFCB	a 16-word FCB assigned to either a permanent file or to SLO. All printing is done in no-wait through this FCB.
080	TCA.UDTA	UDT address of terminal
084	TCA.STK	address of topic stack
088	TCA.STKL	stack level; up to 32 levels
089	TCA.NACT	next action modes
08A	TCA.SACT	sub action mode values
08B	TCA.LINE	number of lines on screen
08C	TCA.CLIN	number of lines currently displayed
08D	TCA.CROW	current row on screen
08E	TCA.ARNO	current arrow key being processed
08F	TCA.RESB	reserved byte for word boundary

Byte (Hex)	Symbol	Description
090	TCA.STAT	status flags:

Bit	Meaning When Set
0	terminal was assigned (TCA.ASSN)
1	multiple topic names detected for topic entry (MTNFLAG)
2	no line feed needed (NOLF)
3	cursor addressing is available (CURADD)
4	end of topic has been reached (EOT)
5	routine TOPISRCH did not find topic name in topic name table (TCA.TNSR)
6	displaying keyword list (TCA.DSKW)
7	display keyword prompt for list (TCA.KWPR)
8	J.HLP has been offlined (OFFLINED)
9	save topic name in audit trail (AT.SAV)
10	currently on print menu (P.MENU)
11	printing in progress (P.PRT)
12	output goes to permanent file (P.FILE)
13	pathname OK to use (P.POK)
14	second pathname prompt issued (P.2ND)
15	number output lines (P.NUM)
16	highlight keywords, if possible (P.HILIT)
17	output file must be opened (P.OPEN)
18	open on output file failed (P.OFAIL)
19	output file must be created (P.CREAT)
20	processing a title line (P.TITLE)
21	processing a multiscreen topic (P.MSCRN)
22	number of screens to print is OK (P.NOK)
23	more screens to print (P.MORE)
24	pathnames are the same length (P.LSAME)
25	output file has been opened (P.OPENED)
26	TERMDEF not available (NO_TDEF)
27	redraw print menu (REDO)
28-31	reserved

Byte (Hex)	Symbol	Description
094	TCA.TYID	terminal ID for TY device
098	TCA.TNO	task number requesting help
09C	TCA.STCA	requesting task's TCA address
0A0	TCA.LBFA	line buffer address
0A4	TCA.FCBA	current help file FCB address
0A8	TCA.LOCW	location word for next help record
0AC	TCA.KWLF	keyword list forward entry pointer
0B0	TCA.KWLL	keyword list last entry pointer
0B4	TCA.PILF	position information list forward entry pointer
0BC	TCA.PILL	position information list last entry pointer
0C0	TCA.CKWA	current keyword entry address
0C4	TCA.ARRO	where addresses of arrow strings are
0C4	TCA.LEFT	address of string used as left arrow
0C8	TCA.RGHT	address of string used as right arrow
0CC	TCA.UP	address of string used as up arrow
0D0	TCA.DOWN	address of string used as down arrow
0D4	TCA.TXLN	current text line on this topic
0D8	TCA.CPIA	current position information entry being used
0DC	TCA.RTRN	return address for end action subroutines
0E0	TCA.KWLN	number of keyword lines displayed
0E4	TCA.ATHC	audit trail linked list head cell
0F0	TCA.PSMA	address of print screen menu data
0F4	TCA.PATH	print screen output pathname
128	TCA.AIDX	allocation index of output pathname
12C	TCA.CEAD	current audit trail entry address
130	TCA.CVOL	user's current working volume name
140	TCA.CDIR	user's current working directory name
150	TCA.OWNR	user's owner name
158	TCA.PROJ	user's project name
160	TCA.PKEY	first character of current print screen keyword
161	TCA.RSVB	reserved bytes
164	TCA.RESW	force doubleword boundary

3.6.2.2 Topic Name Table List (TNTL)

The TNTL contains a list of all the topics and their location in the information files. Each entry is six words that are defined as follows:

Word	Description
0-3	topic name (TN.NAME)
4	file index containing topic (TN.FILIX)
5	topic name location word (TN.LOCW)

3.6.2.3 Keyword List (KWLI)

KWLI contains the keywords that are defined for the help information topic that is currently being displayed. Each entry is 10 words that are defined as follows:

Word	Description
0	forward pointer to next keyword (KW.NKW)
1	reserved space for .KW characters (KW.KWR)
2-5	keyword (KW.KW)
6-9	topic name (KW.TN)

3.6.2.4 Positional Information List (PILI)

PILI contains the keywords that are embedded in the help information topic that is currently being displayed. Each entry is 13 words that are defined as follows:

Word	Description
0	forward pointer to next entry (PI.NPI)
1	backward pointer to last entry (PI.LPI)
2-5	keyword (PI.KW)
6-9	topic name (PI.TN)
10	text line where keyword is located (PI.LIN)
11	column number where keyword is located (PI.COL)
12	length of keyword (PI.LEN)

3.6.2.5 Print Screen Audit Trail (PSAT)

The PSAT contains the names of the screens traversed during a help session. Only the screens visited in a forward direction are kept; those screens displayed using the (R)eturn function are not maintained in the list. Each entry is 6 words defined as follows:

Word	Description
0	forward pointer to next entry (AT.SF)
1-4	screen name (AT.SN)
5	backward pointer to last entry (AT.SB)

3.6.3 Interfacing J.HLP with Other Tasks

Any user task that issues data format inhibit (DFI) reads cannot use the [help] key to invoke J.HLP. Instead, the user must design an interface that allows the user task to generate message requests for J.HLP.

3.6.3.1 Sending Message Requests Via the Interface

To send message requests via the interface to J.HLP, use the send message request (M.SMSGR) service call. For more information about M.SMSGR, refer to the Nonbase Mode System Service chapter in Volume I of the MPX-32 Reference Manual. M.SMSGR requires that a parameter send block (PSB) be set up to send an 18 word data structure in no-wait mode. This data structure contains the following fields:

Word	Byte	Description
0	0	message type (MSG.TYPE); must be set to 0
	1-3	reserved for sender's TCA address in J.TSM (MSG.STCA)
1		terminal's hex device, channel/subaddress, right-justified and zero filled (MSG.TYID)
2-5		topic requested (MSG.TOPI); left-justified, blank-filled
6-9		user's current volume name (MSG.CVOL)
10-13		user's current directory name (MSG.CDIR)
14-15		user's owner name (MSG.OWNR)
16-17		user's project name (MSG.PROJ)

After J.HLP services the message request, it sends its own message request to the user task via the interface. It contains the following two words of status information:

Word	Byte	Description
0	0	contains one of the following values:

Value	Meaning
0	no errors
1	invalid message type received (MSG.TYPE)
2	channel/subaddress not found in TCA (MSG.TYID)
3	help is offline

	1-3	sender's TCA address in J.TSM (MSG.STCA)
1		terminal's hex device, channel/subaddress, right-justified and zero-filled (MSG.TYID)

4 System Generation Task Description

4.1 Task Structure and Functional Organization

The System Generation Task, SYSGEN, is a privileged system task that operates within the framework of a standard MPX-32 system and can be executed in batch or interactive mode. It consists of an executive segment and five overlays. Table 4-2 shows the loading sequence and gives a description of each phase.

System generation for an MPX-32 system involves supplying a set of configuration directives to the SYSGEN task. Table 4-1 shows the functional breakdown of directive processing for each overlay. The end result is the creation of a permanent file containing the installation specific MPX-32 system in memory image absolute format. This file may be subsequently restarted or utilized on a system distribution tape (SDT).

System generation by the SYSGEN utility is described in the MPX-32 Reference Manual Volume III. This chapter provides a functional description.

Table 4-1
SYSGEN Overlays - Overview of Functions

S.EXEC				
S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
Opens statically allocated files: DIR, OBJ, OBR, and SLO. Outputs titles. Sets up map info and variables for remap of target system. Initializes default values in target communications region.	Reads directives from DIR. Writes directives to SLO. Does initial processing of directives.	Initializes tables (memory, ITLB, patch, timer, RTM, activation, sequence, ART, and MVT). Constructs dispatch queue and DQE address tables and links them. Builds DTT, CDT, CHT, and UDT tables. Processes interrupt table entries and builds load module table entries. Builds target system scratch pad image. Obtains space for the SCV 1 and 2 tables, the GPMC table, and the module address table. Builds miscellaneous system parameter (MPL, MIDL, etc.)	Positions object files and allocates temporary disk space for loading processes. Initializes loader variables and sets up pointers to the interrupt table. Scans object input file (OBR) for match on program name in module record file. When match found, copies to temporary disk file and loads it. Scans object input file (OBJ) for match on program name in module record file. When match found, copies to temporary disk file and loads it. Initializes system modules (branches and links to their last entry point). Outputs load map. Builds symbol table and outputs it to file. Allocates memory pool. Loads SYSINIT and unmapped portion of system debugger (if appropriate) at end of image.	Builds memory tables to reflect target system. Constructs partitions in shared memory table. Appends debug block and vector block at end of SYSINIT. Writes image and SYSINIT to file.

Table 4-2
SYSGEN Loading Sequence

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
//HARDWARE /PARAMETERS MACHINE = *type*		Sets C.MACH.			
IPU		Sets C.IPU in C.CONF.			
/MEMORY SIZE = *nn*, TYPE = *c*, CLASS = *x*, MULTI		Builds memory table prototype in scratch space. Parses memory types.	Sets C.MATA. Builds memory allocation table from prototype.		Allocates memory in target system. Builds MIDL.
/CHANNELS CONTROLLER = ttcc, PRIORITY = intlev, CLASS = class, HANDLER = name, MUX = type, SUBCH = *aa*, CACHE DEVICE = *aa*, DISK = devcode, SHR,DTC = *tt*, LINESIZ = *x*, PAGE = *y*, SPOOL = code, HANDLE = name, PHYSA =*ccaa*,OFF IOQ = mode,CACHE, QITD,DEAL START = start		Builds preliminary DTT, CDT, CHT and UDT as linked lists in scratch space. Adds handler to interrupt list. Sets C.CDTN, C.CHTN, and C.UDTN. Builds internal partition table in scratch space for each memory disk.	Builds DTT, CHT, CDT, and UDT for target system. Sets C.DTTA, C.CHTA, C.DTTN, C.CDTA, and C.UDTA. Reserves space for the GPMC jump table. Sets C.MIOP. Builds scratch pad entries.	Loads handler's object and initializes.	Strings CDTs. Uses partition table to initialize a shared memory table for each memory disk.

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/TRAPS					
PROGRAM = (name 1, ..., name 7)		Builds internal interrupt table in scratch space.	Builds load table with interrupt table. Builds scratch pad entries.	Loads program's object and initializes.	
USERPROG = (name 1, ...,name 7)		Builds internal interrupt table in scratch space.	Builds load table with interrupt table. Builds scratch pad entries.	Loads program's object and initializes.	
SYSTRAP = name 1, RETRAP = name 2		Replaces default trap name in trap table.		Loads program's object and initializes.	
/INTERRUPTS					
PRIORITY = intlev RTOM = (channel, subaddress), PROGRAM = name, INTV		Builds internal interrupt table in scratch space.	Builds load table with interrupt table. Builds scratch pad entries.	Loads program's object and initializes.	
/SYSDEVS					
SID = devmnc DENSITY = density PARITY = parity		Sets C.SIDV, C.SIDD, and C.SIDP.	Vertifies C.SIDV.		
LOD = devmnc,IBP		Sets C.LODC and C.SIBP.	Verifies C.LODC.		
POD = devmnc		Sets C.PODC.	Verifies C.PODC.		
SWP		This directive is ignored.			
SWAPDEV= {devmnc | IPLDEV}		Initializes C.SWPDEV and C.SWPRD for SH.SINIT			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/VP					
VP = (aa, number), PROGRAM = module 1		Builds preliminary VP UDT in linked list. Builds preliminary null device CDT if necessary.	Builds VP UDTs. Builds null device CDT if necessary.	Loads VP handler object and intitializes.	Strings CDT, if necessary.
VPID = *aa*, VPTYPE = *tt*, STARTBLK = *blk*, PRIORITY = intlev, INTRPT = (cc,ss), PROGRAM = module 2, IPCA = *ipsize*, BUS0 = *b0size*, BUS1 = *b1size*, BUS2 = *b2size*, BUS3 = *b3size*		Adds VP internal handler to interrupt table in SCR space. Specifies VP device specific information to be stored in preliminary UDT. Builds internal partition table entries for VP in scratch space.	Builds load table with interrupt table. Builds scratch pad enteries. Builds VP UDTs.		Uses partition table to initialize SMT and allocate more memory to the system.
//SOFTWARE /PARAMETERS					
BATCHMSG		Sets C.CONS if keyword NOCONS used Sets C.TERM if keyword NOTERM used			
EXTDMPX	Defaults C.MPXBRD to -2. Resets C.TSA	Sets C.MPXBRD. Sets C.TSA if keyword TSA is used	Builds load module table with extended MPX-32 modules.	Loads EXTDMPX object and non-EXTD object.	Verifies EXTDMPX end address.
DELTA = CC		Sets C.DELTA.		Loads H.EXEC2 and H.CPU2 into image.	

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
DISP = entries	Defaults C.NQUE to ten.	Sets C.NQUE.	Constructs dispatch queue and DQE address table, and links them. Sets C.ADAT and C.DQUE.		Sets C.SWAP
LOGON = MULTI [,NOSYS] SINGLE		Sets C.MLOGIN Sets C.NOSYS in C.BIT1			
POOL = words	Defaults C.POOL to 1000.	Sets C.POOL.		Builds memory pool. Sets C.SBUF.	
IOQPOOL = *n* [PERMIOQ, [NOROLL]] MSGPOOL =*n* [NOROLL]				Builds IOQ pool. Sets C.SBUFA. Builds MSG pool. Sets C.SBUFB.	
NTIM = number	Defaults C.NTIM to 60.	Sets C.NTIM.			
MTIM = number	Defaults C.MTIM to 60.	Sets C.MTIM.			
ITIM = microseconds	Defaults C.ITRS to 384 (38.4 micro-seconds).	Sets C.ITRS.	Used to recom-pute C.IDLC, C.TDQ1, C.TDQ2, and C.TDQ3.		
TSMEXIT NOTSMEXIT		sets C.TSMXIT resets C.TSMXIT			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
ITLB = intlev		Builds entry in internal interrupt table for H.ICP. Sets C.NITL	Initializes indirectly connected interrupt table. Sets C.ITLB. Builds scratch pad entries.	H.ICP loaded from object file and initialized.	
MMSG = *n*	Defaults C.MMSG to 5.	Sets C.MMSG.			
MRUN = *n*	Defaults C.MRUN to 5.	Sets C.MRUN.			
MNWI = *n*	Defaults C.MNWI to 5.	Sets C.MNWI.			
SYSTEM = sysfile		Sets C.SYSTEM.			Uses C.SYSTEM for name of target system file.
SYMTAB = filename		Sets C.SYMTAB.		Uses C.SYMTAB for name of symbol table file.	
MAPOUT NOMAPOUT		Sets C.TSKOUT Resets C.TSKOUT			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
TQFULL = time	Defaults C.IDCL to 26042, C.TDQ3 to 600 and C.TDQ2 to 400.	Sets C.TDQ3.	Recomputes C.TDQ3, C.TDQ2 and C.IDLC.		
TQMIN = time	Defaults C.IDCL to 26042, C.TDQ1 to 200 and C.TDQ2 to 400.	Sets C.TDQ1.	Recomputes C.TDQ2, C.TDQ1 and C.IDLC.		
BATCHPRI = *nn*	Defaults C.BPRI to 61.	Sets C.BPRI.			
TERMPRI = *nn*	Defaults C.TSMPRI to 60.	Sets C.TSMPRI.			
PATCH = number		Sets C.PATCH.	Sets C.MPAA, C.MPAC, C.MPAH, and zeros patch area.		
DEMAND= {NONE|*pp*}		Sets C.DPGPRI.		Vector location C4 points to H.IPPF if mapped out image and C.DPGSYS is set.	
NODEMAND		Resets C.DPGSYS.		If on 32/2000 system, C4 points to H.IP09.	
AGE=*xx*		Sets C.AGE.			
BEGPGOUT=*zz*		Sets C.BEGPGO.			
ENDPGOUT=*aa*		Sets C.ENDPGO.			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
HELP = (volname,dir)		Sets C.HLPVOL and C.HLPDIR.			
MODE = code		Sets C.SCBT, C.NOP, C.SPADOK, C.SIBP, C.SUFA and C.SIMM, C.RTACC as specified.			
SVC = number	Defaults C.SVTN to X'7F'.	Sets C.SVTN.	Sets C.SVTA and C.SVTA2 and zeros SVC tables.		
RMTSIZE = number	Defaults C.RMTL to 32 and C.RMTM to 64.	Sets C.RMTM.	Sets C.RMTA and zeros resource-mark table.		
FLTSIZE		This directive is ignored.			
ACTIVATE = (name1,...,name7)		Builds prototype activation table as linked list in scratch space. Sets C.ACTN.	Builds activation table and sets C.ACTA.		
TRACE = num	Defaults C.TRACE to X'FFFFFFFE'.	Sets C.TRACE.			
DEBUGTLC = cc	Defaults C.DBTLC to X'7E'.	Sets C.DBTLC.			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
PCHFILE = name		Sets C.PCHFLE.			
DBGFILE = name	Defaults C.DBGLM to 'AIDDB'.	Sets C.DBGLM.			
SEQUENCE = (name1,...name7)		Builds prototype sequence table as linked list in scratch space. Sets C.SEQN.	Builds sequence table and sets C.SEQA.		
DPTIMO = number	Initializes C.DPTIMO to zero.	Sets C.DPTIMO.			
DPTRY = num	Initializes C.DPTRY to zero.	Sets C.DPTRY.			
DTSAVE = time		Sets C.DTSAVE.			
SWAPSIZE = size		Sets C.SWAPSZ.			
SWAPLIM =*n*		Sets C.PDQE.			
/MODULES MODULE = (name, module,entpoints)	Defaults C.MODN to twelve.	Builds prototype module table in scratch space. Sets C.MODN.	Uses prototype module table to build load table. Sets C.MODD and zeros module address table.	Loads modules and intializes.	

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/OVERRIDE					
SYSMOD = name1 REPMOD = name2		Removes compatibility modules or replaces modules with names given in the system module defined table in SYSGEN.			
NOLACC		Sets C.NOLACC in C.BIT1			
NOANSI		Excludes support for ANSI labeled tapes from the system image.			
NOBASE		Removes base mode support.			
NOCMS		Excludes H.ALOC, H.FISE, H.MONS, and H.CALM from system image.			
CMIMM		Sets C.CMIMM.			
CMPMM		Sets C.CMPMM.			
NOTDEF		Excludes support for the TERMDEF Facility from the system image.			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/PARTITION					
NAME = name SIZE = *np* STRTPG = *sp* MAP = *pm*		Builds internal partition table in scratch space.			Uses partition table to initialized shared memory table and allocate more memory to the system.
OWNER = name PROJECT = name OTHER = name		Builds internal partition table in scratch space.			Uses partition table to initialized shared memory table and allocate more memory to the system.
/SECURITY					
OWNERNAME = NOECHO		Sets C.NOECHO			
PASSWORD		Sets C.PASSWD.			
SAPASSWD		Sets C.SAPSWD.			
SYSONLY		Sets C.SAONLY.			
/TABLES					
CDOTS = number		Sets parameters for user communication area.	Allocates user communication area.		
JOBS =number	Defaults C.JOBN to one.	Sets C.JOBN.			
MDT = *n*, BLOCK = *b*		Sets C.MDTN. Sets C.MDTA.			

Table 4-2
SYSGEN Loading Sequence - cont.

Directive	Actions Taken by SYSGEN				
	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
SHARE = number		Sets C.SMTN.	Zeros shared memory table. Sets C.SMTA and C.SMTS.		Initializes shared memory table.
TIMER = number		Sets C.TENT.	Zeros timer table. Sets C.TTAB.		
/RMSTABLS					
ARTSIZE = number	Defaults C.ARTN to 100.	Sets C.ARTN.	Allocates and zeros ART. Sets C.ARTA.		
/FILES					
SMD		This directive is ignored.			
SYCSIZE = blocks	Defaults C.SYCS to 32.	Sets C.SYCS to 32.			
SGOSIZE = blocks	Defaults C.SGOS to 32.	Sets C.SGOS to 32.			

4.2 SYSGEN Components

SYSGEN contains the device type table (DTT) definition, the device ID (DID) table definition, and a special scanner used to parse SYSGEN directives.

4.2.1 DID and DTT Definitions

The device identification table (DID) and the device type table (DTT) definitions are defined using the Macro Assembler FORM directive. SYSGEN fills in the DTT with information from the controller definition table (CDT), and vice-versa. The information used by SYSGEN is shown in section 4.2.1.1. This information is used by SYSGEN to build the table layout of the DTT described in Chapter 2.

The DID is an internal structure containing disk identification information. Its layout is shown in section 4.2.1.2. This information is used by SYSGEN to build unit definition table (UDT) entries during the SYSGEN process.

4.2.1.1 Device Type Table

```
*************************************************************************
*                          DEVICE TYPE TABLE                            *
*************************************************************************
            SPACE
            BOUND       1D
DTT.TBL     EQU         $
*
*             . . . . . . . . . . . . . . . . . . . . DEV. TYPE CODE
*             :     . . . . . . . . . . . . . . . . . CDT POINTER
*             :     :     . . . . . . . . . . . . . . # OF CDT'S
*             :     :     :     . . . . . . . . . . . FLAGS
*             :     :     :     :     . . . . . . . . DEVICE NAME
*             :     :     :     :     :   . . . . . . MAX BYTES/XFER
*             :     :     :     :     :   :     . . . ALIAS DTC
*         . . : . . : . . : . . : . . : . : . . : . . FREE BYTE
DTT FORM        8,  24,   8,    8,   16, 16,  8, 8
            SPACE
   DTT  X'00',A(00),X'00',X'00',C'CT', 4096,X'00',0 DUMMY CT
   DTT  X'01',A(00),X'00',X'41',C'DC',16384,X'00',0 ANY DISK EXCEPT
                                                    MEMORY DISK
   DTT  X'02',A(00),X'00',X'40',C'DM',16384,X'01',0 MOVING HEAD OR
                                                    MEMORY DISK
   DTT  X'03',A(00),X'00',X'40',C'DF',16384,X'01',0 FIXED HEAD DISK
   DTT  X'04',A(00),X'00',X'61',C'MT', 8192,X'00',0 ANY MAG.  TAPE
   DTT  X'05',A(00),X'00',X'60',C'M9', 8192,X'04',0 9-TRACK MAG.  TAPE
   DTT  X'07',A(00),X'00',X'01',C'CD', 0120,X'00',0 CARD DEVICE
   DTT  X'08',A(00),X'00',X'00',C'CR', 0120,X'07',0 CARD READER
   DTT  X'0A',A(00),X'00',X'00',C'LP', 0133,X'00',0 LINE PRINTER
   DTT  X'0B',A(00),X'00',X'00',C'PT', 4096,X'00',0 PAPER TAPE
   DTT  X'0C',A(00),X'00',X'00',C'TY', 4096,X'00',0 TELETYPE
   DTT  X'0D',A(00),X'00',X'01',C'CT', 4096,X'00',0 OPERATOR'S CONSOLE
   DTT  X'0E',A(00),X'00',X'60',C'FL',16384,X'00',0 FLOPPY DISK
   DTT  X'0F',A(00),X'00',X'00',C'NU',16384,X'00',0 NULL DEV
   DTT  X'10',A(00),X'00',X'00',C'CA', 4096,X'00',0 CA DEVICE
   DTT  X'11',A(00),X'00',X'00',C'U0', 0000,X'00',0 U0 DEVICE
   DTT  X'12',A(00),X'00',X'00',C'U1', 0000,C'00',0 U1 DEVICE
   DTT  X'13',A(00),X'00',X'00',C'U2', 0000,X'00',0 U2 DEVICE
   DTT  X'14',A(00),X'00',X'00',C'U3', 0000,X'00',0 U3 DEVICE
   DTT  X'15',A(00),X'00',X'00',C'U4', 0000,X'00',0 U4 DEVICE
   DTT  X'16',A(00),X'00',X'00',C'U5', 0000,X'00',0 U5 DEVICE
   DTT  X'17',A(00),X'00',X'00',C'U6', 0000,X'00',0 U6 DEVICE
   DTT  X'18',A(00),X'00',X'00',C'U7', 0000,X'00',0 U7 DEVICE
   DTT  X'19',A(00),X'00',X'00',C'U8', 0000,X'00',0 U8 DEVICE
   DTT  X'1A',A(00),X'00',X'00',C'U9', 0000,X'00',0 U9 DEVICE
   DTT  X'1B',A(00),X'00',X'00',C'LF', 0000,X'00',0 PRINTER/FLOPPY
*
```

4.2.1.2 Device ID Table

```
*************************************************************************
*                        DEVICE ID TABLE                                *
*************************************************************************
         SPACE
         BOUND        1W
DID. TBL EQU          $
*
*DEVICE ID NAME.....................................................:
*TOTAL SECTORS .....................................              :
*BIT MAP SIZE      ..............................      :         :
*NO.  OF HEADS     ........................     :      :         :
*SECTOR SIZE       ...................  :     :      :         :
*SECTORS/TRACK     ..............    :  :     :      :         :
*SECTORS/ALOC. UNIT...........   :    :  :     :      :         :
*SECTORS/BLOCK     ........  :   :    :  :     :      :         :
*OLD DEVICE ID NAME....   :  :   :    :  :     :      :         :
*                     :   :  :   :    :  :     :      :         :
*                .....:...:..:...:....:..:.....:......:.........:
DID   FORM           32, 8, 8,  8,   8, 16,   16,    32,     64
      SPACE
*     CLASS  'F' EXTENDED I/O DISK DEVICES
      DID     C'DF01', 3, 3, 26,  64,  2,     ,   4004, C'FL001 '
      DID     C'DF02', 1, 2, 20, 192,  5,  642,  41100, C'MH040 '
      DID     C'DF03', 1, 2, 20, 192,  5, 1286,  82300, C'MH080 '
      DID     C'DF04', 1, 4, 20, 192, 19, 2444, 312740, C'MH300 '
      DID     C'DF05', 1, 1, 20, 192,  4,  160,   5120, C'FH005 '
      DID     C'DF06', 1, 2, 20, 192,  1,  258,  16460, C'CD032 '
      DID     C'DF06', 1, 2, 20, 192,  1,  258,  16460, C'CD032 '
      DID     C'DF07', 1, 2, 20, 192,  1,  258,  16460, C'CD064 '
      DID     C'DF07', 1, 2, 20, 192,  3,  772,  49380, C'CD064 '
      DID     C'DF08', 1, 2, 20, 192,  1,  258,  16460, C'CD096 '
      DID     C'DF08', 1, 2, 20, 192,  5, 1286,  82360, C'CD096 '
      DID     C'DF09', 1,10, 20, 192, 40, 2635, 674400, C'MH600 '
      DID     C'DF0A', 1,10, 20, 192, 40, 2635, 674400, C'FM600 '
      DID     C'DF0A', 1, 1, 20, 192, 96,   60,   1920, C'FM600 '
      DID     C'DF0B', 1, 4, 20, 192, 10, 1286, 164600, C'MH160 '
      DID     C'DF0C', 1, 2, 20, 192,  5, 1286,     00, C'ANY   '
      DID     C'DF0D', 1, 4, 20, 192, 24, 2670, 341280, C'MH340 '
*
```

	0 7	8 15	16 23	24 31
Word 0	Old device ID name (DID.ONAM)			
1	Sectors per block (DID.SPB)	Sectors per allocation (DID.SPAU)	Sectors per unit track (DID.SPT)	Sector size (DID.SSIZ)
2	Number of heads (DID.NHDS)		Bit map size (DID.MSIZ)	
3	Total sectors (DID.SECS)			
4	Device ID name (5 bytes) (DID.NAME)			
5	5th byte of DID.NAME	Reserved		

4.2.2 SYSGEN Scanner

The SYSGEN scanner parses SYSGEN directives by utilizing linked information tables built by calling system macros. The system macros SECTION, SUBSECT, DIRTV, KEYWD, and PARAM set up tables as shown in section 4.2.2.1. SECTION and SUBSECT correspond to SYSGEN directive sections and subsections. KEYWD and PARAM correspond to the keyword and parameter elements of each directive - DIRTV. See the MPX-32 Reference Manual Volume III, Chapter 7 for directive descriptions.

The action addresses in section 4.2.2.1 are addresses within the SYSGEN program where action should be transferred when the scanner encounters that directive, keyword or parameter type. The SDINIT macro must be called prior to setting up the information tables. It assigns the equates for the parameter type tables. Finally, the macro KWEND must be called after the set-up is complete to specify the end of the tables.

Entry Conditions

Calling Sequence

BL SCANNER

Registers

R1 address of directive definition list; a byte address on a word boundary

R2 address of directive to scan; a byte address on a word boundary

R3 negative length of directive in bytes

Exit Conditions

Return Sequence

TRSW R0 (CC1 = 1 if error detected)
(CC2 = 1 if terminating section directive was encountered. For example, section definition has null subsection link)

Registers

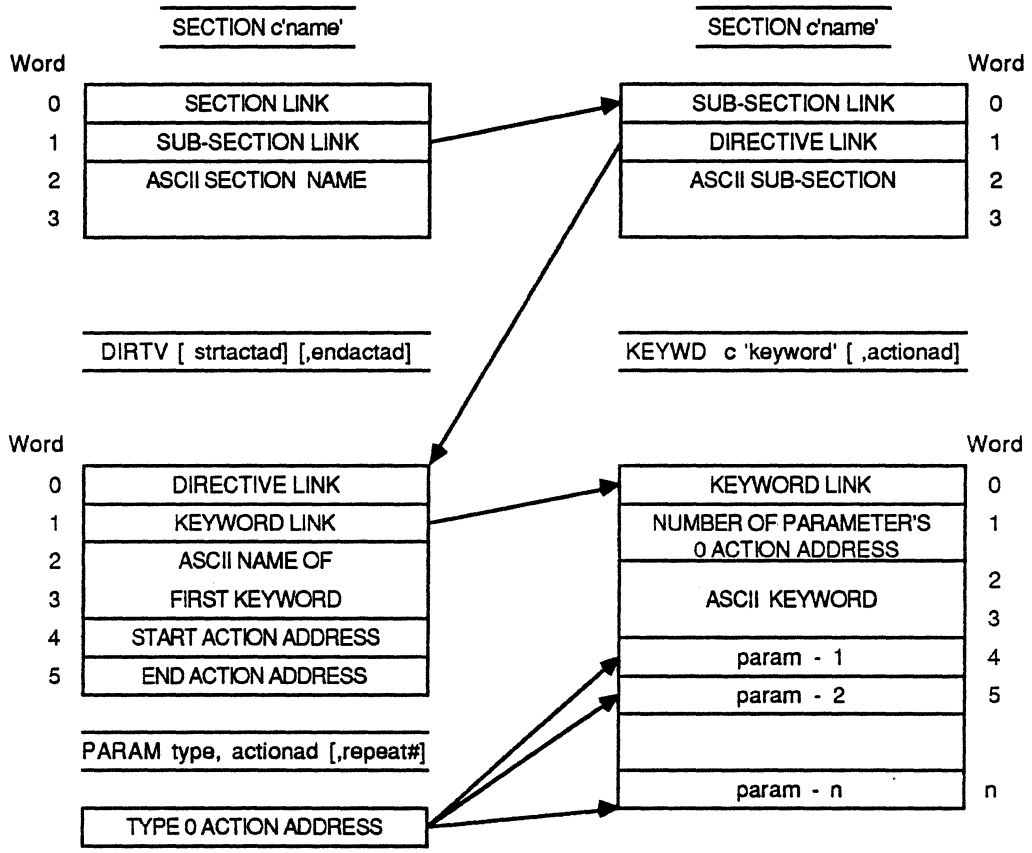
R1 error message TCW if error detected
R2 current directive pointer
R3 negative length of remaining directive

Action Routine Linkage:

Inputs: CC1 = 0
R0 = return address
R2 = byte address of item
R4 = length of item, in bytes
R5 = last character scanned
R6 = first four bytes of string
R7 = second four bytes of string
(or)
R7 = converted decimal number
(or)
R7 = converted hexadecimal number

Outputs: CC1 = 1 if error detected by action routine.

4.2.2.1 Directive Definition List



Type label	Internal value	Description
G 'a'		ASCII Character
DIGIT	EQU X'84'	digit (0-9)
ALPHA	EQU X'88'	alphabetic (A-Z)
SPECL	EQU X'8C'	special (not 0-9 or A-Z)
ANYS	EQU X'90'	anything (X'00' - X'FF')
STRING	EQU X'94'	alphanumeric string
SYMBOL	EQU X'98'	symbol string
DNUMB	EQU X'9C'	decimal number
HNUMB	EQU X'A0'	hexadecimal number

4.3 Table Building

4.3.1 System Tables

The main function of SYSGEN is building the tables used by an MPX-32 system. Utilizing the supplied directives, SYSGEN tailors the tables for the installation required. Some of the system tables which SYSGEN builds are first formed as linked lists in SYSGEN's scratch space and are later inserted in the target file after all pertinent information has been collected. Section 4.3.1.1 provides a list of all the tables with which SYSGEN interfaces, and where information about them can be found.

4.3.1.1 Tables Referenced in SYSGEN

Name of Table	SYSGEN Interaction	Where Documented
Activation Table	Built by SYSGEN	Ref. Man., Vol. III, Ch. 7
Allocated Resource Table - ART	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 2
Channel Definition Table - CHT	Partially built by SYSGEN	Tech. Man., Vol. I, Ch. 2
Controller Definiton Table - CDT	Built by SYSGEN	Tech. Man., Vol. I, Ch. 2
Device Identification Table - DID	Used by SYSGEN	Tech. Man., Vol. I, Ch. 4
Device Type Table - DTT	Used and filled in by SYSGEN	Tech. Man., Vol. I, Ch. 4
DQE Address Table - DAT	Built by SYSGEN	Tech. Man., Vol. I, Ch. 2
DQE Table	Allocated, zeroed and linked by SYSGEN	Tech. Man., Vol. I, Ch. 2
GPMC Jump Table	Allocated and zeroed by SYSGEN	See H.MUX0
Indirectly Connected Task Linkage Table - ITLT	Allocated and initialized by SYSGEN	Tech. Man., Vol. I, Ch. 1
Map Tables	MPL, MSD & MIDL built by SYSGEN	Tech. Man., Vol. I, Ch. 2
Memory Allocation Table	Built by SYSGEN	Tech. Man., Vol. I, Ch. 2
Memory Pool	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 2
Memory Resident Descriptor Table	Allocated and initialized by SYSGEN	Tech. Man., Vol. I, Ch. 2
Module Table	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 1
Module Address Table	Built by SYSGEN	Tech. Man., Vol. I, Ch. 2

Name of Table	SYSGEN Interaction	Where Documented
Mounted Volume Table - MVT	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 2
Patch Area	Allocated and zeroed by SYSGEN	Ref. Man., Vol. III, Ch. 9
Resourcemark Table - RMT	Allocated and zeroed by SYSGEN	Ref. Man., Vol. I, Ch. 2
Scratch Pad	Partially built by SYSGEN	32/70 Tech. Man.
Sequence Table	Built by SYSGEN	Ref. Man., Vol. III, Ch. 7
Shared Memory Table - SMT	Allocated and initialized by SYSGEN	Tech. Man., Vol. I, Ch. 2
SVC1 Table	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 1
SVC2 Table	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 1
Timer Table	Allocated and zeroed by SYSGEN	Tech. Man., Vol. I, Ch. 2
Unit Definition Table - UDT	Built by SYSGEN	Tech. Man., Vol. I, Ch. 2

4.3.2 Internal Tables

As SYSGEN processes directive input, it collects information in its internal tables for utilization later in the task. It has three tables that are limited to internal use: the partition table, the module table and the interrupt/trap table. See section 4.3.2.1.

The partition table is built with information provided from the partition directives in phase one (S.PH01). Later, it initializes the shared memory table in phase four (S.PH04). SYSGEN's internal module table, not to be confused with the system module table, is a table of all system and user modules that are to be loaded in the target file. It is built in phase one and utilized in phase two (S.PH02) to build the load map table. The interrupt/trap table also builds the load map table and creates interrupt entries in the scratch pad in phase two.

4.3.2.1 SYSGEN Internal Tables

Partition Table

	0 7 8	15 16 23 24 31
Word 0	String forward address (PAR.LINK)	
1	Start logical page number (PAR.SPG)	Number of pages (PAR.NPG)
2	Physical map block number (PAR.PBN). See Note 1.	Reserved
3	Reserved	
4	Partition name (PAR.NAME)	
5		

Module Table

	0 7 8	15 16 23 24 31
Word 0	String forward address (MOD.LINK)	
1	Module number (MOD.NO)	Number of entry points (MOD.NEPT)
2	Module name (MOD.NAME)	
3		
4	Reserved	
5	Reserved	Module type (MOD.LTYP). See Note 2.

Interrupt/Trap Table

	0 7 8	15 16	23 24 31	
Word 0	String forward address (INT.LINK)			
1	Interrupt type (INT.TYP). See Note 3.	Priority level (INT.L1)	Controller class (INT.CLS) See Note 4.	Reentrant descriptor (INT.REEN)
2	Interrupt handler name (INT.NAME)			
3				
4	Pointer to device handler list (INT.HNDA)			
5	Device type code (INT.DTC)	Channel number (INT.CHAN)	Subaddress (INT.SUBA)	Module type (INT.LTYP). See Note 2.

Notes:

1. Values for PAR.PBN for a memory disk only are assigned as follows:

Value	Meaning
-1	User did not specify a starting map block number. Bit UDT.MDST of UDT.STA2 in the corresponding UDT is reset.
-ve	User specified the starting map block number. This is the negative of that value. Bit UDT.MDST of UDT.STA2 is set. During phase four (S.PH04), this value is transferred to SMT.PAGE. It is then interpreted by SYSINIT or OPCOM when the position of the memory disk is determined.

2. Values for MOD.LTYP and INT.LTYP are assigned as follows:

Value	Meaning
1	system module (LTYP.SYS)
2	user module (LTYP.USR)
3	I/O handler module (LTYP.IO)
4	non-I/O interrupt handler on trap (LTYP.INT)

3. Values for INT.TYP are assigned as follows:

Value	Meaning
1	indirect (TYP.NDIR)
2	direct (TYP.DIR)
3	service interrupt (TYP.SI)
4	GPMC service interrupt (TYP.GPMC)
5	extended I/O service interrupt (TYP.XIO)

4. Values for INT.CLS are assigned as follows:

Value	Meaning
3	RTOM interval timer (CLS.RTOM)
D	TCW class with extended addressing capability (CLS.TCWB)
F	extended I/O (CLS.XIO)

4.4 Handler and Module Loading and Initialization

In phase three (S.PH03) of the SYSGEN task, the system modules, user modules, interrupt handlers and trap handlers are all loaded and initialized for the target file. SYSGEN reads the object input file (OBJ) and scans the load table built in phase two for a match on the program name in the binary object record file. When a match is found, the module is copied to a temporary disk file and subsequently loaded as often as it appears in the load table. If the object module does not match one entered in the load table, it is skipped.

The last entry point of all modules and handlers is reserved for SYSGEN initialization, and provides the capability of self-initialization. SYSGEN calculates the address of the last entry point of each module and does a branch and link to that location, thus initializing the module. On return from the initialization, by the macro M.XIR (see Chapter 1), the current entry in the load table is cleared, the current address pointer gets updated, and the initialization code is zeroed and overlayed with the next module.

4.5 SYSGEN Load Map Descriptions

Revisions of MPX-32 later than 3.3 have a variety of characters in parentheses located after DEF'ed locations in the SYSGEN load map. The following table describes the meaning of those characters.

Entry	Description
(N)	the address is within a nonbase mode module
(B)	the address is within a base mode (extended) module
(H)	the address is within the HAT of either a nonbase or base mode module
(A)	the location is absolute, usually defining a version number in an older (compatibility) module
(?)	the location is not an address, therefore, neither nonbase nor base mode applies
blank	the address is a normal relative one

These descriptions may occasionally appear together, e. g., (NH), in which case the meaning is derived by combining the two descriptions (e. g., nonbase mode module address in the HAT of the module).

4.6 Special Considerations

4.6.1 MAPTGT/MAPHOST Routines

SYSGEN initially obtains 16KW of extended memory in which to build the target system and acquires additional map blocks as needed. By using the internal routine MAPTGT, it can map its acquired memory to address zero, replacing the host operating system. This enables SYSGEN to use communication region equates as references within the target system instead of within the host. When it becomes necessary for SYSGEN to use host system variables or services, it utilizes the internal routine MAPHOST to put the host system back in place.

4.6.2 Special Case Activation

SYSGEN is a 16KW task. In order to allow SYSGEN to build target systems as large as 44 map blocks, the system allocator loads it at address X'60000', which is the 49th map block. This gives SYSGEN a maximum of 48 map blocks in which to map the target system and to obtain extra memory in nonextended address space for building system tables.

4.6.3 SYSINIT Loading

When SYSGEN has completed building the target image, the load table, and the symbol table, it then loads the object of the SYSINIT task, which directly follows H.SWAPR's object as the last object file on OBJ if the system debugger is not configured. When the system debugger is requested by the USERPROG directive, the unmapped portion of the debugger is loaded on the next page boundary immediately following H.SINIT. When SYSGEN's final output file (see Figure 4-1) is subsequently booted, control is transferred to the SYSINIT task which starts up the image and then exits. When present, the unmapped portion of the system debugger remains memory resident with 6KW of physical memory allocated to it. It is not included as part of the system map and does not increase the size of the logical address space occupied by MPX-32. See MPX-32 Reference Manual Volume III, Chapter 2.

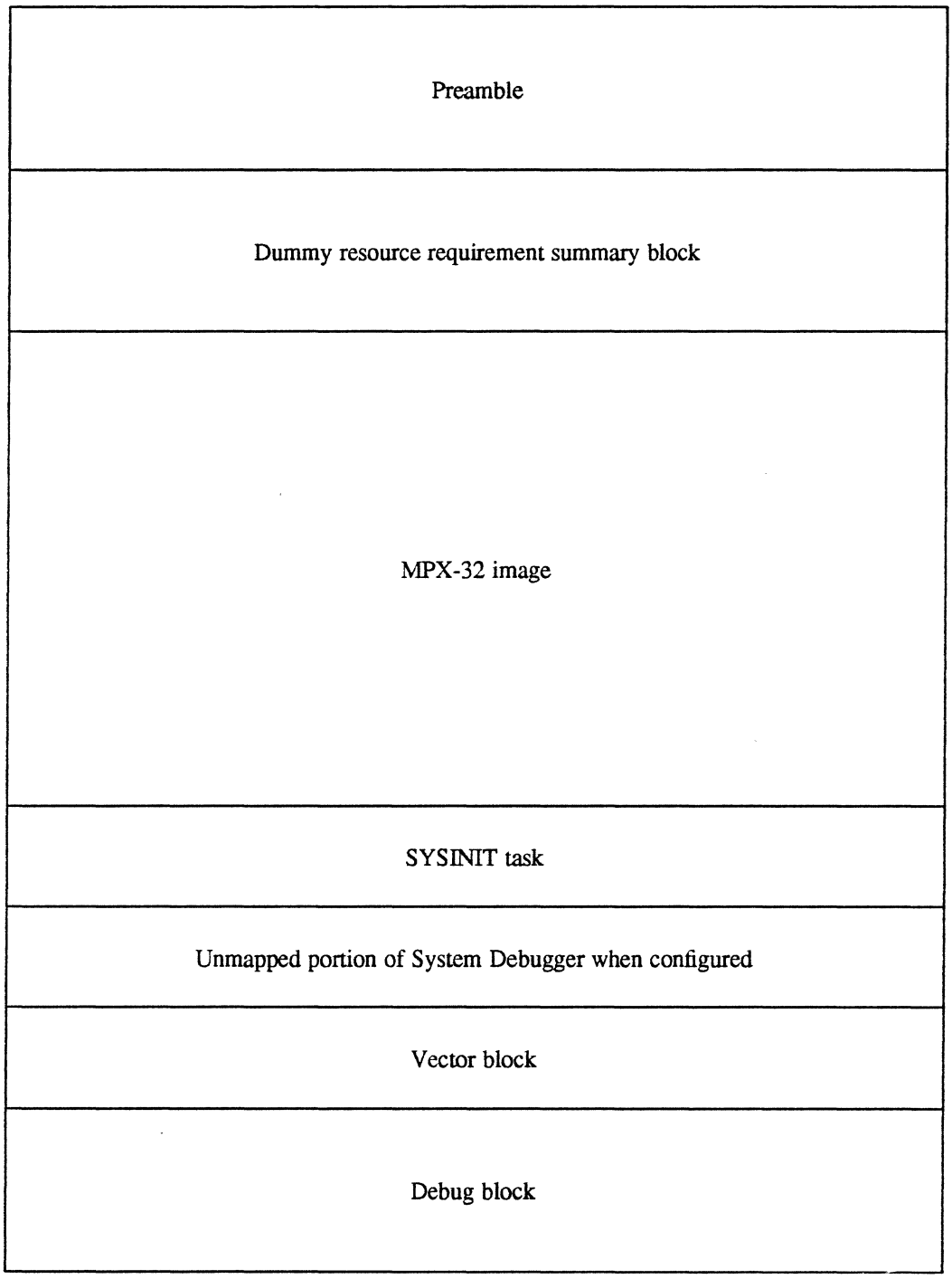


Figure 4-1
SYSGEN Output File Format

5 Batch Task Descriptions

5.1 Cataloger

5.1.1 Introduction

The cataloger builds load modules from an object code file assigned to LFC SGO. External references are resolved from a user specified subroutine library assigned to LFCs DIR and LIB and from the system subroutine library assigned to LFCs LID and LIS. The catalog directives are input from LFC SYC and the load module map is output to LFC SLO. The load module is a permanent file created by the cataloger; its file name is the program name supplied in the CATALOG or BUILD directive.

5.1.1.1 Exit Conditions

Normal Exit:	SVC 1,X'55'
Abnormal Exits:	SVC 1,X'57' Abort

Abort cases are described in the MPX-32 Utilities Reference Manual.

5.1.2 Processing Regions

The cataloger consists of four processing regions. Each is identified by a letter:

- X - external
- M - main
- C - control card interpretation and first object code pass
- B - second object code pass

Program tags, subroutine names, and names of variables begin with the letter of the region that they are associated with.

5.1.2.1 X Region

The X region contains subroutines relating to MPX-32 provided services.

5.1.2.2 M Region

The M region contains subroutines, variables, and tables which are referenced by more than one region. It also contains the entry point called by MPX in response to the $EXECUTE CATALOG job control statement. When the entry point is called, the limits of the general table area are established and control is transferred to the C region. The general table area occupies the free memory allocated to the cataloger following the cataloger program logic. The utilization of this area is depicted in Figure 5-1.

5.1.2.3 C Region

The C region interprets cataloger directives and makes the first pass over the object code comprising each segment being cataloged. Information about each program element, its external definitions and common blocks is extracted and stored in the symbol table (SYMTAB). SYMTAB is built from the high memory end of the general table area toward low memory. SYMTAB data restored with the SYMTAB directive is stored first in the table. SYMTAB entry formats are presented in Figure 5-1. The first entry for each segment is the control entry. The control entry is followed by a program name entry.

For each segment, a table of external references from EXCLUDE directives is built from the low memory end of the general table area. This table is followed by a table of undefined external references. Names from INCLUDE directives are placed in the undefined external references table. The table also contains any unsatisfied external references encountered during the processing of a segment. If unsatisfied externals exist after all program elements have been processed from the SGO file for a segment, the subroutine libraries are searched for the externals. Program elements that satisfy external references are selected from the libraries. Any remaining undefined external references are ignored since they may be satisfied by segments that are subsequently processed.

If the first pass over the object code for all segments is successful, SYMTAB addresses are made module relative, and control is transferred to the B region.

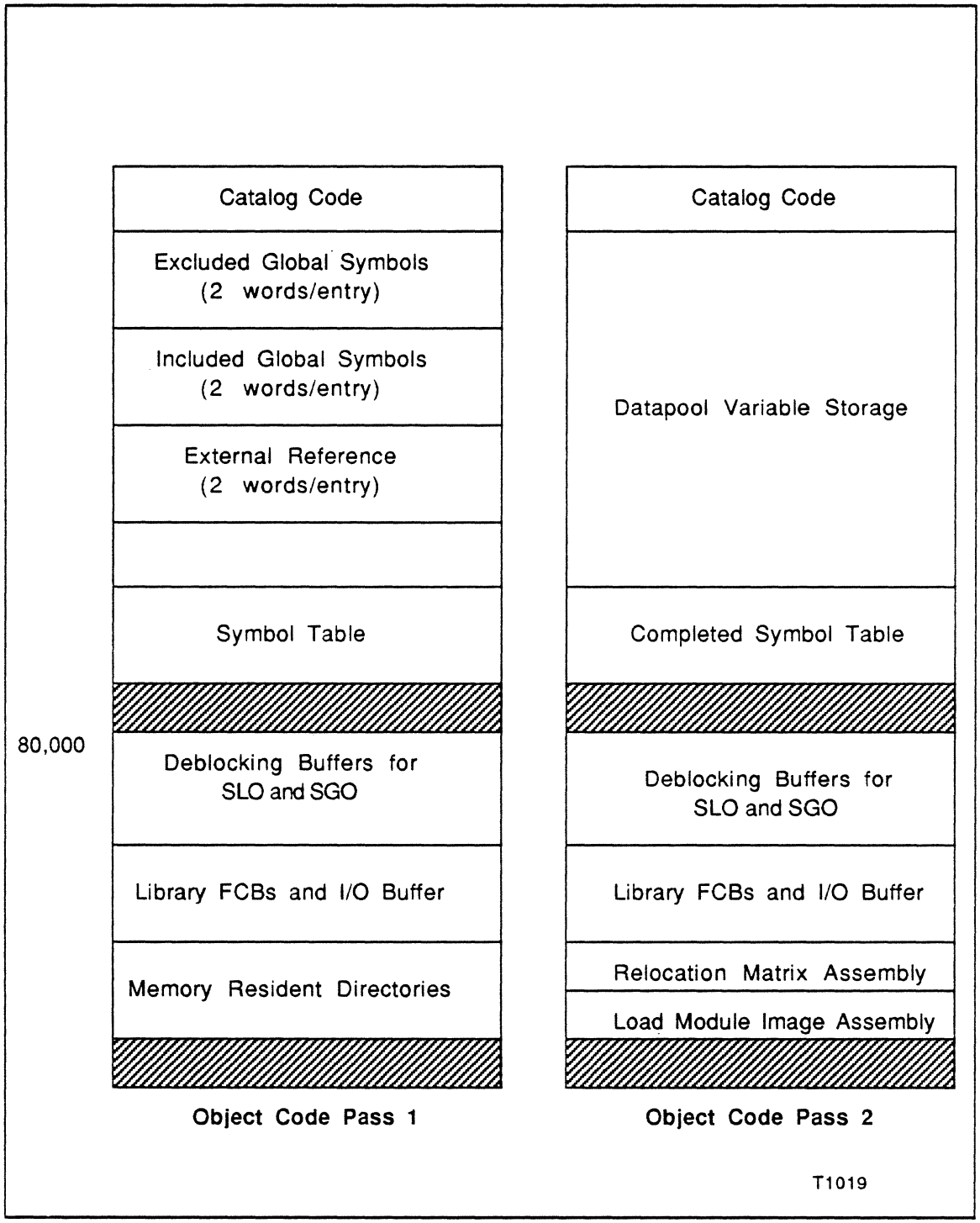


Figure 5-1
General Table Area

5.1.3 SYMTAB Entries

5.1.3.1 Linkback Entries

	0 – 5	6 – 7	8 – 15	16 – 31
Word 0	000000	00	Overlay level	Sequence Number
1-3	Must be zeros			

If the Linkback directive is used, the identities of specified overlay segments are saved in entries built toward low memory in four-word blocks.

5.1.3.2 Segment (Module) Entry

	0 – 5	6 – 7	8 – 15	16 – 31
Word 0	ID 100000	Flags See Note 1.	Option flags. See Note 2.	Number of linkback entries
1	000000	00	Overlay level (Main is zero)	Sequence number (Main is zero)
2-3	Left-justified ASCII segment name			
4	000000	00	Transfer address	
5			Origin of segment TSA relative. See Note 3.	
6	Reserved			
7	Last (END) address of segment			

Notes:

1. Flags are as follows:

Flag	Description
1x	CATALOG directives for the overlay were preceded by an ORIGIN or LORIGIN directive
x1	segment has a transfer address

2. Option flags are as follows:

Flag	Description
xxxxxxx1	suppress printing of the module map
xxxx1xxx	suppress output of load module to disk file
xx1xxxxx	output segment's SYMTAB

3. Bit 0 of the field is set if the origin is the end of a specified segment.

5.1.3.3 Defined Entry Point

	0 5	6 7	8 15	16 31
Word 0	ID 010000	00	Reserved	
1	Section number		Module relative address	
2-3	Left-justified ASCII name			

5.1.3.4 Common Entry

	0 5	6 7	8 15	16 31
Word 0	ID 001000 See Note 1.	Flags See Note 2.	Size in bytes (zero if allocated in another element)	
1	Block number		Module relative address. See Note 3.	
2-3	Left-justified ASCII name			

Notes:

1. The ID is 001100 for the first common entry for a given global block.
2. The flags are as follows:

Flag	Description
11*	common block is program initialized in this element
x1	common block is Cataloger allocated in this element

* Catalog always allocates initialized common in the program that initalizes it.

3. If allocated in another element, the address of the symbol table entry where the common is allocated.

5.1.3.5 Section Entry

	0 5	6 7	8 15	16 31
Word 0	ID 000100	00	Section size in bytes	
1	Section number		Section origin	
2-3	Left-justified ASCII name			

5.1.3.6 Program Name

	0 5	6 7	8 15	16 31
Word 0	ID 000010	Flags See Note 1.	Subroutine library logical record number	Subroutine library block number containing first record
1	Subroutine library file index	See Note 2.	Module transfer address if contained in this element	
2-3	Left-justified ASCII name			

Notes:

1. Flags are as follows:

Flag	Description
1x	word 1 of entry contains module transfer address
x1	program is from a subroutine library

2. If bit 7 is set, the transfer address is in the CSECT; otherwise, the transfer address is in the DSECT.

5.1.3.7 Control Entry

	0 . . . 5	6 . . . 7	8 . . . 15	16 . . . 31
Word 0	ID 000001	Flags See Note 1.	Control flags	Total of SYMTAB entries for this element
1	Maximum bound required in bytes	Number of bytes (mod 4) allocated this element for common and bounding		
2	Number of bytes (mod 4) of object code in this element			
3	Module relative address of this element, i.e., the address of its common if any			

Notes:

1. The flags are as follows:

Flag	Description
1x	program element is from a subroutine library
x1	element is last in segment

2. If section code, bit 8 is set.

5.1.3.8 B Region

The B region makes a second pass over the object code and outputs the cataloged segments in load module format. Absolute overlays are output in absolute format.

At the beginning of the B region, the space in the general table area not occupied by SYMTAB is partitioned. An area large enough to build the core image of the largest program element in the segments being cataloged is reserved. An area for an associated relocation matrix is also reserved. Each bit in the matrix corresponds to a word in the program element data area, and is set to one if the word contains relative data. The remaining space is allocated for a datapool table. The three-word datapool table entries contain the datapool item name in the first and second words and the item's address in the third word. During the second pass over the object code, the datapool item table is searched for each datapool reference that is encountered. If not found, an attempt is made to locate the item in the datapool dictionary. If the item is found, it is added to the datapool table. Items are added sequentially to the table. Wraparound occurs when table space is exhausted with new items replacing previously stored items.

An image of each program element comprising each segment is built in memory before being written to the segment's disk file (if not suppressed). A module map of a segment is printed before the next segment is formatted. After all segments are processed, request SYMTABs are output.

5.1.4 Load Module Structure

A load module consists of one or more program elements in a form that requires only load origin biasing at load time. A program element is the unit of program organization bounded by the macro assembler PROGRAM and END directives. Program elements include programs written by the user and any subroutines called from subroutine libraries. All program elements must be assembled in the relative mode.

The six load module segments are: preamble, resource requirement summary, CSECT data, CSECT relocation matrix, DSECT data, and DSECT relocation matrix. CSECT segments or DSECT segments can be omitted if they are empty. The resource requirement summary block is always present even if it is empty.

The load module preamble equates are described in Chapter 2.

Common blocks are allocated within a segment according to these rules:

- A common block is allocated preceding the program element that contains a common origin referencing the block.
- If a common block is not referenced by a common origin, it is allocated preceding the program element that defines the largest area.

Program areas, which are reserved but do not contain data and are not included in the module common delta, exist as words of zeros on disk. Therefore, those areas are initialized to zero when loaded into core. The module common delta is an area at the beginning of the module which occupies no space on disk and which is not initialized when loaded. It includes bounding and common which precedes the first program element and which precedes any common which is referenced by a common origin.

Each program element's assembled relative zero is placed on a doubleword boundary. Common blocks are placed on eight-word boundaries. The main segment of a module with overlays is placed on an eight-word boundary. If necessary, the size of the transient area is increased so that it consists of an integral number of eight-word units.

References can be made from an overlay segment to symbols contained within the main segment. These symbols can be contained in subroutines called from the subroutine libraries. The symbols are established using the assembler directives DEF and COMMON. Within the overlay, the assembler EXT and COMMON directives allow the symbols to be referenced. The main segment can reference symbols which are DEFs in overlay segments as main overlays reference symbols in lower level overlays. Other linkage depends on the use of the LINKBACK directive.

5.1.5 Symbol Table Output Format

The symbol table (SYMTAB) is output by the cataloger in a format similar to object records output by the assembler. Each record format is:

Byte 0	1	2,3	4,5	6
FF	Byte count*	Checksum	Sequence Number	Data blocks

*Number of bytes in data block on card

Each data block is preceded by a control byte in the form XXXXNNNN. XXXX identifies the data block type and NNNN specifies the number of bytes of data in the block. If NNNN is zero, the number of bytes is sixteen. Data block types and their contents are as follows:

Type(Hex)	No. Bytes Data	Contents
0	16	symbol table data output as four-word entries from high to low memory.
5	1 to 8	name of the segment during whose cataloging the SYMTAB was output
F	1	none — signals end of output.

A type five block is output first. Type zero blocks are output next and are terminated by a type F block. All cards, except the last, contain six data blocks. The last card can contain up to seven blocks (six data and one end).

5.1.6 Object Language

The object code is the output of the language processors and is the primary input to the cataloger. It describes the contents to be placed into the load module when the object module is included into the cataloger's input stream.

5.1.6.1 Object Module Records

The object module consists of one or more variable length records up to 120 bytes in length. Each record contains six bytes of header information describing the object record.

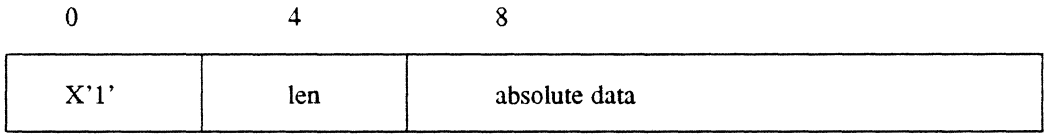
Byte	Field label	Contents
0	record type	X'FF' or X'DF'. A value of X'DF' indicates the last record of the current object module.
1	byte count	number of data bytes in the current record. It ranges from 2 to 114 (X'2', to X'72') and does not include the six bytes of header information.
2,3	checksum	checksum of the data bytes within the record. It is computed by adding the data bytes and truncating the sum to a halfword.
4,5	sequence	sequence number of the current object record. The initial value is one. If the field overflows, the count is reset to one.

5.1.7 Object Commands

The data portion of the object records consists of a series of object commands. Each object command is described by a control byte. The control byte is the first byte (byte zero) of each object command and contains two fields, the function code and the byte count. The function code is the left four bits and ranges from X'0' to X'F'. The byte count is the right four bits and is the count of data bytes for each command, exclusive of the control byte. A byte count of zero is interpreted as X'10' data bytes.

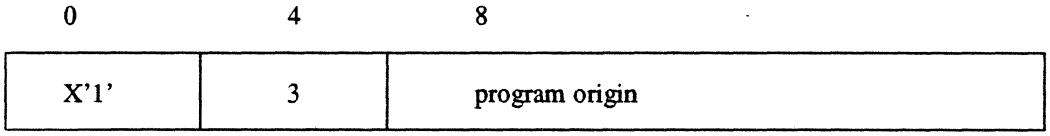
A stringback is a linked list of data that is terminated by a zero address. The word containing the zero address is absolute rather than relocatable. The cataloger makes that word relocatable if necessary. The addresses in the list are module relative and are nineteen-bit word addresses. The cataloger preserves bits 30 and 31 in stringing back the data.

5.1.7.1 Absolute Data



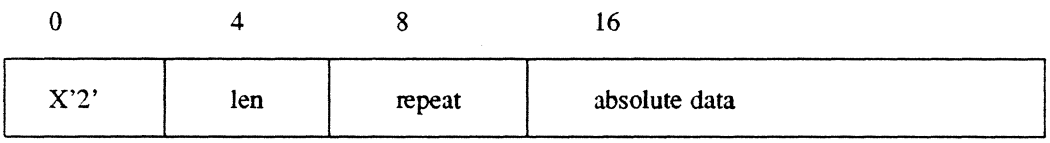
len — is the number of bytes of absolute data, 1 to 16. A value of zero is equivalent to a value of sixteen.

5.1.7.2 Program Origin



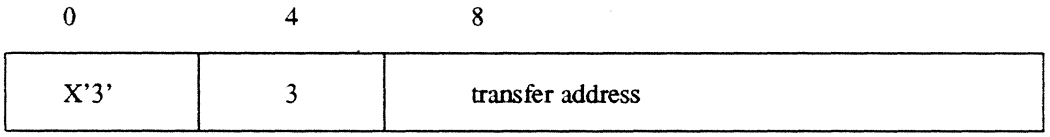
program origin	is a right-justified three-byte field containing a 19-bit origin
bit 8	is set if the address is relocatable

5.1.7.3 Absolute Data Repeat



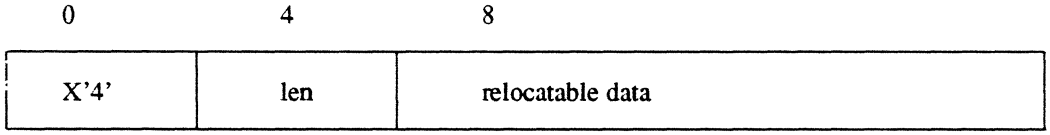
len	is the number of bytes in the command
repeat	is the number of times to repeat data, 1 to 255. A value of zero is equivalent to a value of one.

5.1.7.4 Transfer Address



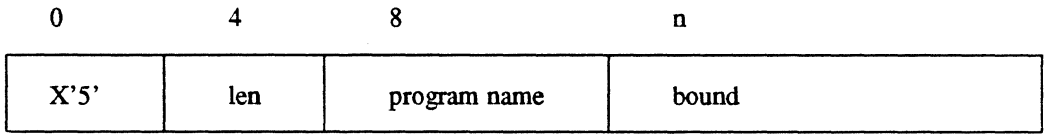
transfer address	is a right-justified three-byte field containing a 19-bit transfer address. Bit zero of the field must be set.

5.1.7.5 Relocatable Data



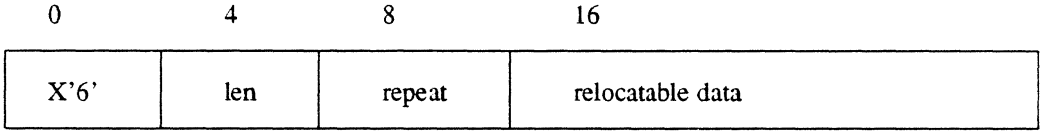
len	is the number of bytes of relocatable data, 4 to 16. A value of zero is equivalent to a value of sixteen. Len must be a multiple of four.

5.1.7.6 Program Name



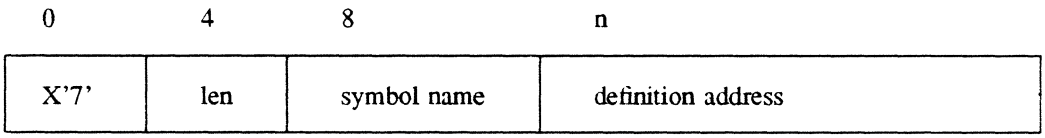
len	is the number of bytes in the program name plus three
program name	is the 1- to 8-character program name
bound	is a 3-byte field containing the minimum bounding requirement for the program. The minimum value is X'8' maximum value is X'20' or eight words.

5.1.7.7 Relocatable Data Repeat



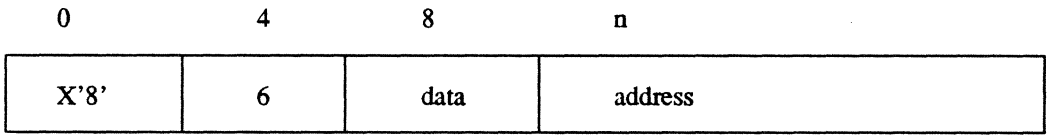
len	is the number of bytes of relocatable data (4, 8, or 12) plus one
repeat	is the number of times to repeat data, 1 to 255. A value of zero is equivalent to a value of one.

5.1.7.8 External Definition



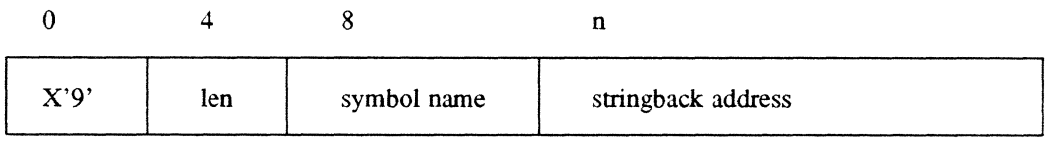
len	is the number of bytes in the symbol name plus three
symbol name	is the 1- to 8-character name of the symbol being defined
definition address	is a 3-byte field containing a right-justified 19-bit address. Bit 0 of the field is set if the address is relocatable.

5.1.7.9 Forward Reference



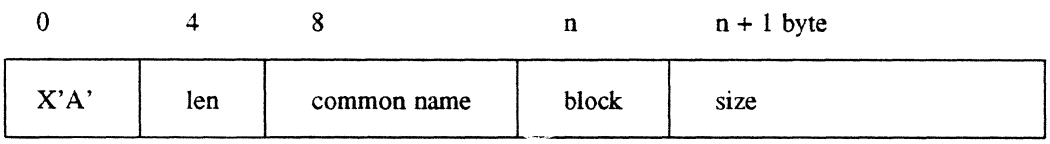
data	is a 3-byte field containing a right-justified 19-bit address. Bit zero of the field is set if the address is relocatable.
address	is a 3-byte field containing the right-justified 19-bit address of the stringback list. The contents of the data field are put into each word in the list. List is terminated by absolute zero link. Bit zero of the field must be set.

5.1.7.10 External Reference



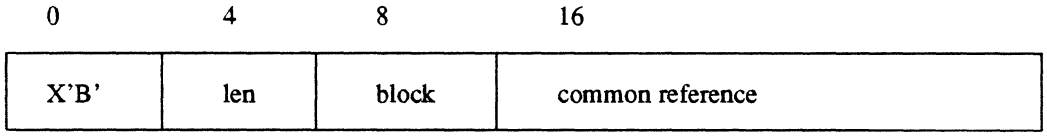
len	is the number of bytes in the symbol name plus three
symbol name	is the 1- to 8-character name of the symbol being referenced
stringback address	is a 3-byte field containing the 19-bit address of the stringback list. External address replaces the low order 19 bits in each word of the list. List is terminated by the absolute zero link. Bit zero of the field is set if the address is relocatable.

5.1.7.11 Common Definition



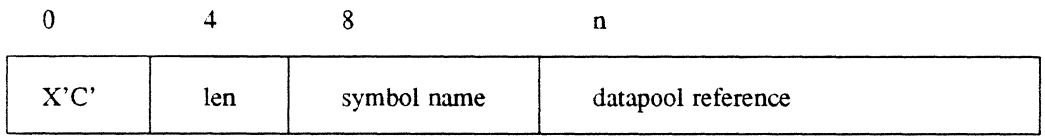
len	is the number of bytes in the common name plus three
common name	is the 1- to 8-character name of common
block	is a 1-byte block number assigned by the compiler
size	is a 2-byte size of common in bytes

5.1.7.12 Common Reference



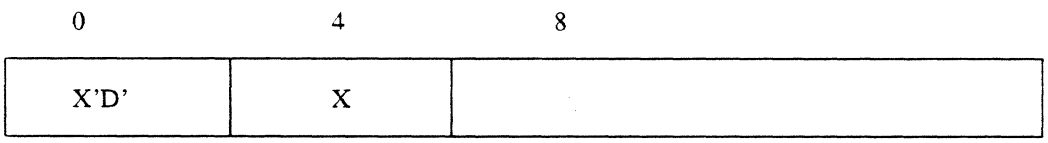
len	is the number of bytes in the common reference plus one
block	is a one-byte common block number referenced by data
common reference	is 4, 8, or 12 bytes of data that reference the common block. The base address of the common block is added to the low order 19 bits of each reference word.

5.1.7.13 Datapool Reference



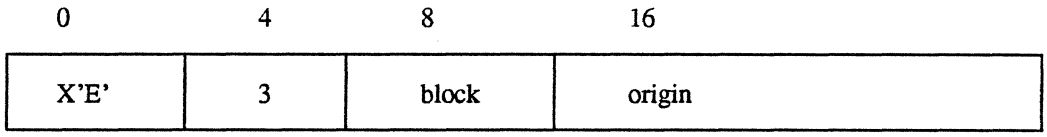
len	is the number of bytes in the symbol name plus four
symbol name	is the 1- to 8-character name of the symbol in datapool
datapool reference	is a 4-byte datapool reference. Symbol's value is added to the low order 19 bits of the datapool reference.

5.1.7.14 Escape to Extended Functions



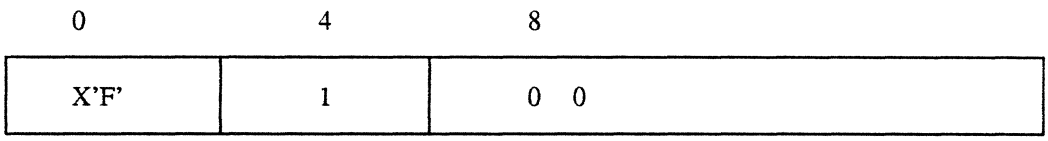
Function code of X'D' indicates extended item. (See section 5.1.8).

5.1.7.15 Common Origin



block	is a 1-byte common block number
origin	is a 2-byte offset from beginning of common block

5.1.7.16 Object Termination



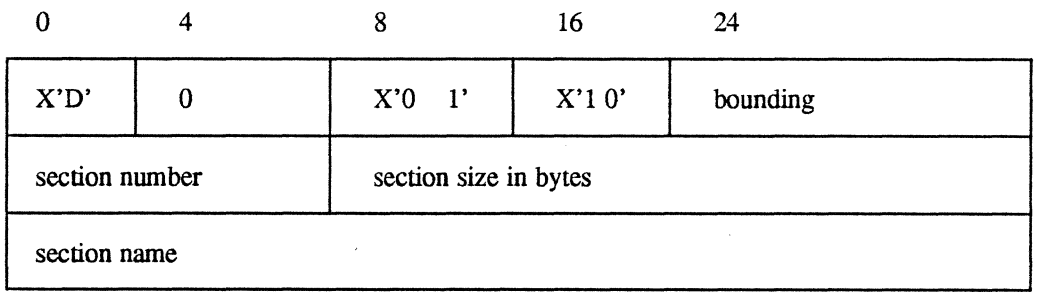
This record terminates the object code for the current module.

5.1.8 Extended Object Commands

The extended object commands differ from object commands in the following ways:

- Byte 1 contains the function code ranging from hexadecimal 1 to B.
- Byte 2 contains the length of the item including overhead bytes 0 to 3.

5.1.8.1 Section Definition



bounding	is the section bounding requirement (X'8' to X'20')
section number	is a one-byte field containing zero if DSECT or one if CSECT
section name	is an eight-byte field containing **DSECT* or **CSECT*

5.1.8.2 Section Origin

0	4	8	16	24
X'D'	0	X'0 2'	X'0 8'	bounding
section number		origin		

bounding — is the section bounding requirement (X'8' to X'20')

origin — is the offset within the section to establish as the new origin

5.1.8.3 Section Relocatable Reference

0	4	8	16	24
X'D'	0	X'0 3'	len	0 0
section number		repeat count	4 to 248 bytes of relocatable data	

len — is the number of bytes in the command

section number — is the section number where the relocatable data references

repeat count — is the number of times to repeat the data. A value of zero is equivalent to a value of one.

relocatable data — is the data in multiples of four bytes whose right 19 bits are to be relocated by the section base

5.1.8.4 Section Transfer Address

0	4	8	16	24
X'D'	0	X'0 4'	X'0 8'	0 0
section number		transfer address		

transfer address — is the offset within the section that is the transfer address

5.1.8.5 Section External Definition

0	4	8	16	24
X'D'	0	X'0 5'	len	0 0
section number		definition address		
1- to 8-character symbol name				

len	is the number of bytes in the command
section number	is the section number where the symbol is defined
definition address	is the offset within that section

5.1.8.6 Section External Reference

0	4	8	16	24
X'D'	0	X'0 6'	len	0 0
section number		stringback address		
1- to 8-character symbol name				

len	is the number of bytes in the command
section number	is the section where the stringback list begins
stringback address	is the offset within that section
symbol name	is the global symbol referenced

Note: If the address is zero and bit zero of the address is set, a stringback is performed to address zero of the section.

5.1.8.7 Section Forward Reference

0	4	8	16	24
X'D'	0	X'0 7'	X'0 C'	0 0
section number		definition address		
section number		stringback address		

definition address — is the offset within the "section number" section where the symbol is defined

stringback address — is the offset within the "section number" section where the stringback list begins

5.1.8.8 Large Common Definition

0	4	8	16	24
X'D'	0	X'0 8'	len	0 0
common number		common size in bytes		
1- to 8-character common name				

len — is the length of the command

common number — is the identifier assigned by the compiler to the common block

common name — is the name of common block

5.1.8.9 Large Common Origin

0	4	8	16	24
X'D'	0	X'0 9'	X'0 8'	0 0
common number		common origin		

common number — is the number assigned by the compiler to the common block

common origin — is the offset from beginning of common

5.1.8.10 Large Common Reference

0	4	8	16	24
X'D'	0	X'0 A'	len	0 0
common number		repeat count	4 to 248 relocatable data	

len	is the length of the command
common number	is the identifier assigned by the compiler
repeat count	is the number of times to repeat the data. A value of zero is equivalent to a value of one.
relocatable data	is 4-byte multiples of data which have the address of the referenced block added to the low order 19 bits

5.1.8.11 Debugger Information

'0	4	8	16	24
X'D'	flg	X'0 B'	len	flag
type		address		
size		left-justified 8-character symbol name		

left-justified 8-character common name

len is the number of bytes in the command

Value	Meaning if Set
18	symbol is not in common
26	symbol is in common

flg is as follows:

Bit	Meaning if Set
4	symbol is in extended memory (address is that of a 24-bit pointer to the symbol)
5	symbol is a formal parameter (address is that of a pointer to the symbol)
6	symbol is in the common. The common name follows the symbol's name.
7	symbol is in the datapool

flag — is as follows:

Bit	Meaning if Set
6	address is absolute
7	symbol is in CSECT

type — is as follows:

Type	Description
0	integer*1
1	integer*2
2	integer*4
3	integer*8
4	real*4
5	real*8
6	complex*8
7	complex*16
8	bit logical
9	logical*1
10	logical*4
11	character
14	statement label
15	procedure

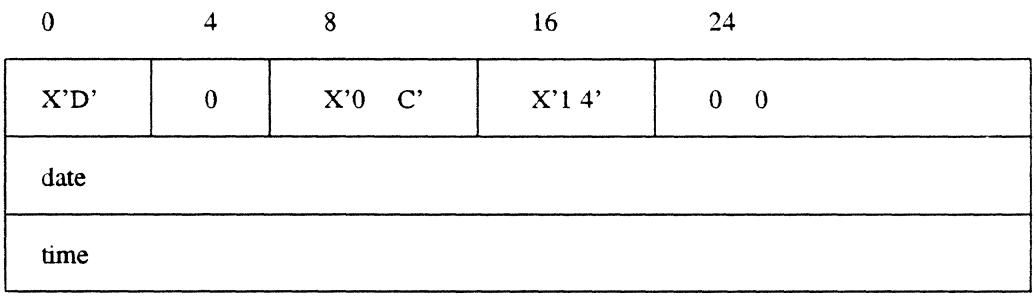
address — 23-bit address. Bits 9 to 28 indicate byte address, and bits 29 to 31 indicate bit within byte.

size — length of datum in bytes

symbol name — is an 8-character, left-justified, blank-filled variable name

common name — is an 8-character, left-justified, blank-filled common name

5.1.8.12 Object Creation Date/Time



date/time — is the day and time the object code was generated

date — is the 8-byte ASCII date (*mm/dd/yy*)

time — is the eight-byte ASCII time (*hh:mm:ss*)

5.1.8.13 Product Identification Information Leader

0	4	8	16	24
X'D'	0	0 C	len	0 0
product identification				

len	is the number of bytes in the command
product identification	is a user supplied string of up to 32 bytes of text identifying the generated object code

5.1.8.14 Multiple Datapool Reference

0	4	8	16	24
X'D'	0	0 D	len	0 0
symbol name				
datapool reference				
pool number				

len	is the number of bytes in the command
symbol name	is an 1- to 8-character name of a symbol in the datapool
datapool reference	is a 4-byte datapool reference. The datapool symbol's value is added to the datapool reference.
pool number	is a value from 0 to 99 that identifies DPOOL00 to DPOOL99

5.1.9 Assembler Instructions and Generated Object Commands

A source listing is provided for an example program OBJTCOMD. This program is not meant to have any utility or to suggest any recommended programming practices. It is written to show assembly instructions and the object commands generated by those instructions. The comments appearing on the right side of the listing lines are added to help compare the assembler instructions with the generated object commands.

Compare the source listing in Figure 5-2 with the object code dump in Figure 5-3. This dump demonstrates the object code generated when instructions are assembled, and the position of object commands within the project.

```
00014                          C.3227   SETF
00015                          C.TRACF  SETF
00016                          C.MEMO   SETF
00017                          C.SALONE SETF
00018                          C.SYSGON SETF
00019                          *                                          OBJECT COMD   TYPE
00020                          *                                          ------------------
00021                                   PROGRAM  OBJTCOMD OBJECT_COMD_DEMO   5-  D10C
00022                                   DEF      OBJTCOMD                        D005
00023                          DATAPOOL COMMON   D1(1),D2(2)
00024                                   EXT      EXTEOBJT
00025   00000                           ABS
00026   80000                           ORG      X'80000'                        A-
00027                          COMMLARG COMMON   L1(1)                             E3
00028  C00000                  ORGCOM   ORG      L1                              0-
00029  C00000  00000001                DATAW    1
00030  P00000                          REL                                      D002
00031  P00000                  OBJTCOMD EQU      $                               D00B
00032  P00000  AD800000  A00000       LW       3,D1                            C-
00033 *P00000                          CSECT                             D002  D001
00034 *P00000  AC800000  C00000       LW       1,L1                            B-
00035 *P00004  F8800001  X00000       BL       EXTEOBJT                  0-    D006
00036  P00004                          DSECT                             D002  D001
00037  P00004  EC000009  P00008       BU       TRANSFER                        D003
00038  P00008            P00000  TRANSFER END      OBJTCOMD                  D00B  D004

*  0000  ERRORS IN OBJTCOMD


                                                                          T1020
```

Figure 5-2
Sample Source Listing

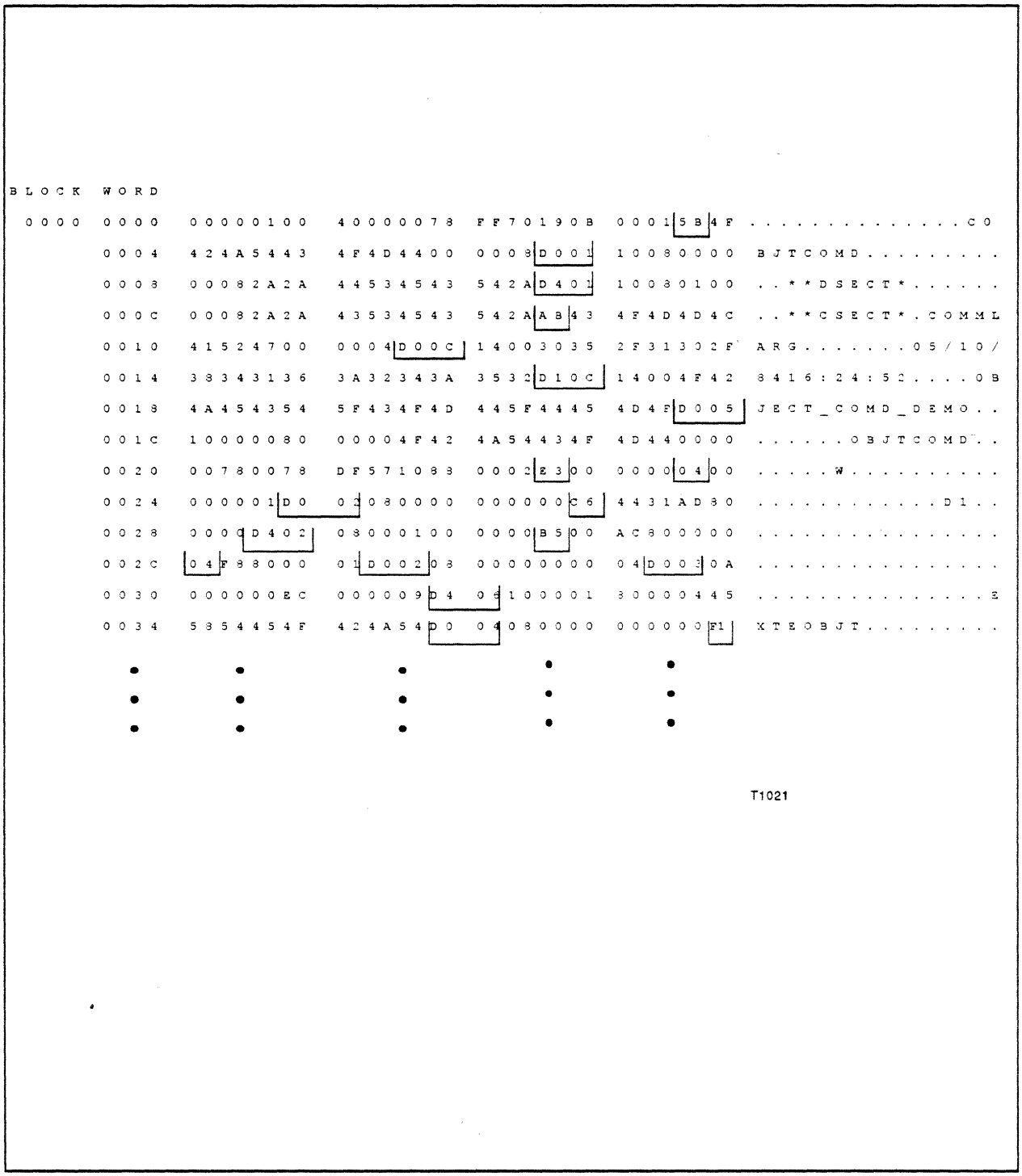


Figure 5-3
Sample Object Code Dump

5.2 AIDDB

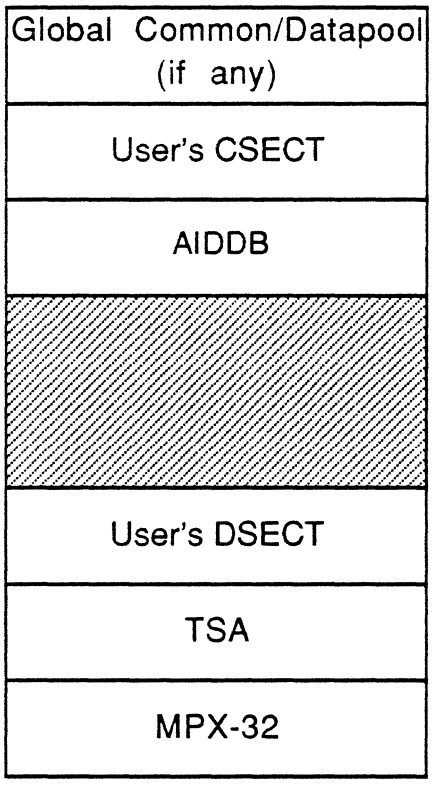
AIDDB functions as an unsolicited overlay of a nonbase mode task being debugged. When attached to a user task, it provides a set of commands that the user can use to monitor and control the execution of the task. It is an interactive tool for the online user operating under TSM, and can be used to debug a task in batch mode.

5.2.1 The AIDDB Environment

The activation sequence for a nonbase mode task to be run under AIDDB is similar to the normal activation sequence except that:

- the size of the address space constructed is increased by the size of AIDDB
- control is given to the AIDDB startup entry point instead of the transfer address of the user task

The address space constructed for AIDDB is as follows:



T1022

The AIDDB environment is established for a task by a call to H.REXS,29 (M.DEBUG service). The task can call H.REXS,29 anytime. The TSM DEBUG directive and the JCL $DEBUG directive cause H.REXS,29 to be called as part of the activation sequence for a user task.

Notes:

The combination of AIDDB and the user's code is a single task with a single TSA and a single dispatch queue entry. When AIDDB gains control at its start-up entry point, it makes dynamic assignments for its file codes according to whether it is running online or batch. The user task cannot make any dynamic assignments for these file codes. To minimize conflict with user file codes, all AIDDB file codes begin with the character #.

When AIDDB gains control, whether at activation or upon the occurrence of a trap or abort, it runs privileged, so it can replace user instructions with traps. When AIDDB transfers control to the user's task, it restores the privilege state (from privilege state at point of activation or interruption) of the user's task.

5.2.2 Entry Points

AIDDB begins with a halfword address table (HAT) in the following format:

```
DEBUG   DATAW   5
        ACH     DEBUG.1
        ACH     RESERVED
        ACH     DEBUG.3
        ACH     DEBUG.4
        ACH     DEBUG.5
        ACH     DEBUG.6
```

The entry points have the following functions:

Entry point	Function
1	start-up
2	reserved
3	trap/break receiver
4	re-entry after BREAK directive
5	user abort receiver
6	user overlay load courtesy call

5.2.2.1 Entry Point 1 - Start-up

File codes are assigned to the operating mode (online or batch), and files are opened. The first immediate command is read from #IN and DEBUG proceeds under control of the command stream.

Entry Conditions

The user PSD in T.CONTXT points to the cataloged transfer address of the user task. The user registers in T.CONTXT all contain zeroes. T.REGP points to T.REGS+0W.

EP1 is called because of a call to H.REXS,29 (M.DEBUG service) by the user task or as part of the activation sequence for the user task.

Exit Conditions

Exit is through any of the H.EXEC calls described in section entitled H.EXEC Calls, or M.EXIT in response to an immediate EXIT directive, or H.REXS,50 described in the H.REXS Calls section of this chapter.

5.2.2.2 Entry Point 2 - Reserved

5.2.2.3 Entry Point 3 - Trap/Break Receiver

T.CONTXT and the trap table are analyzed to distinguish breaks from traps. For a trap, the count for that trap is incremented by one.

For a conditional trap whose IF expression equals zero, control is passed back to the user task. For conditional traps whose IF expression does not equal zero, a trap report is issued and the IF command is displayed on #OT. For unconditional traps, a trap report is issued on #OT. In either case, AIDDB proceeds under control of the commands in the trap list.

For a break, the parameter input file reverts to #IN, a break report is issued on #OT, and DEBUG reads the next immediate command from the parameter input file.

Entry Conditions

This entry point is called when the user task executes an SVC 1,X'66' (AIDDB trap) instruction or receives a break. T.CONTXT indicates the user context following the execution of the last user instruction. T.REGP, T.REGS, and flags in DQE.ATI allow DEBUG to report the nesting, if any, of push-down levels due to any task interrupts active at the time of the trap or break.

Exit Conditions

Exit is through any of the H.EXEC calls described in section entitled H.EXEC Calls, or M.EXIT in response to an immediate EXIT directive, or H.REXS,50 described in the H.REXS Calls section of this chapter.

5.2.2.4 Entry Point 4 - M.BRKXIT Receiver

Execution of the user's M.BRKXIT is reported on #OT. AIDDB then reads the next immediate command from the parameter input file.

Entry Conditions

This entry point is called as the result of the user's execution of M.BRKXIT. The user's break receiver is run only as the result of an AIDDB call to H.EXEC,23. T.CONTXT, T.REGS, and T.REGP are the same as they were immediately before AIDDB called H.EXEC,23.

Note: After AIDDB calls H.EXEC,23, if the user task never executes M.BRKXIT, the break receiver push-down level in T.REGS will never be cleared. Each trap or break report on #OT will remind the user of this condition with its push-down analysis.

Exit Conditions

Exit is through any of the H.EXEC calls described in section entitled H.EXEC Calls, or M.EXIT in response to an immediate EXIT directive, or H.REXS,50 described in the H.REXS Calls section of this chapter.

5.2.2.5 Entry Point 5 - Abort Receiver

A report of the user abort, similar in form to a trap or break report, is displayed on #OT. The abort report includes the abort code message found in the dispatch queue entry. AIDDB then reads the next immediate command from the parameter input file.

Entry Conditions

This entry point is called when the user task encounters an abort condition. T.CONTXT indicates the user context following the execution of the last user instruction. T.REGS, T.REGP, and flags in DQE.ATI allow AIDDB to report the nesting, if any, of push-down levels due to any task interrupts active at the time of the abort. This entry point is never entered while the user task has an abort receiver established (M.SUAR service).

Exit Conditions

Exit is through any of the H.EXEC calls described in section entitled H.EXEC Calls, or M.EXIT in response to an immediate EXIT directive, or H.REXS,50 described in the H.REXS Calls section of this chapter.

5.2.2.6 Entry Point 6 — User Overlay Load Courtesy Call

AIDDB sets any traps in the user overlay that was just loaded prior to calling this entry point. After setting any traps, AIDDB returns to its caller.

Entry Conditions

H.REXS calls this entry point after a user overlay is loaded and before return to the user task. H.REXS has added the overlay name to the task's overlay name stack. R6 and R7 have the overlay name, left-justified, and blank filled. R2 points to the first instruction to be executed if the overlay was loaded with the execute flag on.

Exit Conditions

Exit is through M.RTRNOS.

5.2.3 H.EXEC Calls

DEBUG uses two special entry points in H.EXEC for control transfers, as follows:

H.EXEC,22

H.EXEC,22 is called by AIDDB in response to an immediate go or track directive, to begin or continue execution of the user task.

H.EXEC,23

H.EXEC,23 is called in response to an immediate break directive, to pass control to the user's break receiver. When the user task executes M.BRKXIT, control is passed to AIDDB entry point 4.

5.2.4 H.REXS Calls

This section describes the H.REXS calls which perform special AIDDB-related functions. AIDDB makes free use of many other H.REXS calls.

H.REXS,29

H.REXS,29 (M.DEBUG) is not called by AIDDB. It is called by a user task, or as part of the activation sequence, to establish the AIDDB environment for a user task.

H.REXS,30

H.REXS,30 is called in response to an immediate kill directive. Its function is to destroy the AIDDB environment previously established by H.REXS,29 (M.DEBUG), leaving the user task intact and transferring control to it at a specified context. In particular, H.REXS,30 makes AIDDB memory available for allocation by the user task.

H.REXS,42

H.REXS,42 (SVC 1,X'66') is the AIDDB trap instruction. It is stored in the user task, replacing the user's instruction, in response to the set, go, and track directives. Execution of SVC 1,X'66' by the user task causes control to pass to AIDDB entry point three after T.CONTXT is loaded with the user context.

H.REXS,50

H.REXS,50 (SVC 1,X'7E') is called by the user program to exit. If AIDDB is associated with the task, entry point 5 is entered and a user exit message is generated.

5.2.5 File Code Usage

AIDDB has no cataloged assignments. When it gains control at entry point 1, it dynamically assigns #IN, #OT, #01 and #04 according to the operating mode, online or batch. It assigns #02 and #03 in response to log, dump, file, and store directives. The following table lists the AIDDB file codes and their uses:

#IN Control input (commands)

Online: AS #IN TO LFC=UT
Batch: AS #IN TO SYC (only if not already assigned)

#OT Primary output (displays, trap reports, diagnostics, etc.)

Online: AS #OT TO LFC=UT
Batch: AS #OT TO SLO (only if not already assigned)

#01 Log file (log of all I/O on UT)

Online: AS #01 TO TEMP SIZE=300
Batch: not used

#02 SLO files for log and dump directives

Online: AS #02 TO SLO
or
AS #02 TO <filename> (when using the LOG <filename> command)
Batch: not used

#03 FILE and STORE files

Online: AS #03 TO <file> (where file is as specified in FILE or STORE command)
Batch: same as online

#04 Patch file (saved CM commands)

Online: AS #04 TO TEMP SIZE=100
Batch: not used

#SM Symbol table file

Online: AS #SM TO <loadmodule>
Batch: same as online

#HP Help file

Online: AS #HP TO DBHELP.H
Batch: same as online

#OV User's overlay load module

Online: AS #OV TO <overlay loadmodule>
Batch: same as online

#OD Overlay directory

Online: AS #01 TO TEMP SIZE=20
Batch: same as online

#LM AIDDB load module (used for overlay processing)

Online: AS #LM TO <loadmodule>
Batch: same as online

5.2.6 TSA References

The following TSA areas are referenced by AIDDB:

T.REGS	with DQE.ATI to analyze
T.REGP	user task interrupt status for abort, trap and break reports
T.CONTXT	user task context as of its last executed instruction
T.TRAD	user task transfer address

5.2.7 Communication Region References

C.MACH	to determine the machine type
C.CURR	to access the task's DQE entry
C.SMTA	to determine memory usage
C.MGRAN	to determine memory usage
C.SMTS	to determine memory usage

5.2.8 Dispatch Queue Entry (DQE) References

The following areas of the DQE are referenced by AIDDB:

DQE.USHF	to determine operating mode, online or batch
DQE.ATI	together with T.REGS and T.REGP to analyze user task interrupt status for abort, trap, and break reports

6 System Trace

6.1 Introduction

System Trace is an MPX-32 debug facility that assists in determining the cause of system crashes. System events are recorded circularly in a trace table which can be dumped in the event of a system crash.

Thirty trace event types are available for recording as follows:

1	Task activation
2	Task termination
3	Dispatch CPU to task
4	Task relinquishes CPU
5	Queue I/O
6	End I/O
7	Interrupt/trap handler entry
8	Interrupt/trap handler exit
9	M.SHUT
10	M.OPEN
11	M.IOFF or BEI
12	M.IONN or UEI
13	M.CALL
14	SVC Type 1
15	M.RTRN or M.RTNA
16	Inswap Task
17	Outswap Task
18	Dispatch IPU Task
19	Relinquish IPU Task
20	CALM
21-22	Mobile Event Trace
23	SVC Type 15
24	SVC Type 2
25-30	Reserved

Occurrences of trace events are signaled by SVC instructions. These SVCs are generated by the expanded BEI and UEI macros for trace types 11 and 12, and by the expanded M.TRAC macro for all other trace types. The M.TRAC macro is included in MPX-32 as required by each trace type. SVC types X'A', X'B', X'C', and X'D' are used by the System Trace event recorder.

The System Trace event recorder is an SVC processor which is assembled within H.IP06. Recorded events each use an eight-word entry in the trace table which occupies memory from absolute locations 78000 to 7FFFF. The table is circular - when the last entry in the table is used, the first entry is reused to record the next event.

System Trace event recording is controlled by flags in a word in the communications region, C.TRACE. Bit zero of C.TRACE controls all trace types. If this bit is set, no tracing is performed. If this bit is not set, tracing is controlled by bits 1 through 30 of C.TRACE. Bits 1 through 30 correspond to trace types 1 through 30, respectively, and may be set to turn each trace type off. Bit 31 of C.TRACE is reserved as an indicator that the trace table recording control words have been initialized by the event recorder. If the value of this bit is not disturbed, recording is continuous; for example, the control words are not reset. If the value of this bit is set to zero, the event recorder initializes the recording control words to indicate that the trace table is empty.

The first eight words of the trace table are reserved for control information. The first word contains the absolute memory address within the trace table at which the last trace entry was stored. Bit 0 of the second word is a wraparound indicator. This bit is set if wraparound from the last to first trace table entry has occurred.

The trace table dump routine is incorporated in resident MPX-32. This routine formats each trace table entry and writes it to the line printer. The routine is controlled by the contents of C.TRACD which is equated to absolute memory location 78008. Bits 1 through 15 of C.TRACD correspond to trace types 1 through 15, respectively, and may be set to inhibit printing of any trace types. Bits 16 through 31 of C.TRACD may be used to limit the number of trace table entries eligible for printing. If this field contains zero, all trace table entries are eligible for printing. If this field contains a nonzero value, this is the number of most recently recorded entries eligible for printing.

To use the trace table dump routine, the contents of C.TRACD should be set as desired and control transferred to the routine at its entry point. The symbol TRACDUMP is associated with the entry point via a DEF. The routine must be entered unmapped. After the dump is completed, the routine halts. Each trace type is detailed next. Fields in printout formats are underlined to indicate actual values. Numeric hexadecimal fields are indicated by "x". Numeric decimal fields are indicated by "d". The printout of each trace table entry occupies one printer line.

6.2 Trace Type 1 - Task Activation

Macro:

```
M.TRAC    1

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',1
```

Implanted in H.ALOC so that the address of the task's dispatch queue entry is contained in R7 to be returned and may be obtained as follows:

```
LW        R2,C.TSAD
LW        R2,T.REGP,R2
LW        R2,7W,R2
```

Trace Table Entry:

Word 0	Type 01	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  ACTIVATE TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.3 Trace Type 2 - Task Termination

Macro:

```
M.TRAC      2

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',2
```

Implanted in H.EXEC so that C.CURR contains the address of the task's dispatch queue entry number.

Trace Table Entry:

Word 0	Type 02	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  TERMINATE TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.4 Trace Type 3 - Dispatch CPU to Task

Macro:

```
M.TRAC      3

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',3
```

Implanted in H.EXEC so that R2 contains the address of the task's dispatch queue entry number.

Trace Table Entry:

Word 0	Type 03	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  DISPATCH TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.5 Trace Type 4 - Task Relinquishes CPU

Macro:

```
M.TRAC    4

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',4
```

Implanted in H.EXEC so that R2 contains the address of the task's dispatch queue entry number.

Trace Table Entry:

Word 0	Type 04	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  RELINQ TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.6 Trace Type 5 - Queue I/O

Macro:

```
M.TRAC    5

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',5
```

Implanted in H.IOCS so that R3 contains the I/O queue entry address.

Trace Table Entry:

Word 0	Type 05	FCB or TCPB address (IOQ.FCBA)	
1	Interrupt counter (C.INTC)		
2	Handler function word 1 (IOQ.FCT1)		
3	Handler function word 2 (IOQ.FCT2)		
4	Handler function word 3 (IOQ.FCT3)		
5	32-bit flag word (IOQ.FLGS)		
6	Task activation sequence number (C.TSKN)		
7	Channel # (IOQ.CHNO)	Subaddress (IOQ.SUBA)	

Printout:

```
C.INTC  QUE I/O TASK  = xxxxxxxx

        DEV  = xxxx IOQ.FLGS  = xxxxxxxx FN WDS  = xxxxxxxx
        xxxxxxxx xxxxxxxx

        FCB  = xxxxxxxx
```

6.7 Trace Type 6 - End I/O

Macro:

```
M.TRAC      6

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',6
```

Implanted in H.EXEC (S.EXEC1, S.EXEC2, S.EXEC3 and S.EXEC4) so that R1 contains the task's dispatch queue entry number.

Trace Table Entry:

Word 0	Type 06	
1	Interrupt counter (C.INTC)	
2-5		
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7		

Printout:

```
C.INTC   END I/O TASK  = xxxxxxxx
```

6.8 Trace Type 7 - Interrupt/Trap Handler Entry

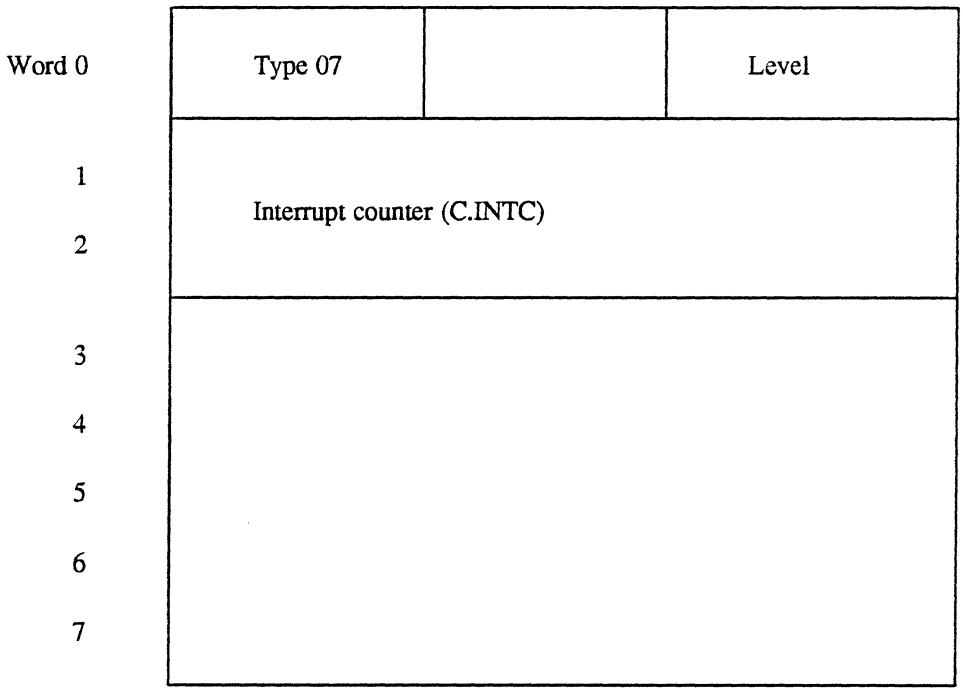
Macro:

```
M.TRAC    7,level

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'B',X'level'
```

Implanted in interrupt/trap handler.

Trace Table Entry:



Printout:

```
C.INTC   ENTERINT  xxxx
```

6.9 Trace Type 8 - Interrupt/Trap Handler Exit

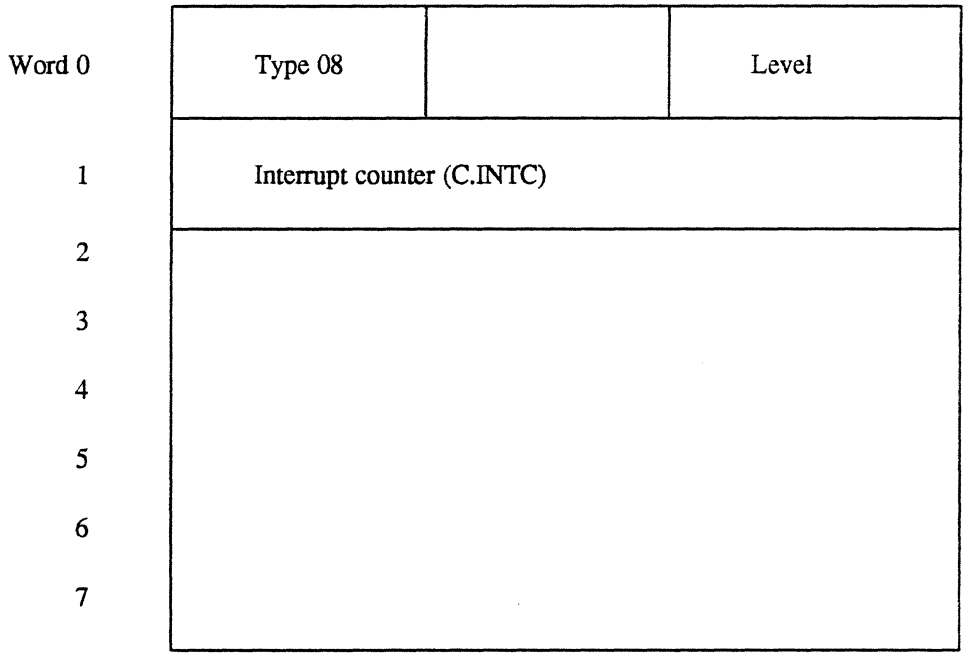
Macro:

```
M.TRAC    8,level

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'C',X 'level'
```

Implanted in the interrupt/trap handler.

Trace Table Entry:



Printout:

```
C.INTC   EXITINT  xxxx
```

6.10 Trace Type 9 - M.SHUT

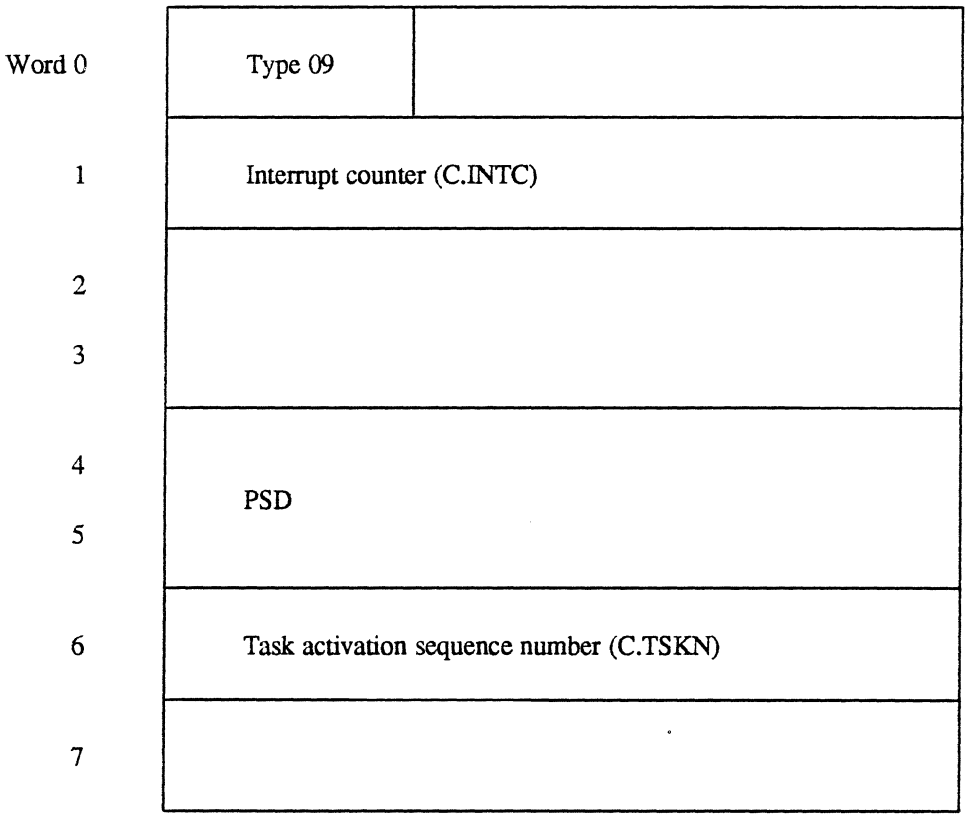
Macro:

```
M.TRAC      9

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',9
```

Implanted in M.SHUT macro.

Trace Table Entry:



Printout:

```
C.INTC   M.SHUT TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
```

6.11 Trace Type 10 - M.OPEN

Macro:

```
M.TRAC      10

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',10
```

Implanted in M.OPEN macro.

Trace Table Entry:

Word 0	Type 10	
1	Interrupt counter (C.INTC)	
2		
3		
4	PSD	
5		
6	Task activation sequence number (C.TSKN)	
7		

Printout:

```
C.INTC   M.OPEN TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
```

6.12 Trace Type 11 - M.IOFF or BEI

Implemented by BEI macro whose prototype is as follows:

```
BEI   DEFM
      TBM       0,C.TRACE
      BCT       1,$+3W
      DATAW     X'00060002'
      SVC       X'A',11
      ENDM
```

Trace Table Entry:

Word 0	Type 11	
1	Interrupt counter (C.INTC)	
2		
3		
4	PSD	
5		
6	Task activation sequence number (C.TSKN)	
7		

Printout:

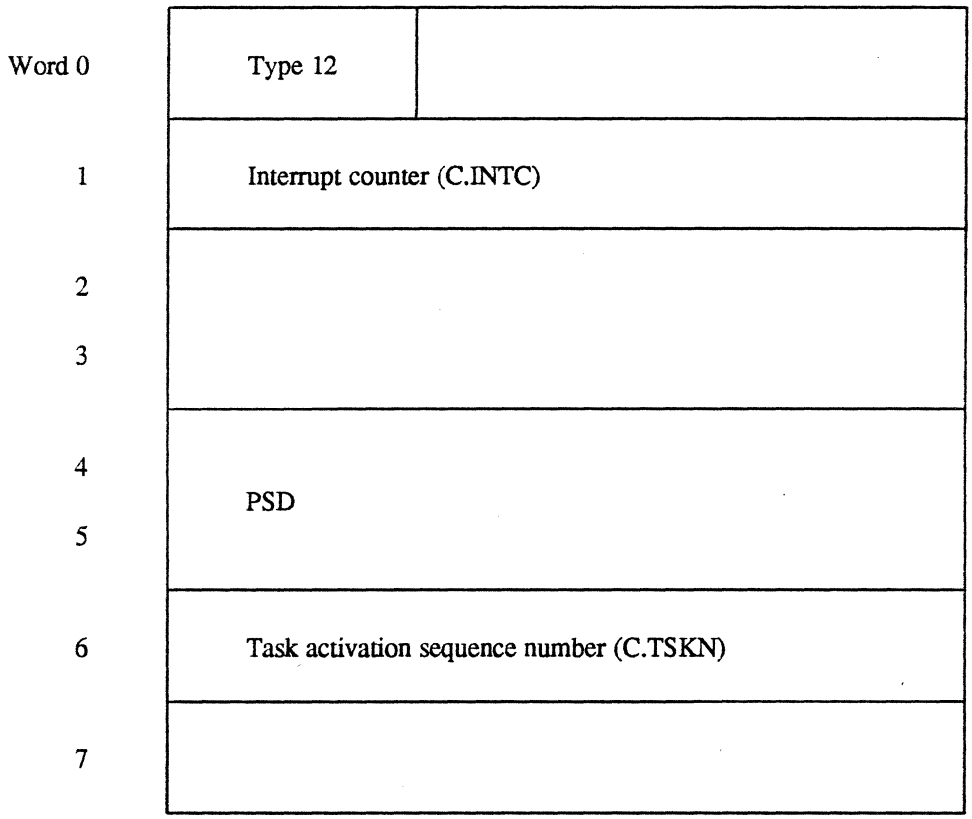
```
C.INTC   M.IOFF TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
```

6.13 Trace Type 12 - M.IONN or UEI

Implemented by UEI macro whose prototype is as follows:

```
UEI   DEFM
      TBM       0,C.TRACE
      BCT       1,$+3W
      DATAW     X'00070002'
      SVC       X'A',12
      ENDM
```

Trace Table Entry:



Printout:

```
C.INTC  M.IONN TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
```

6.14 Trace Type 13 - M.CALL

Macro:

```
M.TRAC    13

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',13
```

Implanted in H.IP06.

Trace Table Entry:

Word 0	Type 13		Bits 20-13 of the SVC
1	Interrupt counter (C.INTC)		
2			
3			
4	PSD		
5			
6	Task activation sequence number (C.TSKN)		
7	Stack frame pointer (T.REGP)		

Printout:

```
C.INTC   M.CALL TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
         MODULE  = name, dd
```

6.15 Trace Type 14 - SVC Type 1

Macro:

```
M.TRAC    14

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',14
```

Implanted in H.IP06.

Trace Table Entry:

Word 0	Type 14		Bits 20-31 of the SVC
1	Interrupt counter (C.INTC)		
2 3			
4 5	PSD		
6	Task activation sequence number (C.TSKN)		
7	Stack frame pointer (T.REGP)		

Printout:

```
C.INTC   SVC1TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
         SVC  = dddd
```

6.16 Trace Type 15 - M.RTRN or M.RTNA

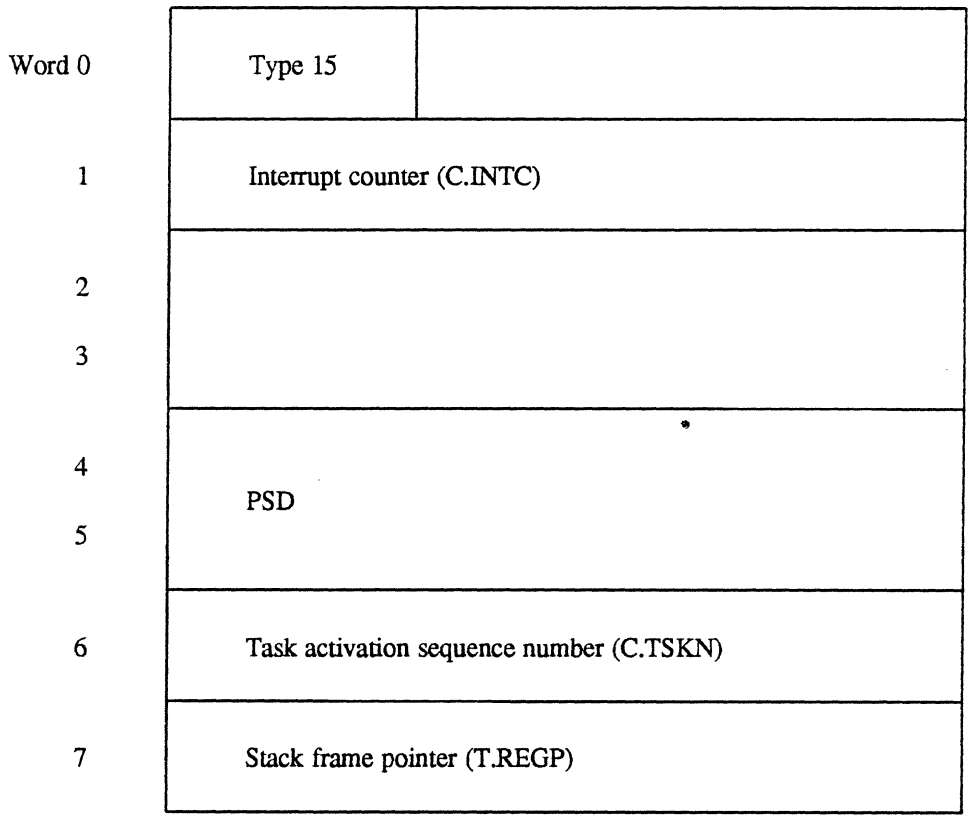
Macro:

```
M.TRAC    15

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',15
```

Implanted in M.RTRN and M.RTNA macros.

Trace Table Entry:



Printout:

```
C.INTC  M.RTRN/A TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
```

6.17 Trace Type 16 - Inswap Task

Macro:

```
M.TRAC      16

TBM         0.C.TRACE
BCT         1,$+2W
SVC         X'A',16
```

Implanted in H.IP06.

Trace Table Entry:

Word 0	Type 16	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC   INSWAP TASK  = xxxxxxxx DQE.LMN DQE.ON

         DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.18 Trace Type 17 - Outswap Task

Macro:

```
M.TRAC      17

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',17
```

Implanted in IP06.

Trace Table Entry:

Word 0	Type 17	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  OUTSWAP TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.19 Trace Type 18 - Dispatch IPU Task

Macro:

```
M.TRAC      18

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',18
```

Implanted in H.CPU so that R2 contains the address of the dispatch queue entry address.

Trace Table Entry:

Word 0	Type 18	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Schedule flags (DQE.USHF)	

Printout:

```
C.INTC  DISP IPU TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.20 Trace Type 19 - Relinquish IPU Task

Macro:

```
M.TRAC      19

TBM         0,C.TRACE
BCT         1,$+2W
SVC         X'A',19
```

Implanted in H.CPU so that R2 contains the address of the dispatch queue entry address.

Trace Table Entry:

Word 0	Type 19	TSA address (DQE.TAD)
1	Interrupt counter (C.INTC)	
2 3	Load module name (DQE.LMN)	
4 5	Owner name (DQE.ON)	
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Scheduling flags (DQE.USHF)	

Printout:

```
C.INTC  RELINQ IPU TASK  = xxxxxxxx DQE.LMN DQE.ON

        DQE.USHF  = xxxxxxxx DQE.TAD  = xxxxxxxx
```

6.21 Trace Type 20 - Reserved

6.22 Trace Type 21 - Mobile Event Trace 1

Implanted by the system debugger by the ET command.

Trace Table Entry:

Word 0	Type 21	Stack frame address (T.REGP)
1	Interrupt counter (C.INTC)	
2	DQE schedule flags (DQE.USHF)	
3	Requested task interrupts (DQE.RTI)	Active task interrupts (DQE.ATI)
4	PSD	
5		
6	DQE entry # (DQE.NUM)	Task activation sequence number (DQE.TAN)
7	Swap inhibit flags (DQE.SWIF)	System action interrupt requests (DQE.SAIR)

Printout:

```
C.INTC  ET #1 TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
        USHF  = xxxxxxxx RTI/ATI  = xxxxxxxx T.REGP  = xxxxxxxx
        SWIF  = xx SAIR  = xx
```

6.23 Trace Type 22 - Mobile Event Trace 2

Implanted by the system debugger by the ET command.

Trace Table Entry:

Word 0	Type 22	Contents of GPR 0
1	Contents of GPR 1	
2	Contents of GPR 2	
3	Contents of GPR 3	
4	Contents of GPR 4	
5	Contents of GPR 5	
6	Contents of GPR 6	
7	Contents of GPR 7	

Printout:

```
R0 = xxxxxxxx R1 = xxxxxxxx R2 = xxxxxxxx R3 = xxxxxxxx
R4 = xxxxxxxx R5 = xxxxxxxx R6 = xxxxxxxx R7 = xxxxxxxx
```

6.24 Trace Type 23 - SVC Type 15

Macro:

```
M.TRAC    23

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',23
```

Implanted in H.IP06.

Trace Table Entry:

Word 0	Type 23		Bits 20-31 of the SVC
1	Interrupt counter (C.INTC)		
2 3			
4 5	PSD		
6	Task activation sequence number (C.TSKN)		
7	Stack frame pointer (T.REGP)		

Printout:

```
C.INTC  SVC15 TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
        SVC  = dddd
```

6.25 Trace Type 24 - SVC Type 2

Macro:

```
M.TRAC    24

TBM       0,C.TRACE
BCT       1,$+2W
SVC       X'A',24
```

Implanted in H.I06.

Trace Table Entry:

Word 0	Type 24		Bits 20-31 of the SVC
1	Interrupt counter (C.INTC)		
2			
3			
4	PSD		
5			
6	Task activation sequence number (C.TSKN)		
7	Stack frame pointer (T.REGP)		

Printout:

```
C.INTC   SVC2 TASK  = xxxxxxxx PSD  = xxxxxxxx xxxxxxxx
         SVC  = dddd
```

7 System Initializers and Builders

7.1 Introduction

This chapter describes the components of MPX-32 that load and initialize the system when it is booted. The components involved are:

- SDT loader contained within the Volume Manager
- SYSINIT, J.INIT, J.TINIT, and RESTART system tasks

A system boot can be performed in various operating environments and involve various system devices. In a given environment, a system boot can require all of the above components or a subset.

Figures 7-1 through 7-3 describe the components required when the system is booted from a Software Distribution Tape (SDT), from the IOP console, and from an online RESTART directive.

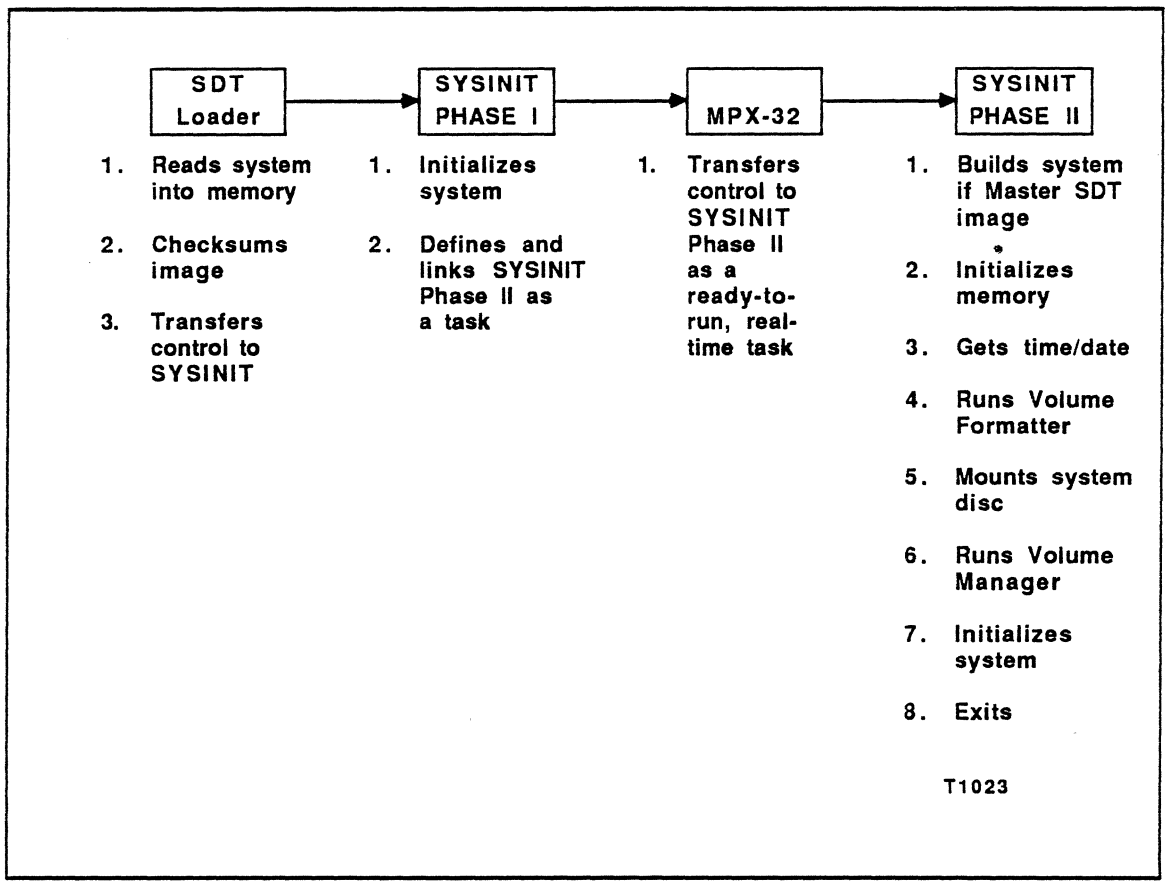


Figure 7-1
Components and Functions in Boot from an SDT

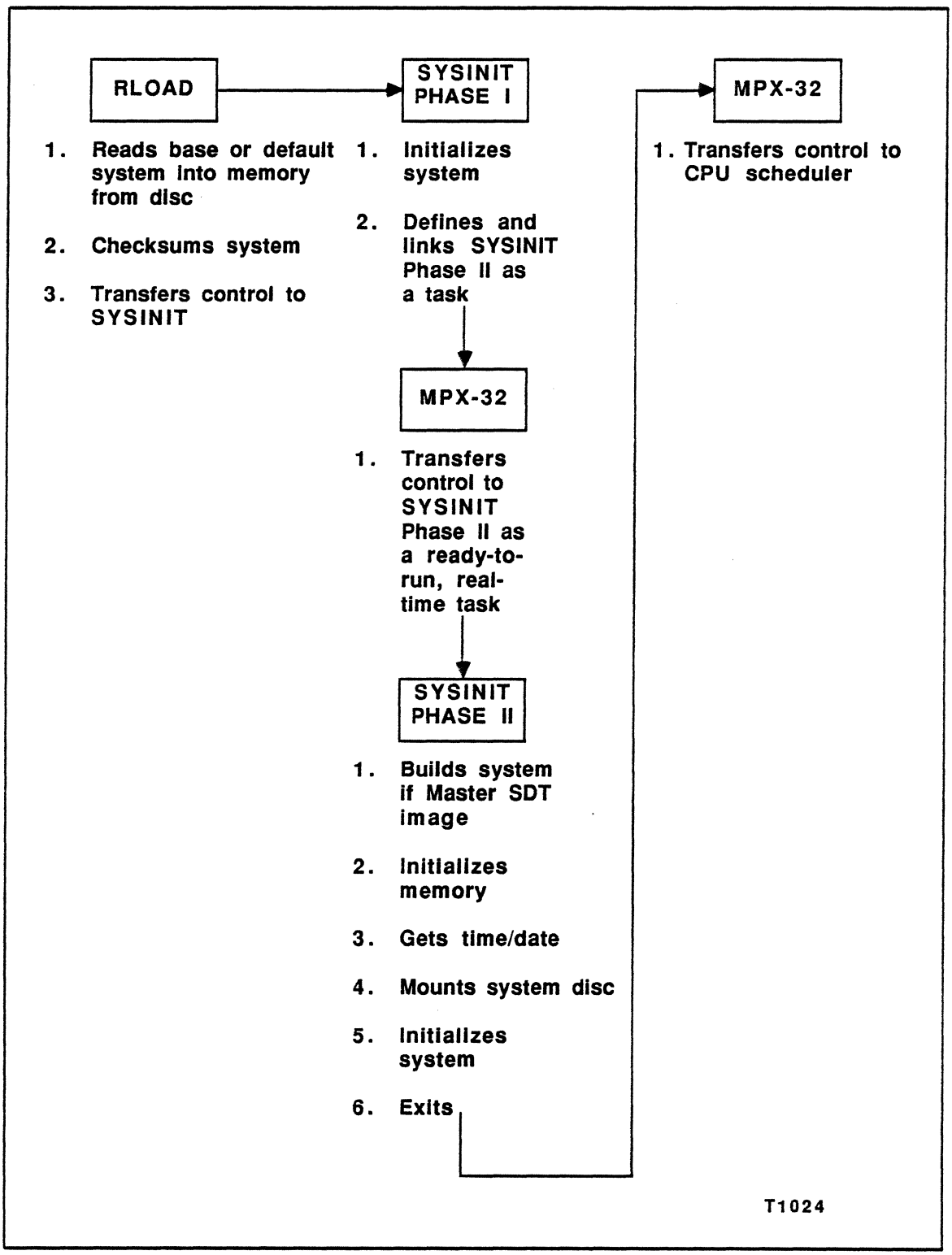


Figure 7-2
Components and Functions in Boot from IOP Console

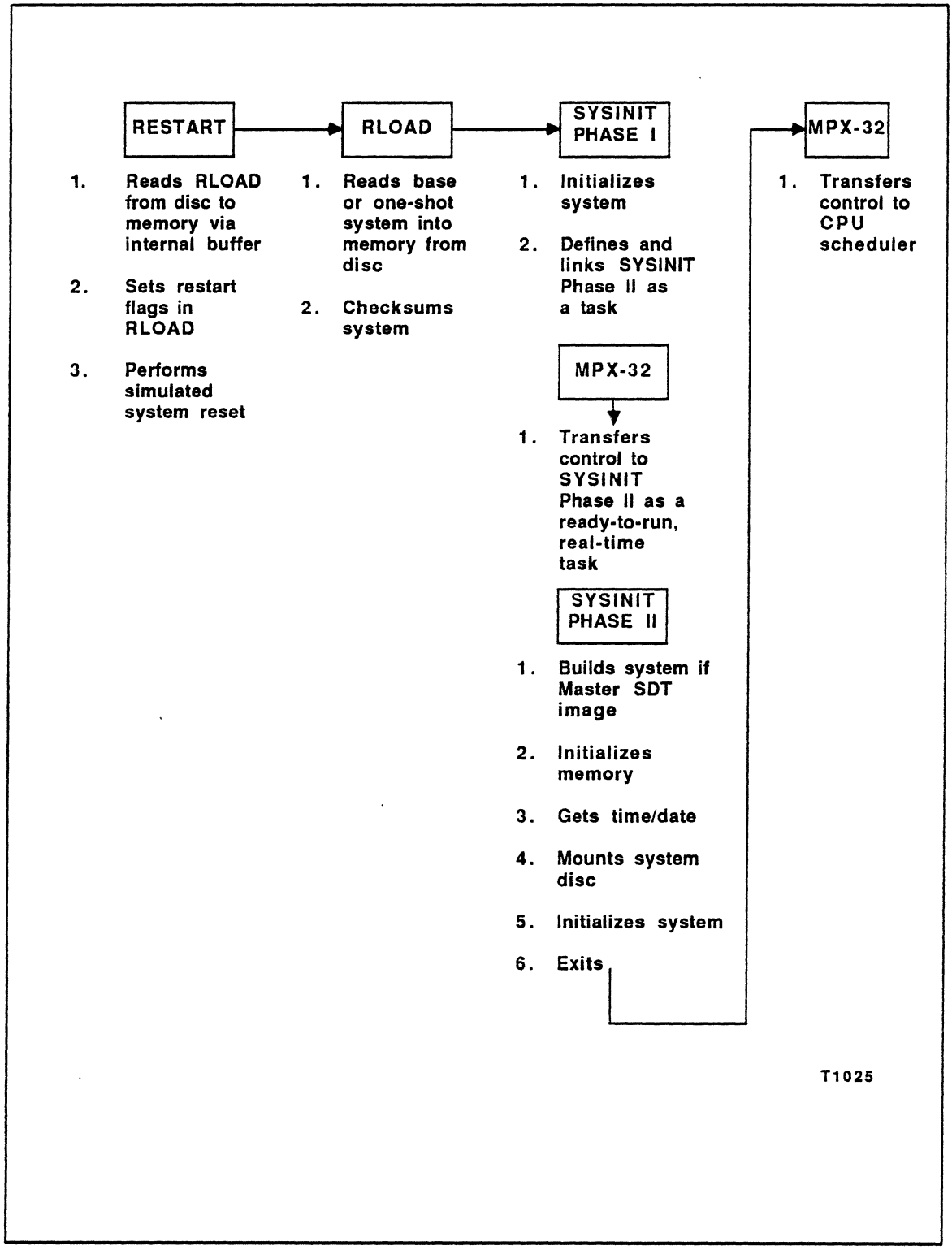


Figure 7-3
Components and Functions in Boot from Online RESTART

7.2 SDT Loader

The SDT loader is a section of code written to a system distribution tape (SDT) by the portion of the Volume Manager that processes the SDT directive. Its purpose is to read the system image from the SDT into memory. Control is then transferred to the MPX-32 initialization program SYSINIT which builds a functional system.

7.2.1 Activating

The SDT loader is activated from the SDT by the IPL = *devaddr* (device address) console command. The loader code must be the first piece of information contained on the SDT.

7.2.2 Required Input

The SDT loader requires no input. However, the starting load point may be altered by supplying a value other than zero in R3 prior to depressing the IPL button. The load point defaults to X'780'.

7.2.3 Processing

The SDT loader is written to tape as absolute code by supplying the SDT directive to the Volume Manager. The loader code is then followed by the system image whose name is supplied in the directive.

When an IPL is performed, firmware reads the SDT loader from the IPL medium into memory, starting at absolute address zero. Control is then transferred to location zero. The loader then reads the system image into memory from tape or floppy disk sequentially starting at the load point specified in R3 or the default load point if R3 is zero.

The image is read in 192W blocks by means of a buffer reserved in low memory. The load module preamble is read first and the appropriate information is extracted from it. Then the resource requirement summary (RRS) block of the preamble is ignored, and the remainder of the image is loaded into memory.

A checksum is performed while loading the image. The checksum is compared with the checksum value supplied in the preamble. If the checksum value does not agree, the loader halts execution.

After loading the system image, the loader transfers control to SYSINIT Phase I to allow initialization of the operating system to proceed.

7.2.4 Results

Loading has commenced at the appropriate load point, with all memory locations below the load point undefined. For a normal boot from a system distribution tape (SDT), memory resident code consists of the version of MPX-32 specified in the SDT directive, immediately followed by the SYSINIT load module. From this point, control is transferred to the SYSINIT entry point.

7.3 The DBOOT Program Section

7.3.1 Activating

The system disk bootstrap is placed at sector 0 of all system disks by the Volume Formatter. This enables booting from any properly formatted disk.

To boot the computer, the console is entered in panel mode and given the command IPL = *xxxx*, where *xxxx* is the channel and subaddress of the system disk.

The CPU firmware reads 120 bytes from sector 0 of the disk.

The IOCL starting at location eight is executed. When this IOCL terminates, the program status doubleword (PSD) is loaded with the contents of locations 0 and 4 and I/O status is placed back into locations 0 and 4.

7.3.2 Processing

The bootstrap code loads the operating system, checksums it, and transfers control to SYSINIT Phase I to complete the initialization of the MPX-32 system.

7.4 The SYSINIT Program Section

7.4.1 Activating

SYSINIT is comprised of two sections: Phase I and Phase II. Phase I is entered by either the SDT loader or the disk bootstrap loader and runs as stand-alone code. Phase II is actually the first MPX-32 task and is entered by the MPX-32 execution scheduler as a result of the first real-time clock interrupt.

Control transfers from either loader via a context switching type process. The address of SYSINIT's TSA is stored at C.CTSAD by SYSGEN. The loader uses this address to load general purpose registers 2 through 7 from the current general purpose stack frame in SYSINIT's TSA. (Note: GPR 0 is used to pass parameters from the loader to SYSINIT. GPR 1 is used by the loader in the hand-off process.) Base registers are loaded from the current base register stack frame in SYSINIT's TSA. The loader then performs a load program status double word (LPSD) using the PSD stored in the current frame of the general purpose stack to transfer control to SYSINIT. SYSINIT will pop off the current frames of the general purpose and base register stacks and continue with normal initialization procedures.

7.4.2 Processing

Phase I of SYSINIT runs as stand alone code because, when it is entered, the hardware and the software are not ready for OS operation. Phase I performs the required hardware initialization functions while Phase II performs the software initialization functions.

The hardware initialization functions performed by Phase I are as follows:

1. perform auto console configuration if booting a CONCEPT 32/2000 system
2. load CPU scratchpad from the image built by SYSGEN
3. update memory allocation tables to reflect the memory occupied by SYSINIT
4. build a dispatch queue entry for SYSINIT and link it on the ready-to-run state queue
5. set up the interrupt vector locations
6. determine the unit definition table (UDT) address of the IPL device for all startups except for a Master SDT boot, (see section on booting from a Master SDT for details)
7. enable all peripheral interrupt levels
8. enable all software interrupt levels
9. enable traps
10. set the CPU mode
11. set mapped mode, unblock interrupts, and wait for a clock interrupt

The software initialization functions performed by Phase II are as follows:

1. memory initialization
2. request date and time to be entered
3. disk start-up initialization
4. User SDT start-up initialization
5. Master SDT start-up initialization

7.4.2.1 Memory Initialization

The first function performed by SYSINIT on a nonmaster SDT boot is the initialization of memory. This clears parity errors which occur in MOS memory after power up, locates nonpresent sections of physical memory, and locates defective memory modules.

The initializer revectors parity and nonpresent memory traps to point to handlers within SYSINIT. Then, for MOS memory, every location is read, and if it is not in a multi-processor shared section of memory, written back. This clears all potential parity errors. If a parity error is detected on the read, a test data pattern is written to the location and then read back. If another parity error or a mismatched data pattern is detected, the module is marked as malfunctioning and the testing continues. If a nonpresent memory trap is encountered on a read, the module is flagged as nonpresent and testing continues.

For core memory, one location is checked in each module to determine if the module is present.

For the CONCEPT 32/2000 processor, memory types defined at SYSGEN must be verified at SYSINIT. Any discrepancies between SYSGEN definitions of memory and physically present SRAM and DRAM memory are handled as nonpresent memory. (Any physically present DRAM memory which is SYSGENed as SRAM is declared nonpresent. Any physically present SRAM memory which is SYSGENed as DRAM is declared nonpresent.) When this occurs the following message is displayed.

```
***WARNING: SYSGEN DEFINITION OF MEMORY DOES NOT MATCH PHYSICALLY
CONFIGURED MEMORY.
```

This message is followed by one of the following.

If the physical amount of SRAM is greater than the amount of SYSGENed E, H, and S memory, the following message is displayed.

```
ACTUAL SRAM:   0 - addr1
SRAM FROM  addr2 - addr1 IS UNUSED
```

and the following if part of the SYSGENed D memory physically resides in SRAM:

```
TYPE 'D' FROM  addr2 - addr1 MARKED NONPRESENT
```

or the following if all the SYSGENed D memory physically resides in SRAM:

```
TYPE 'D' FROM  addr3 - addr4 MARKED NONPRESENT
```

If the physical amount of SRAM is less than the amount of SYSGENed E, H, or S memory, the following message is displayed:

```
ACTUAL SRAM:          0 - addr1
TYPE 'type' FROM addr1 - addr5 MARKED NONPRESENT
```

addr1 is the hexadecimal address of the physical end of SRAM

addr2 is the hexadecimal address of the start of memory SYSGENed as DRAM (or nonpresent) which is physically SRAM

addr3 is the hexadecimal address of the start of SYSGENed DRAM memory which is physically SRAM

addr4 is the hexadecimal address of the end of SYSGENed DRAM memory which is physically SRAM

addr5 is the hexadecimal address of the end of memory SYSGENED as *type* which is physically DRAM

type is E, H, or S

At the end of the routine, the maps are reset to point to their normal handlers and SYSINIT continues software initialization.

7.4.2.2 System Date and Time

This routine prompts the user to enter the date and time for use by the system. See the MPX-32 Reference Manual Volume III, Chapter 2 for valid entry formats.

7.4.2.3 Disk Start-up Final Initialization

For disk start-up, the system volume must be mounted. SYSINIT automatically allocates the volume, reads the volume descriptor, and builds a mounted volume table entry (MVTE). This allows a call to be made to the mount service H.REMM,17 which causes a run request to be sent to J.MOUNT (the system nonresident media mounting program) which actually performs the mount.

Next, J.SWAPR is built into the system by SYSGEN. Its SYSGEN initialization entry point links its own dispatch queue entry to the suspend queue. Execution is held until SYSINIT finishes its functions, then SYSINIT issues a resume request for J.SWAPR.

Once the swapper is running, SYSINIT completes system initialization by activating the following sequence of tasks:

1. J.INIT — installs system patches, mounts default public volumes, and loads ACS
2. J.TINIT — initializes user terminals
3. J.TSM — builds environment for interactive users
4. J.TDEFI — initializes the TERMDEF facility, if present
5. user sequentially run tasks, see SYSGEN SEQUENCE directive
6. user activate table, see SYSGEN ACTIVATE directive

SYSINIT then exits.

7.4.2.4 Tape Start-up Final Initialization

For tape boots, it is assumed there is no information on the disk to become the system volume. As a result, all tasks needed to build the disk environment are included on the system distribution tape (SDT). These tasks are:

1. Volume Formatter (J.VFMT) — This task builds the maps and data structures needed for file maintenance, places a disk bootstrap at sector 0, and writes a copy of the operating system to the desired disk. This assumes an empty disk. J.VFMT can also be used to replace an operating system image only, leaving other data intact, thus giving the user warm start capability. See MPX-32 Reference Manual Volume III, Chapter 13.
2. Mount Program (J.MOUNT) — This task is activated to mount the formatted volume created by J.VFMT.
3. Volume Manager Program (VOLMGR) — This program is activated so files in the file save area following the system information on an SDT can be restored.

After all tasks on the tape have been activated, J.SWAPR is invoked. Once the swapper is running, SYSINIT completes system initialization by activating the following sequence of tasks:

1. J.INIT — installs system patches, mounts default public volumes, and loads ACS
2. J.TINIT — initializes user terminals
3. J.TSM — builds environment for interactive users
4. user sequentially run tasks, see SYSGEN SEQUENCE directive
5. user activate table, see SYSGEN ACTIVATE directive

SYSINIT then exits.

7.4.2.5 Master SDT

The Master System Distribution Tape for MPX-32 contains three system images, MSTRALL, the default image for all CONCEPT 32/*xx* computers; MSTREXT, the extended-mode image and MSTROUT, the default image for CONCEPT 32/2000 computers.

The format of the Master SDT is shown below and described in the paragraphs which follow.

BOOT	IMAGE MSTRALL	EOF	IMAGE MSTREXT	EOF	IMAGE MSTROUT	EOF	J.VFMT	EOF	J.MOUNT,J.SWAPR, VOLMGR	EOF	EOF	SAVED FILES

Tape Boot Loader — The first record on an SDT is the boot loader. This code is contained within the Volume Manager and written to the tape as a result of the SDT command. The boot loader first determines the type of machine it is executing on by reading the CPU status word. It then sets up the CPU scratchpad RAM to allow the IPL device to be read as logical device X'1000'.

Two flags are set inside the bootstrap code by the Volume Manager. One specifies the IPL device is a floppy disk. The other indicates a master, as opposed to a user, SDT format.

If the flag indicating Master SDT is set, the boot loader uses the machine type to determine if it is necessary to set command chain bits in the initialization IOCD list. These chained commands are one or two skipfile commands used to advance the tape to the proper system image for the CPU type being IPLed.

Once the tape has been positioned, the image is read into memory. Control then passes to SYSINIT, and system initialization begins.

SYSINIT - Phase I Initialization — Phase I of SYSINIT prepares the MPX-32 environment. This allows the Phase II portions of SYSINIT to run as an MPX-32 task; therefore, system service calls can be used to perform I/O rather than stand-alone I/O routines.

The first functions SYSINIT performs are reading the IPL device and data return transfer response locations in scratchpad, loading scratchpad from the C.SPAD area created by SYSGEN, initializing the interrupt vector area in low memory, allocating SYSINIT's memory space in the MPX-32 memory tables, and building and linking a CPU dispatch queue entry for SYSINIT. If booting from a disk, SYSINIT may abort. See the MPX-32 Reference Manual, Volume III, Chapter 6.

If the machine is a 32/67 or 32/97, the shared memory region's high and low bounds are set in C.SHRHI and C.SHRLO.

After these operations, SYSINIT compares the doubleword value at C.SYSTEM with the three names reserved for Master SDT starter systems. If a match is found, a flag bit is set and a skip count control word is loaded to allow SYSINIT to correctly process a Master SDT IPL sequence for the current machine type. The skip count word is used to determine where the next desired information resides on the SDT.

The three reserved system names are:

MSTRALL - default image for all CONCEPT 32/*xx* computers

MSTREXT - split-mode image

MSTROUT - default image for CONCEPT 32/2000 computers

The final activity in Phase I of SYSINIT is enabling all interrupts and then unblocking them. This allows the clock interrupt to occur, causing a context switch to SYSINIT Phase II, the task portion of SYSINIT.

SYSINIT - Phase II Initialization — Phase II of SYSINIT checks for a Master SDT boot. This is indicated by the flag set in Phase I. If set, the proper peripheral configuration must be set up to allow further initialization.

The Master SDT system images are SYSGENed in a special way. The following controller and device entries are included:

```
/CHANNELS
CONTROLLER=DM03,PRIORITY=06,CLASS=F,MUX=XIO,HANDLER=(H.IFXIO,I)
DEVICE=(00,3,2),DISK=ANY,HANDLER=(H.DCXIO,S),OFF
CONTROLLER=DM08,PRIORITY=07,CLASS=F,MUX=XIO,HANDLER=(H.IFXIO,I)
DEVICE=(00,2,2),DISK=ANY,HANDLER=(H.DCXIO,S),OFF
CONTROLLER=DM7E,PRIORITY=13,CLASS=F,MUX=MFP,HANDLER=(H.IFXIO,I)
DEVICE=00,DISC=ANY,HANDLER=(H.DCSCI,S),OFF
DEVICE=08,DISC=ANY,HANDLER=(H.DCSCI,S),OFF
CONTROLLER=DM76,PRIORITY=12,CLASS=F,MUX=MFP,HANDLER=(H.IFXIO,I)
DEVICE=00,DISC=ANY,HANDLER=(H.DCSCI,S),OFF
CONTROLLER=M910,PRIORITY=08,CLASS=F,MUX=XIO,HANDLER=(H.IFXIO,I),CACHE
DEVICE=00,DTC=M9,HANDLER=(H.MTXIO,S),OFF
CONTROLLER=M97E,PRIORITY=13,CLASS=F,MUX=MFP,SUBCH=4,CACHE
DEVICE=40,DTC=M9,HANDLER=(H.MTSCI,S),OFF
CONTROLLER=M976,PRIORITY=12,CLASS=F,MUX=MFP,SUBCH=4,CACHE
DEVICE=40,DTC=M9,HANDLER=(H.MTSCI,S),OFF
CONTROLLER=CT02,PRIORITY=02,CLASS=F,MUX=XIO,SUBCH=F,HANDLER=(H.IFXIO,I)
DEVICE=FC,DTC=CT,HANDLER=H.CTXIO,LINSIZ=80,PAGE=20
```

```
CONTROLLER=LF7E,PRIORITY=13,CLASS=F,MUX=IOP,SUBCH=F
DEVICE=F0,DTC=FL,HANDLER=H.DCXIO,DISC=FL001,OFF
DEVICE=F8,DTC=LP,SPOOL=(BL,RL),HANDLER=H.LPXIO
CONTROLLER=DM7E,PRIORITY=13,CLASS=F,MUX=IOP,SUBCH=C
DEVICE=(C0,2,2),DISC=ANY,HANDLER=H.DCXIO,OFF
CONTROLLER=DM04,PRIORITY=09,CLASS=F,MUX=XIO,HANDLER=(H.IFXIO,I)
DEVICE=(00,2,2),DISC=ANY,HANDLER=(H.DPXIO,S),OFF
CONTROLLER=NU00
DEVICE=00,SHR,DTC=NU,SPOOL=(BB,RB)
```

All possible handlers for a one disk, one tape system are included. Three dummy disks are included so the UDT configurations are similar, and the IOP disk is configured.

The first device initialized is the tape. The IPL device address scratchpad word, saved in Phase I, provides the address of the tape. Using this information and known UDT indices due to the special format SYSGEN directives, SYSINIT configures the UDT, CDT, DCA, and CHT tables, and marks the tape on-line.

The second device initialized is the disk. SYSINIT prompts the operator for the device address and the type of controller (XIO or IOP).

Using answers to the prompts and the known UDT indices, SYSINIT selects the UDT entry with the proper handler for the disk. SYSINIT uses a copy of the SYSGEN module SJ.STBLS, which contains device dependent parameters for the various disk drives, to fill in the following UDT areas: sectors per block, sectors per allocation unit, sectors per track, total sectors, sector size, and number of heads on the unit. The CDT, DCA, and CHT are modified to contain the correct channel and device information. The drive attribute registers are then constructed and the disk is marked online.

If the disk specified is a cartridge disk, the even subaddress must be specified when SYSINIT prompts for the device address. SYSINIT then sets up two UDTs and associated tables, one each for the fixed media and the removable media parts of the disk. Either portion may be used in the Volume Formatter step and as a response to SYSINIT's prompt for the system disk address.

After the tape and disk devices are configured, SYSINIT loops through the first seven UDT entries. The device address fields for offline entries are filled with X'FFFF'. This prevents UDT searches from finding the wrong entry.

After the UDT device address fields are filled, SYSINIT activates the Volume Formatter from the Master SDT.

SYSINIT contains a skip file control word which is used to locate the Volume Formatter. The control word consists of byte fields indicating the relative file positions on the tape. The word defaults to the values required for a user SDT and is modified during Phase I of SYSINIT if a Master SDT is being used. When the tape is positioned, SYSINIT reads the Volume Formatter load module preamble into the system buffer and performs a parameter task/run request activation with the bit indicator 'C.TAPACT' set. The bit variable informs H.REMM, which performs the activation sequence, that the preamble is in the system buffer and activation is from a tape. The bit C.SYSB is also set, which indicates to H.REMM that no swap file should be obtained. During the parameter task part of the activation, SYSINIT passes assignments for the system console and IPL device to the Volume Formatter. During the run request part, SYSINIT waits for completion of the Volume Formatter before continuing the activation sequence.

The activation call causes J.VFMT to be loaded, then placed, into the suspend queue. Control then passes back to SYSINIT. Before SYSINIT resumes J.VFMT, the tape must be positioned at the start of the correct system image. SYSINIT rewinds the tape, advances one record, then uses the skip file control word to advance the required number of files to reach the desired image. For example, MSTRALL would require no skip files where the MSTROUT would require two. After the tape is positioned, J.VFMT is resumed. The Volume Formatter copies the tape image of the MPX-32 operating system to disk. The copy becomes the default system image.

When SYSINIT is informed of J.VFMT's completion, the parameter send block associated with the run request is checked for completion errors. If an error is reported, the error is displayed on the system console and SYSINIT aborts.

After running J.VFMT, SYSINIT runs J.MOUNT, J.SWAPR, and the Volume Manager in the same manner. J.MOUNT mounts the system volume, J.SWAPR is the swapper, and the Volume Manager allows the restoration of saved files from the SDT.

7.4.3 Autodisk Subroutine

The autodisk subroutine determines the geometry of any F-class disk except memory disk. This subroutine is used by any program that communicates with an unmounted disk.

Entry Conditions

Calling Sequence

```
EXT     AUTODISK, AUTOFLAG
LW      R7, DISKSPEC
BL      AUTODISK
```

where:

DISKSPEC — contains a word describing the device to be verified. See the MPX-32 Reference Manual Volume I, Chapter 5, Resource Requirement Summary description.

AUTODISK AND AUTOFLAG — are defined labels in the autodisk subroutine.

Exit Conditions

Return Sequence

Registers

Normal Return

None

CC1 is zero

Abnormal Return

CC1 is set

R6 error types as follows:

Error Type	Description
1	error in disk assignment
2	I/O error reading track label zero
3	media verification pointer not in track label zero
4	I/O error reading media verification sector
5	SYSGEN attributes do not match actual disk parameters and ANY was not specified as the disk type
6	device not a disk
7	I/O error initializing an IOP disk
8	device inoperable
9	contents of first word of media verification sector invalid
10	formatted sector size is not 768 bytes
11	I/O error attempting to initialize disk processor
12	disk track/sector labels corrupted, reinitialize disk
13	SYSGEN configured handler does not match specified disk type

R7 error message transfer word address

Autodisk subroutine performs the following:

1. Verifies that the requested device is a nonfloppy disk.
2. Assigns the disk.
3. Reads track label zero.
4. Compares the device dependent parameters computed from the track label with those generated by SYSGEN. If the parameters match, the autodisk subroutine returns to the caller and indicates a successful match. If the parameters do not match, the autodisk subroutine continues processing.
5. Determines if the relevant operating system tables should be modified. There are two ways to determine this:
   - The total sectors field of the UDT is zero. This indicates the disk type was SYSGENed as ANY.
   - Bit two of AUTOFLAG is set.

   If neither of these conditions exists, error type 5 is generated.
6. Modifies the following table entries:

   UDT.SPT
   UDT.STA2
   UDT.SPAU
   UDT.NHDS
   UDT.SECS
   DCA.SCYL
7. Updates the drive attribute information (DATR). If a disk processor is used, there is no onboard RAM to hold the DATR. A 224-word buffer, containing the DATR information, is allocated following the interrupt fielding module. See H.XIOS in Volume II for details. The autodisk subroutine updates the buffer and places the updated DATR in the MPX-32 drive initialization list.

   If an IOP disk is used, there is an onboard RAM which holds the DATR. The IOP allows an initialize controller command to reload the DATRs without a preceding reset channel command. The updated DATR is also placed in the MPX-32 drive initialization list.

   It is necessary to update the MPX-32 drive initialization list as RESTART uses the MPX-32 copy of the DATR to construct the IOCL to load a new MPX-32 image.
8. Deallocates the device and returns to the caller.

7.4.4 Memory Disk

Each memory disk has an associated UDT and SMT. The UDT is constructed by SYSGEN in the same manner as for any other device. The SMT is constructed by SYSGEN in the same manner as for a memory partition, with the exception of SMT.PAGE. For a memory disk, it contains the following values:

value = -1 start of disk not specified at SYSGEN
value = -ve start of disk specified at SYSGEN, value is the negative of the specified starting map block.

If the DEAL parameter was specified in the SYSGEN DEVICE directive for a memory disk, bit UDT.MDAL in UDT.STA2 is reset. As a result, SYSINIT will not attempt to allocate memory for the memory disk. The memory can be allocated later by the OPCOM ONLINE directive. The value of SMT.PAGE retains the value explained above until the memory disk is marked ONLINE by OPCOM. After memory disk is marked online, the memory disk starting page number is stored by OPCOM into the shared memory table (SMT).

If the DEAL parameter was not specified (UDT.MDAL set), SYSINIT attempts to locate enough contiguous free memory for the memory disk. If the START parameter was specified in the SYSGEN DEVICE directive for the memory disk, bit UDT.MDST in UDT.STA2 is set. As a result, SYSINIT attempts to allocate the memory starting at the specified map block number.

If a starting map block was not specified, UDT.MDST is reset and SYSINIT allocates the memory wherever possible, starting at the high end of the presently configured memory. In either case, SYSINIT does not locate a single-ported memory disk in Multiprocessor Shared Memory System (MSMS) memory, nor does it locate a dual-ported memory disk in non-MSMS memory.

If memory disks or static partitions are defined in DRAM, (CONCEPT 32/2000 processors only), SYSGEN verifies that they reside fully within the DRAM memory. If these definitions are incorrect the following messages are displayed on the console and SYSGEN aborts with no image being produced.

```
***Memdiscname DOES NOT RESIDE COMPLETELY IN TYPE 'D' MEMORY
```

or

```
***partname DOES NOT RESIDE COMPLETELY IN TYPE 'D' MEMORY
```

A dual ported memory disk always has its memory allocated by SYSINIT. If SYSINIT fails to allocate memory for a memory disk, the shared memory table (SMT) is cleared for the memory disk that was attempting to allocate memory.

Provided that the memory for the disk was allocated successfully, SMT.PAGE is set to the starting page number of the memory disk. UDT.MDAL is set to indicate that the memory has been allocated.

7.5 Online RESTART

RESTART is a privileged task which simulates an IPL from the IOP console. RESTART can test a newly SYSGENed version of MPX-32, or replace the current default system image with a new image.

7.5.1 Activating

RESTART runs as a privileged, interactive TSM task. A user activating RESTART must be privileged. See M.KEY, MPX-32 Reference Manual Volume III, Chapter 10.

7.5.2 Required Input

RESTART accepts a pathname as an optional parameter of an activation request. For example:

```
RESTART@VOL1(SYSTEM)TEST.SYS
```

In this case, the image `TEST.SYS` contained in the system directory of volume `VOL1` is booted and `VOL1` becomes the system volume. If a volume other than the current system volume is specified in the pathname, the selected volume must be formatted as a system volume. That is, it must contain bootstrap code at sector 0.

If a pathname is not specified at activation, the default image on the current system volume is rebooted.

7.5.3 Processing

The functions performed by RESTART are as follows:

1. The TSM line buffer is checked for a user-supplied pathname. If one is specified, it is moved to a resource requirement summary (RRS) buffer to allow the file to be assigned. If one is not specified, a flag is set to indicate the default image should be used.

2. If a pathname is specified, the file is assigned. The unit definition table (UDT) address of the device it resides on is determined, and the first five sectors of that device, boot code and volume descriptor are read by space definition.

3. The image preamble is used to determine the size of the image and the system name. The name is compared with the user-supplied name. The size is placed in a table to be placed into the boot code, and an IOCD list is constructed that will be used to read the new image.

4. The required drive attribute registers are constructed.

5. At this point, the operator is prompted before proceeding with reboot. If the request was for the default system, a request for reboot is issued. If a pathname was specified, the user is also asked if this image should be made the default system image. If the user replies yes to the default system option, the volume descriptor on the target volume is updated with the new system image definition. This allows IOP console restarts to locate the correct image. In addition, the resource descriptor for the new image is marked as not deletable. This prevents inadvertent relocation of the default system image by SYSGEN, SAVE, RESTORE, COPY, etc. which causes an IOP console IPL to fail. The previous default image, if not the SDT image, has this flag reset so it may be deleted.

6. The CPU scratchpad IPL device address location is updated to reflect the new IPL device. This location is used by the bootstrap code as it sets up scratchpad to perform all of its I/O from logical channel 08.

7. A simulated system reset is performed. All interrupts are disabled and channels are reset.

8. The bootstrap code read from the target disk is updated by RESTART and moved into low memory.

9. Control is passed to the bootstrap code, which reads the image and transfers control to the system initializer program, SYSINIT.

8 Internal Processing Unit (IPU)

8.1 Overview

The IPU is a parallel processor connected directly to the SelBUS. Synchronization between the CPU and the IPU is maintained by one CPU trap and sixteen IPU traps. The traps and default trap vector locations are shown in Table 8-1.

Task execution in the IPU is transparent to the user. Scheduling for the IPU is accomplished by MPX-32 with no user intervention. However, IPU biasing or inhibiting can be used to maximize the performance of processor or I/O bound tasks.

When an IPU is configured in a system, two modules must be included in the resident operating system to perform IPU biased scheduling and to provide trap handlers for IPU related traps. These modules are H.CPU and H.IPU.

IPU accounting can be performed by a second interval timer (RTOM). IPU execution time and idle time are tabulated by the resident handler, H.IPUIT.

When both a CPU and an IPU are configured, memory read and lock is automatically enabled. When one processor is accessing a memory location, the other processor is prohibited from accessing that location. Memory locations are unlocked when they are not being accessed.

8.1.1 IPU - Memory Interface

The IPU can address all locations of physical memory. Task loading and initialization are performed by the CPU before the task is queued for IPU execution. The task service area (TSA) includes pointers to all physical map blocks used by the task, allowing the IPU to remap to the task's address space. This mechanism allows the CPU and IPU to be coordinated in use of memory.

8.1.2 IPU - CPU Interface

Table 8-1 shows IPU related traps and default trap vector locations. The trap vectors may be stored at alternative locations if the IPU scratchpad is set up appropriately.

Table 8-1
IPU Trap Structure

IPU Trap Vector Location	CPU Trap Vector Location	Trap Condition
20	80	power fail
24	84	power on/autostart
28	88	memory parity
2C	8C	nonpresent memory
30	90	undefined instruction
34	94	privilege violation
38	98	supervisor call
3C	9C	machine check
40	A0	system check
44	A4	map fault
48		undefined IPU instruction
4C		CPU issued SIPU instruction
	AC	IPU issued SIPU instruction
50	B0	address specification error
54	B4	console attention
58	B8	privilege mode halt
5C	BC	arithmetic exception
60	C0	cache fault

8.2 Task Scheduling and Execution

8.2.1 Task Biasing

There are three scheduling options related to IPU task execution: IPU biased, CPU only, and unbiased. IPU biased tasks are scheduled for IPU execution whenever possible. Some SVCs are executed directly by the IPU, but most SVCs and all privileged instructions cause a task to be returned to the CPU for execution. An IPU biased task is rescheduled for IPU execution at the instruction following the instruction which caused the trap.

A CPU only task is never scheduled for IPU execution.

An unbiased task can be scheduled in the CPU or the IPU depending on available resources. The CPU scheduler, S.EXEC20, is responsible for selecting unbiased tasks for IPU or CPU execution.

8.2.2 Standard CPU/IPU Scheduling

There are two head cell addresses used by the operating system to control IPU task execution: C.CIPU and C.RIPU. The currently executing IPU task is linked to the head cell, C.CIPU. C.RIPU is a standard linked list head cell containing the dispatch queue (DQE) addresses of all IPU biased tasks awaiting IPU execution.

Tasks are linked to the above state queues by the CPU, and control is passed to the IPU after the task has been linked to the C.CIPU head cell.

8.2.3 Optional CPU/IPU Scheduling

IPU-biased tasks are not automatically queued on the RIPU list as they are in the standard CPU/IPU scheduler. The RIPU list is used only when the task in the CPU is replacing the IPU task.

8.2.4 Standard Scheduling of IPU-Biased Tasks

Tasks are biased to the IPU by specifying OPTION IPUB at either catalog or run time, or by calling the dynamic IPU bias service, M.IPUBS. Biased tasks are queued on the IPU ready-to-run queue by priority with the highest priority task at the head. Tasks linked to C.RIPU are executed by the IPU ahead of higher priority unbiased tasks. Task replacement of IPU biased tasks occurs when a higher priority task is linked to C.RIPU or the currently executing IPU task executes an instruction causing control to be returned to the CPU. If a biased task is linked to C.RIPU while a higher priority unbiased task is currently executing in the IPU, the higher priority task continues to run in the IPU.

8.2.5 Optional Scheduling of IPU-Biased Tasks

The optional CPU/IPU scheduler is enabled by the SYSGEN DELTA directive. This directive replaces system modules H.EXEC and H.CPU with H.EXEC2 and H.CPU2. The optional scheduling approach does not queue IPU-biased tasks on the RIPU list as the standard scheduler does.

Instead, all tasks are linked to their appropriate ready-to-run state chains. When the delta value is zero, scheduling is performed on both processors according to the tasks' original priorities. When the delta value is greater than zero and less than 55, the value is subtracted from the original priority of the IPU-biased task to create a new priority. The new priority is used during IPU scheduling and when the IPU-biased task needs the CPU for system service execution. When IPU-biased tasks run on the CPU, the new priority does not apply.

8.2.6 Scheduling Unbiased Tasks

When an IPU biased task is not linked to C.CIPU or C.RIPU, IPU task selection proceeds with unbiased tasks. The ready state queues are searched for the first eligible task, starting with the highest priority real-time task. The conditions for IPU eligibility are:

- task is not CPU only
- task is not inhibited for IPU execution because it has executed an instruction not available to the IPU
- there are no run requests or messages outstanding against the task
- there are no system action requests outstanding against the task

If all the above conditions are met, the task is linked to C.CIPU and control is passed to the IPU. If there are no ready-to-run tasks meeting all of the above conditions, the IPU remains idle.

8.2.7 Scheduling CPU Only Tasks

Tasks with OPTION CPUO specified are never scheduled for IPU execution. Typically, these tasks are I/O bound.

8.2.8 IPU Task Execution

A task ceases execution in the IPU when one of the following events occur:

- the IPU encounters a system service request (SVC or CALM). Some SVCs are executed by the IPU and control is not returned to the CPU.
- the IPU encounters an exceptional or error condition; for example, privilege violation, undefined instruction.
- the CPU executes an SIPU instruction.

Tasks running with batch priorities (55 through 64) are not subject to time distribution while being executed in the IPU.

8.3 IPU Executive Module Description

The resident module H.IPU is responsible for initializing the IPU, dispatching tasks linked to the IPU current state queue, C.CIPU, handling all IPU traps, and returning control to the CPU on exceptional or error conditions.

8.3.1 Entry Point 1 - IPU Executive

This entry point is used when the CPU issues an SIPU instruction. The first time the trap occurs, a branch is made to the initialization subroutine, S.IPU2. The online restart in progress flag is checked and, if set, a simulated IPU reset is performed. If the IPU is currently executing a task, the CPU is requesting a context switch in the IPU. The context of the task is preserved and the IPU issues an SIPU instruction to allow the CPU to link the new task to C.CIPU.

If there is not a current task, the IPU dispatches control to the task linked to C.CIPU.

If there is a current task but the IPU is executing within H.IPU, the IPU is in the process of handling a trap and is allowed to continue.

8.3.2 Entry Point 2 - Undefined IPU Instruction

This entry point sets the IPU inhibit bit in the current task's DQE, which corrects the PSD to point to the last instruction executed by the task, and returns control to the CPU to re-execute the instruction.

8.3.3 Entry Point 3 - Memory Parity Error

This entry point sets the IPU inhibit bit in the current task's DQE, which corrects the PSD to point to the last instruction executed by the task, and returns control to the CPU to re-execute the instruction.

8.3.4 Entry Point 4 - Nonpresent Memory

This entry point causes the current task to be aborted. The error code is stored in the DQE and the abort request bit is set before returning control of the task to the CPU.

8.3.5 Entry Point 5 - Undefined Instruction

This entry point causes the current task to be aborted. The error code is stored in the DQE and the abort request bit is set before returning control of the task to the CPU.

8.3.6 Entry Point 6 - Privilege Violation

This entry point causes the current task to be aborted. The error code is stored in the DQE and the abort request bit is set before returning control of the task to the CPU.

8.3.7 Entry Point 7 - Map Fault

This entry point causes the current task to be aborted. The error code is stored in the DQE and the abort request bit is set before returning control of the task to the CPU.

8.3.8 Entry Point 8 - SVC Trap Handler

This entry point contains a secondary vector table for the IPU SVCs. SVC types 0, 3, and 5 through 15 are returned to the CPU for processing. The PSD is corrected to point to the SVC instruction, the current context is saved, and control is returned to the CPU.

Some SVC types 1 and 2 are executable directly by the IPU. These SVCs have bit 1 in the SVC table set. The SVC number is retrieved from the trap status word and the SVC table entry is checked. If the SVC is not executable by the IPU, control is returned to the CPU as for an SVC type 0. If the service is executable by the IPU, a PSD is constructed and control is transferred to the correct SVC processor.

An SVC type 4 is the mechanism for returning from an SVC. This entry point is reached by the macro M.IPURTN, which determines if any registers are to be returned and issues an SVC type 4. The registers and PSD are popped from the stack and control is returned to the task.

8.3.9 Entry Point 9 - Arithmetic Exception Trap Handler

When an arithmetic exception occurs, the arithmetic exception bit in the current task's TSA is set and may be tested by the M.TSTE service. The return registers are set to the following values, depending on the cause of the exception:

underflow	zero
positive overflow	maximum positive value
negative overflow	maximum negative value

8.3.10 Entry Point 10 - Privilege Mode Halt

The function this entry point performs depends on where an error occurs. If an error occurs in a task, an abort request is set up for the task and control is returned to the CPU. If an error occurs in the operating system, the registers are set up as for the CPU M.KILL macro and stored in the buffer HLT.REG within H.EXEC. The register contents are as follows:

Register	Meaning
0	old PSD word 1
1	old PSD word 2
2	address of the instruction causing the error
3	instruction causing the error
4	trap status word
5	ASCII code of reason for trap, for example, HT02
6	address of register save block
7	ASCII 'TRAP'

A flag is set requesting the IPU to be marked offline so no further tasks are scheduled for the IPU.

8.3.11 Entry Point 11 - Address Specification

The function this entry point performs depends on where an error occurs. If an error occurs in a task, an abort request is set up for the task and control is returned to the CPU. If an error occurs in the operating system, the registers are set up as for the CPU M.KILL macro and stored in the buffer HLT.REG within H.EXEC. The register contents are as follows:

Register	Meaning
0	old PSD word 1
1	old PSD word 2
2	address of the instruction causing the error
3	instruction causing the error
4	trap status word
5	ASCII code of reason for trap, for example, AD01
6	address of register save block
7	ASCII 'TRAP'

A flag is set to request the IPU be marked offline so no further tasks are scheduled for the IPU.

8.3.12 Entry Point 12 - Cache Fault

The function this entry point performs depends on where an error occurs. If an error occurs in a task, an abort request is set up for the task and control is returned to the CPU. If an error occurs in the operating system, the registers are set up as for the CPU M.KILL macro and stored in the buffer HLT.REG within H.EXEC. The register contents are as follows:

Register	Meaning
0	old PSD word 1
1	old PSD word 2
2	address of the instruction causing the error
3	instruction causing the error
4	trap status word
5	ASCII code of reason for trap (for example, CP01)
6	address of register save block
7	ASCII 'TRAP'

A flag is set to request the IPU be marked offline so no further tasks are scheduled for the IPU.

8.3.13 Entry Point 13 - Machine Check

The function this entry point performs depends on where an error occurs. If an error occurs in a task, an abort request is set up for the task and control is returned to the CPU. If an error occurs in the operating system, the registers are set up as for the CPU M.KILL macro and stored in the buffer HLT.REG within H.EXEC. The register contents are as follows:

Register	Meaning
0	old PSD word 1
1	old PSD word 2
2	address of the instruction causing the error
3	instruction causing the error
4	trap status word
5	ASCII code of reason for trap, for example, MC01
6	address of register save block
7	ASCII 'TRAP'

A flag is set to request the IPU be marked offline so no further tasks are scheduled for the IPU.

A second trap within a task causes the IPU to halt as above since this is assumed to be a hardware failure.

8.3.14 Entry Point 14 - System Check

The function this entry point performs depends on where an error occurs. If an error occurs in a task, an abort request is set up for the task and control is returned to the CPU. If an error occurs in the operating system, the registers are set up as for the CPU M.KILL macro and stored in the buffer HLT.REG within H.EXEC. The register contents are as follows:

Register	Meaning
0	old PSD word 1
1	old PSD word 2
2	address of the instruction causing the error
3	instruction causing the error
4	trap status word
5	ASCII code of reason for trap, for example, SC01
6	address of register save block
7	ASCII 'TRAP'

A flag is set to request the IPU be marked offline so no further tasks are scheduled for the IPU.

A second trap within a task causes the IPU to halt as above since this is assumed to be a hardware failure.

8.3.15 Entry Point 15 - Power Fail Trap

This trap saves the general purpose and base registers, and saves the IPU scratchpad key in memory location X'6D4'. This provides the required parameters for a power-up auto-restart when power is restored to the system. The privileged mode halt trap is disabled and the IPU halts.

8.3.16 Subroutine S.IPU1 - Perform Stack Push

This subroutine pushes the registers and program status doubleword (PSD) of the current task into the next stack frame as defined by T.REGP. This is a clean-up activity in preparation for the return of task control to the CPU.

Calling Sequence

```
LA      R2,addr
LD      R6,psd
BL      S.IPU1
```

addr is the address of the register save block

psd is the current task PSD

Exit Sequence

```
TRSW    R0
```

8.3.17 Subroutine S.IPU2 - IPU Initialization

This subroutine initializes the IPU by storing the master process list (MPL) address in the IPU scratchpad, loading the shared map registers for the operating system, setting up the address of the IPU history buffer, and enabling the privilege mode halt trap.

Calling Sequence

```
BL      S.IPU2
```

Exit Sequence

```
TRSW    R0
```

8.3.18 Subroutine S.IPU3 - Terminate IPU Execution

This subroutine generates an SIPU instruction which causes a trap in the CPU to indicate the IPU has finished processing.

Calling Sequence

```
BL      S.IPU3
```

Exit Sequence

None

8.3.19 Subroutine S.IPU4 - Generate IPU History Buffer

This subroutine maintains a circular buffer containing information on the last twenty traps in the IPU. The buffer contains the task name, PSD at the time of the trap, and a code of the reason for the trap. Each entry is four words long in the following format:

Word	Meaning
0	PSD word 1
1	PSD word 2 with bits 10 - 14 containing function code as follows:
2-3	taskname

Code	Meaning
2	nonpresent memory
3	undefined instruction
4	privilege violation
5	SVC type 0 or 3
6	SVC type 1
7	SVC type 2
8	machine check
9	system check
10	map fault
11	unidentified IPU instruction or memory parity error
12	start IPU
13	address specification
14	privilege mode halt
15	arithmetic exception
16	cache fault
17	SVC type 4

8.4 IPU Auto Start Trap Processor - H.IPUAS

The interrupt or trap signal occurs at priority level X'01'. This trap occurs during the power up sequence, provided the following operating conditions are met:

1. The CPU, IPU, and system software traps are enabled.
2. The CPU scratchpad image is contained in dedicated memory locations X'300' through X'6FC'.
3. The memory scratchpad image contains the CPU and IPU scratchpad keys.
4. A successful power down trap has been executed.
5. The integrity of the memory has been preserved. The system memory configuration must be core and/or MOS memory with a battery backup.

If any of the conditions are not met, an automatic trap halt is exectued.

If the conditions are met, H.IPUAS disables the privileged halt trap and halts the IPU.

The H.IPUAS trap handler can be replaced with a user-supplied routine as follows:

1. Specify the new trap in the SYSGEN SYSTRAP directive.
2. At SYSGEN, the address of the user-supplied routine's trap context block must be saved in memory location X'24'.
3. The five operating conditions must be met.

8.5 IPU Task Scheduler - H.CPU/H.CPU2

Two IPU task schedulers are provided. H.EXEC and H.CPU are the standard IPU scheduling modules; H.EXEC2 and H.CPU2 are the optional IPU scheduling modules. The differences that apply to H.CPU2 are noted in the following sections.

8.5.1 Entry Point 1 - Field IPU Halt

This entry point fields the IPU 'HALT' trap. It schedules IPU tasks based on the following criteria:

1. If the head cell count of the IPU current state is zero, the IPU is idle. Entry point two schedules the current task.
2. If the head cell count is greater than zero and the inhibit IPU flag is set, entry point two unlinks the current task, relinks it at its base priority state, then schedules the new IPU task.
3. If the head cell count is greater than zero and the inhibit IPU flag is reset, entry point two unlinks the task from the current state, relinks it to the IPU request queue, then schedules the new IPU task.

Note: For the optional scheduler, H.CPU2 never links the task to the IPU request queue. The task is always linked to the ready to run queue. If it is a real-time task, it is linked at its base priority minus its delta value. All other tasks are linked at their base priority.

8.5.2 Entry Point 2 - Schedule IPU Biased Tasks

If tasks are queued to the IPU request queue, this entry point unlinks the highest priority task, relinks it to the IPU current state, and calls the IPU start subroutine. If there are no tasks on the IPU request queue, this entry point goes to entry point 3 for unbiased task selection.

8.5.3 Entry Point 3 - Schedule Unbiased Tasks

This entry point begins at the real-time state queue to select an IPU candidate. It tests each encountered task for IPU eligibility as follows:

1. IPU inhibit flag = reset
2. CPU only flag = reset
3. no system actions (DQE.SAIR = 0)
4. no run requests (DQE.RTI = 0)
5. execution address is not in the operating system

This entry point continues testing each lower priority task until an eligible candidate is found. It unlinks that task from its ready state and relinks it to the IPU current state. This entry point then calls the start IPU subroutine. If no eligible task is found, the IPU remains idle.

8.5.4 Subroutine S.CPU1 - Link Task to IPU Request State

This subroutine links a task to the IPU request queue. If the task is higher priority than the current IPU task, IPU task replacement takes place.

8.5.5 Subroutine S.CPU2 - IPU Eligibility Test

This subroutine contains the tests for task eligibility to run in the IPU. The items checked are:

- is the task executing in the monitor (DQE.OSD)
- is the task CPU only (DQE.IPUR)
- is the task IPU inhibited (DQE.IPUH)
- is a system action request pending against the task (DQE.SAIR)

8.6 IPU Accounting Module Descriptions

8.6.1 Entry Point 1 - Field Interval Timer Interrupt

This entry point fields the IPU accounting interval timer interrupt. If there is a current IPU task, it updates a local IPU execution time accumulator. If the IPU is idle, the IPU idle time accumulator (C.IDLA1) is updated. The timer is then reset for one second and the handler is exited.

8.6.2 Subroutine S.IPUIT1 - Perform Accounting After IPU Trap

This subroutine is called by H.CPU after an IPU HALT trap is fielded. It updates the TSA of the current IPU task with the accumulated IPU execution time and resets the interval timer to accumulate idle time.

8.6.3 Subroutine S.IPUIT2 - Perform Accounting Before Starting IPU

This subroutine is called by H.CPU just prior to calling S.EXEC80 to start the IPU. It updates the IPU idle time accumulator and resets the timer to accumulate execution time.

8.7 IPU SYSGEN Directives

To generate a system with an IPU configured, the following SYSGEN directives must be used:

```
//HARDWARE
/PARAMETERS
MACHINE=type
IPU
  .
  .
  .
/TRAPS
```

If an IPU accounting interval timer is present, the following SYSGEN directives should be used:

```
/INTERRUPTS
PRIORITY=xx,RTOM=(channel,subaddress),PROGRAM=H.IPUIT,INTV
```

Note: It is recommended that *xx* be 5E. However, if a scientific accelerator is also configured, priority 3F should be used for the IPU accounting interval timer.

8.8 SVCs Executable by an IPU

Certain SVCs are executable directly by the IPU. When modifying these SVCs or when writing new services which are executed by the IPU, the following guidelines should be followed:

- The service must not be called by SYSGEN or J.SWAPR.
- The M.GTSAD macro must be called to load the TSA address
- T.PRNO must be used instead of C.CURR.
- M.IPURTN must be used instead of M.RTRN.
- The macro M.SVCP must be used in the SYSGEN initialization entry point to set bit 1 in the SVC table.

9 Converting Modules for Extended MPX-32

9.1 General Information

The modules that can operate in extended MPX-32 on delivery are:

- Resource Allocator (H.ALOC)
- Executive Subroutine Module (H.EXSUB)
- File System Executive (H.FISE)
- Memory Management Module (H.MEMM)
- System Services (H.MONS)
- Resource Management Module (H.REMM)
- Resident Execution Services Module (H.REXS)
- Task Management Module (H.TAMM)
- Volume Management Module (H.VOMM)

Any of these modules that were modified by the user can be converted to operate in extended MPX-32. Any user-created modules that meet the programming considerations can also be converted for extended MPX-32.

Good candidates for conversion to extended MPX-32 are:

- user SVC's
- user modules

The requirements for converting a module for extended MPX-32 are in the Programming Considerations section. The candidates that cannot meet the programming considerations and cannot be converted to execute extended MPX-32 are:

- tasks
- interrupt handlers (i.e., H.F8XIO)
- trap handlers (i.e., H.IP00, H.IP03)
- code that is callable from an interrupt level (i.e., real-time clock, H.IPCL, H.IPIT, H.IPUIT, H.MEMM2, H.XIOS)

9.2 Programming Considerations

Existing user modules, such as I/O handlers and service routines, can execute without code changes if positioned in the nonextended MPX-32. If user modules are positioned in the extended MPX-32, the following guidelines must be observed:

- Insert SSECT directives around source lines that contain dynamic address fields in the source module. This causes the assembler to generate the required sectioned object code.
- Use macro calls (MBR_*xxx*) and Macro Assembler directives designed for converting modules to extended MPX-32. For detailed description of the macro calls, see the Macros for Extended MPX-32 section. For detailed information on the Macro Assembler directives, see the Macro Assembler and Extended MPX-32 section.
- Reassemble the module with logical file code PRE assigned to MPX_EXT during assembly of code. Then, the modules can be compressed into OH.32_E, the object file assigned to OBR, for input to SYSGEN.

Extended MPX-32 modules cannot:

- Include code that contain:
  - subroutines that are callable from an interrupt level
  - routines that depend on logical equals physical addressing
  - nontransparent instruction addressing, such as passing return addresses to external routines or providing end-action addresses in file control blocks
- Use variable points of return when designing external references. This can cause rapid expansion of the size of the adaptive code.
- Use indirect addressing code that will run in the extended code region.
- Use nonbase mode instructions, such as CEA, LEA, or SEA.
- Run unmapped.
- Use data parameter lists in which a branch and link is followed by data or address constant words in-line for external reference.
- Use code or data that is accessed by the SYSGEN initialization entry point of another nonextended MPX-32 module.

A module containing any of the above can be split into two modules — one that is moved to extended MPX-32 and one that remains in nonextended MPX-32.

9.3 Macros for Extended MPX-32

The macros for extended MPX-32 allow existing user modules and service routines to run in extended MPX-32. These macros generate extended or nonextended code depending on the state of the BOPT_MPX flag. After the state of BOPT_MPX is tested, code embedded in the macro sets or resets assembler option 16 with Macro Assembler OPTR and OPTS directives to generate the appropriate code.

BOPT_MPX is set true by assigning logical file code PRE to MPX_EXT. Then, code embedded in the macro sets option 16 with the assembler OPTS directive. This causes the macro to generate extended code.

BOPT_MPX is set false by assigning logical file code PRE to MPX_NON. Then, code embedded in the macro resets option 16 with the assembler OPTR directive. This causes the macro to generate nonextended code.

By default, BOPT_MPX is false generating nonextended code.

The following macros convert existing modules for extended MPX-32:

Macro	Description
MBR_BL	branch and link
MBR_BU	branch unconditional
MBR_B*xx*	conditional branch
MBR_DBG	calls the system debugger
MBR_DEF	identifies linkage symbols that can be referenced by another program or subroutine
MBR_DSCT	directs data into the DSECT
MBR_ENT	generates the adaptive sequence (ADP_MPX), if required, for nonextended modules to reference routines in extended MPX-32
MBR_EXT	identifies linkage symbols that are entry points or subroutines in another program, but are referenced by this program
MBR_INIT	tests the state of the BOPT_MPX macro flag and sets Macro Assembler option 16 appropriately
MBR_OFFS	specifies offset mode of operation to SYSGEN
MBR_REL	specifies relative mode of operation to SYSGEN
MBR_SSCT	returns to a local code section after an MBR_DSCT has been specified.
MBR_TRSW	transfer register to program status word

9.3.1 MBR_BL - Branch and Link Macro

The MBR_BL macro tests the state of the BOPT_MPX flag. If BOPT_MPX is true, the assembler generates the adaptive sequence required to branch and link to a module in the opposite mode of execution. See Figure 2-1 and 2-2. If BOPT_MPX is false, the assembler BL instruction is generated.

Syntax

MBR_BL <*label*>

This adaptive sequence is generated when a nonextended MPX-32 module branches and links to a subroutine in an extended MPX-32 module. Boldface type represents base mode instructions.

```
        EXT     TARGET
ORG     BL      TARGET
*Subroutine Return Site
```

(or)

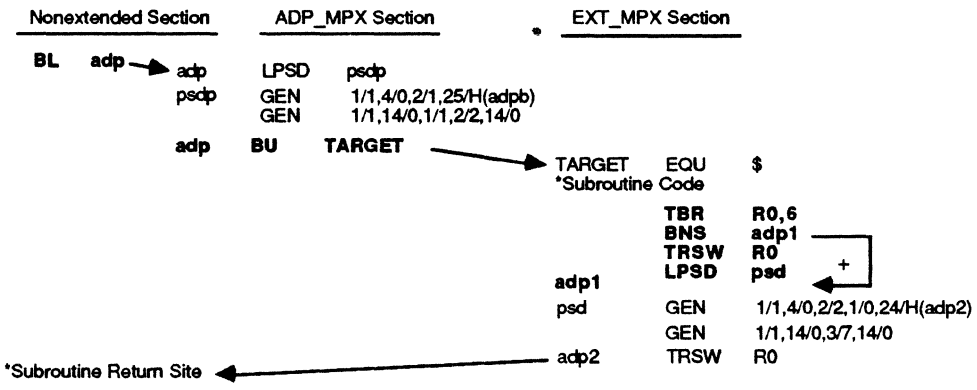
```
        MBR_EXT   TARGET
ORG     MBR_BL    TARGET
*Subroutine Return Site
```

When BOPT_MPX is false, both examples generate the same adaptive sequence. The second subroutine call example shows a code sequence in a source module that has been converted to execute in extended MPX-32 depending on the setting of BOPT_MPX.

```
        MBR_DEF   TARGET
        MBR_ENT   TARGET
*Subroutine Code
        MBR_TRSW  R0
```

Example of a code sequence in a subroutine module that has been converted to execute in extended MPX-32 depending on the setting of BOPT_MPX. When assembled with BOPT_MPX set true, this example generates an extended MPX-32 object module.

The following information shows the placement of instructions and the order of execution:



* Indicates a comment line.
\+ Although this instruction is present in this example, it is not executed.

T1027

Figure 9-1
Adaptive Sequence Generated By a Branch and Link From a Nonextended to an Extended MPX-32 Module for Extended MPX-32

This adaptive sequence is generated when an extended MPX-32 module branches and links to a subroutine in a nonextended MPX-32 module. Boldface type represents base mode instructions.

```
        MBR_EXT   TARGET
        MBR_BL    TARGET
*Subroutine Return Site
```

Subroutine call example assembled with BOPT_MPX set true to generate an extended MPX-32 object module.

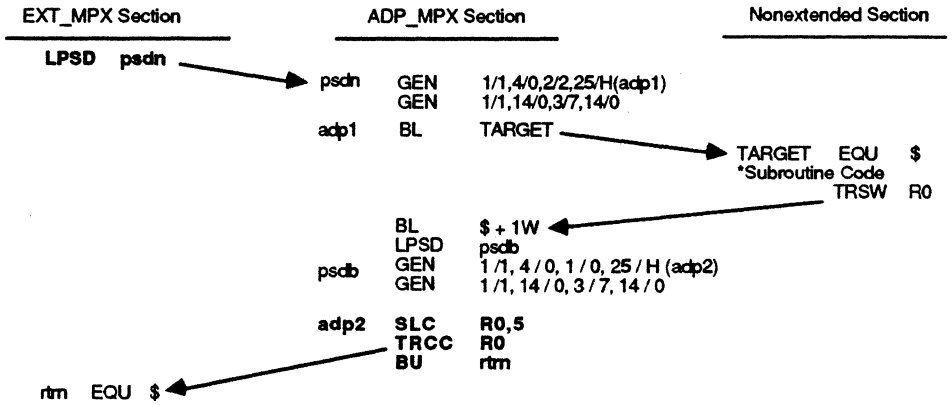
```
        DEF       TARGET
TARGET  EQU       $
*Subroutine Code
        TRSW      R0
```

(or)

```
        MBR_DEF   TARGET
        MBR_ENT   TARGET
*Subroutine Code
        MBR_TRSW  R0
```

When BOPT_MPX is false, both examples generate the same adaptive sequence. The example shows a source module that has been converted to execute in extended MPX-32 depending on the setting of BOPT_MPX.

The following information shows the placement of instructions and the order of execution:



* Indicates a comment line.

Figure 9-2
Adaptive Sequence Generated By a Branch and Link From Extended to a Nonextended MPX-32 Module

9.3.2 MBR_BU - Branch Unconditional Macro

The MBR_BU macro branches unconditionally to externally defined labels that are in the opposite mode of execution (i.e., the program is in extended MPX-32 and the branch is to a nonextended section of code).

The MBR_BU macro also tests the state of the BOPT_MPX flag. If BOPT_MPX is true, MBR_BU generates:

- control object records for SYSGEN with the code for an adaptive sequence if the branch target is in nonextended MPX-32. See Figures 2-3 and 2-4.
- a BU instruction if the branch target is in EXT_MPX.

If BOPT_MPX is false, a BU instruction is generated.

To convert existing modules for extended MPX-32, replace appropriate unconditional branch instructions with the MBR_BU macro.

If the label branched to is internal to the module, use the assembler BU instruction. The MBR_BU macro is not required because adaptive sequences are not needed.

Syntax

BMBR_BU *<label>*

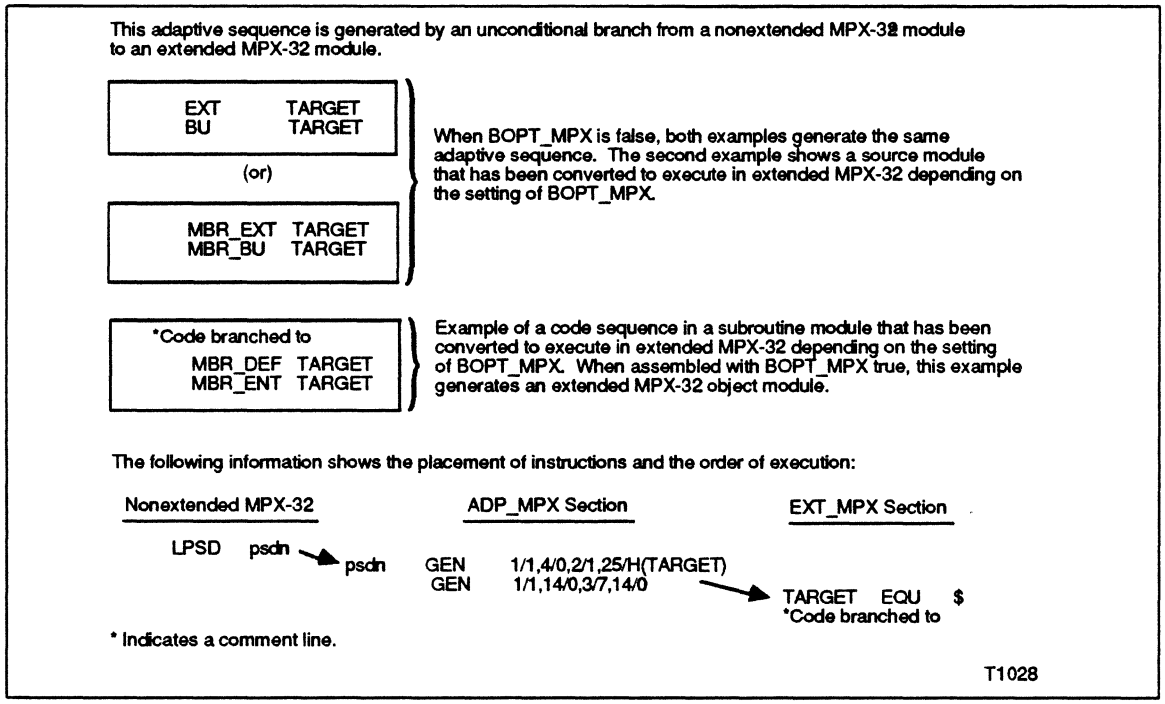


Figure 9-3
Adaptive Sequence Generated By an Unconditional Branch From Nonextended to an Extended MPX-32 Module

This adaptive sequence is generated by an unconditional branch from an extended MPX-32 module to a nonextended MPX-32 module.

```
        MBR_EXT   TARGET
        MBR_BU    TARGET
```

Example of code sequence in a subroutine module that has been converted to execute in extended MPX-32 depending on the setting of BOPT_MPX. When assembled with BOPT_MPX set true, this example generates an extended MPX-32 object module.

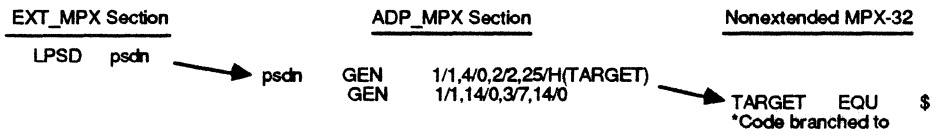
```
        DEF       TARGET
TARGET  EQU       $
*Code branched to
```

(or)

```
        MBR_DEF   TARGET
        MBR_ENT   TARGET
*Code branched to
```

When BOPT_MPX is false, both examples generate the same adaptive sequence. The second example shows a source module that has been converted to execute in extended MPX-32.

The following information shows the placement of instructions and the order of execution:



* Indicates a comment line

T1029

Figure 9-4
Adaptive Sequence Generated By an Unconditional Branch From Extended to a Nonextended MPX-32 Module

9.3.3 MBR_B*xx* - Conditional Branch Macro

The MBR_B*xx* macros conditionally branch to externally defined labels that are in the opposite mode of execution (i.e., the program is in extended MPX-32 and the branch is to a nonextended section of code).

MBR_B*xx* tests the state of the BOPT_MPX flag. If BOPT_MPX is true, the macro generates:

- control object records for SYSGEN with code for an adaptive sequence if the branch target is in nonextended MPX-32. See Figures 9-5 and 9-6.
- a Bxx instruction if the branch target is in EXT_MPX.

If BOPT_MPX is false, the assembler directive B*xx* is generated. To convert existing modules for extended MPX-32, replace appropriate conditional branch instructions with the corresponding MBR_B*xx* macro. See Table 9-1.

If the label that is branched to is internal to the module, use the assembler directive B*xx*. The MBR_B*xx* macro is not required because adaptive sequences are not needed.

Syntax

MBR_B*xx* <*label*>

Table 9-1
Conditional Branch Macros for Extended MPX-32

Macro	Assembler Extended Mnemonic Code	Machine Instruction	Description
MBR_BEQ replaces	BEQ or	BCT 4,*m*,*x*	Branch if equal to
MBR_BGE replaces	BGE or	BCT 5,*m*,*x*	Branch if greater than or equal to
MBR_BGT replaces	BGT or	BCT 2,*m*,*x*	Branch if greater than
MBR_BLT replaces	BLT or	BCT 3,*m*,*x*	Branch if less than
MBR_BNE replaces	BNE or	BCF 4,*m*,*x*	Branch if not equal to
MBR_BNS replaces	BNS or	BCF 1,*m*,*x*	Branch if not set
MBR_BS replaces	BS or	BCT 1,*m*,*x*	Branch if set

For the machine instructions, *m* indicates the memory address and *x* indicates the index register.

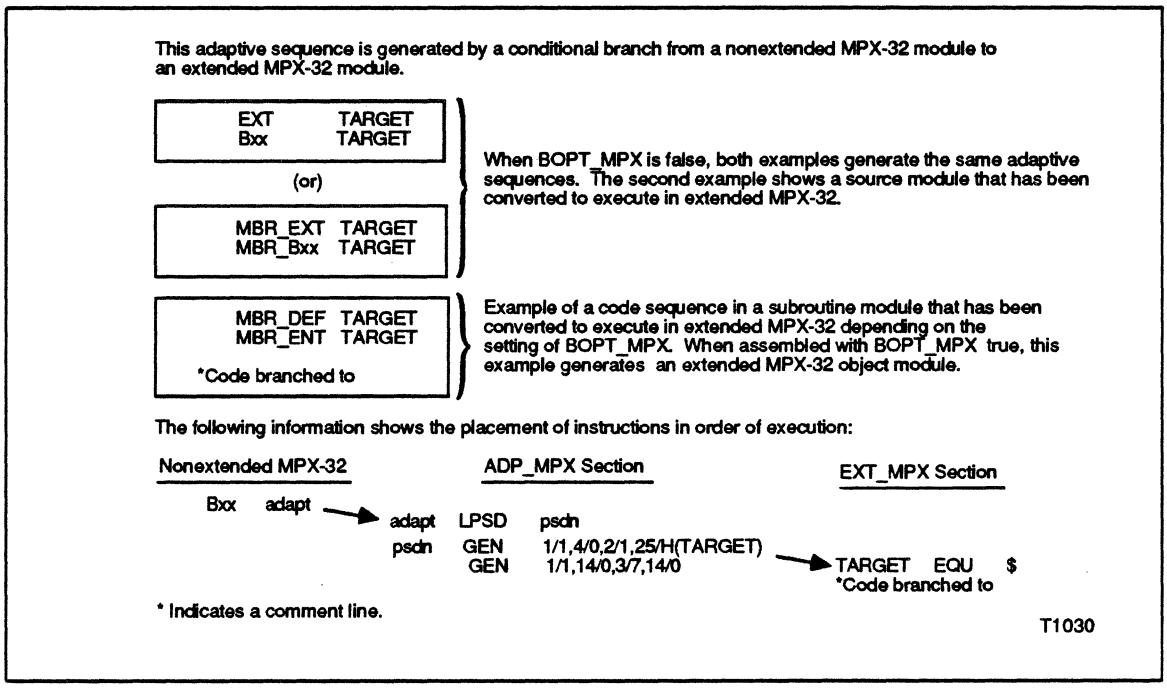


Figure 9-5
Adaptive Sequence Generated By a Conditional Branch From a Nonextended to an Extended MPX-32 Module

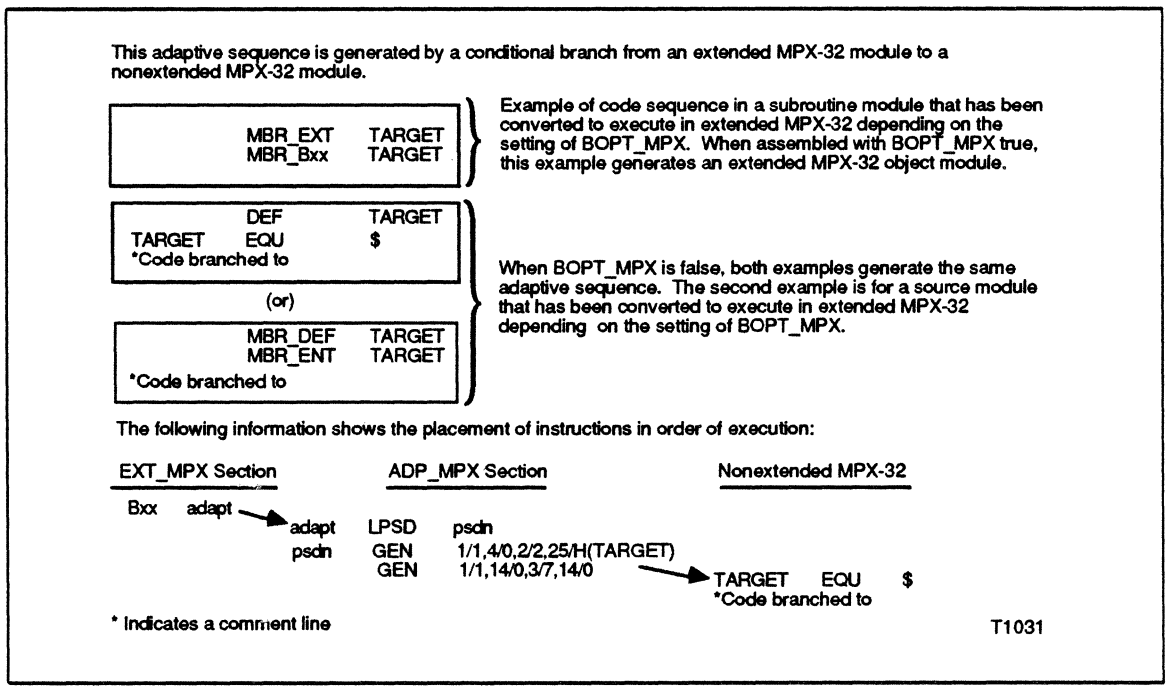


Figure 9-6
Adaptive Sequence Generated By a Conditional Branch From an Extended to a Nonextended MPX-32 Module

9.3.4 MBR_DBG - Calls to System Debugger Macro

The MBR_DBG macro calls the system debugger from the target extended module. This macro references the system debugger extended code entry within the H.DBUG1 module at label BR.DBUG.

MBR_DBG tests the state of BOPT_MPX flag. If BOPT_MPX is true, MBR_DBG generates he following assemble statements:

```
SEXT    BR.DBUG
BL      BR.DBUG
```

H.DBUG1 contains code to change from the extended mode to the nonextended mode in order for the system debugger to properly execute its nonextended code.

If BOPT_MPX is false, MBR_DBG generates an assemble "BL *C.DEBUG" statement.

Syntax

MBR_DBG

9.3.5 MBR_DEF - Identify Linkage Symbols Macro

The MBR_DEF macro identifies linkage symbols that are within a program and can be referenced by another program or subroutine. This macro tests the state of the BOPT_MPX flag. If BOPT_MPX is false, a Macro Assembler DEF is generated. If BOPT_MPX is true, a Macro Assembler SDEF is generated. The SDEF directive is used by the assembler to provide linkages for EXT_MPX.

Syntax

MBR_DEF *label*

9.3.6 MBR_DSCT - DSECT Data Separation Macro

The MBR_DSCT macro specifies that the code and data following this macro is to be placed in the DSECT section. Data and variable constants can be separated for inclusion in the DSECT section.

MBR_DSCT tests the state of the BOPT_MPX flag. If BOPT_MPX is true, MBR_DSCT generates an assembler SSECT DSECT directive. If BOPT_MPX is false, no assembler code or directive is generated and the next assembler line is processed.

Syntax

MBR_DSCT

9.3.7 MBR_ENT - Extended Code Routine Entry Macro

The MBR_ENT macro must define labels in an extended MPX-32 program that may be branched to from nonextended MPX-32.

This macro is not required for labels that are not external to the program because no adaptive sequences are required.

MBR_ENT tests the state of BOPT_MPX flag. IF BOPT_MPX is true, the MBR_ENT macro generates the adaptive sequence required to reference a label that is in the opposite mode of execution (e.g. a program is in nonextended MPX-32 and is branching to an extended section of code).

Syntax

MBR_ENT <*symbol*> Replaces <*symbol*> EQU $

9.3.8 MBR_EXT - Identify External Linkage Symbols Macro

The MBR_EXT macro identifies external linkage symbols (entry points or data) that are referenced by the program.

MBR_EXT tests the state of the BOPT_MPX flag. If BOPT_MPX is false, an assembler EXT directive is generated. If BOPT_MPX is true, an assembler SEXT directive is generated. The SEXT directive is used by the assembler to provide linkage for EXT_MPX.

To convert existing modules for extended MPX-32, replace assembler EXT directives with the MBR_EXT macro.

Syntax

MBR_EXT *label*

9.3.9 MBR_INIT - Module Initialization Macro

The MBR_INIT macro tests the state of the BOPT_MPX flag. If BOPT_MPX is true, MBR_INIT sets assembler option 16. When assembler option 16 is set, the assembler generates extended code that can be placed in extended MPX-32 via SYSGEN.

If BOPT_MPX is false, the assembler generates nonbase code that cannot be placed in extended MPX-32.

This macro is required before any assembler directive that generates code or data.

Syntax

MBR_INIT

9.3.10 MBR_OFFS - Offset Mode Macro

The MBR_OFFS macro defines the offset mode of operation for SYSGEN. By default, the offset mode is in effect (i.e., relocatable object records placed in the EXT_MPX section by SYSGEN are resolved using base registers) until relative mode is set using the MBR_REL macro.

MBR_OFFS tests the state of the BOPT_MPX flag. If BOPT_MPX is true, the macro generates a control object record for SYSGEN. This allows SYSGEN to fill in the base register field of base mode instructions in the EXT_MPX section. The control object record consists of an ASCII string "BROFFSET" that is placed in the FLG_MPX section. The EXT_MPX section is established upon exit from the macro.

IF BOPT_MPX is false, no assembler code or directive is generated and the next assembler line processed.

Syntax

MBR_OFFS

9.3.11 MBR_REL - Relative Mode Macro

The MBR_REL macro defines the relative mode of operation for SYSGEN. By default, the offset mode is in effect (i.e., relocatable object records placed in the EXT_MPX section by SYSGEN are resolved using base registers) until relative mode is set using this macro.

MBR_REL tests the state of the BOPT_MPX flag. If BOPT_MPX is true, the macro generates a control object record for SYSGEN. This allows SYSGEN to treat the subsequent relocatable object using the relative offset from the beginning of extended MPX-32. The EXT_MPX section is established upon exit from the macro.

IF BOPT_MPX is false, no assembler coding or directive is generated and the next assembler line is processed.

Syntax

MBR_REL

9.3.12 MBR_SSCT - System Code Separation Macro

THE MBR_SSCT macro tests the state of BOPT_MPX. If BOPT_MPX is true, an assembler SSECT EXT_MPX directive is generated. See the SSECT EXT_MPX directive. If BOPT_MPX is false, no assembler coding or directive is generated, and the next assembler line is processed.

Syntax

MBR_SSCT

9.3.13 MBR_TRSW - Transfer Register Status Word Macro

For MBR_TRSW, if BOPT_MPX is true, the appropriate code sequence is generated to return to nonextended or extended MPX-32. The generated code tests the base register mode bit, bit 6, in the specified register. See Figure 9-2.

If BOPT_MPX is false, a TRSW instruction is generated.

Syntax

MBR_TRSW

9.4 Macro Assembler and Extended MPX-32

In addition to the Macro Assembler functions stated in the MPX-32 Utilities Manual, the assembler can generate extended or nonextended code depending on the condition of Macro Assembler option 16.

Extended MPX-32 modules control option 16 with the Assembler OPTS directive. This directive generates the appropriate code and adaptors that allow a module to execute in extended memory. The OPTR (Option Reset) and OPTT (Option Test) Macro Assembler directives are used to test the state of option 16 and change the mode of the instruction set during the generation of adaptive sequences.

Note: Option 16 is declarable internally and in-line only with the OPTS directive. If option 16 is set externally (i.e. at assemble time), it has no effect on the generation of extended code.

9.5 Macro Assembler Directives for Extended MPX-32

Some Macro Assembler directives form the adaptor source code that must be used for a module to run in extended MPX-32. The adaptive code is generated by a combination of Macro Assembler directives and special communications sequences. The adaptive code enables the properly sectioned location of nonextended code, extended code, adaptive code, and DSECT data with the required changes in the instruction mode.

The following assembler directives can be used for nonextended to extended MPX-32 conversions.

Directive	Function
OPTR	resets assembly option 16
OPTS	sets assembly option 16
OPTT	tests assembly option 16
SDEF*	identifies SYSGEN linkage symbols within a program that can be externally referenced
SEXT*	identifies externally defined SYSGEN linkage symbols that are referenced by the program
SORG	assigns a specified value to the location counter
SSECT	assembles the SYSGEN section of program source code
SSECTFLG_MPX	allows communication of state changes in the object code to identify and load the code and data sections

* This directive has an MBR_*xxx* macro equate. The macro equate tests BOPT_MPX and generates the appropriate extended or nonextended code. The Assembler directive generates code that runs only in extended MPX-32.

9.5.1 OPTR Directive

The OPTR directive resets option 16, regardless of the option's previous value. It then assigns the new option value to the symbol.

Syntax

Label	Operation	Operand
symbol	OPTR	16

symbol is optional, and the value assigned to it can be redefined by the OPTR, OPTS, and OPTT directives

Usage: See the SSECT FLG_MPX Directive Usage section.

9.5.2 OPTS Directive

The OPTR directive resets option 16, regardless of the option's previous value. It then assigns the new option value to the symbol.

Syntax

Label	Operation	Operand
symbol	OPTS	16

symbol is optional, and the value assigned to it can be redefined by the OPTR, OPTS, and OPTT directives

Usage: See the SSECT FLG_MPX Directive Usage section.

9.5.3 OPTT Directive

The OPTT directive tests the selected assemble option and assigns a value of one to the symbol if option 16 is set. A value of zero is assigned to the symbol if option 16 is not set.

Syntax

Label	Operation	Operand
symbol	OPTT	16

symbol is optional and the value assigned to it can be redefined by the OPTR, OPTS, and OPTT directives

9.5.4 SDEF Directive

The SDEF directive identifies linkage symbols that are within a given program and may be referenced by another program or subroutine as entry points or data.

The symbols referenced in the operand field must be defined in the program where the directive is used.

SDEF directive must precede data definitions and executable statements in the source program.

Syntax

Label	Operation	Operand
label	SDEF	*sym*[,*sym*]

sym is the symbolic name local to the program

Usage: See the SEXT directive description.

9.5.5 SEXT Directive

The SEXT directive identifies linkage symbols that are entry points or data in another program or subroutine, but referenced by the given program.

The symbols referenced in the operand field must be defined in a program other than the program where the SEXT directive is specified. The symbols are given defined addresses at load time if corresponding SDEF directives in another program or subroutine are present.

Symbols defined by SEXT directives may not be used within a common definition or in the operand field of the EQU directive.

Syntax

Label	Operation	Operand
label	SEXT	*sym* [,*sym*]

sym is a symbolic name defined in another program or subroutine

Usage: These samples of listed output illustrate the use of the SEXT and SDEF directives.

Referencing Program

Location Counter	Machine Instruction	Byte Address	Label	Operation	Operand
				PROGRAM	EXTDEF1
				SEXT	CAL4
S00000				SSECT	CAL
S00000			CAL5	EQU	$
S00000	D4000018	S00018		STW	0,CAL5R0
S00004	F8800001	X00000		BL	CAL4
S00008	F8800005	X00000		BL	CAL4
S0000C	F8800009	X00000		BL	CAL4
S00010	F880001D	S0001C		BL	CAL2
S00014	EC100019	S00018		BU	*CAL5R0
S00018	00000000		CAL5R0	DATAW	0
S0001C	D4000028	S00028	CAL2	STW	0,CAL2RO
S00020	C9800003			LI	3,3
S00024	EC100029	S00028		BU	*CAL2R0
S00028	00000000		CAL2R0	DATAW	0
P00000				END	

Referenced Program

Location Counter	Machine Instruction	Byte Address	Label	Operation	Operand
				PROGRAM	EXTDEF2
				SDEF	CAL4
S00000				SSECT	CAL
S00000			CAL4	EQU	$
S00000	D4000014	S00014		STW	0,CAL4R0
S00004	D5800018	S00018		STW	3,WORD3
S00008	D600001C	S0001C		STW	4,WORD4
S0000C	D6800020	S00020		STW	5,WORD5
S00010	EC100015	S00014		BU	*CAL4RO
S00014	00000000		CAL4RO	DATAW	0
S00018	00000000		WORD3	DATAW	0
S0001C	00000000		WORD4	DATAW	0
S00020	00000000		WORD5	DATAW	0
P00000				END	

9.5.6 SORG Directive

The SORG directive assigns the value specified in the operand field to the location counter. Symbolic names are assigned absolute or relocatable values relative to the point of origin until a subsequent ABS, REL, ORG or SORG directive is encountered.

Syntax

Label	Operation	Operand
label	SORG	*value*

value a previously defined operand that is not an external reference.

Usage: This example assigns the value `1000` (hexadecimal) to `TAGA` and `START`.

```
TAGA    SORG    X'1000'
START   LW      R2,TAGA
```

9.5.7 SSECT Directive

This directive determines the location of code and data sequences, and assembles the SYSGEN section of the program source code. All symbolic names are assigned relocatable memory addresses relative to the beginning of the SYSGEN section. The total number of COMMON blocks and SYSGEN sections must be less than 254.

Syntax

Label	Operation	Operand
label	SSECT	*sname*

sname is an ASCII constant indicating the section where successive code or data should be positioned. If *sname* is not specified, the default is in the nonextended code section. Valid *snames* are:

sname	Code/data position
ADP_MPX	adaptive code sequence section
DSECT	DSECT data section
EXT_MPX	extended MPX-32 code section
ALT_MPX	alternate extended MPX-32 code section to support the IFBASE/ELSE/ENDIF construct

Usage:

```
RM17.LPN    EQU     $
            BU      RM17.NOW
            SSECT   EXT_MPX
RM17.NOW    EQU     $
            GOTO    %SKIP2
            ANOP
```

For an additional example, see the SSECT FLG_MPX Directive Usage section.

9.5.8 SSECT FLG_MPX Directive

The FLG_MPX SSECT directive signals SYSGEN to conditionally change states when processing the object code. This directive is followed by an ASCII data doubleword that indicates:

- the conditions SYSGEN tests to load the correct object records into the selected section
- whether the object code is extended or nonextended code

Syntax

Label	Operation	Operand
label	SSECT	FLG_MPX
	DATAD	C'*keyword*'

keyword is an ASCII doubleword. Only one keyword can be specified for each FLG_MPX SSECT directive. Valid keywords are:

Keyword	Description
BASECODE	switch instruction set processing mode to base instruction set. Marks extended code and the start of the adaptive code for DEFed entry followed by ASCII label for DEFed entry.
NONBASE	switch instruction set processing mode to nonbase instruction set
BASENTRY	extended module BL entry point sequence follows. An additional ASCII doubleword containing the entry point symbolic name follows the BASENTRY data doubleword.
ENDADAPT	indicates the end to BASENTRY adaptor sequence. If, during SYSGEN processing, the entry is not referenced by a nonextended module, SYSGEN eliminates the adaptive sequence to conserve memory space.
IFBASE	indicates the beginning of an IFBASE/ELSE/ENDIF statement. Code following the directive to the next occurrence of an ELSE directive will be included in the system image if the extended module addresses another extended module. This keyword is used when processing a module in the extended base code object format.
ELSE	divides the code for an IFBASE/ELSE/ENDIF statement. Code preceding the directive is used for an extended-to-extended control transfer. Code following the directive is used for a base to nonbase control transfer.
ENDIF	indicates the conclusion of an FBASE/ELSE/ENDIF statement
BROFFSET	relocatable object in EXT_MPX is resolved with base registers
RELATIVE	relocatable object is resolved by adding the relative offset from the beginning of extended MPX-32.

Usage: The following examples illustrate the FLG_MPX SSECT directive and keywords.

```
MBR_BEQ   DEFM    TARGET
          IFT     BOPT_MPX,%NOPT
          SSECT   EXT_MPX
X.DOLLAR  SET     $-X.BEGIN.          ABS POSITION IN EXT_MPX
*                                     CHECK ON ODD HALFWORD
          IFT     X.DOLLAR+2/4.NE. X.DOLLAR/4,NONOOP
          NOP                         BOUND TO NEXT HIGHEST WORD
NONOOP    ANOP
          SSECT   FLG_MPX
          DATAD   C'IFBASE'           START IF/THEN/ELSE
          SSECT   EXT_MPX
          BEQ     %TARGET             NON-ADAPTIVE REFERENCE
          SSECT   FLG_MPX
          DATAD   C'ELSE'             END NON-ADAPTIVE REFERENCE
          SSECT   ALT_MPX
          BEQ     %ADAPT              ALTERNATIVE ADAPTIVE REFER.
          SSECT   ADP_MPX
%ADAPT    LPSD    %PSDN               CONVERT TO NON-BASE
          SSECT   FLG_MPX
          DATAD   C'NONBASE'          MARK NON-BASE
          OPTR    16                  TELL ASSEMBLER
          SSECT   ADP_MPX
%PSDN     GEN     1/1,4/0,2/2,25/H (%TARGET)      NON-BASE PSD
          GEN     1/1,14/0,3/7,14/0
          SSECT   FLG_MPX
          DATAD   C'ENDIF'             END ALTERNATIVE SEQUENCE
          DATAD   C'BASECODE'          RETURN TO BASEMODE
          OPTS    16                   TELL ASSEMBLER
          SSECT   EXT_MPX
          GOTO    %SKIP
%NOPT     ANOP
          BEQ     %TARGET              NON-EXTDMPX VERSION OF CODE
%SKIP     ANOP
          ENDM
***********
** BRANCH AND LINK TO EXTERNAL ROUTINE MACRO
***********
MBR_BL    DEFM    TARGET
          IFT     BOPT_MPX,%NOPT
          SSECT   EXT_MPX
X.DOLLAR  SET     $-X.BEGIN.           ABS POSITION IN EXT_MPX
*                                      CHECK ON ODD HALFWORD
          IFT     X.DOLLAR+2/4.NE. X.DOLLAR/4,NONOOP
          NOP     0                    BOUND TO NEXT HIGHEST WORD
NONOOP    ANOP
          SSECT   FLG_MPX
          DATAD   C'IFBASE'            START IF/THEN/ELSE SEQUENCE
          SSECT   EXT_MPX
          BL      %TARGET              NON-ADAPTIVE REFERENCE
```

```
            SSECT   FLG_MPX
            DATAD   C'ELSE'              START ADAPTIVE SECTION
            SSECT   ALT_MPX
            LPSD    %PSDN                ADAPTIVE REFER (VIA ADAPT)
            SSECT   FLG_MPX
            DATAD   C'NONBASE'           CHANGES TO NON-BASE
            OPTR    16                   TELL ASSEMBLER
            SSECT   ADP_MPX
            BOUND   1W
%PSDN       GEN     1/1,4/0,2/2,25/H (%ADP1)      NON-BASE PSD
            GEN     1/1/,14/0,3/7,14/0
%ADP1       BL      %TARGET              CALL TARGET, NON-BASE
            BL      $+1W                 SAVE CONDITION CODES IN R0
            LPSD    %PSDB                BACK TO BASE MODE
%PSDB       GEN     1/1,4/0,1/0,1/1,25/H (%ADP2)
            GEN     1/1,14/0,3/7,14/0
            SSECT   FLG_MPX
            DATAD   C'BASECODE'          TELL SYSGEN
            OPTS    16                   TELL ASSEMBLER
            SSECT   ADP_MPX
%ADP2       SLC     R0,5                 MOVE CCS TO PUT IN PSW
            TRCC    R0                   PUT CCS IN PSW
            BU      %RTRN                BACK TO IN-LINE CODE
            SSECT   FLG_MPX
            DATAD   C'ENDIF'             END ALTERNATIVE SEQUENCE
            SSECT   EXT_MPX
%RTRN       EQU     $
            GOTO    %SKIP
%NOPT       ANOP
            BL      %TARGET              NON-BASE VERSION OF CODE
%SKIP       ANOP
            ENDM
***********
* BRANCH UNCONDITIONALLY TO EXTERNAL ROUTINE MACRO
***********

MBR_BU      DEFM    TARGET
            IFT     BOPT_MPX,%NOPT
            SSECT   EXT_MPX
X.DOLLAR    SET     $-X.BEGIN.           ABS POSITION IN EXT_MPX
*                                        CHECK ON ODD HALFWORD
            IFT     X.DOLLAR+2/4.NE. X.DOLLAR/4,NONOOP
            NOP                          BOUND TO NEXT HIGHEST WORD
NONOOP      ANOP
            SSECT   FLG_MPX
            DATAD   C'IFBASE'            START IF/THEN/ELSE SEQUENCE
            SSECT   EXT_MPX
            BU      %TARGET              NON-ADAPTIVE REFERENCE
            SSECT   FLG_MPX
            DATAD   C'ELSE'              ALTERNATIVE ADAPTIVE REFER
            SSECT   ALT_MPX
            LPSD    %PSDN                CHANGE TO NON-BASE, TO
```

```
*                                          TARGET
          SSECT    FLG_MPX
          DATAD    C'NONBASE'              TELL SYSGEN
          OPTR     16                      TELL ASSEMBLER
          SSECT    ADP_MPX
          BOUND    1W
%PSDN     GEN      1/1,4/0,2/2,25/H (%TARGET)    NON-BSE PSD TO TRG
                   (%TARGET)
          GEN      1/1,14/0,3/7,14/0
          SSECT    FLG_MPX
          DATAD    C'ENDIF'                END ALTERNATIVE SEQUENCE
          DATAD    C'BASECODE'             BACK TO BASECODE
          OPTS     16                      TELL ASSEMBLER
          SSECT    EXT_MPX
          GOTO     %SKIP
%NOPT     ANOP
          BU       %TARGET            NON-BASE VERSION
%SKIP     ANOP
          ENDM
***********
* SWITCH SYSGEN TO BR OFFSET ADDRESSING MODE
***********
MBR_OFFS  DEFM
          IFT      BOPT_MPX,%NOPT
          SSECT    FLG_MPX
          DATAD    C'BROFFSET'
          SSECT    EXT_MPX
%NOPT     ANOP
          ENDM
***********
* RETURN TO EXTERNAL CALLING ROUTINE MACRO
***********

MBR_TRSW  DEFM     REG
          IFT      BOPT_MPX,%NOPT
          TBR      %REG,6
          BNS      %ADP1
          TRSW     %REG
*
%ADP1     EQU      $
          LPSD     %PSD
          BOUND    1W
%PSD      GEN      1/1,4/0,2/2,1/0,24/H(%ADP2)
          GEN      1/1,14/0,3/7,14/0
          SSECT    FLG_MPX
          DATAD    C'NONBASE'
          SSECT    ADP_MPX
          OPTR     16
%ADP2     TRSW     %REG
          NOP
          SSECT    FLG_MPX
          DATAD    C'BASECODE'
```

```
          SSECT     EXT_MPX
          OPTS      16
          GOTO      %SKIP
%NOPT     ANOP
          TRSW      %REG
%SKIP     ANOP
          ENDM
***********
* SWITCH SYSGEN TO RELATIVE ADDRESSING MODE
***********

MBR_REL   DEFM
          IFT       BOPT_MPX,%NOPT
          SSECT     FLG_MPX
          DATAD     C'RELATIVE'
          SSECT     EXT_MPX
%NOPT     ANOP
          ENDM
***********
* BASE CODE ROUTINE ENTRY MACRO
***********

MBR_ENT   DEFM      ENTRY
          IFT       BOPT_MPX,%SKIP
          SSECT     FLG_MPX
          DATAD     C'BASENTRY'
          DATAD     C'%ENTRY'
          DATAD     C'NONBASE'
          SSECT     ADP_MPX
          OPTR      16
          LPSD      %PSDB
          BOUND     1W
%PSDB     GEN       1/1,4/0,2/1,25/H(%ADPB)
          GEN       1/1,14/0,1/1,2/2,14/0
*
          SSECT     FLG_MPX
          DATAD     C'BASECODE'
*
          SSECT     ADP_MPX
          OPTS      16
%ADPB     EQU       $
          BU        %ENTRY
*
          SSECT     FLG_MPX
          DATAD     C'ENDADAPT'
          SSECT     EXT_MPX
%SKIP     ANOP
%ENTRY    EQU     $
          ENDM
```

9.6 Macro Assembler Options for Extended MPX-32

In addition to the options defined in the Macro Assembler section of the MPX-32 Utilities Manual, the assembler has an option for control of macro percentage parameters:

Option	Description
16	generates extended MPX-32 opcodes and instruction formats. This option is declarable internally and in-line with source code only (the OPTS directive). If option 16 is set externally, it has no affect on the generation of extended code.

9.7 Macro Assembler Errors and Aborts for Extended MPX-32

This error flag is generated by the assembler:

Error	Description
S	identifies the usage of indirect addressing when the base register code generation option bit is set

9.8 Extended MPX-32 Examples

This section of the addendum contains examples of extended MPX-32. The following subjects are included:

- Nonextended MPX-32 SVC
- Extended MPX-32 SVC
- Assemble assignment for extended MPX-32
- JH.32_E File Sample
- JCL for compressing the extended MPX-32 SVC
- JCL for SYSGENing an extended MPX-32 system

These examples are in order of completion. For example, the existing nonextended SVC is converted to an extended MPX-32 SVC. Then, it must be assembled, JH.32_E must be edited, and COMPRESS must be run. After this is completed, an extended MPX-32 operating system can be SYSGENed.

9.8.1 Nonextended SVC (H.NONEXT)

The following example is the nonextended MPX-32 SVC that will be converted to an extended MPX-32 SVC. This SVC operates only in nonextended MPX-32. See the extended MPX-32 SVC for a comparison of nonextended versus extended SVC.

```
        PROGRAM H.NONEXT
        EXT S.EXEC38
        EXT S.EXEC39
        M.EQUS
        M.TBLS
HAT     DATAW       5
        ACH         EP1         Makes calling task privileged
        ACH         EP2         Makes calling task non-privileged
        ACH         EP3         Makes calling task resident
        ACH         EP4         Makes calling task non-resident
        ACH         INIT        SYSGEN initialization entry point
EP1     EQU         $
        LW          3,C.REGS
        SBM         0,T.REGS+8W,3
        LW          3,C.CURR
        SBM         DQE.PRIV,DQE.USHF,3
        M.RTRN
EP2     EQU         $
        LW          3,C.REGS
        ZBM         0,T.REGS+8W,3
        LW          3,C.CURR
        ZBM         DQE.PRIV,DQE.USHF,3
        M.RTRN
EP3     EQU         $
        BL          S.EXEC38
        M.RTRN
EP4     EQU         $
        BL          S.EXEC39
        M.RTRN
INIT    EQU         $
        M.EIR
        M.MODT      HAT,13
        M.SVCT      TAB,4
        M.XIR       HAT
*
TAB     GEN         8/X'80',1/0,23/H(EP1)
        GEN         8/X'81',1/0,23/H(EP2)
        GEN         8/X'82',1/0,23/H(EP3)
        GEN         8/X'83',1/0,23/H(EP4)
*
        END
```

9.8.2 Extended MPX-32 SVC (H.EXTMOD)

The following SVC is the modified version of H.NONEXT. The macro/command in boldface type is code that differs from the nonextended SVC.

Note: This code operates in extended or nonextended MPX-32 depending on the state of option 16.

```
        PROGRAM   H.EXTMOD
        MBR_INIT
        MBR_EXT   S.EXEC38
        MBR_EXT   S.EXEC39
        MBR_DEF   EP1
        MBR_DEF   EP2
        MBR_DEF   EP3
        MBR_DEF   EP4
        M.EQUS
        M.TBLS
        M.BREGS
HAT     DATAW     5
        ACH       EP1
        ACH       EP2
        ACH       EP3
        ACH       EP4
        ACH       INIT
EP1     EQU       $
        MBR_ENT   EP1
        LW        R3,C.REGS              Get TSA address
        SBM       0,T.REGS+8W,X3         Set privilege bit in PD stack
        LW        R3,C.CURR              Get DQE address
        SBM       DQE.PRIV,DQE.USHF,X3   Set privilege bit in DQE
        LIST      NOMAC
        M.RTRN                           Return to caller
        LIST      MAC
EP2     EQU       $
        MBR_ENT   EP2
        LW        R3,C.REGS              Get TSA address
        ZBM       0,T.REGS+8W,X3         Set unprivilege bit in PD stack
        LW        R3,C.CURR              Get DQE address
        ZBM       DQE.PRIV,DQE.USHF,X3   Set unprivilege bit in DQE
        LIST      NOMAC
        M.RTRN
        LIST      MAC
EP3     EQU       $
        MBR_ENT   EP3
        MBR_BL    S.EXEC38                Force unswappable resident
        LIST      NOMAC
        M.RTRN                           Return to caller
        LIST      MAC
EP4     EQU       $
        MBR_ENT   EP4P
        MBR_BL    S.EXEC39                Allow swapping nonresident
        LIST      NOMAC
```

```
        M.RTRN                                  Return to caller
        LIST      MAC
INIT    EQU       $
        M.EIR
        M.MODT    HAT,13
        M.SVCT    TAB,4
        M.SVCP    TAB1,4
        M.XIR     HAT
        MBR_REL                                 Establish relative mode
TAB     GEN       8/X'80',1/0,23/H(EP1)
        GEN       8/X'81',1/0,23/H(EP2)
        GEN       8/X'82',1/0,23/H(EP3)
        GEN       8/X'83',1/0,23/H(EP4)
        MBR_OFFS                                Set up flags for SVCs
TAB     GEN       8/X'80',8/X'20'
        GEN       8/X'81',8/X'20'
        GEN       8/X'82',8/X'20'
        GEN       8/X'83',8/X'20'
        END
```

9.8.3 Assemble Assignment for Extended MPX-32 SVC

The following example is an interactive assignment that assembles H.EXTMOD, the extended MPX-32 SVC example. Assigning PRE to MPX_EXT sets option 16 to generate extended MPX-32 coding. This example can be completed in batch mode.

```
TSM> $JOB MWTEST SLOF=XX

TSM> $AS PRE TO ^(SYSTEM) MPX_EXT

TSM> $AS SI TO SH.EXTMOD

TSM> $AS GO TO OH.EXTMOD

TSM> $OPTI 2 3 4 5

TSM> $ASSEMBLE

TSM> $EOJ

TSM> $$
```

9.8.4 JH.32_E File Sample

The following example is a list of the contents of JH.32_E after it has been edited to include the extended MPX-32 SVC example H.EXTMOD. These pathnames are COMPRESS input directives.

^(OBJECT_E)OH.REMM	Resource Management Module
^(OBJECT_E)OH.MEMM	Memory Management Module
^(OBJECT_E)OH.VOMM	Volume Management Module
^(OBJECT_E)OH.REXS	Resident Executive Services Module
^(OBJECT_E)OH.TAMM	Task Management Module
^(*username*)OH.EXTMOD	User Created/Modified SVC

9.8.5 JCL for Compressing the Extended MPX-32 SVC

The following example is the JCL for compressing the extended MPX-32 SVC example, H.EXTMOD. Compress is run after the JH.32_E file has been edited.

```
TSM> $AS IN TO JH.32_E

TSM> $AS OT TO OH.32_E

TSM> $COMPRESS
```

9.8.6 JCL for SYSGENing an Extended MPX-32 Operating System

The following example is the JCL for SYSGENing an extended MPX-32 operating system.

```
TSM> ASSIGN DIR TO DIRECTIVES BLO=Y

TSM> ASSIGN OBR TO @SYSTEM(SYSTEM)OH.32_E BLO=Y

TSM> ASSIGN OBJ TO @SYSTEM(SYSTEM)OH.32    BLO=Y

TSM> ASSIGN SLO TO SLO

TSM> SYSGEN
```

10 RTOM Interval Timer

10.1 General Information

The Real-Time Option Module (RTOM) provides an interval timer that can be used to measure the speed at which code executes. To use the RTOM interval timer:

- Install an RTOM board on the SelBUS. For installation information, see the RTOM Technical Manual.
- SYSGEN the RTOM interval timer at any priority level from X'01' through X'6E'.
- Generate code to direct the interval timer.

All communication with the RTOM interval timer is done with R0.

This chapter explains the basic use of the RTOM interval timer. For more information, refer to the RTOM Technical Manual.

10.2 SYSGENing RTOM

After the RTOM board is installed on the SelBUS, access the RTOM interval timer by adding the following command to the /INTERRUPTS section of the SYSGEN file.

Syntax

PRIORITY = *pp*, RTOM = (*hh*,04), INTV

pp is a 2-digit hexadecimal priority level of the RTOM. The RTOM priority level must not be used in any other PRIORITY directive.

hh is a 2-digit hexadecimal hardware address of the RTOM. *hh* must be between X'79' and X'7E', inclusive. Determine the hardware address by inspecting jumper assembly U13 on the RTOM board or by referring to the RTOM Technical Manual.

Note: This directive assumes that the RTOM interval timer has a subaddress of X'04'. While this is usually true, some RTOMs are at subaddress X'0B'. In this case, substitute 0B for 04. The subaddress of the interval timer can be determined by looking at jumper assembly U107 or referring to the RTOM Technical Manual.

10.3 Frequency Rate of the Interval Timer

The frequency rate of the interval timer (i.e., the length of time represented by each clock tick) is determined by the physical jumpering of the RTOM board. There are eight frequency rates: four high frequency and four low frequency. See Table 10-1 for the eight frequency rates and jumpering information. The RTOM board is jumpered for one high and one low interval timer frequency rate. The user chooses between these two rates when programming the timer.

Table 10-1
RTOM Frequency Rates and Jumper Addresses

	Frequency Rate	Jumper Address
High	300 nanoseconds	U158-8 to U158-9
	600 nanoseconds	U158-7 to U158-10
	1.2 microseconds	U158-6 to U158-11
	2.4 microseconds	U158-5 to U158-12
Low	4.8 microseconds	U158-4 to U158-13
	9.6 microseconds	U158-3 to U158-14
	19.2 microseconds	U158-2 to U158-15
	38.4 microseconds	U158-1 to U158-16

10.4 Controlling the Interval Timer

The CD (Command Device) command starts, reads, or stops the RTOM interval timer.

Syntax

CD X'*pp*',X'*ffff*'

pp is a hexadecimal priority level of the interval timer as specified in the SYSGEN directive

ffff is a hexadecimal function code for basic use of the timer.

Hex Value	Description
X'38'	load and start timer at high frequency
X'39'	load and start timer at low frequency
X'40'	read timer
X'50'	stop timer

See the RTOM Technical Manual for additional information.

Communication to the RTOM is done through R0. The load and start function codes transfer the contents of R0 into the interval timer. The interval timer then begins counting down from that value.

The read function code transfers the current value of the timer to R0.

Because the timer counts down, the time interval between the two reads is the first read minus the second. For accurate results, this overhead must be estimated and subtracted from the results from each machine type. (The overhead is about 4 microseconds on for a CD command on a CONCEPT 32/87.)

10.5 Examples

The following examples show how to:

- enable and read the interval timer at priority X'04' and high frequency
- estimate read overhead
- read the timer before and after the code to be timed
- time a section of code.

Note: Because the CD command is privileged, the task must be cataloged/linked as privileged.

10.5.1 Example 1: Enabling and Reading the Timer

This example shows how to enable and read the timer. In this example X'7FFFFFFF' is the initial count value, but any value can be used.

```
                        Code
                         .
                         .
                         .
                TIME = X'7FFFFFFF'
                         .
                         .
                         .
*
* Start the timer and estimate read overhead
*
    INLINE
    BEI
    LW 0,TIME            Start count value is high positive value
    CD X'04',X'38'       Start timer
    CD X'04',X'40'       Read timer
    UEI
    TRN 0,0              Make value negative
    ADMW 0,TIME          Add starting value
    STW 0,OVERHEAD       Save as the overhead
    ENDI

*
* The timer is now running and the overhead clock tick is
* stored in the variable OVERHEAD.
*
                         .
                         .
                         .

                        Code
                         .
                         .
                         .
```

```
*
* This is the top of the loop to be timed.
*

    INLINE
    LW 0,TIME           Load the timer with high positives
    CD X'04',X'38'      Enable the timer
    ENDI
```

.
.
.

Timing sequence

.
.
.

```
    INLINE
    CD X'04',X'40'      Read the timer
    TRN 0,0             Make value negative
    ADMW 0,TIME         Add beginning value
    STW 0,TOTAL         Store total counts
    SUMW 0,OVERHEAD     Store results
    ENDI
```

.
.
.

Code output to the results

10.5.2 Example 2: Reading the Timer

This example illustrates how to read the timer before and after a piece of code. This technique can be used when multiple parts of the code are using the timer.

Code

.
.
.

```
*
* Start the timer
*
    INLINE
    LI 0,0
    CD X'04',X'38'
    ENDI

*
* Estimate the read overhead
*

    INLINE
    BEI
    CD X'04',X'40'      Read the timer
```

Examples

```
        STW 0,SAVESTART
        CD X'04',X'40'      Read the timer
        UEI
        TRN 0,0             Make value positive
        ADMW 0,SAVESTART    Add start value
        STW 0,OVERHEAD      Save the overhead
        ENDI

*
* The timer is now running and the overhead in clock ticks
* is stored in the variable OVERHEAD.
*
                            .
                            .
                            .
                          Code
                            .
                            .
                            .
*
* This is the top of the loop to be timed.
*

        INLINE
        CD X'04'            Read the timer
        STW 0,SAVESTART
        ENDI
                            .
                            .
                            .
                   Timing sequence
                            .
                            .
                            .
        INLINE
        CD X'04',X'40'      Read the timer
        TRN                 Make value negative
        ADMW 0,SAVESTART    Add beginning value
        STW 0,TOTAL         Store total counts
        SUMW 0,OVERHEAD     Subtract out overhead
        STW 0,RESULT        Store result
        ENDI
                            .
                            .
                            .
              Code output to the result
```

A System Tables and Variables

A.1 Cross-Reference

TABLE NAME	DESCRIPTION	SECTION
ART	Allocated Resource Table	2.4
CDT	Controller Definition Table	2.8
CHT	Channel Definition Table	2.7
CNP	Caller Notification Packet	2.6
DAT	Dispatch Queue Address Table	2.14
DCA	Device Context Area	2.9
DMAP	Descriptor Map Deallocation File Descriptor	2.45.6
DQE	Dispatch Queue Entry	2.13
DTT	Device Type Table	2.10
FAT	File Assignment Table	2.15
FCB	File Control Block	2.16
FPT	File Pointer Table	2.18
IOQ	I/O Queue (IOQ) Entry	2.19
J.SOEX	Run Request	2.37.3
J.SOUT	Run Request	2.37.4
J.SSIN	Run Request	2.37.1
J.TSM	Run Request	2.37.2
M.BB.DEQ	Bad Block Descriptor	2.45.3
M.BD.DEQ	Descriptor Map (DMAP) Deallocation File Descriptor	2.45.6
M.BO.EQU	New Boot Macro Offset	2.47.1
M.BS.DEQ	Space Map (SMAP) Deallocation File Descriptor	2.45.11
M.DD.DEQ	Descriptors Descriptor	2.45.5
M.DI.DEQ	Directory Descriptor	2.45.7

Cross-Reference

TABLE NAME	DESCRIPTION	SECTION
M.DM.DEQ	Descriptor Allocation Map Descriptor	2.45.4
M.DN.TEQ	Directory Entry Table	2.11
M.FI.DEQ	File Descriptor	2.45.8
M.ME.DEQ	Memory Parition Descriptor	2.45.9
M.SM.DEQ	Space Allocation Map Descriptor	2.45.10
M.VO.DEQ	Volume Descriptor	2.45.12
M.DPT	Disk Parameter Table Offset	2.47.2.1
M.KEY	M.KEY Entry Format	2.20
M.PRJCT	M.PRJCT Format	2.21
M.RDACC	Resource Descriptor Access Parameters	2.45.1
M.RDACT	Resource Descriptor Accounting Parameters	2.45.1
M.RDCOM	Resource Descriptor	2.45.1
M.RDID	Resource ID	2.45.1
M.RDSPD	Resource Descriptor Space Definition	2.45.2
M.RIQ	Resource Inquiry Table	2.33
MATA	Memory Allocation Table	2.24
MDT	Memory Resident Descriptor Table	2.27
MEML	Memory Attribute List	2.25
MIDL	Map Image Descriptor List	2.22
MPTL	Memory Allocation Pointer List	2.23
MRRQ	Message or Run Request Queue	2.28
MVT	Mounted Volume Table	2.30
PSM	Physical Shared Memory Table	2.31
RD.SEGDF	Segment Definitions	2.45.13
RD.USER	User Area	2.45.14
RCB	Resource Create Block	2.32
RLB	Resource Logging Block	2.34
RRS	Resource Requirement Summary Entries	2.35
SJ.VFDPT	Disk Parameter Table Format	2.47.2.2
SMD	System Master Directory	2.38
SMAP	Space Map Deallocation File Descriptor	2.45.11
SMT	Shared Memory Table	2.36

TABLE NAME	DESCRIPTION	SECTION
T.MEMLA	Memory Attribute List	2.25
T.MIDLA	Map Image Descriptor List	2.22
TCPB	Type Control Parameter Block	2.42
TSA	Task Service Area	2.39
UDT	Unit Definition Table	2.43
VAT	Volume Assignment Table	2.44
N/A	Blocking Buffer Control Cells	2.5
N/A	Communications Region	2.3
N/A	Dispatch Queue Area	2.12
N/A	Disk Parameter Table Structures	2.47.2
N/A	Disk Resident Resource Descriptors	2.45
N/A	COFF Executable Image Preamble	2.46.10
N/A	COFF Load Module Structure	2.46.9
N/A	COFF Shared Image Preamble	2.46.11
N/A	Internal J.VFMT Structures	2.47
N/A	I/O Queue (IOQ) Entry	2.19
N/A	Load Module Preamble	2.46.3
N/A	Load Module Structure	2.46.2
N/A	Memory Layout	2.2
N/A	Module Address Table	2.29
N/A	Memory Pool Management	2.26
N/A	Executable Image Preamble	2.46.5
N/A	Executable Image Structure	2.46.4
N/A	Shared Executable Image Preamble	2.46.7
N/A	Shared Executable Image Structure	2.46.6
N/A	Shared Image Descriptors	2.46.8
N/A	Spooled File Data Structures	2.37
N/A	Terminal Line Buffer	2.40
N/A	Timer Table	2.41
N/A	Volume Format	2.46.1

MPX-32™
Technical Manual
No. 322-001551-500

READER'S COMMENT FORM

Please use this form to communicate your views about this manual. The form is preaddressed and stamped for your convenience.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy	____	____	____	____
Clarity	____	____	____	____
Completeness	____	____	____	____
Examples	____	____	____	____
Figures	____	____	____	____
Index	____	____	____	____
Organization	____	____	____	____
Retrievability of Information	____	____	____	____

Additional comments:

_____

_____

_____

_____

_____

_____

If you wish a reply, please print your name and mailing address:

_____

_____

_____

_____

What is your occupation/title? _____

_____

Thank you for your cooperation.

Note: Copies of Encore publications are available through your Encore representative or the customer service office serving your locality.

FOLD HERE

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 2356 FORT LAUDERDALE, FL

POSTAGE WILL BE PAID BY ADDRESEE

ENCORE COMPUTER CORPORATION
ATTENTION: DOCUMENTATION COORDINATOR
6901 W. SUNRISE BLVD.
P.O. BOX 409148
FT. LAUDERDALE, FL 33340-9970

FOLD HERE

PLEASE TAPE DO NOT STAPLE

CUT ALONG LINE