Gould MPX-32™

Release 1.5E

Technical Manual

April 1985

Publication Order Number: 322-003660-000

GOULD
Electronics

This manual is supplied without representation or warranty of any kind. Gould Inc., Computer Systems Division 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.

LIMITED RIGHTS LEGEND

for

PROPRIETARY INFORMATION

The information contained herein is proprietary to Gould CSD and/or its vendors, and its use, disclosure or duplication is subject to the restrictions stated in the Gould CSD license agreement Form No. 620-06(1/82) or the appropriate third-party sublicense agreement. The information is provided to government customers with limited rights as described in DAR 7-104.9A.

HISTORY

The MPX-32 1.0 Technical Manual, Publication Order Number 322-001010-000, was printed September, 1979.

Publication Order Number 322-001010-100 (Revision 1, Release 1.3) was printed March, 1980.

Publication Order Number 322-001010-200 (Revision 2, Release 1.4) was printed August, 1980.

Publication Order Number 322-003660-000 (Revision 3, Release 1.5B) was printed September, 1982.

Publication Order Number 322-003660-001 (Change 1 to Revision 3, Release 1.5D) was printed February, 1984.

Publication Order Number 322-003660-002 (Change 2 to Revision 3, Release 1.5E) was printed April, 1985. The updated manual contains the following pages:

	* Change Number		* Change Number
Title page	2	3-177 through 3-279	0
Copyright page	2	3-280 and 3-281	2
iii/iv Change 2	2	3-282 through 3-284	0
iii/iv Change 1	1	4-1 through 4-11/4-12	0
iii/iv	0	5-1 through 5-20	0
v through xix	0	6-1 through 6-22	0
xx	2	6-23 through 6-31/6-32	2
xxi through xxxii	0	7-1 through 7-25/7-26	0
1-1 through 1-61/1-62	0	8-1 through 8-13/8-14	0
2-1 through 2-10	0	9-1 through 9-9/9-10	0
2-11	1	10-1 through 10-12	0
2-12 through 2-30	0	11-1 and 11-2	0
2-31	1	12-1 through 12-28	0
2-32 through 2-90	0	13-1 through 13-10	0
3-1 through 3-10	0	14-1 through 14-41/14-42	0
3-11	1	15-1 through 15-44	0
3-12 through 3-32	0	16-1 through 16-4	0
3-33	1	A-1 through A-6	0
3-34 through 3-174	0	B-1/B-2	0
3-175 and 3-176	1		

* Zero in this column indicates an original page.

On a change page, the portion of the page affected by the latest change is indicated by a vertical bar in the outer margin of the page. However, a completely changed page will not have a full length bar, but will have the change notation by the page number.

TABLE OF CONTENTS

Page

1. SYSTEM DESCRIPTION

3. MODULE DESCRIPTIONS

Page

Page

Page

Page

Page

4. SYSTEM TASK DESCRIPTIONS

7. SYSTEM TRACE

8. SYSTEM INITIALIZERS AND BUILDERS

9. INTERNAL PROCESSING UNIT (IPU)

14. DISC PROCESSOR HANDLER

15. INPUT/OUTPUT PROCESSOR (IOP) HANDLER

Page

16. IOP EIGHT-LINE FULL DUPLEX HANDLER (H.F8IOP)

APPENDIXES

ILLUSTRATIONS

Documentation Conventions

Notation conventions used in command syntax and message examples throughout this manual are described below.

lowercase letters

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

```
!ACTIVATE taskname
```

means replace taskname with the name of a task, e.g.,

```
!ACTIVATE DOCCONV
```

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

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

means a break occurred on the specified task.

UPPERCASE LETTERS

In command syntax, uppercase letters specify a keyword must be entered as shown for input, and will be printed as shown in output. For example,

```
SAVE filename
```

means enter SAVE followed by a filename, e.g.,

```
SAVE DOCCONV
```

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

```
taskname,taskno ABORTED

*YOUR TASK IS IN HOLD. ENTER CONTINUE TO RESUME IT
```

Braces { }

Elements placed one under the other inside braces specify a required choice. You must enter one of the arguments from the specified group. For example,

$$\left\{\begin{array}{l}\text{counter}\\ \text{startbyte}\end{array}\right\}$$

means enter the value for either counter or startbyte.

Brackets [ ]

An element inside brackets is optional. For example,

```
[CURR]
```

means the term CURR is optional.

Items placed one under the other within brackets specify you may optionally enter one of the group of options or none at all. For example,

```
[base name]
[progname ]
```

means enter the base name or the program name or neither.

Items in brackets within encompassing brackets specify one item is required only when the other item is used. For example,

```
TRACE [lower address [upper address]]
```

means both the lower address and the upper address are optional, and the lower address may be used alone. However, if the upper address is used, the lower address must also be used.

Commas between multiple brackets within an encompassing set of brackets are semi-optional; that is, they are not required unless subsequent elements are selected. For example,

```
M.DFCB fcb,lfc[,[a], [b] , [c] , [d] , [e]]
```

could be coded as

```
M.DFCB FCB12,IN

    or

M.DFCB FCB12,IN,,ERRAD

    or

M.DFCB FCB13,OUT,,ERAD,,PCK
```

Horizontal Ellipsis ...

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

```
name ,...,name
```

means you may enter one or more name values separated by commas.

Vertical Ellipsis

.
.
.

The vertical ellipsis specifies commands, parameters, or instructions have been omitted. For example,

```
COLLECT 1
  .
  .
  .
LIST
```

means one or more commands have been omitted between the COLLECT and LIST commands.

Numbers and Special Characters

In a syntax statement, any number, symbol, or special character must be entered as shown. For example,

```
(value)
```

means enter the proper value enclosed in parentheses; e.g., (234).

Underscore

In syntax statements, underscoring specifies the letters, numbers or characters that may be typed by the user as an abbreviation. For example,

```
ACTIVATE  taskname
```

means spell out the command verb ACTIVATE or abbreviate it to ACTI.

```
RESET
```

means type either RESET or RST.

In examples, all terminal input is underscored; terminal output is not. For example,

```
TSM > EDIT
```

means TSM> was written to the terminal; EDIT is typed by the user.

Subscript Delta ▲

A subscript delta specifies a required space. For example,

```
EDT > STO▲TSSPGM
```

means a space is required between O and T.

1. SYSTEM DESCRIPTION

1.1 Naming Conventions

To assist in the identification of system components, the following naming conventions are used in MPX-32 software and documentation.

1.1.1 Communications Region

Names of variables within the MPX-32 communications region are prefixed by the characters "C.". Their 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.". Their 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. Their general form is x.y where x consists of 2-4 characters which identify the table and y consists of 3-6 characters which identify the variable. Table or file name prefixes (x) are as follows:

CDT	Controller Definition Table
CHT	IOP Channel Definition Table
DAT	Dispatch Queue Address Table
DFT	Disc 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 Table
MEML	Memory Attribute List
MIDL	Map Image Descriptor List
MQ	Message or Run Request Queue Entry
PRB	Parameter Receive Block
PSB	Parameter Send Block
RRS	Resource Requirement Summary Entry
RXB	Receiver Exit Block
SD	System Input Directory (M.SID)/System Output Directory (M.SOD) Entry
SMD	System Master Directory Entry
SMT	Shared Memory Table
TCA	Terminal Context Area
TCP	Type Control Parameter Block
UDT	Unit Definition Table

1.1.4 System Modules and Interrupt Handlers

Names of system modules and interrupt handlers are prefixed by the characters "H.". Their general form is H.x where x is a string of one to six characters. Entry points within 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.". Their 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 system macros are prefixed by the characters "M.". Their general form is M.x where x is a string of one to six characters.

1.1.7 System Task Load Module Files

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

1.1.8 Batch Task Load Module 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.". Their general form is M.x where x is a string of one to six characters.

1.2 Scheduler - IOCS Interface

I/O Initiation

A user task issues an SVC to enter IOCS. I/O services for pre-transfer processing are then executed at the software priority level of the requesting task. Once the I/O request has been 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

Wait I/O Post Processing

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

No-Wait I/O Post Processing

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

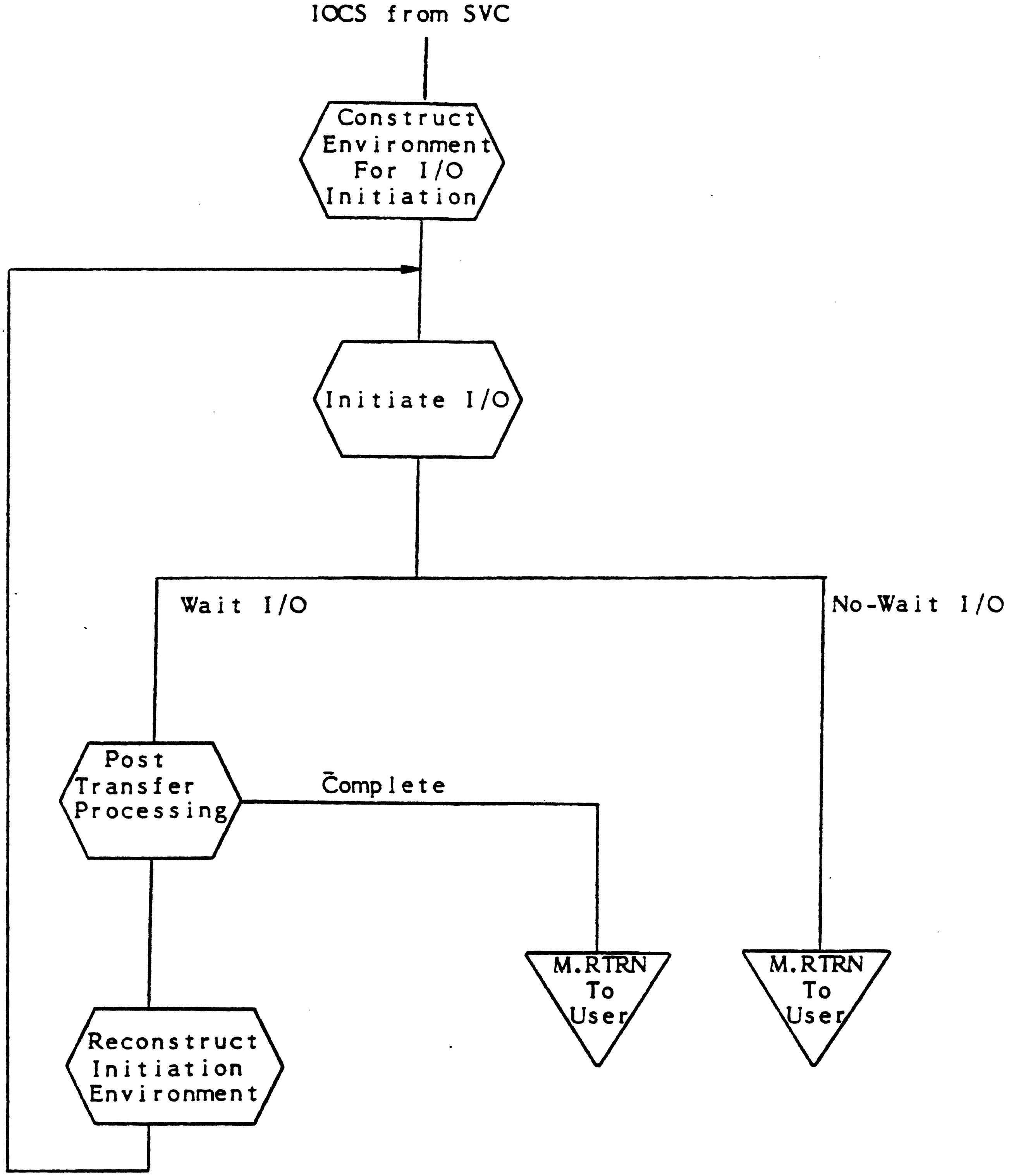
No-Wait I/O Completion Task Interrupt Service

When the I/O handler interrupt service routine fields a completion interrupt for a no-wait I/O request, it will call the executive subroutine S.EXEC4 to report the event. The I/O queue entry associated with the call will be linked to the task interrupt list in the DQE of the task which made the I/O request. When the scheduler attempts to dispatch control to the task, it will discover that a task interrupt is outstanding. It should be noted that task interrupts are inhibited during execution of any system service on behalf of a task. It should be also noted that no task interrupt will be honored while a previous task interrupt is active. When the task interrupt is honored, control will be transferred to the IOCS routine specified in the Preemptive System Service Header of the I/O queue entry. Post transfer processing may then occur at the software priority level of the requesting task. When post processing of the no-wait I/O request is complete, the task interrupt service may be exited by a call to S.EXEC6 or H.EXEC,12.

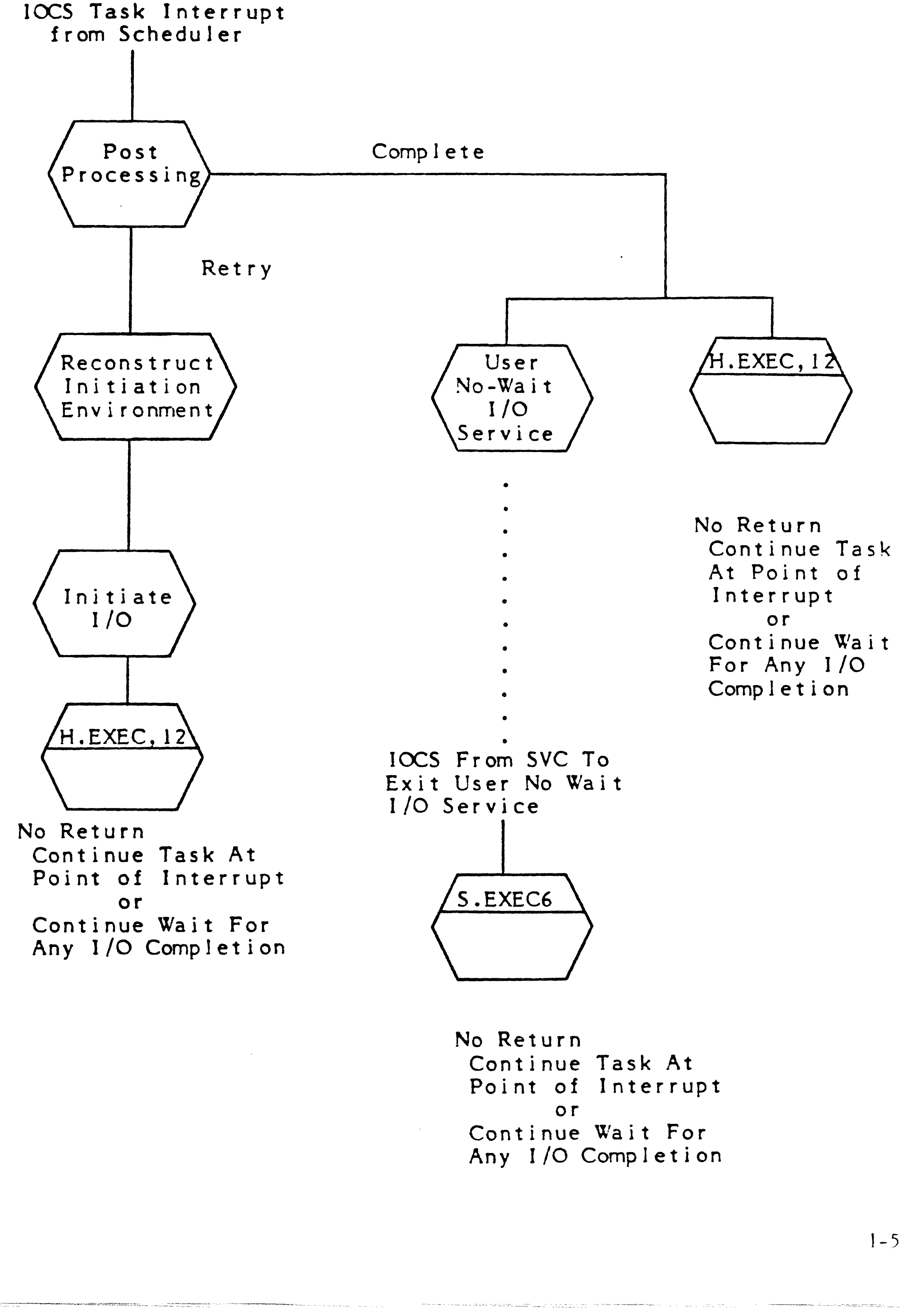
No-Wait I/O Restrictions for System Services

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 to this rule 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 will be honored if the task is in the wait-for-any-no-wait-I/O-completion state. A system service desiring to do no-wait I/O may issue a series of no-wait calls followed by a wait-for-any-call. Care should be exercised to insure that all outstanding calls are completed as appropriate.

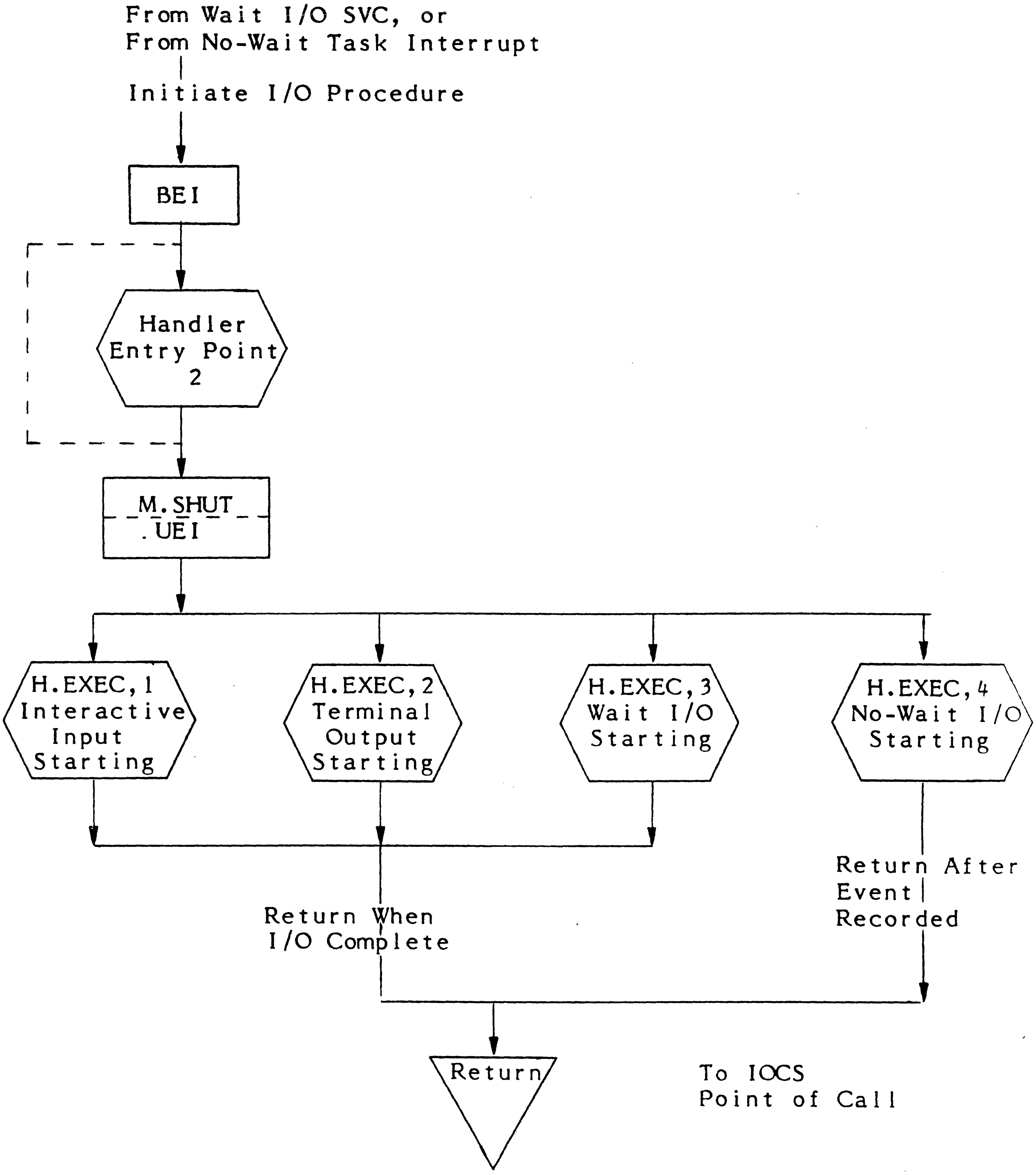
Scheduler - IOCS Interface - IOCS I/O SVC Processing Overview



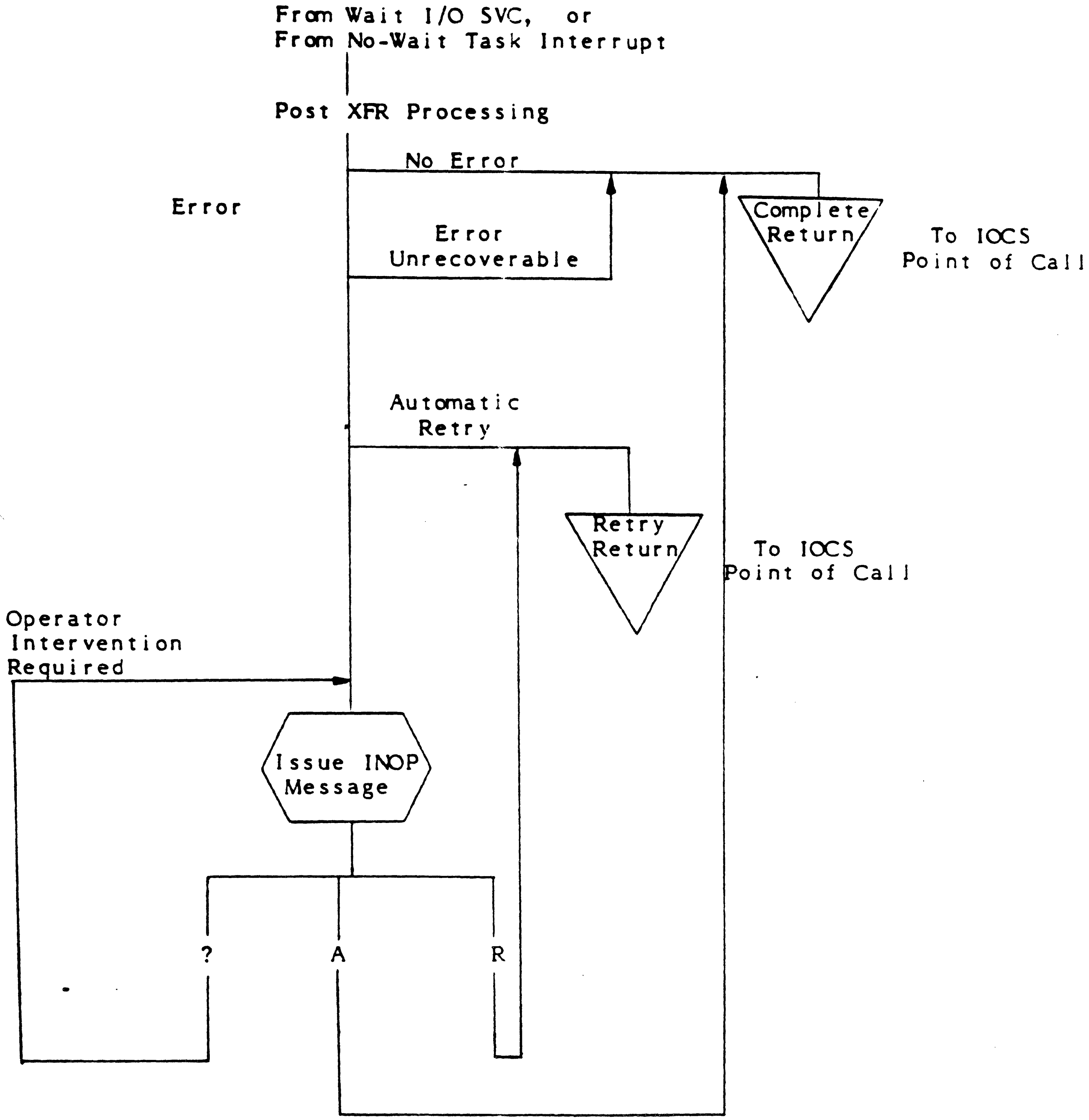
Scheduler - IOCS Interface - IOCS No-Wait I/O Post Processing Overview


IOCS Task Interrupt from Scheduler
Post Processing
Complete
Retry
Reconstruct Initiation Environment
Initiate I/O
H.EXEC,12
No Return Continue Task At Point of Interrupt or Continue Wait For Any I/O Completion
User No-Wait I/O Service
IOCS From SVC To Exit User No Wait I/O Service
S.EXEC6
No Return Continue Task At Point of Interrupt or Continue Wait For Any I/O Completion
H.EXEC,12
No Return Continue Task At Point of Interrupt or Continue Wait For Any I/O Completion

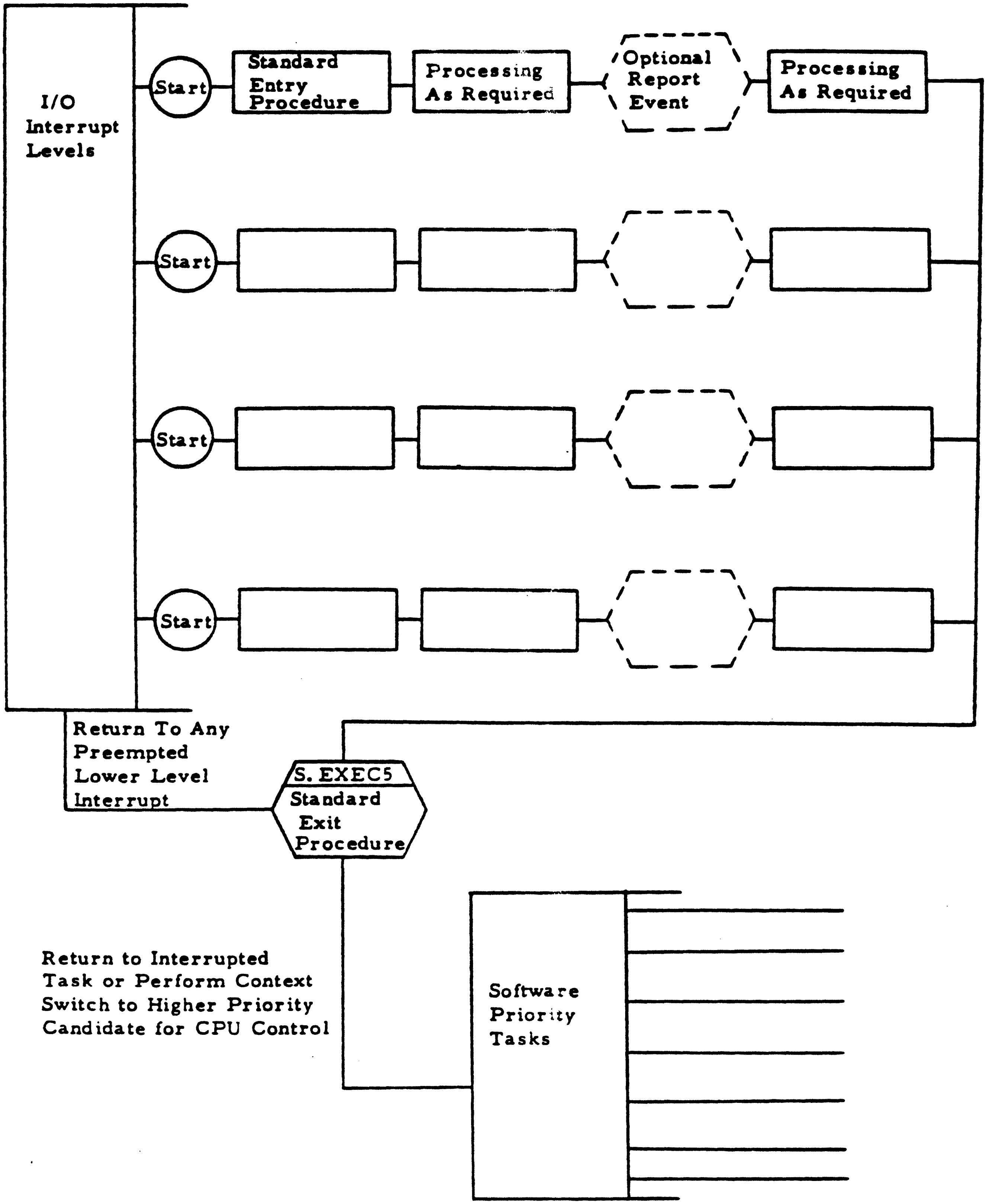
Scheduler - IOCS Interface - IOCS Initiate I/O Procedure


From Wait I/O SVC, or
From No-Wait Task Interrupt
Initiate I/O Procedure
BEI
Handler Entry Point 2
M.SHUT
.UEI
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
Return When I/O Complete
Return After Event Recorded
Return
To IOCS Point of Call

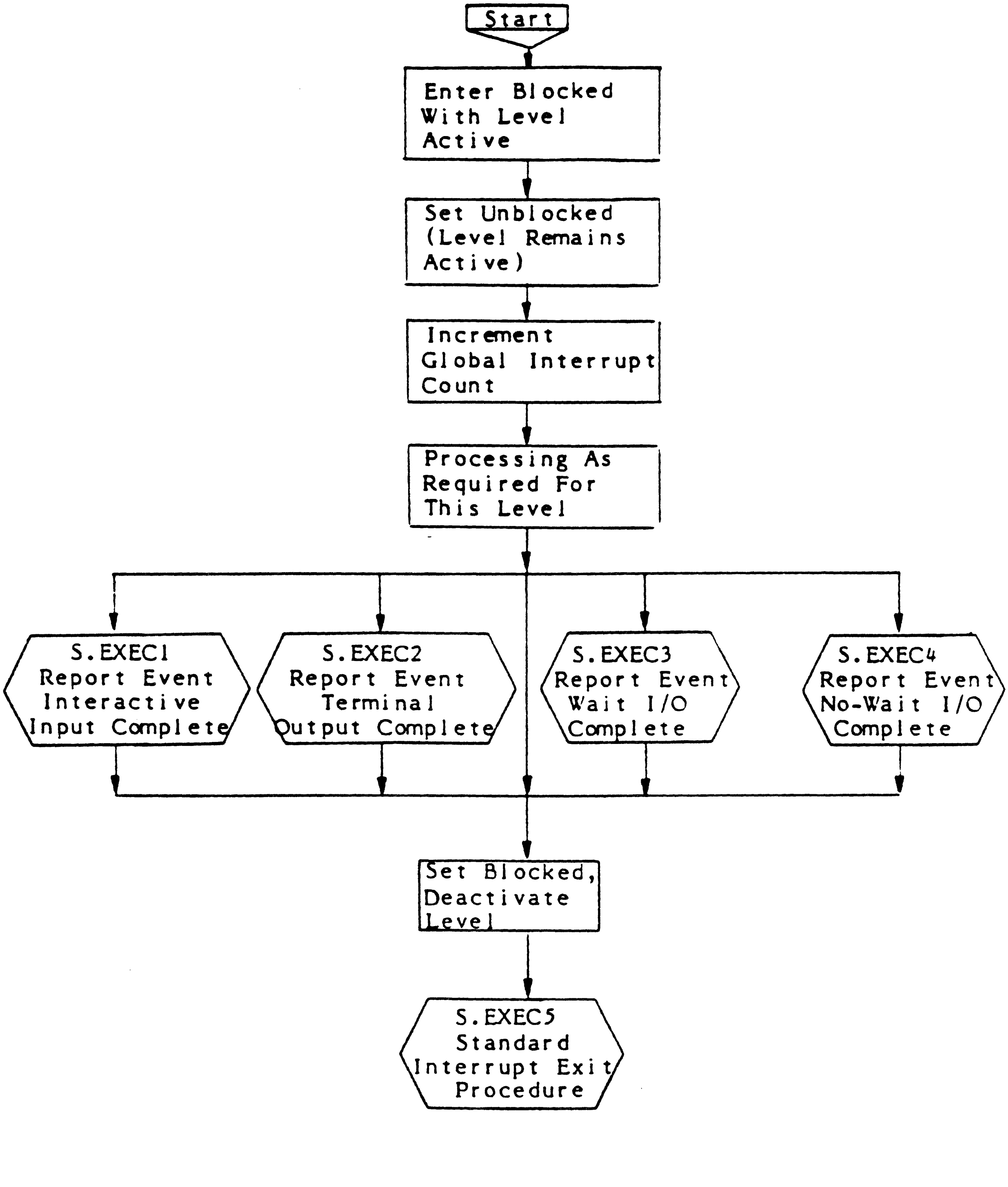
Scheduler - IOCS Interface - IOCS Post Processing Procedure



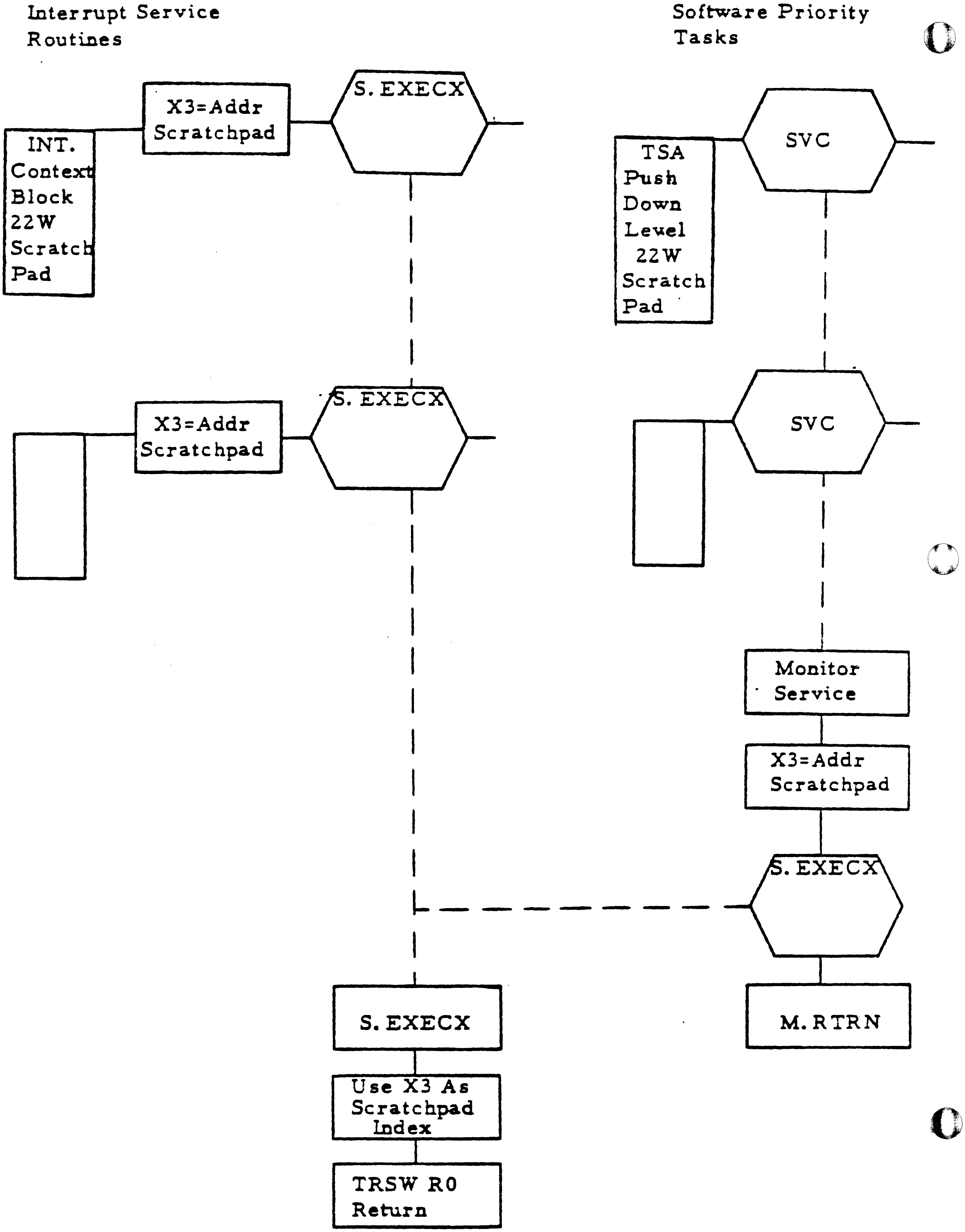
Scheduler - I/O Interrupt Interface Overview



Scheduler - I/O Interrupt - Interface, Procedures


Start
Enter Blocked With Level Active
Set Unblocked (Level Remains Active)
Increment Global Interrupt Count
Processing As Required For This Level
S.EXEC1 Report Event Interactive Input Complete
S.EXEC2 Report Event Terminal Output Complete
S.EXEC3 Report Event Wait I/O Complete
S.EXEC4 Report Event No-Wait I/O Complete
Set Blocked, Deactivate Level
S.EXEC5 Standard Interrupt Exit Procedure

Scheduler - I/O Interrupt Interface, Reentrant Subroutines

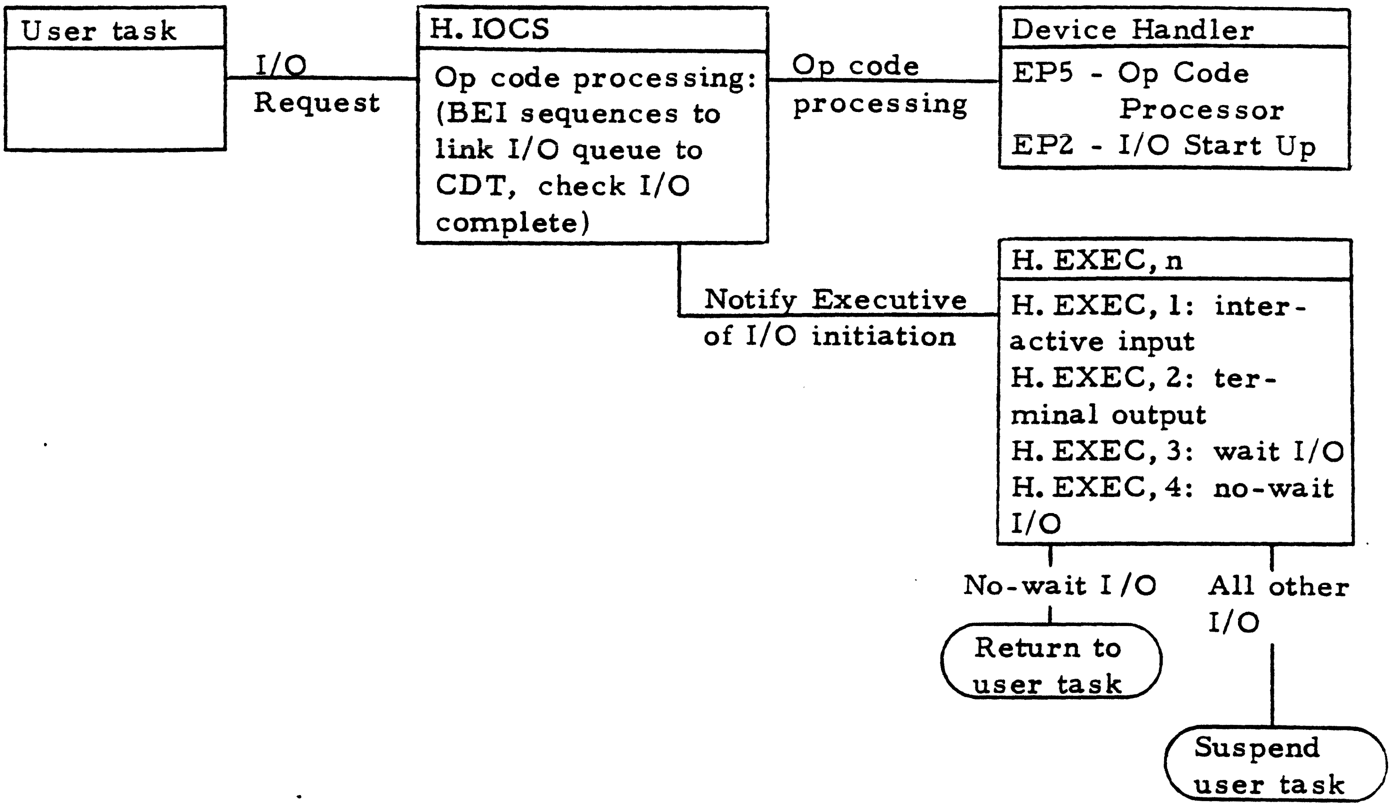


Preemptive System Service List Entry Header Format

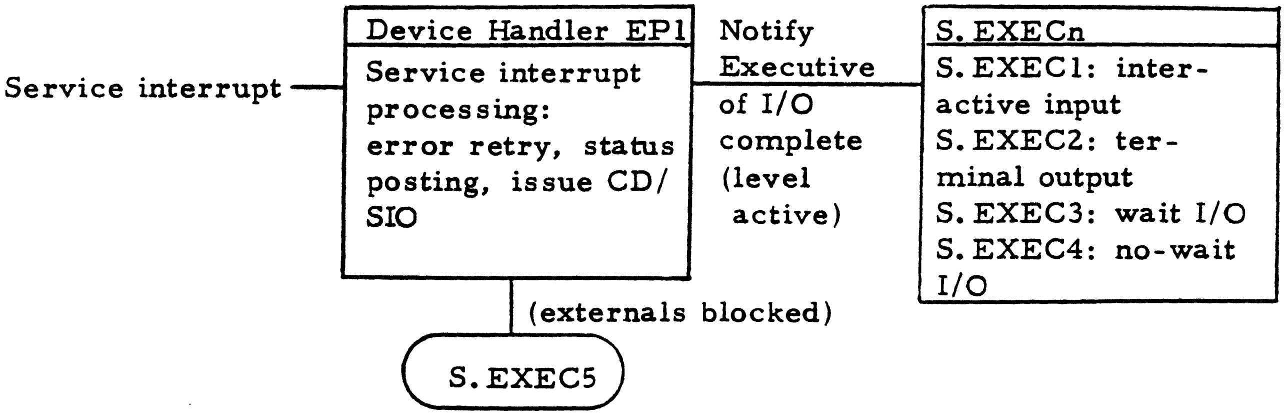
0	String Forward Address	
1	String Back Address	
2	Priority	
3		
4	PSD Word 1	
5	PSD Word 2	
6		
7		

1.3 I/O Overview from User Request to I/O Complete

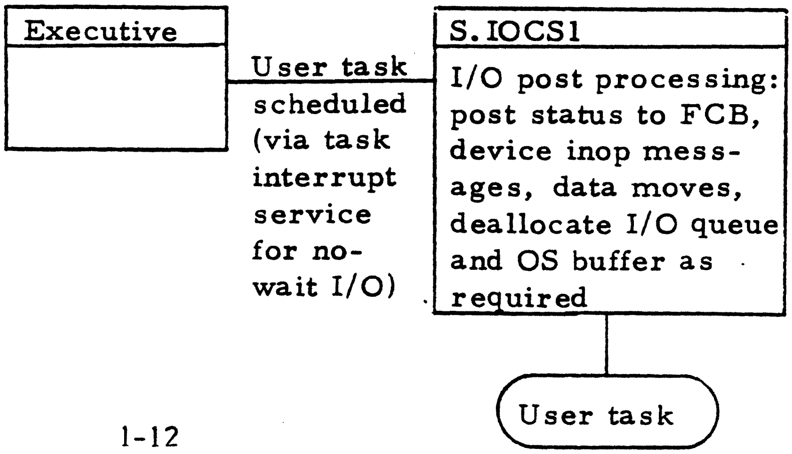
I/O Request Processing



Service Interrupt Processing


Service interrupt
Device Handler EP1
Service interrupt processing: error retry, status posting, issue CD/SIO
Notify Executive of I/O complete (level active)
S.EXECn
S.EXEC1: interactive input
S.EXEC2: terminal output
S.EXEC3: wait I/O
S.EXEC4: no-wait I/O
(externals blocked)
S.EXEC5

Post I/O Processing


Executive
User task scheduled (via task interrupt service for no-wait I/O)
S.IOCS1
I/O post processing: post status to FCB, device inop messages, data moves, deallocate I/O queue and OS buffer as required
User task

1.4 Scheduler - Task Termination Interface

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

1.4.1 Exit Task

The exit task service is called by a task that wishes to terminate its execution in a normal fashion.

Outstanding I/O (Exit)

If an exiting task has outstanding I/O, further exit processing will be deferred until all outstanding I/O is complete. Any user end action routines associated with I/O which completes while a task is exiting, will result in a task abort.

Messages in Receiver Queue (Exit)

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

Outstanding Run Requests (Exit)

If an exiting task has outstanding run requests (with call back) for other tasks, further exit processing will be deferred until all such requests are complete. End action routine execution will be inhibited and end action not processed status will be reflected in the Run Request Block.

Run Requests in Receiver Queue (Exit)

If an exiting task has requests in its run receiver queue, they will be unlinked and treated as complete with abnormal status.

Task Abort Receiver (Exit)

A task abort receiver is not processed on task exit.

Files (Exit)

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

Resources (Exit)

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

1.4.2 Abort Task

The abort task service is called by a task that wishes to terminate its execution in an abnormal fashion. It may also be initiated by the system when a task encounters a system trap condition (e.g., undefined instruction, privilege violation, or non-present memory); or by a system service because of a parameter validation error. This service may also be asynchronously initiated by another task of the same ownername or by operator communications.

Asynchronous Abort

When a task wishes to abort another task, of the same ownername, it calls the asynchronous abort service. The task to be aborted may be in a ready-to-run state, or it may be in 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
Disc 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. The abort request processing will then proceed on behalf of the aborting task when it is selected for execution by the CPU scheduler.

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

Synchronous Aborts

When the currently executing task encounters an abort condition, the abort bit will be set in the DQE. The CPU scheduler will then process the abort request.

Outstanding I/O (Abort)

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

Messages in Receiver Queue (Abort)

All outstanding messages sent to an aborting task will be unlinked from the message receiver queue, and treated as complete with abnormal status.

Outstanding Run Requests (Abort)

If an aborting task has outstanding run requests (with call back) for other tasks, further abort processing will be deferred until all such requests are complete. End action routine execution will be inhibited, and task abort status will be reflected in the Run Request Block.

Run Requests in Receiver Queue (Abort)

If an aborting task has requests in its run receiver queue, it will be unlinked and treated as complete with abnormal status. If any run requests still remain in the queue, a new copy of the task is activated.

Abort Receiver (Abort)

If an aborting task has an abort receiver, control will be transferred to it. All outstanding operation or resource waits will have been completed, and all no-wait I/O or no-wait run requests (with call back) will have been completed when the abort receiver is entered. End action routines associated with no-wait operations which completed while the abort request was outstanding will not have been executed. Status bits reflecting this will be posted in the appropriate FCB's and RRB's. Any files open at the time the abort request was received will remain open on an abort receiver entry. Any resources allocated at the time the abort request was received will also remain allocated when the abort receiver is executed.

Files (Abort)

If an aborting task has no intercepting abort receiver, all files open at the time the abort request was encountered will be automatically closed.

Resources (Abort)

If an aborting task has no intercepting abort receiver, all previously allocated resources will be deallocated, and the task will no longer be active in the system.

1.4.3 Delete Task

The delete task service is called by the system on behalf of a task that encounters a second abort condition during processing of an initial abort request. This service may also be asynchronously initiated by another task of the same ownername or by operator communications.

Asynchronous Delete

When a task wishes to delete another task, of the same ownername, it calls the asynchronous delete service. The task to be deleted may be in a ready-to-run state, or it may be in a wait state (e.g., 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 will not be linked into the ready-to-run queue until memory scheduler processing is complete.

Synchronous Deletes

When the currently executing task encounters a delete condition, the delete task bit will be set in the DQE. The CPU scheduler will then process the delete request.

Outstanding I/O (Delete)

Delete processing will cause all outstanding I/O to be terminated (killed); if I/O is no wait it is forced into its end action receiver (if it has one) then I/O is terminated.

Messages in Receiver Queue (Delete)

All outstanding messages sent to a task being deleted will be unlinked from the message receiver queue, and treated as complete with abnormal status.

Outstanding Run Requests (Delete)

If a task being deleted has outstanding run requests for other tasks, any call back will be ignored.

Run Requests in Receiver Queue (Delete)

If a task being deleted has requests in its run receiver queue, it will be unlinked and treated as complete with abnormal status. If any run requests still remain in the queue, a new copy of the task is activated.

Abort Receiver (Delete)

Abort receivers are not processed for tasks being deleted.

Files (Delete)

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

Resources (Delete)

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

1.5 Scheduler-Debug Interface

Design Goals

The structure of the Scheduler-Debug interface is dictated by the following major design goals:

(1) DEBUG may optionally be associated with a task at task activation time, or subsequently associated with a terminal task when the break key is struck. DEBUG may also be associated with a task dynamically through a system service call.

(2) When a task that has DEBUG associated with it is executing, two methods of entering DEBUG are provided: (a) The executing task encounters a previously set DEBUG trap instruction, or (b) The terminal operator depresses the break key.

(3) Entering DEBUG mode via trap or break is allowed during execution of software (task) interrupt receivers (e.g., message, end action, and break).

(4) DEBUG will intercept any task aborts, automatically enter the DEBUG mode, and inform the operator of the abort reason.

(5) System entry into the abort receiver will be "soft" in that outstanding I/O requests will be completed, and files will remain open and allocated. This allows the operator the ability to correct and proceed from the environment that caused the abort condition.

Debug Entry Points

To accommodate the scheduler interface, and achieve the DEBUG design goals, DEBUG is organized into five entry points. These entry points are reflected by an address table (HAT) structure at the beginning of the DEBUG program. When DEBUG is loaded, the address of the DEBUG HAT is stored in T.DBHAT in the TSA. The first word of the HAT contains the number of DEBUG entry points. Subsequent words contain the address of the individual DEBUG entry points. The entry points provided are:

ENTRY POINT	DESCRIPTION
1	Debug startup
2	Debug restart
3	Trap/break
4	User break exit
5	Abort

Task Interrupt Status

DEBUG may examine a byte (DQE.ATI) in the dispatch queue entry to determine the status of task interrupts. When DEBUG is entered, DQE.ATI contains the definition of all active task interrupts.

BIT	LABEL	MEANING
0	Reserved	
1	DQE.AEA1	Active end action interrupt 1
2	DQE.ADM	Active Debug mode interrupt
3	DQE.AUB	Active user break interrupt
4	DQE.AEA	Active end action interrupt 2
5	DQE.AMI	Active message interrupt
6-7	Reserved	

TSA Stack Pushdown Level Interpretation

For all DEBUG entry points, except Restart, the context associated with the most recently interrupted task level will be contained in T.CONTXT. Nested levels of task interrupt will be contained in the TSA stack. Unless one of the task interrupt levels (other than DQE.ADM) is active, the TSA stack will be clean (empty) on entry to DEBUG. If task interrupts are active, the context storage in the TSA will be in reverse order of priority (e.g., highest priority = most recent). Note that in the active task interrupt bit assignments bit 0 is the lowest priority, etc.

Exit from DEBUG Mode

Whenever DEBUG is executing (regardless of entry point) the task is said to be in DEBUG mode. DEBUG mode may be 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

1.5.1 Entry Point 1 - Startup

This entry point is accomplished in either of two methods: (1) DEBUG is activated with the user task, or (2) the user task may issue a SVC call to load and execute DEBUG.

DEBUG Activated with User Task

The program activation service which runs on behalf of the task being activated, detects that DEBUG is to be activated with the task. When the task has been loaded, a special service is called to load DEBUG. Once DEBUG is loaded, the service will store the normal startup registers and PSD in a DEBUG context block in the TSA (T.CONTXT). The service will then adjust the stack in the TSA to enter DEBUG at the DEBUG startup entry point. When DEBUG is entered the stack is clean, DEBUG mode is set, and T.CONTXT contains the user task startup registers and PSD.

DEBUG Activated by Load and Execute (M.DEBUG) SVC

When the user task issues a load and execute DEBUG SVC, the system service will load DEBUG, store the user's registers and PSD in T.CONTXT, set DEBUG mode, and adjust the TSA stack for entry at DEBUG's startup entry point.

1.5.2 Entry Point 2 - Restart

This entry point will be executed when DEBUG wishes to terminate any outstanding I/O, discard any outstanding messages, and clear the TSA stack. A DEBUG restart is invoked by a DEBUG call to H.EXEC,24.

1.5.3 Entry Point 3 - Trap/Break

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

1.5.4 Entry Point 4 - User Break Exit

This entry point will be executed when the user task being debugged executes a break exit. Note that a user task being debugged may only execute his break receiver by giving a break command to DEBUG. DEBUG will in turn call H.EXEC,23. Normal break receiver entry is reserved for DEBUG use when DEBUG is associated with a task. At the time DEBUG's user break exit entry point is entered, T.CONTXT will contain the most recent level of pushdown from the TSA stack. The number of pushdowns in the TSA stack will vary based upon the number of active task interrupts (e.g., message, end action).

1.5.5 Entry Point 5 - Abort

This entry point will be executed when an abort request is received for the user task and no user abort receiver has been specified. The user task context at the time the abort was received will be in T.CONTXT in the TSA. If a task interrupt (e.g., message or break receiver) was in effect at the time the abort request was received, the TSA stack will be at the associated level of pushdown. Otherwise, the TSA stack will be clean.

Wait I/O Operation Status on Abort Receiver Entry

When the abort receiver is entered, any wait I/O operation will have first been completed. This means that if an abort request is received for a task with wait I/O outstanding, abort processing will be deferred until the wait I/O is complete. It should be noted that 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 will be posted in the FCB.

No-Wait I/O Operation Status on Abort Receiver Entry

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

File Status on Abort Receiver Entry

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

Inhibit of Abort Receiver Entry

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

Re-use of Abort Receiver

Privileged tasks may re-establish an abort receiver from within an abort receiver. This allows privileged tasks to enter their abort receiver more than once. Unprivileged tasks may establish a one-shot abort receiver, but will be aborted if an attempt is made to re-establish this receiver.

1.6 MPX-32 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.

Task Interrupt Priorities

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

Level Priority	Description
0	Reserved for operating system use
1	DEBUG
2	Break
3	End Action
4	Message
5	Normal Execution (Run Request)

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 DEBUG interrupt level is reserved for system use by tasks running in DEBUG mode. The end action interrupt level is used for system post processing of no-wait I/O, message, or run requests. It is also used for execution of user task specified end action routines. The normal execution level is used for run request processing and general base level task execution.

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 will defer the interrupt until a return is made to the user task execution area.

System Service Calls from Task Interrupt Levels

A task may 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 therefore illegal to issue an I/O request on any FCB which is busy or may have post processing outstanding.

Task Interrupt Context Storage

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

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 will be processed.

In addition, the following services can be used to 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 via 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 via 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.

Note that any deferred task interrupts will be processed when a lower level task interrupt calls the M.ANYW,M.EAWAIT or M.WAIT services.

User Break Interrupt Receivers

A task may 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 a M.INT service call which 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 pseudo interrupt (break) level.

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 adddress may optionally be specified. If specified, the routine will be entered at the end action priority level from the appropriate system post processing 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 "continue-wait-for-any" option. An interrupt exit will normally return to the interrupted context. A task may, however, have issued 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) will place the task in an interruptible wait state, allowing the execution of post processing 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 either 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.

User Message Receivers

A task may 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. Upon entry to the message interrupt receiver, R1 contains the address of the queue entry (MRRQ) in memory pool. The receiver may optionally call a service M.GMSGP to store the message in a user receiver buffer. No-wait I/O is permitted in conjunction with the M.WAIT service. After appropriate processing, the message interrupt level may be reset by calling the M.XMSGR system service to exit from the message interrupt receiver.

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 will be queued from the DQE of the specified task. If the specified task is not active, it will first be activated. When a task begins execution as the result of a run request, R1 contains the address of the run request queue entry. The receiver may optionally call a service M.GRUNP to store the run parameters in a user receiver buffer. After appropriate processing, the run receiver task may exit by calling the M.XRUNR system service. Any queued run requests will then be processed.

User Abort Receivers

User abort receivers execute at the normal task execution (base) level. The user task may optionally establish an abort receiver by calling the M.SUAR monitor service. If an abort condition is encountered during task operation, control will be transferred to it. Upon entry, any active software interrupt level will have been reset, all outstanding operations or resource waits will have been completed, and all no-wait requests will have been processed. End action routines associated with no-wait requests which completed while the abort was outstanding will not have been executed. Status bits reflecting this will be posted in the appropriate FCB's and PSB's. Any files opened or resources allocated at the time the abort condition was encountered will remain opened and/or allocated when the abort receiver is executed. The TSA stack will be clean, and the context at the time the abort condition was encountered will be stored in T.CONTXT. When the abort receiver is entered, R6 contains a status byte reflecting task interrupt status at the time the abort condition was encountered.

Bit	Meaning When 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 will be deleted.

1.7 MPX-32 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 which issue the requests are called "sending task services".

1.7.1 Receiving Task Services

Establishing Receiver Capability

Establishing Message Receivers -- In order 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 monitor service M.RCVR, and providing the receiver routine address as an argument with the call.

Establishing Run Receivers -- Any valid task may be a run receiver. Although a set of special run receiver services are provided, in the most simple case, they need not be used. 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 will be 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 will be activated (multi-copied) to process this request. When a single-copied task exits, any queued run requests will be executed. If a run request is issued for a task that is not currently active, an activation will automatically be performed.

Execution of the Receiver Program

Execution of Message Receiver Programs -- When a task is active and has a message receiver established, it may receive messages sent from other tasks. A message sent to this task will cause 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 will be entered at its cataloged transfer address. A run receiver executes at the normal task execution (base) level.

Obtaining the Passed Parameters

Obtaining Message Parameters -- When the message receiver is entered, R1 contains the address of the message queue entry in memory pool. The task may optionally retrieve the message directly from memory pool, or the task may 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, R1 contains the address of the run request queue entry in memory pool. The task may optionally retrieve the run request parameters directly from memory pool, or the task may 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.

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 the return parameter buffer, and the number of bytes (if any) to be returned to the sending task. The RXB will also contain 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 will be examined for any additional messages to process. If none exist, a return to the base level interrupted context will be performed.

Exiting the Run Receiver Task -- When run request processing is complete, the task may use 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 will be returned. Only completion statuus will be 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, any queued run request will cause the task to be re-executed. If the run receiver queue for this task is empty, a standard exit will be 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 arguument with the call. The RXB will contain the address of the return parameter buffer, and the number of bytes (if any) to be returned to the sending task. The RXB will also contain a return status byte to be stored in the Parameter Send Block (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 will be put into a wait-state, waiting for the receipt of new run requests. Execution of the task will not resume until such a request is received. If the "terminate" exit option is used, any queued run requests will be processed. If the run receiver is empty, however, a standard exit will be performed.

Waiting for the Next Request

In addition to the wait options described under the heading "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 may invoke a service call (M.SUSP) to enter a wait-state until the next message is received.

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

1.7.2 Sending Task Services

Sending the Request

Message Send Service -- A task may send a message to another active task, providing that task 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, as well as 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 may send a run request to any active or inactive task, identifying the task by load module name or by task number if the task is multicopied. When the run request service (M.SRUNR) 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 run request parameters to be sent, any parameters to be returned, scheduler and user status, as well as the address of a user end action routine. No-wait and no-call-back mode control options are also provided.

Waiting for Request Completion

Waiting for Message Completion -- A message may be sent in either the wait or no-wait mode. If the wait mode is used, execution of the sending task will be deferred until processing of the message by the destination task is complete. If the no-wait mode is used, execution of the sending task will continue as soon as the request has been queued. The operation in progress bit in the scheduler status field of the PSB may be examined to determine completion. A sending task may issue 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 will cause 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.

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 will have been 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.

Parameter Send Block (PSB)

The Parameter Send Block (PSB) is used to describe 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 (doubleword bounded) must be presented as an argument when either the M.SMSGR or M.SRUNR services are invoked.

Parameter Send Block (PSB)

Word	0 7	8 15	16 23	24 31
0 1	Load Module Name (PSB.LMN) or Task Number if Message (PSB.TSKN) or if run request to multicopied load module			
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 (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 Name - Characters 1-4 of the name of the load module to receive the run request, or
Task Number - The task number of the task to receive the message, or task number of the multicopied task to receive the run request.

WORD 1

Bits 0-31 Load Module Name - Characters 5-8 of the name of the load module to receive the run request, or zero if Word 0 contains the task number.

WORD 2

Bits 0-7 Priority - This field contains the priority of the send request (1-64). If the value of this field is zero, the priority used will default to the execution priority of the sending task. This field is not examined if the sending task is not a privileged program.

Bits 8-15 Reserved.

Bits 16-31 Number of Bytes to be Sent - This field specifies the number of bytes to be passed (0-768) with the message or run request.

WORD 3

Bits 0-7 Reserved.

Bits 8-31 Send Buffer Address - This field 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 (0-768) that may be accepted as returned parameters.

Bits 16-31 Number of Bytes Actually Returned - This field is 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 into which any returned parameters will be 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 - This bit encoded field contains completion status information posted by the operating system as follows:

Bit	Meaning When Set
0	Operation in progress (busy) (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 (may be examined in abort receiver to determine incomplete operation) (PSB.EANP)

6-7 Reserved.

Bits 8-15 Processing Start (Initial) Status - This value encoded field contains initial status information posted by the operating system as follows:

Code Definition

0 Normal initial status (PSB.IST)

1 Message request task number invalid (PSB.TSKE)

2 Run request load module name not found in System Master Directory (SMD) (PSB.LMNE)

3 File associated with run request load module name is password protected (PSB.LMPE)

4 File associated with run request load module name does not have a valid load module format (PSB.LMFE)

5 Dispatch Queue Entry (DQE) space is unavailable for activation of the load module specified by a run request (PSB.DQEE)

6 An I/O error was encountered while reading the SMD to obtain the file definition of the load module specified in a run request (PSB.SMIO)

7 An I/O error was encountered while reading the file containing the load module specified in a run request (PSB.LMIO)

8-9 Reserved

10 Invalid priority specification. Note: An unprivileged task may not specify a priority which is higher than its own execution priority (PSB.PRIE)

11 Invalid send buffer address (PSB.SBAE)

12 Invalid return buffer address (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 - As defined by destination task.

Bits 24-31 Options - This field contains user request control specification. It is bit encoded as follows:

Bit	Meaning When 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 is said to have been issued in the "no call-back mode". (PSB.NCBM)

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 both message and run requests. The address of the PRB must be presented when either the M.GMSGP or M.GRUNP services are invoked by the receiving task.

Parameter Receive Block (PRB)

<table>
<tr><th>Word</th><th>0 7</th><th>8 15</th><th>16 31</th></tr>
<tr><td>0</td><td>Status
(PRB.ST)</td><td colspan="2">Parameter Receiver Buffer Address
(PRB.RBA)</td></tr>
<tr><td>1</td><td colspan="2">Receiver Buffer
Length(PRB.RBL)</td><td>Number of Bytes
Actually Received
(PRB.ARQ)</td></tr>
<tr><td>2</td><td colspan="3">Ownername of Sending Task (Word 1) (PRB.OWN)</td></tr>
<tr><td>3</td><td colspan="3">Ownername of Sending Task (Word 2)</td></tr>
<tr><td>4</td><td colspan="3">Task Number of Sending Task (PRB.TSKN)</td></tr>
</table>

WORD 0

Bits 0-7 Status-value encoded status byte

Code	Definition
0	Normal status
1	Reserved
2	Invalid receiver buffer address (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 - This field contains the word address of the buffer, into which the sent parameters will be stored.

WORD 1

Bits 0-15 Receiver Buffer Length - Contains the length of the receiver buffer (number of bytes).

Bits 16-31 Number of Bytes Actually Received - This value is set by the operating system and is clamped to a maximum equal to the receiver buffer length.

WORDS 2,3

Bits 0-63 Ownername of Sending Task - Set by the operating system to contain the ownername of the task which issued the parameter send request.

WORD 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.

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-type 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.

Receiver Exit Block (RXB)

Word	0 7	8 15	16 31
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 which issued the send request.

WORD 1

Bits 0-7 Options - This field contains receiver exit control options. It is encoded as follows:

Value	Exit Type	Meaning
0	M.XRUNR	Wait for next run request.
	M.XMSGR	Return to point of task interrupt.
1	M.XRUNR	Exit task, process any additional run requests. If none exist, perform a standard exit.
	M.XMSGR	N/A

Bits 8-15 Reserved.

Bits 16-31 Number of Bytes to be Returned - contains the number of bytes to be returned on a message or receiver run exit.

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 manufactured 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 will contain the address of the MRRQ in memory pool. The receiver program may optionally 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.

Message or Run Request Queue Entry (MRRQ)

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	Post Processing Service PSD Word 1, or Sending Task Ownername Word 1 (MQ.PPSD)			
5	Post Processing Service PSD Word 2, or Sending Task Ownername Word 2 (MQ.PPSD)			
6	Passed Parameter Quantity (Bytes) or Number Bytes of Storage Space (MQ.PPQ)		Return Parameter Buffer Length (Bytes) or Number 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)
	Variable Length Storage Area for Passed and Returned Parameters			

WORD 0

Bits 0-31 String Forward Address - Address of next entry of top-to-bottom list.

WORD 1

Bits 0-31 String Backward Address - Address of next entry in bottom-to-top list.

WORD 2

Bits 0-7 Priority - Priority (1-64) of this request.

Bits 8-31 Address of Parameter Send 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 - This field contains the task activation sequence number of the task which issued the request.

WORDS 4-5

Bits 0-63 Post Processing Service PSD - The PSD of the appropriate post processing service which runs on behalf of the task which issued the request. This slot is also used to contain sending task ownername.

WORD 6

Bits 0-15 Passed Parameter Quantity - The number of bytes sent to the destination task.

Bits 16-31 Return Parameter Buffer Length - Contains the length (bytes) of the return parameter buffer in the task which issued the request.

WORD 7

Bits 0-15 Scheduler Status - This field is used to contain status information to be posted in the scheduler status field of the PSB upon request completion (see PSB format).

Bits 16-23 User Status - As defined by destination task.

Bits 24-31 Options - This field contains user request control specifications. It is bit encoded as follows:

Bit	Meaning When Set
24	Request is in no-wait mode.
25	Request is in no-call-back mode (no-wait, no status, no return parameters)

Messages and Run Request Services Summary

The following table is provided as a summary of message and run request services provided by the MPX-32 system.

Message and Run Request Services Summary

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,time1 or M.EXIT	M.XMSGR rxbaddr	Exit receiver
N/A	M.ANYW time1 message or M.SUSP taskno,time1	Wait for receipt of next
Send Services		
M.SRUNR psbaddr	M.SMSGR psbaddr	Send request
M.ANYW time1	M.ANYW time1 M.EAWAIT time 1	Wait for any request competition
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, otherwise contains negative number of time units to be used as a wait time-out value.

1.8 MPX-32 Device Address Specification

Under MPX-32 device addresses are specified using a combination of three levels of identification. They are Device Type, Device Channel/Controller Address, and Device Address/Subaddress.

A device may be specified using the generic device type only, which will result in allocation of the first available device of the type requested.

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 and final method of device selection requires specification of the Device Type, Channel/Controller, and Device Address/Subaddress. This method allows specification of a specific device.

Examples of the three methods of device specification follow:

Type 1 - Generic Device Class

```
$ASSIGN3 DEV=M9
```

In this example the file associated with LFC "DEV" will be allocated to any 9-track tape unit on any channel.

Type 2 - Generic Device Class and Channel/Controller

```
$ASSIGN3 DEV=M910
```

In this example the file associated with LFC "DEV" will be 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

```
$ASSIGN3 DEV=M91001
```

In this example the file associated with LFC "DEV" will be allocated to the 9-track tape unit 01 on channel 10. The specification is invalid if unit 01 on channel 10 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:

```
$ASSIGN3 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 would be constructed as follows:

DISC

DC	Any Disc
DM	Any Moving Head Disc
DM08	Any Moving Head Disc on Channel 08
DM0801	Moving Head Disc 01 on Channel 08
DF	Any Fixed Head Disc
DF04	Any Fixed Head Disc on Channel 04
DF0401	Fixed Head Disc 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
M7	Any 7-track Magnetic Tape
M712	Any 7-track Magnetic Tape on Channel 12
M71201	7-track Magnetic Tape 01 on Channel 12

CARD EQUIPMENT

CD	Any CR/CP
CR	Any CR
CR78	Any CR on Channel 78
CR7800	CR on Channel 78 Subaddress 00
CP	Any CP
CP7C	Any CP on Channel 7C
CP7C00	CP on Channel 7C Subaddress 00

LINE PRINTER

LP	Any LP
LP7A	Any LP on Channel 7A
LP7A00	LP on Channel 7A Subaddress 00

MPX-32
Device Type Codes

Dev Type Code	Device Mnemonic	Device Description
00	CT	Operator Console (Not Assignable)
01	DC	Any Disc Unit
02	DM	Any Moving Head Disc
03	DF	Any Fixed Head Disc
04	MT	Any Magnetic Tape Unit
05	M9	Any 9-track Magnetic Tape Unit
06	M7	Any 7-track Magnetic Tape Unit
07	CD	Any Card Reader/Reader Punch
08	CR	Any Card Reader
09	CP	Any Card Reader Punch
0A	LP	Any Line Printer
0B	PT	Any Paper Tape Reader Punch
0C	TY	Any Teletype Compatible Device (other than Console)
0D	CT	Operator Console (Assignable)
0E	FL	Floppy Disc
0F	NU	Any Null Device
10	CA	Communications Adapter
11	U0	User Supplied Device
12	U1	User Supplied Device
13	U2	User Supplied Device
14	U3	User Supplied Device
15	U4	User Supplied Device
16	U5	User Supplied Device
17	U6	User Supplied Device
18	U7	User Supplied Device
19	U8	User Supplied Device
1A	U9	User Supplied Device
1B	LF	Line Printer/Floppy Controller (used only with SYSGEN)

1.9 MPX-32 CPU Scheduling

The MPX-32 CPU scheduler is responsible for allocating 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 enters (is cataloged into) the system. Execution eligibility is determined by the task's readiness to run.

Execution Priorities

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

Real Time Priority Levels (1-54)

Scheduling of Real Time Tasks in MPX-32 occurs on a strict priority basis. The system does not impose time-slice, priority migration, or any other scheduling algorithm which will interfere 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 may, of course, always be preempted by a higher priority real time task that is ready to run.

Time Distribution Priority Levels (55-64)

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

Priority Migration

The specified execution priority of a time distribution task is used as the tasks 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 preempted 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 clamped at the task's base execution level.

Situational Priority Increments

Time distribution tasks are given situational priority increments in order 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 will migrate 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

Time Quantum Controls

The MPX-32 system allows for the specification of two time quantum values at system generation time. If these values are not specified, system default values will be 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 preemption 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 may be preempted and/or outswapped by any higher priority task. When a task's full time quantum has expired, it is relinked 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 is reset when the task voluntarily relinquishes CPU control (e.g., suspend, wait I/O, etc.).

State Chain Management

The current state of a task (e.g., ready to run, waiting for I/O, etc.) 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 Disc Space - SWDC
25. Waiting for Peripheral Device - SWDV
26. Waiting for FISE - SWFI
27. Waiting for Memory - MRQ
28. Waiting for Memory Pool - SWMP

1.10 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 have allocated concurrently. The number of entries in each area is established as follows.

FAT/FPT Area

Non-shared task: one FAT and FPT entry for each cataloged assignment plus one entry for each Job Control or TSM assignment that does not override a cataloged assignment plus the number specified on the FILES Cataloger directive.

Shared task: the number specified on the FILES Cataloger directive.

Blocking Buffer Area

Non-shared task: from the assignments resulting from merging Cataloger and Job Control or TSM assignments, one buffer for each ASSIGN1, plus one buffer for each ASSIGN3 to a magnetic tape or disc unit on which the unblocked option is not specified plus one buffer for each ASSIGN2 plus the number specified on the BUFFERS Cataloger directive.

Shared task: the number specified on the BUFFERS Cataloger directive.

Cataloger, Job Control, and TSM ASSIGN1 and ASSIGN3 directives are modified by the addition of an "unblocked" specification as follows.

```
ASSIGN1  fc=file,(password),(U)
ASSIGN3  fc=device,(U)
```

where: U specifies that I/O to the file or device is to be unblocked. If this parameter is absent, I/O to the file or magnetic tape or disc device is blocked.

Files specified on ASSIGN2 directives are blocked by default.

The following Cataloger directives are added.

FILES number

where: number specifies the maximum number of dynamically allocated file codes that a non-shared task has allocated concurrently. It specifies the maximum number of file codes that a shared task has allocated concurrently.

BUFFERS number

where: number specifies the maximum number of dynamically allocated file codes assigned to blocked files or devices that a non-shared task has allocated concurrently. It specifies the maximum number of file codes assigned to blocked files or devices that a shared task has allocated concurrently.

"Files" and "buffers" override parameters may be specified to the Parameter Task Activation (M.PTSK) system service. These parameters allow addition of FILES and BUFFERS Job Control and TSM directives if required by a future "load and go" capability.

1.11 Indirectly Connected Interrupts

An indirectly connected interrupt is an interrupt that may be associated with a MPX-32 task. When the interrupt occurs, the associated task will be resumed. An interrupt is declared as indirectly connected at system generation (SYSGEN) time. This declaration will cause 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 a MPX-32 task when the M.CONN system service is invoked. A task may be disconnected from an interrupt level by invoking the M.DISCON system service.

Connect Task To Interrupt Service (M.CONN)

The M.CONN system service associates a MPX-32 task with an external interrupt that was declared at system generation time to be indirectly connected. When called, M.CONN is presented the priority level of the interrupt and the task activation sequence number (TASKNO) of the task. The TASKNO is first validated to insure that it is both currently active and of the same ownername as the calling task. 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 will contain the interrupt priority level if the task is already connected. If the task is not previously connected, the M.CONN service will search 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 1 of the ITLB to reflect the linkage. DQE.ILN in the DQE is then set to contain the interrupt priority level. Note: The task will be automatically disconnected from the interrupt on abort, delete, or exit.

Disconnect Task From Interrupt Service (M.DISCON)

The M.DISCON system service disconnects a MPX-32 task from an external interrupt to which it had previously been connected. When called, M.DISCON is presented the task activation sequence number (TASKNO) of the task as an argument with the call. If the specified task is not connected to an interrupt, DQE.ILN in the DQE will be equal to zero and the request will be ignored. Otherwise, DQE.ILN will contain the external interrupt priority level. M.DISCON will use this priority level to locate the linkage block (ITLB) in the linkage table (ITLT). The DQE address (word 1 of the ITLB) will then be cleared to mark the level as disconnected. DQE.ILN will also be cleared in the DQE of the specified task.

Indirectly Connected Task Linkage Table (ITLT)

The Indirectly Connected Task Linkage Table (ITLT) is a variable length table built by the system generation program (SYSGEN) and it contains an entry for each interrupt specified as being 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.

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.

Indirectly Connected Task Linkage Block (ITLB)

Word	0	31
0	Priority Level	[DATAW X'YY']
1	DQE Address of Ind. Conn Prog.	[DATAW 0]
2	Old PSD Word 1	[DATAW 0]
3	Old PSD Word 2	[DATAW 0]
4	New PSD Word 1	[GEN 1/1, 12/0, 19/W($+2W)]
5	New PSD Word 2	[GEN 1/1, 14/0, 1/1, 1/0, 1/0, 14/0]
6	Increment Global Interrupt Count Inst.	[ABM 31,C.GINT]
7	Save Register Inst.	[STF R0,$+9W]
8	Branch and Link to ICP Routine	[BL ICP]
9	Address of Reg. Save Area for S.EXEC5 Call	[LA X2, $+7W]
10	Old PSD for S.EXEC5 Call	[LD R6, $-8W]
11	Block External Interrupts	[BEI]
12	Deactivate Interrupt	[DAI X'YY']
13	Branch Back for S.EXEC5 Call	[BL ICP.20]
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 equal to zero by SYSGEN. Initialized by M.CONN system service.

Words 2,3

Bits 0-63 Old PSD - Old PSD slot of Interrupt Control Block. Used for storage of the PSD associated with the interrupted context. Initially set equal to zero by SYSGEN. The dedicated interrupt location (IVL) is initialized by SYSGEN to contain the address of word 2 of the ITLB.

Words 4,5

Bits 0-63 New PSD - New PSD slot of Interrupt Control Block. Contains the PSD 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 - Execution will begin at this location upon occurrence of the associated interrupt. Contains an add bit in memory instruction to increment the global interrupt count. It must be the first instruction executed in ICP. This location is initialized by SYSGEN to contain a ABM 31,C.GINT.

Word 7

Bits 0-31 Save Registers Instruction - Contains a store file instruction to save all 8 registers in words 16-23 of the ITLB. This location is initialized by SYSGEN to contain a 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 9-13 Control is returned to this location after S.EXEC14 is called in the ICP routine. Set up is made for exiting the interrupt, and then control is tranferred back to ICP for the S.EXEC5 call.

Words 14-15 Reserved for future use.

Words 16-23 Register Save Area.

Indirectly Connected Interrupt Program (ICP)

The Indirectly Connected Interrupt Program (ICP) is a single copied routine that processes all indirectly connected external interrupts. It is entered in unblocked mode with the end address (+1W) of the linkage block (ITLB) in R0. The global interrupt count will have been incremented within the ITLB and the registers from the interrupted context will have been stored in Words 16-23 of the block. When ICP is entered, it will check ITLB word 1 to verify connection of the interrupt to a MPX-32 task. If the interrupt is not connected, it will be ignored and ICP will transfer back to the ITLB to exit the interrupt. If ITLB word 1 contains a DQE address, the associated task will be resumed by calling S.EXEC14 who will return control to word 9 of the ITLB. Set up is made for exiting the interrupt within the ITLB, and then execution is transferred back to ICP.20 for the S.EXEC5 exit.

```
H.ICP       TRR     R0,R1          ITLB END ADDRESS TO R1
            LW      R2,-8W,R1      GET DQE ADDR OF IND CONN TASK
            BCF     ZR,ICP.10      CONTINUE IF CONNECTED
            TRSW    R0             BRANCH BACK TO ITLB TO EXIT
ICP.10      BU      S.EXEC14       RESUME PROGRAM
ICP.20      BL      S.EXEC5
```

1.12 Miscellaneous System Macros

1.12.1 M.BACK

Functional Description

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

Macro Call

Calling Sequence:

M.BACK ARG1, [ARG2], ARG3

where:

ARG1 is the FCB address

ARG2 is the optional character R to specify record (otherwise file is assumed)

ARG3 contains the number of record or file marks to be backspaced (the number specified must be word-scaled, e.g., 1W for 1 record)

1.12.2 M.CALL

Functional Description

This macro generates a supervisor call instruction.

Macro Call

Calling Sequence:

```
        M.CALL  ARG1,ARG2
```

where:

ARG1 is the name of a system module

ARG2 is an entry point number (1,2,3,...) within the system module

1.12.3 M.CLSE

Functional Description

This macro marks the file closed to subsequent service, writes an optional end-of-file mark, and performs an optional rewind.

Macro Call

Calling Sequence:

```
        M.CLSE  ARG1,[ARG2],[ARG3]
```

where:

ARG1 is the FCB address

ARG2 is the optional character string EOF, the presence of which will cause an end-of-file mark to be written

ARG3 is the optional character string REW, the presence of which will cause the file to be rewound

1.12.4 M.DFCB

Functional Description

This macro allows the user to create a file control block (FCB) and set the appropriate parameters and specifications common to I/O requests which will be issued for the file.

Macro Call

Calling Sequence:

```
M.DFCB   ARG1,ARG2,ARG3,ARG4,ARG5, [ARG6] ,
         [NWT],   [NER],   [DFI],   [NST],   [RAN] ,
         ⎡ASC⎤    ⎡LDR⎤    ⎡INT⎤    ⎡EVN⎤    ⎡556⎤
         ⎣BIN⎦ ,  ⎣NLD⎦,   ⎣PCK⎦,   ⎣ODD⎦,   ⎣800⎦
```

where:

ARG1 — is the ASCII character string to be used as the symbolic label for the address of the FCB

ARG2 — is the ASCII character string to be used as the logical file code in the FCB (this file code can be one to three characters in length)

ARG3 — is the optional transfer count (bytes)

ARG4 — is the optional data transfer address

ARG5 — is the optional error return address

ARG6 — is the optional random access address expressed as the hexadecimal block number (original zero) relative to the base of the random access file

NWT — is the optional character string which sets the no-wait I/O specification indicator

NER — is the optional character string which sets the inhibit peripheral error processing indicator

DFI — is the optional character string which sets the inhibit data formatting indicator

NST — is the optional character string which sets the inhibit status testing indicator

RAN — is the optional character string which sets the random access mode indicator

ASC or BIN — is the optional character string which will set the forced ASCII or forced binary mode specification, respectively, for read or punch operations performed when the file code for this file is assigned to a card reader or card reader/punch device

LDR or NLD — is the optional character string which will specify skip leader or do not skip leader, respectively, when the file code for this file is assigned to a paper tape reader/punch device

INT or PCK — is the optional character string which will specify interchange or packed modes, respectively, when the file code for this file is assigned to a magnetic tape device

EVN or ODD — is the optional character string which will specify even or odd parity, respectively, when the file code for this file is assigned to a magnetic tape device

556 or 800 — is the optional character string which will specify 556 or 800 bpi tape densities, respectively, when the file code for this file is assigned to a magnetic tape device

1.12.5 M.DFCBE

Functional Description

This macro allows the user to create an expanded file control block (FCB) and set the appropriate parameters and specifications common to I/O requests which will be issued for the file.

Macro Call

Calling Sequence:

```
M.DFCBE ARG1,ARG2,ARG3,ARG4,ARG5, [ARG6]
        [NWT],  [NER],  [DFI],  [NST],    [RAN],
        ⎡ASC⎤   ⎡LDR⎤   ⎡INT⎤   ⎡EVN⎤     ⎡556⎤
        ⎣BIN⎦,  ⎣NLD⎦,  ⎣PCK⎦,  ⎣ODD⎦,    ⎣800⎦,ARG7,ARG8
```

where:

ARG1 — is the ASCII character string to be used as the symbolic label for the address of the FCB

ARG2 — is the ASCII character string to be used as the logical file code in the FCB (this file code can be one to three characters in length)

ARG3 — is the optional transfer count (bytes)

ARG4 — is the optional data transfer address

ARG5 — is the optional wait I/O error return address

ARG6 — is the optional random access address expressed as the hexadecimal block number (original zero) relative to the base of the random access file

NWT — is the optional character string which sets the no-wait I/O specification indicator

NER — is the optional character string which sets the inhibit peripheral error processing indicator

DFI — is the optional character string which sets the inhibit data formatting indicator

NST — is the optional character string which sets the inhibit status testing indicator

RAN — is the optional character string which sets the random access mode indicator

ASC or BIN — is the optional character string which will set the forced ASCII or forced binary mode specification, respectively, for read or punch operations performed when the file code for this file is assigned to a card reader or card reader/punch device

LDR or NLD — is the optional character string which will specify skip leader or do not skip leader, respectively, when the file code for this file is assigned to a paper tape reader/punch device

INT or PCK — is the optional character string which will specify interchange or packed modes, respectively, when the file code for this file is assigned to a magnetic tape device

EVN or ODD — is the optional character string which will specify even or odd parity, respectively, when the file code for this file is assigned to a magnetic tape device

556 or 800 — is the optional character string which will specify 556 or 800 bpi tape densities, respectively, when the file code for this file is assigned to a magnetic tape device

ARG7 — is the optional no-wait I/O normal end-action service address

ARG8 — is the optional no-wait I/O error end-action service address

1.12.6 M.EIR

Functional Description

This macro is called by the resident system modules initialization entry points at entry. It stores register 0 for later recall by M.XIR, the initialization entry point exit macro (see Section 1.12.24).

Macro Call

Calling Sequence:

```
        M.EIR
```

1.12.7 M.FWRD

Functional Description

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

Macro Call

Calling Sequence:

```
        M.FWRD  ARG1,[ARG2],ARG3
```

where:

ARG1	is the FCB address
ARG2	is the optional character R to specify record (otherwise file is assumed)
ARG3	contains the number of record or file marks to be advanced (the number specified must be word-scaled, e.g., 1W for 1 record)

1.12.8 M.INIT

Functional Description

This macro is used to provide for user initialization of device handler parameters via entry point 8. The code generated by this macro is executed by SYSGEN and overlayed.

Macro Call

Calling Sequence:

```
        M.INIT  ARG1,[ARG2],[ARG3],[ARG4],[ARG5],[ARG6],
                [ARG7],[ARG8],[ARG9],[ARG10],[ARG11],
                [ARG12],[ARG13],[ARG14],[ARG15],[ARG16],
                [ARG17]
```

where:

ARG1	is the entry point truncated label, e.g., MT0 for magnetic tape handler. This argument must be three ASCII characters. The first two represent the device mnemonic and the third is zero (0).
ARG2	is the optional specification for modification of the TD 2000 level test to no-operation
ARG3-ARG17	are the optional special parameter addresses to be initialized, i.e., SPA's

Usage

M.INIT MT0,,SPA1,SPA2,SPA3

This macro, when placed as the last source statement in the device handler, will provide the necessary code to initialize the handler. The 'HAT' must be modified to specify entry point 8 and an additional entry must be made in the table (ACH MT00.8).

1.12.9 M.INITX

Functional Description

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.

Macro Call

Calling Sequence:

```
M.INITX ARG1,ARG2
```

where:

ARG1	contains the basic instruction or command
ARG2	contains a mask which is ORed with ARG1

NOTE:

Whenever this macro is called, register 5 must be preloaded with the properly positioned priority level or device address.

1.12.10 M.IOFF

Functional Description

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

Macro Call

Calling sequence:

```
M.IOFF
```

1.12.11 M.IONN

Functional Description

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

Macro Call

Calling Sequence:

```
M.IONN
```

1.12.12 M.IVC

Functional Description

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

Macro Call

Calling Sequence:

```
        M.IVC  ARG1,ARG2
```

where:

ARG1	is the register number containing the interrupt level
ARG2	is the handler entry point address label

1.12.13 M.KILL

Functional Description

This macro generates a request for the System Override Interrupt/Trap level. A HALT instruction follows the request in the event that the system is not in the Protect mode.

Macro Call

Calling Sequence:

```
        M.KILL  ARG1
```

where:

ARG1	contains the address of a four character ASCII crash code

1.12.14 M.MODT

Functional Description

This macro builds an entry in the module address table.

Macro Call

Calling Sequence:

```
        M.MODT  ARG1,ARG2
```

where:

ARG1	is the address label of the module's HAT table
ARG2	is the module number

1.12.15 M.OPEN

Functional Description

This macro is used for control gating purposes. It results in the termination of context switching being inhibited.

Macro Call

Calling Sequence:

```
M.OPEN
```

1.12.16 M.RTNA

Functional Description

This macro provides the facility to return to the caller from a system module to some address other than that specified by the saved PSW. It is intended to be used primarily for denial returns. It operates functionally in the same way as the M.RTRN macro. The interrupt handler tests for the presence of an address specification in the parameter and replaces the saved PSW if an address is found.

Macro Call

Calling Sequence:

```
M.RTNA addr,r1,r2,r3,...,r8
```

where:

addr — is the register number of the register containing the address to which to return control

r1,...,r8 — is a list of the register numbers (0,1,2,...,7) identifying the registers to be preserved through the register pop-up

1.12.17 M.RTRN

Functional Description

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 to be preserved) and returns to the location specified by the saved PSW.

Macro Call

Calling Sequence:

```
M.RTRN r1,r2,r3,...,r8
```

where:

r1...,r8 is a list of register numbers (0,1,2,...,7) identifying the registers that are to be preserved through the register pop-up

1.12.18 M.SHUT

Functional Description

This macro is used for control gating purposes. It results in context switching being inhibited. This macro should not be used in a user task that is eligible for IPU execution (see M.USHUT).

Macro Call

Calling Sequence:

```
M.SHUT
```

1.12.19 M.SPAD

Functional Description

At each register push-down level, twenty-two 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 will perform any memory reference operating on at least a word boundary (i.e., LW, STF, ARMD, DVMW, etc.) or any bit in memory operation (i.e., TBM, SBM, ABM, ZBM).

Macro Call

Calling Sequence:

```
M.SPAD mnem,r,s,x
```

where:

mnem is an instruction mnemonic defining the operation that is to be performed

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

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

x is an index register number (1,2, or 3) that will be utilized in performing the operation

1.12.20 M.SVCT

Functional Description

This macro builds one entry in the SVC table for each of the SVC's 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.

Macro Call

Calling Sequence:

```
M.SVCT  ARG1,ARG2
```

where:

ARG1 is the address label for the calling module's prototype SVC table

ARG2 is the number of SVC entries in the module's prototype SVC table

1.12.21 M.TRAC

See Chapter 7 in the MPX-32 Technical Manual.

1.12.22 M.TYPE

Functional Description

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

Macro Call

Calling Sequence:

```
M.TYPE  ARG1,ARG2 ,ARG3,ARG4
```

where:

ARG1 is the output message address

ARG2 is the output transfer count

ARG3 is the optional input message address

ARG4 is the optional input transfer count

1.12.23 M.USHUT

Functional Description

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

Macro Call

Calling Sequence:

```
        M.USHUT
```

1.12.24 M.XIR

Functional Description

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.

Macro Call

Calling Sequence:

```
        M.XIR  ARG1
```

where:

ARG1 — is the address label of the module's HAT table

1.12.25 HMP.INIT

Functional Description

This macro is used to provide for user initialization of MIOP device handler parameters via entry point 8. The code generated by this macro is executed by SYSGEN and overlayed.

Macro Call

Calling Sequence:

```
        HMP.INIT ARG1
```

where:

ARG1 — is the entry point truncated label, e.g., AS0 for the Asynchronous Communications Handler. This argument must be three ASCII characters. The first two represent the device mnemonic and the third is zero (0).

1.12.26 IB.INIT

Functional Description

This macro is used for MIOP initialization via entry point 8, where register 7 contains the CDT address and register 2 contains the address of the current context block.

Macro Call

Calling Sequence:

```
        IB.INIT
```

2. SYSTEM TABLES AND VARIABLES

2.1 Communications Region

The Communications Region is an area of main memory reserved for use by MPX-32 for storage of common data. This data is referenced via symbols which are equated to absolute memory locations. Together with each symbol given below 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 equated to the address of the first word. Bit variables are equated to bit positions relative to C.BIT. Bit variables are referenced by a combination of the variable symbol and C.BIT, e.g.,

```
        TBM     C.AFLK,C.BIT
```

C.ACTA (Word) Address of activation table.

C.ACTN (Byte) Number of entries in activation table.

C.ACTSEQ (Word) 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.

C.ADAT (Word) Contains the address of the DQE address table (DAT).

C.ADMASK (Word) Maximum address bit mask for machine.

C.AFLK (Bit) Accounting File Locked Indicator.

C.AFLW (Bit) Request for J.JOBC to list contents of accounting file.

C.ANYW (3 Words) 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.

C.BIT (2 Words) Symbol associated with the beginning of the bit variables.

C.BPRI (Byte) The default software priority level at which batch jobs execute.

C.BTCH (Bit) When set, task currently in execution is a batch task.

C.BUP (Byte) Base execution priority of currently executing task.

C.CAL (Dwd) Calendar devices as follows: century (binary), year (binary), month (binary), day (binary), and number of interrupts from midnight.

C.CDTA (Word) Address of Controller Definition Table.

C.CDTN (Hwd) Number of entries in Controller Definition Table.

C.CENT (Byte) Current century in binary.

C.CHTA (Word) Address of Channel Definition Table (CHT).

C.CHTN (Hwd) Number of entries in the Channel Definition Table (CHT).

C.CIPU (3 Words) Standard format linked list head cell for all IPU tasks ineligible for CPU control, waiting in general queue.

C.CONF (Byte) Configuration flags. Bit 0 is set if CPU accelerator is present; bit 1 is set if IPU accelerator is present; bit 2 is set if IPU is present; bit 3 is set if a memory-only system.

C.CPRI (Word) Byte 0 contains C.CUP; byte 1 contains C.BUP; byte 2 contains C.IOP; byte 3 contains C.US.

C.CPUACC (Bit) If set, CPU accelerator is present.

C.CSWI (Bit) Task to task context switch inhibited. This bit is set by the M.SHUT procedure and cleared by the M.OPEN procedure.

C.CUP (Byte) Current execution priority of currently executing task.

C.CURR (3 Words) Standard format linked list head cell for the CPU Dispatch Queue entry of the currently executing task. This list may 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).

C.DALMAP (Word) Contains the address of the Disc Allocation Map Buffer (initialized by FISE).

C.DAMAPT (Word) Contains the address of the Disc Allocation Map Table (initialized by SYSGEN).

C.DAMCST (Word) Address of Disc Allocation Map Checksum Table (initialized by SYSGEN).

C.DATE (Dwd) The current date (Gregorian) as input by the operator in the format: MM/DD/YY.

C.DAY (Byte) Current day in binary.

C.DBTLC (Byte) TLC used for System Debug.

C.DEBUG (Word) Address location of Debugger.

C.DQUE (Word) Address of CPU Dispatch Queue Area. The CPU Dispatch Queue Area is a variable length table built by SYSGEN. It contains the number of 42-word Disoatch Queue Entries (DQE's) specified at system generation time

C.DTTA (Word) Address of Device Type Table.

C.DTTN (Byte) Number of entries in Device Type Table.

C.EMAC (Hwd) Total count of valid E type memory modules available.

C.EMCC (Hwd) Total count of valid E type memory modules configured (minus 1 if the swap device is E-class and extended memory is present in the system).

C.EMTA (Word) Address of eventmark table.

C.EMTL (Byte) Low address of eventmark area.

C.EMTM (Hwd) Maximum number of eventmarks.

C.ETLOC (Word) Address of event trace logic.

C.FADR (Word) Reserved.

C.FGONR (Word) DQE address of task gating FISE.

C.FGPM (Bit) Post-mortem dump requested when a real-time task aborts.

C.FIRST Symbol equated to the absolute memory location at which the Communications Region begins.

C.FLTA (Word) Address of file lock table.

C.FLTC (Hwd) Number of File Lock Table (FLT) entries currently in use.

C.FLTM (Hwd) Maximum number of file locks.

C.FREE (3 Words) Standard format linked list head cell for free entries in the CPU Dispatch Queue. C.FREE is the first of a set of Communications Region variables which are contiguous in memory. These variables, listed in the order in which 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

C.FSFLGS (Byte) FISE flags. Bit 0 indicates FISE busy; bit 1 indicates FISE gated externally; bit 2 is set if temporary allocation or reset if permanent allocation.

C.GINT (Word) 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.

C.HIMAP (Hwd) Number of the last map block of logical address space available to a task.

C.HMAC (Hwd) Total count of valid H type memory modules available.

C.HMCC (Hwd) Total count of valid H type memory modules configured.

C.HOLD (3 Words) Standard format linked list head cell for all tasks that are ineligible for CPU control, waiting for a continue request to be received.

C.ICS (Bit) Reserved for ICS.

C.IDLA (Word) CPU idle time accumulation value in seconds, cleared by SYSGEN. This value is incremented when the countdown value in C.IDLC expires.

C.IDLA1 (Word) IPU idle time accumulation value in seconds, cleared by SYSGEN. This value is incremented when the countdown value in C.IDLC1 expires.

C.IDLC (Word) CPU idle time countdown value, cleared by SYSGEN. This value is used to load the interval timer when no tasks are ready to run. When a task becomes ready to run, the interval timer is read and the value is stored in this word.

C.IDLC1 (Word)

IPU idle time countdown value cleared by SYSGEN. This value is used to load IPU accounting interval timer (if present) when no tasks are 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.

C.INTC (Word)

Interrupt counter (number of interrupts from zero which is midnight) used for time of day calculations.

C.IOP (Byte)

I/O priority of currently executing task.

C.IPU (Bit)

If set, IPU is present.

C.IPUACC (Bit)

If set, IPU accelerator is present.

C.IPUIT (Bit)

If set, IPU accounting interval timer is present.

C.IPUIT1 (Word)

Address of the IPU accounting routine, S.IPUIT1, which performs accounting functions after an IPU trap is fielded. Initialized by SYSGEN.

C.IPUIT2 (Word)

Address of the IPU accounting routine, S.IPUIT2, which performs accounting functions prior to the starting of the IPU. Initialized by SYSGEN.

C.IPUOFF (Bit)

If set, IPU is off-line.

C.ITLT (Word)

Contains the address of the Indirectly Connected Task Linkage Table (ITLT). Initialized by SYSGEN.

C.ITRS (Hwd)

Interval timer resolution, in tenths of microseconds, as derived from the SYSGEN ITIM directive.

C.JOBA (Word)

Contains memory address of the Job Table.

C.JOBL (Bit)

Set by a System Input task after a file has been linked into the System Input Directory (M.SID).

C.JOBN (Word)

Number of entries in the Job Table.

C.LODC (Dwd)

The system listed output device to be used as a default in Operator Communications commands. Bytes 0 and 1 contain the ASCII device type code. Bytes 2 and 3 contain the ASCII channel number. Bytes 4 and 5 contain the ASCII subaddress.

C.LSPT (Bit)

List patches indicator.

C.MACH (Byte)

Machine currently in use as follows:
0 = 32/55 (not applicable)
1 = 32/75
2 = 32/27
3 = Reserved
4 = 32/87
5-15 = Reserved

C.MATA (Word)

Address of the memory tables.

C.MEMNLY (Bit)

If set, memory-only system.

C.MERR1 (Bit)

A memory error has been detected by H.IP02.

C.MERR2 (Bit)

A memory error has been detected by J.SWAPR.

C.MERR3 (Bit)

Nonpresent memory has been detected by J.SWAPR.

C.MGRAN (Word)

Machine dependent map granularity.

C.MIDL (Word)

Address of the list of map registers used by the operating system.

C.MIOP (Word)

Address of first entry of MIOP jump table.

C.MODD (Word)

Address of variable length Module Address Table. Initialized by SYSGEN. The Module Address Table contains entries in module number sequence. Each entry consists of one word which contains the address of the entry point transfer list (HAT) of the associated module.

C.MODN (Byte)

Number of entries in the Module Address Table. Initialized by SYSGEN.

C.MONTH (Byte)

Current month in binary.

C.MPAA (Word) Low address of the patch area.

C.MPAC (Word) Current address of the patch area.

C.MPAH (Word) High address of the patch area.

C.MPL (Word) Address of Master Process List. Length of list is (in words) C.NDQE+1. First entry points to C.MSD (hardware requirement).

C.MRQ (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting for memory to become available.

C.MSD (Word) Contains map segment descriptor for OS (BPIX). It points to C.MIDL (hardware requirement).

C.MTIM (Word) Number of clock interrupts per second. Initialized by SYSGEN.

C.MVTA (Word) Address of mounted volume table.

C.MVTN (Hwd) Number of entries in mounted volume table.

C.NITI (Byte) Contains the number of 24-word Indirectly Connected Task Linkage Block (ITLB) entries in the Indirectly Connected Task Linkage Table (ITLT). Initialized by SYSGEN.

C.NOLOAD (Bit) Module which cannot be loaded (SYSGEN flag).

C.NOS (Hwd) Number of blocks required for SYSGEN code.

C.NQUE (Byte) Number of entries (255 maximum) in CPU Dispatch Queue.

C.NRST (Hwd) Number of blocks required for restart code.

C.NTBA (Word) Total space currently available on all disc units.

C.NTIM (Word) Number of clock interrupts per time unit. Initialized by SYSGEN.

C.PODC (Dwd) The system punched output device to be used as a default in Operator Communications commands. Bytes 0 and 1 contain the ASCII device type code. Bytes 2 and 3 contain the ASCII channel number. Bytes 4 and 5 contain the ASCII subaddress.

C.POOL (Word) Address of Memory Pool.

C.PREA (3 Words) Standard format linked list head cell for CPU Dispatch Queue entries that are in the pre-activation state.

C.PRIV (Bit) When set, task currently in execution is a privileged task.

C.PRNO (Byte) Contains the DQE entry number of the currently executing task. It is in the range 1-255, and (when word format adjusted) may be used as an index to the DQE address table (DAT), to obtain the DQE for the associated task. Note: the address of the DQE address table (DAT) is contained in C.ADAT.

C.PSWRD (Dwd) Indicates a valid password to use for system files.

C.REGS (Word) TSA address of the current task.

C.REV (Word) MPX-32 release and interim release.

C.RIPU (3 Words) Standard format linked list head cell for all IPU tasks ready to run, waiting in general queue.

C.RMTA (Word) Address of resourcemark table.

C.RMTL (Byte) Low address of user resourcemark area.

C.RMTM (Hwd) Maximum number of resourcemarks.

C.RRUN (Byte) 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.

C.RUNW (3 Words) 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.

C.SBUF (Dwd) First word contains address of memory pool. Second word contains the number of words in memory pool.

C.SCBT (Bit) Continuous batch mode indicator.

C.SCDIPU (Word) Schedule IPU routine address.

C.SGOS (Word) The default size of the SGO file to be allocated for each batch job.

C.SIBP (Bit) When set, indicates to J.SOUT the banner page should be inhibited.

C.SICTD (Word) Address of MIOP test device status processor, H.SICTD.

C.SIDD (Bit) Set when batch system input device is 7-track magnetic tape and its density is 556.

C.SIDP (Bit) Set when batch system input device is 7-track magnetic tape and its parity is odd.

C.SIDV (Dwd) The system input device to be used as a default in Operator Communications commands. Bytes 0 and 1 contain the ASCII device type code. Bytes 2 and 3 contain the ASCII channel number. Bytes 4 and 5 contain the ASCII subaddress.

C.SIMM (Bit) When set, inhibits magnetic tape mount message.

C.SLEN (Byte) Number of blocks valid in System Input File (SIF) data.

C.SMAC (Hwd) Total count of valid S type memory modules available.

C.SMCC (Hwd) Total count of valid S type memory modules configured.

C.SMDD (Dwd) First word contains starting disc address of the SMD. Second word contains SMD length in 192-word blocks.

C.SMDS (Word) Number of entries in the SMD.

C.SMDUDT (Hwd) Contains UDT index of device on which the SMD resides.

C.SMTA (Word) Address of the Shared Memory Table area. Size is determined by SYSGEN SHARE directive.

C.SMTN (Byte) Number of entries in Shared Memory Table.

C.SMTS (Byte) Number of bytes in Shared Memory Table entry.

C.SPAD (Word) Address of CPU scratchpad image.

C.SPADOK (Bit) Do not zero unused scratchpad locations mode.

C.SQ55 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 55 time distribution tasks.

C.SQ56 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 56 time distribution tasks.

C.SQ57 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 57 time distribution tasks.

C.SQ58 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 58 time distribution tasks.

C.SQ59 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 59 time distribution tasks.

C.SQ60 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 60 time distribution tasks.

C.SQ61 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 61 time distribution tasks.

C.SQ62 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 62 time distribution tasks.

C.SQ63 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 63 time distribution tasks.

C.SQ64 (3 Words) Standard format linked list head cell for the list of ready-to-run priority level 64 time distribution tasks.

C.SQRT (3 Words) Standard format linked list head cell for the list of ready-to-run real time (priority level 1-54) tasks.

C.SRTA (Word) Address of shared resource table.

C.SRTN (Hwd) Number of entries in shared resource table.

C.STRTR (Bit) SYSGEN is in the starter system mode of the accounting file.

C.SUDT (Hwd) Swap device UDT index set by SYSGEN.

C.SUFA (Bit) Unidirectional file allocation mode indicator.

C.SUSP (3 Words) 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.

C.SVTA (Word) Address of variable length SVC Table. Initialized by SYSGEN. The SVC table contains entries in SVC (type 1 or 2) number sequence. Each entry consists of one word which contains the address of the service associated with the SVC number. Type 2 entries (restricted to privileged users) will have bit 0 set.

C.SVTN (Hwd) Number of entries in the SVC Table. Initialized by the System Generation (SYSGEN) program.

C.SWAP (Word) Contains the DQE address of the memory scheduler (swapper) when it is waiting for a memory event.

C.SWDC (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting for disc space to become available.

C.SWDV (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting for a peripheral device to become available.

C.SWFI (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting for the File System to be ungated.

C.SWGQ (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting in general queue.

C.SWIO (3 Words) Standard format linked list head cell for all tasks waiting for the completion of wait-mode I/O requests.

C.SWLO (3 Words) Standard format linked list head cell for all tasks waiting for the completion of low speed output.

C.SWMP (3 Words) Standard format linked list head cell for all tasks ineligible for CPU control, waiting for memory pool to become available.

C.SWSM (3 Words) Standard format linked list head cell for all tasks waiting for the completion of a wait-mode send message request.

C.SWSR (3 Words) Standard format linked list head cell for all tasks waiting for the completion of a wait-mode send run request.

C.SWTI (3 Words) Standard format linked list head cell for all tasks waiting for the completion of wait-mode interactive (terminal) input.

C.SYCS (Word) The initial size in 192-word blocks to be allocated for each SYC file as the job is being spooled to disc.

C.SYMTAB (Dwd) Name of the symbol table file.

C.SYSB (Bit) SYSBUILD is active in memory only.

C.SYSGEN (Word) SYSGEN scratch area.

C.SYSTEM (Dwd) Name of current system image.

C.TABLES Symbol equated to the absolute memory location at which SYSGEN built tables begin. This location is on a word boundary.

C.TDQ1 (Word)

Time distribution quantum stage 1, 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:

1) The task is initially selected after activation;
2) A task is initially selected after the termination of a voluntary wait state (e.g., wait I/O, or timed suspend);
3) A task is initially selected after inswap;
4) A task is re-selected after completion of its full quantum.

During the quantum stage 1 interval, the currently executing task is not eligible for outswap, and may not be preempted from CPU control by a higher priority time-distribution task. If preempted by a realtime task, the stage 1 time-distribution task remains ineligible for outswap until it has been re-selected and its stage 1 quantum expires.

C.TDQ2 (Word)

Time distribution quantum stage 2, in interval timer units. Initialized by SYSGEN. This value is used to load the interval timer when the stage 1 quantum for the currently executing task expires. (The quantum stage 2 value may be added to the quantum stage 1 value to define the full task quantum.)

C.TDQ3 (Word)

Time distribution full quantum value, in interval timer units. Initialized by SYSGEN. This value is the sum of the quantum stage 1 and stage 2 values.

C.TENT (Byte)

Number of timer table entries.

C.TERM (Bit)

When set, task currently in execution is a terminal task.

C.TMAC (DWD)

Counts contained in four halfwords of all memory types available. It is derived from configured size minus the O.S. size and any running task. It is constructed as follows:

1) Total amount of memory available
2) Amount of 'E' memory available (C.EMAC)
3) Amount of 'H' memory available (C.HMAC)
4) Amount of 'S' memory available (C.SMAC)

C.TMCC (DWD) Counts contained in four halfwords of all memory types configured and is constructed as follows:

1) Total amount of memory configured
2) Amount of 'E' memory configured (C.EMCC)
3) Amount of 'H' memory configured (C.HMCC)
4) Amount of 'S' memory configured (C.SMCC)

Note: Total memory configured (C.TMCC) and available (C.TMAC) and memory outside the first 128K, are decremented by the corresponding number of map blocks of any partitions, such as shared memory and Globals.

C.TRACE (Word) System trace (M.TRAC) control word.

C.TSAD (Word) Contains the address of the TSA for the currently executing task.

C.TSKN (Word) Task activation sequence number of currently executing task. Note: byte 0 of this word contains C.PRNO.

C.TSMCNT (Byte) Number of currently active TSM devices. Maintained by J.TSM.

C.TSMDQA (Word) Address of DQE for J.TSM. Required for ring processing and message sending.

C.TSMPRI (Byte) Priority default for TSM-ACTIVATED tasks. Overrides cataloged priority.

C.TSMTOT (Byte) Number of TSM devices. Initialized by entry point 8 of H.IBAS and H.TY10.

C.TTAB (Word) Address of the timer table.

C.TTBT (8 Words) Task Timer Bit Table. The Task Timer Bit Table is an 8-word table containing 256 bits. Each bit corresponds to a C.DQE entry and is accessed by the DQE entry number (1-255). A bit set in this table indicates that the associated DQE has an active task timer.

C.UDTA (Word) Address of Unit Definition Table (UDT).

C.UDTN (Hwd) Number of entries in Unit Definition Table (UDT).

C.US (Byte) State chain index of currently executing task.

C.YEAR (Byte) Current year in binary.

2.2 Task Service Area (TSA)

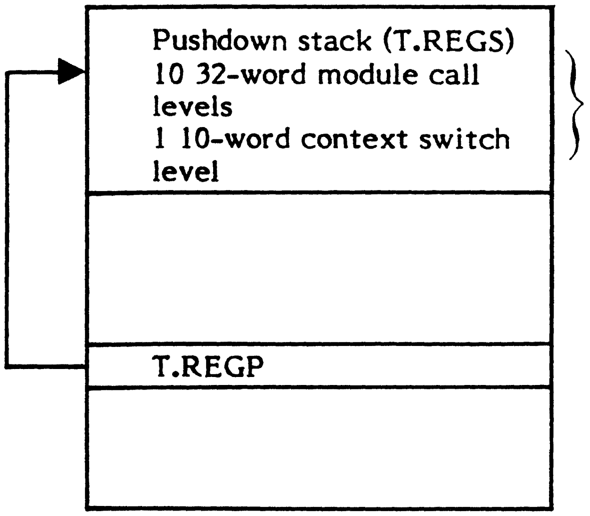
The Task Service Area (TSA) is a section of memory associated with each active task which is used by MPX for storage of task-unique information. A TSA is allocated for each task when the task becomes active and is deallocated when the task terminates. The size of each task's TSA is fixed for the duration of the task's execution. However, the sizes of TSA's among tasks is variable and is 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 buffer, FAT and FPT entry are reserved for MPX use and are present in every TSA.

The pushdown area in the TSA provides reentrancy 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 bing called. T.REGP is decremented to the previous pushdown level upon return to the entry point caller. Upon context switch away from a task, 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.

TSA Structure

TSA Fixed Area



Module Call Level

Word	
0	General Purpose Registers 0-7
7	
8	PSD
9	
10	Scratchpad Storage
31	

Context Switch Level

Word	
0	General Purpose Registers 0-7
8	PSD
9	

TSA Variable Area

T.FATA	1 to 255 16-word FAT entries. First is reserved for system use.
T.FPTA	1 to 255 3-word FPT entries. First is reserved for system use.
T.BBUFA	1 to 255 192-word blocking buffers. First is reserved for system use.

TSA Structure

Word # (Decimal)	Byte (Hex)	0 7	8 15	16 23	24 31
0-329	0	T.REGS			
330-331	528	T.USER			
332	530	T.PGOW			
333	534	T.PARENT			
334	538	T.REGP			
335	53C	T.ITAC			
336-351	540	T.BFCB			
352-359	580	T.PROT			
360-369	5A0	T.CONTXT			
370	5C8	T.DBHAT			
371	5CC	T.PRNO			
372	5D0	T.ABRTA			
373	5D4	T.BBUFA			
374	5D8	T.VATA			
375	5DC	T.VATN	T.WORK	T.UNUSED	
376	5E0	T.FATA			
377	5E4	T.FPTA			
378	5E8	T.BREAK			
379	5EC	T.MSGR			
380	5F0	T.LINBUF			
381	5F4	T.BIAS			
382	5F8	T.TEND			
383	5FC	T.END			
384	600	T.TRAD			
385	604	T.LINNO		T.UKEY	
386	608	T.BIT1	T.BIT2	T.BBUFN	T.FILES
387	60C	T.DSOR	T.DSSZ	T.CSOR	T.CSSZ
388	610	T.MPOR	T.MPSZ	T.EAOR	T.EASZ
389	614	T.IPUAC			
390-391	618	T.ACCESS			
392-395	620	T.SGOS			
396-399	630	T.SYCS			
400-401	640	T.CVOL			
402-403	648	T.CDIR			
404	650	T.CURH			
405	654	T.CRHX			
406-415	658	T.SPARES			
416-431	680	T.MIDL (words 416-543 on CONCEPT/32 computer)			
432-447	6C0	T.MEML (words 544-671 on CONCEPT/32 computer)			

Word	Name	Description
0	T.REGS	Task pushdown stack (330 words).
330	T.USER	Current username for file allocation (2 words).
332	T.PGOW	Task option word.
333	T.PARENT	Task sequence number of parent task (activator).
334	T.REGP –	Current level of pushdown in stack area.
335	T.ITAC	Interval timer based accounting.
336	T.BFCB	System service file control buffer (16 words).
352	T.PROT	Memory protection image (8 words); not used on CONCEPT/32.
360	T.CONTXT	Debug context area (10 words).
370	T.DBHAT	Address of Debug halfword address table.
371	T.PRNO	Address of task's dispatch queue.
372	T.ABRTA	Address of task's abort receiver.
373	T.BBUFA	Address of task's blocking buffer.
374	T.VATA	Address of task's volume assignment table.
375	T.VATN	Number of entries in volume assignment table (1 byte).
	T.WORK	Word space scratch area.
	T.UNUSED	Reserved (2 bytes).
376	T.FATA	Address of task's file assignment table.
377	T.FPTA	Address of task's file pointer table.
378	T.BREAK	Address of task's break receiver.
379	T.MSGR	Address of task's message receiver.
380	T.LINBUF	Address of TSM's line buffer.
381	T.BIAS	Starting address of task's DSECT area.
382	T.TEND	Ending address of TSA when task is loaded
383	T.END	Ending address of task's DSECT area.

Word	Name	Description		
384	T.TRAD	Transfer address of task's main segment.		
385	T.LINNO	Line counts (1 halfword). Byte 0 = maximum line count. Byte 1 = current line count.		
	T.UKEY	Compressed original user key (1 halfword).		
386	T.BIT1	Bit variables (1 byte) assigned as follows:		
		0	T.EI01	Set if first 4K of E-class I/O buffer in use
		1	T.EI02	Set if second 4K of E-class I/O buffer in use
		2	T.EXCP	Set on arithmetic exception trap
		3	T.EBUF	E-class buffer present (8KW map only)
		4	T.WAIT	Set to indicate suspend after activation
		5	T.DBG	Set to indicate Debugger requested
		6	T.SHR	Set to indicate CSECT to share
		7	T.COMFIL	Set to indicate command file is active in TSM
	T.BIT2	Bit variables (1 byte) assigned as follows:		
		0	T.EBUF1	Set if one E-class I/O has been allocated
		1	T.EBUF2	Set if a second E-class I/O buffer has been allocated. Used in 2KW map granularity systems.
		2-7	Reserved	
	T.BBUFN	Number of blocking buffers associated with task (1 byte).		
	T.FILES	Number of FAT/FPT pairs associated with task (1 byte).		
387	T.DSOR	DSECT origin within T.MEML/T.MIDL (1 byte). Usually zero.		
	T.DSSZ	DSECT size in map blocks (1 byte).		
	T.CSOR	CSECT origin within T.MEML/T.MIDL (1 byte). If size is zero, T.CSOR is set equal to T.MPOR contents.		

Word	Name	Description
	T.CSSZ	CSECT size in map blocks (1 byte).
388	T.MPOR	Memory partition origin within T.MEML/T.MIDL (1 byte). If size is zero, contains map blocks for OS.
	T.MPSZ	Memory partition size in map blocks (1 byte).
	T.EAOR	Extended address origin in T.MEML/T.MIDL (1 byte). If size is zero, contains map blocks for OS.
	T.EASZ	Extended address size in map blocks (1 byte).
389	T.IPUAC	IPU RT clock accounting.
390	T.ACCESS	Privileged flags (2 words). See MPX-32 Reference Manual, Volume 2, Chapter 7.
392	T.SGOS	SGO definition (4 words).
396	T.SYCS	SYC definition (4 words).
400	T.CVOL	Name of current disc volume (2 words).
402	T.CDIR	Name of current directory (2 words).
404	T.CURH	Current high address in map.
405	T.CRHX	Current high address in extended space.
406	T.SPARES	Reserved for TSA expansion (10 words).
416	T.MIDL	32/7x Halfword map image descriptor list (16 words).

0	MIDL.RES	Reserved
1	MIDL.VAL	Set if map block number is valid
2	MIDL.PRO	Set if map block protected
3-15		Physical map number

CONCEPT/32 Halfword map image descriptor list (128 words).

0	MIDL.VAL	Set if number is valid
1	MIDL.PR0	Protection granule
2	MIDL.PR2	Protection granule
3	MIDL.PR3	Protection granule
4	MIDL.PR4	Protection granule
5-15		Physical map number

Word	Name	Description		
432	T.MEML	32/7x Halfword memory attribute list (16 words).		
		0	MEML.TYE	Set if E-type memory
		1	MEML.TYH	Set if H-type memory
		2	MEML.TYS	Set if S-type memory
		3	MEML.SHR	Set if map block is shared
		4	MEML.SWP	Set if map block is swappable
		5	MEML.VAL	Set if map is valid
		6	MEML.RS1	Reserved
		7	MEML.RS2	Reserved
		8-15		Shared index if map block shared, otherwise zero
544	T.MEML	CONCEPT/32 Halfword memory attribute list (128 words).		
		0	MEML.TYE	Set if E-type memory
		1	MEML.TYH	Set if H-type memory
		2	MEML.TYS	Set if S-type memory
		3	MEML.SHR	Set if map block is shared
		4	MEML.SWP	Set if map block is swappable
		5	MEML.VAL	Set if map is valid
		6	MEML.RS1	Reserved
		7	MEML.RS2	Reserved
		8-15		Shared index if map block shared, otherwise zero

Map Image Descriptor List (T.MIDL)

The MIDL describes the maps that are allocated to a particular task. The MIDL and MEML correspond one to one. The MIDL is located in the TSA at location T.MIDL.

If C.MACH = 1 indicating 32/7x machine type, there are 16 words in the MIDL, each word containing two entries, giving a maximum of 32 entries or map blocks per task.

If C.MACH = 2 indicating 32/27 machine type or 4 indicating 32/87 machine type, there are 128 words in the MIDL, giving a maximum of 256 entries or map blocks per task.

32/7x Machine

0 2	3 15	0 2	3 15
Flags	Physical Map Number	Flags	Physical Map Number

1. Flag bits are assigned as follows.

MIDL.RES	0	Reserved
MIDL.VAL	1	Map number is valid if set
MIDL.PRO	2	Reserved

2. If MIDL.VAL is set, the physical map number contains a valid map block number that represents an entry in the Memory Allocation Table (MATA).

32/27 or 32/87 Machine

0 4	5 15	0 4	5 15
Flags	Physical Map Number	Flags	Physical Map Number

1. Flag bits are assigned as follows.

MIDL.VAL	0	Map number is valid if set
MIDL.PRO	1	1st protection granule
MIDL.PR2	2	2nd protection granule
MIDL.PR3	3	3rd protection granule
MIDL.PR4	4	4th protection granule

2. If MIDL.VAL is set, the physical map number contains a valid map block number that represents an entry in the Memory Allocation Table (MATA).

Memory Attribute List (T.MEML)

The MEML describes the maps that are allocated to a particular task. The MEML and MIDL correspond one to one. The MEML is found in the TSA at location T.MEML.

If C.MACH = 1 indicating 32/7x machine type, there are 16 words in the MEML, each word containing two entries, giving a maximum of 32 entries or map blocks per task.

If C.MACH = 2 indicating 32/27 machine type or 4 indicating 32/87 machine type, there are 128 words in the MEML giving maximum of 256 entries or map blocks per task.

The MEML structure is the same for both machine types.

0 7	8 15	0 7	8 15
Flags	Shared Index	Flags	Shared Index

1.Flag bits are assigned as follows.

MEML.TYE	0	If set, E-type memory
MEML.TYH	1	If set, H-type memory
MEML.TYS	2	If set, S-type memory
MEML.SHR	3	If set, map is shared
MEML.SWP	4	If set, map is swappable
MEML.VAL	5	If set, map is valid
MEML.RS1	6	Reserved
MEML.RS2	7	Reserved

2. If MEML.SHR is set, the shared index will contain the index into the associated shared memory table in which this map block has been allocated.

2.3 Executive (H.EXEC) Data Areas and Tables

2.3.1 CPU Dispatch Queue Area

The CPU Dispatch Queue Area is a variable length doubleword bounded table built by the system generation program. It contains a maximum of 255 Dispatch Queue Entries (DQE's). 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 maintaind in the Task Service Area (TSA). While a task is active, its DQE will be linked to one of the various ready-to-run or wait state chains provided by the scheduler to describe the tasks current status. When a task exits, its DQE is again linked to the free list.

2.3.2 CPU 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

Decimal Word Number	Hexa-Decimal Offset	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	10	DQE.ON			
5	14				
6	18	DQE.LMN			
7	1C				
8	20	DQE.PSN			
9	24				
10	28	DQE.USW			
11	2C	DQE.USHF			
12	30	DQE.MSD			
13	34	DQE.MRT	DQE.MEM	DQE.RMMR	DQE.MEMR
14	38	DQE.MMSG	DQE.MRUN	DQE.MNWI	DQE.GQFN
15	3C	DQE.UF2	Reserved		DQE.SOPO
16	40	DQE.IPUF	Reserved		
17	44	DQE.UXRF		DQE.TIFC	DQE.RILT
18	48	DQE.UTS1			
19	4C	DQE.UTS2			
20	50	DQE.NWIO/DQE.TDA			
21	54	DQE.PRS			
22	58	DQE.PRM			
23	5C	DQE.CME	DQE.CMH	DQE.CMS	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			
35	8C	DQE.MRPR	DQE.MRCT	DQE.NWRR	DQE.NWMR
36	90	DQE.RTI		DQE.ATI	
37	94	DQE.SAIR/DQE.TAD			
38	98	DQE.ABC			
39	9C				
40	A0				
41	A4	DQE.CQC			
42	A8	Reserved (7 words)			

WORD	SYMBOL	ITEM DESCRIPTION
*****	*******	******************

00	DQE.SF	STRING FORWARD LINKAGE ADDRESS;
		STANDARD LINKED LIST FORMAT;
		CONTAINS ADDRESS OF NEXT (TOP-TO-BOTTOM) ENTRY IN CHAIN.

01	DQE.SB	STRING BACKWARD LINKAGE ADDRESS;
		STANDARD LINKED LIST FORMAT;
		CONTAINS ADDRESS OF NEXT (BOTTOM-TO-TOP) ENTRY IN CHAIN.

02	DQE.CUP	CURRENT USER PRIORITY;
		FIELD LENGTH = 1B;
		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.

	DQE.US	STATE CHAIN INDEX FOR THIS USER TASK;
		FIELD LENGTH = 1B;
		RANGE: ZERO THRU "N";
		INDICATES CURRENT STATE OF THIS TASK E.G. READY-TO-RUN PRIORITY, I/O WAIT, RESOURCE BLOCK, ETC..

LABEL	INDEX	DESCRIPTION
FREE	00	DQE IS AVAILABLE (IN FREE LIST)
PREA	01	TASK ACTIVATION IN PROGRESS
CURR	02	TASK IS CURRENTLY EXECUTING TASK OR IS PREEMPTED TIME DISTRIBUTION TASK IN QUANTUM STAGE 1.
SQRT	03	TASK IS READY TO RUN (PRI. LEV. 1-54)
SQ55	04	TASK IS READY TO RUN (PRI. LEV. 55)
SQ56	05	TASK IS READY TO RUN (PRI. LEV. 56)
SW57	06	TASK IS READY TO RUN (PRI. LEV. 57)

SW58	07	TASK IS READY TO RUN (PRI. LEV. 58)
SQ59	08	TASK IS READY TO RUN (PRI. LEV. 59)
SQ60	09	TASK IS READY TO RUN (PRI. LEV. 60)
SQ61	0A	TASK IS READY TO RUN (PRI. LEV. 61)
SQ62	0B	TASK IS READY TO RUN (PRI. LEV. 62)
SQ63	0C	TASK IS READY TO RUN (PRI. LEV. 63)
SQ64	OD	TASK IS READY TO RUN (PRI. LEV. 64)
SWTI	0E	TASK IS WAITING FOR TERMINAL INPUT
SWIO	0F	TASK IS WAITING FOR I/O
SWSM	10	TASK IS WAITING FOR MESSAGE COMPLETE
SWSR	11	TASK IS WAITING FOR RUN REQ COMPLETE
SWLO	12	TASK IS WAITING FOR LOW SPEED OUTPUT
SUSP	13	TASK IS WAITING FOR: 1) TIMER EXPIRATION, OR 2) RESUME REQUEST, OR 3) MESSAGE INTERRUPT.
RUNW	14	TASK IS WAITING FOR: 1) TIMER EXPIRATION, OR 2) RUN REQUEST.
HOLD	15	TASK IS WAITING FOR A CONTINUE REQ.
ANYW	16	TASK IS WAITING FOR: 1) TIMER EXPIRATION, OR 2) NO-WAIT I/O COMPLETE, OR 3) NO-WAIT MSG COMPLETE, OR 4) NO-WAIT RUN REQ COMPLETE, OR 5) MESSAGE INTERRUPT, OR 6) BREAK INTERRUPT.
SWDC	17	TASK IS WAITING FOR DISC SPACE
SWDV	18	TASK IS WAITING FOR DEV ALLOCATION
SWFI	19	TASK IS WAITING FOR FILE SYSTEM
MRQ	1A	TASK IS WAITING FOR MEMORY
SWMP	1B	TASK IS WAITING FOR MEMORY POOL
SWGQ	1C	TASK IS WAITING IN GENERAL WAIT QUEUE
CIPU	ID	CURRENT IPU TASK IN EXECUTION
RIPU	IE	IPU REQUESTING STATE

03 DQE.NUM DQE ENTRY NUMBER;
FIELD LENGTH = 1B;
USED AS AN INDEX TO DQE ADDRESS TABLE (DAT);
RANGE: ONE THRU "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.
NOTE: THE SPECIFIED ENTRY IN THE MPL (ONE PER TASK) CONTAINS THE COUNT AND ADDRESS OF THE MSD IN THE DQE, WHICH IN TURN POINTS TO THE MIDL IN THE TSA.

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 MAY BE ACCESSED AS DQE.NUM.

04 DQE.ON OWNER NAME;
FIELD LENGTH = 1D

06 DQE.LMN LOAD MODULE NAME;
FIELD LENGTH = 1D

08 DQE.PSN PSEUDONYM ASSOCIATED WITH TASK;
FIELD LENGTH = 1D;
THIS PARAMETER IS AN OPTIONAL ARGUMENT ACCEPTED BY THE PSEUDOTASK ACTIVATION SERVICE.IT MAY BE USED TO UNIQUELY IDENTIFY A TASK WITHIN A SUBSYSTEM (E.G. 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.

10 DQE.USW USER STATUS WORD;
FIELD LENGTH = 1W

11 DQE.USHF SCHEDULING FLAGS;
FIELD LENGTH = 1W;
USED BY THE SCHEDULER TO INDICATE SPECIAL STATUS CONDITIONS.

BIT	LABEL	DESCRIPTION
00	AVAILABLE	
01	DQE.SING	SINGLE COPY LOAD MODULE
02	DQE.INDC	TASK IS INDIRECTLY CONNECTED.
03	DQE.PRIV	TASK IS PRIVILEGED.
04	DQE.MSGR	TASK HAS MESSAGE RECEIVER.
05	DQE.BRKR	TASK HAS BREAK RECEIVER.
06	DQE.QS1X	TASK QUANTUM STAGE 1 EXPIRED.
07	DQE.QS2X	TASK QUANTUM STAGE 2 EXPIRED.
08	DQE.INER	INSWAP I/O ERROR.
09	DQE.WIOA	WAIT I/O REQUEST OUTSTANDING.
10	DQE.WIOC	WAIT I/O COMPLETE BEFORE IN-PROGRESS NOTIFICATION.
11	DQE.INMI	INHIBIT MESSAGE PSEDUO INTERRUPT
12	DQE.BAOR	BATCH ORIGIN TASK.
13	DQE.TMOR	TERMINAL ORIGIN TASK.
14	DQE.ABRT	TASK ABORT IN PROGRESS.
15	DQE.PRXT	TASK IS IN PRE-EXIT STATE.
16	DQE.RRMD	RUN RECEIVER MODE.
17	DQE.WMSA	WAIT-SEND MSG OUTSTANDING.
18	DQE.WMSC	WAIT MSG COMPLETE BEFORE LINK TO WAIT QUEUE.
19	DQE.WRRA	WAIT MODE SEND RUN REQUEST OUTSTANDING.
20	DQE.WRRC	WAIT MODE SEND RUN REQUEST COMPLETE BEFORE LINK TO WAIT QUEUE.
21	DQE.DBAT	DEBUG ASSOCIATED WITH TASK.
22	DQE.RT	REAL TIME TASK.
23	DQE.TDID	TIME DISTRIBUTION TASK INITIAL DISPATCH. SET BY: 1) H.ALOC1 ON ACTIVATION OF T/D TASK. 2) S.EXEC51 WHEN TASK IS LINKED TO WAIT STATE. 3) H.EXEC7 ON COMPLETION OF INSWAP OR OTHER MEMORY REQUEST. CLEARED BY S.EXEC20 ON INITIAL DISPATCH OF TASK AFTER ACTIVATION, WAIT-STATE TERMINATION, OR INSWAP.
24	DQE.DELP	TASK DELETE IN PROGRESS.
25	DQE.ABRA	TASK ABORT (WITH ABORT RECEIVER) IN PROGRESS.
26	DQE.ABRC	ABORT RECEIVER ESTABLISHED.
27	DQE.ADIN	ASYNCHRONOUS ABORT/DELETE INHIBITED.
28	DQE.ADDF	ASYNCHRONOUS DELETE DEFERRED.
29	DQE.INAC	TASK IS INACTIVE.

30 DQE.AADF ASYNCHRONOUS ABORT DEFERRED.
31 DQE.ACTT ACTIVATION TIMER IN EFFECT.

12 DQE.MSD MSD ENTRY POINTING TO MIDL IN TSA FOR 7X PROCESSORS; FIELD LENGTH = 1W

13 DQE.MRT MEMORY REQUEST TYPE CODE;
FIELD LENGTH = 1B;
00 = INSWAP ONLY.
01 = PREACTIVATION REQUEST.
02 = ACTIVATION REQUEST.
03 = MEMORY EXPANSION REQUEST.
04 = IOCS BUFFER REQUEST.
05 = SHARED MEMORY REQUEST. BYTES TWO AND THREE EQUAL THE ADDRESS OF THE REQUESTED USER SMT

DQE.MEM TYPE OF MEMORY REQUESTED;
FIELD LENGTH = 1B;
01 = CLASS 'E' MEMORY.
02 = CLASS 'H' MEMORY.
03 = CLASS 'S' MEMORY.

DQE.RMMR MAP REGISTER FOR REQUESTED MEMORY;
FIELD LENGTH = 1B

DQE.MEMR NUMBER OF MEMORY BLOCKS REQUIRED;
FIELD LENGTH = 1B

14 DQE.MMSG ONE LESS THAN MAXIMUM NUMBER OF NO WAIT SEND MESSAGE REQUESTS ALLOWED TO BE CONCURRENTLY OUTSTANDING FOR THIS TASK. DEFAULT IS 5 (6 NO-WAIT REQUESTS) FOR UNPRIVILEGED TASKS, 0 (255 NO-WAIT REQUESTS) FOR PRIVILEGED.

DQE.MRUN ONE LESS THAN MAXIMUM NUMBER OF NO-WAIT SEND RUN REQUESTS ALLOWED TO BE CONCURRENTLY OUTSTANDING FOR THIS TASK. DEFAULT IS 5 (6 NO-WAIT REQUESTS) FOR UNPRIVILEGED TASKS, 0 (255 NO-WAIT REQUESTS) FOR PRIVILEGED.

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.
01 = QUEUED FOR FILE EXCLUSIVE LOCK
02 = QUEUED FOR FLT SPACE
03 = QUEUED FOR FILE SYNCHRONIZATION LOCK
04 = QUEUED FOR RESOURCEMARK LOCK
05 = RESERVED FOR EVENTMARK
06 = QUEUED FOR MISCELLANEOUS ID
07 = QUEUED FOR SHARED MEMORY TABLE

15 DQE.UF2 SCHEDULING FLAGS;
FIELD LENGTH = 1B.

BIT	LABEL	DESCRIPTION
0	DQE.EDB	ENABLE DEBUG MODE BREAK
1	DQE.GQTO	GENERALIZED WAIT QUEUE TIME-OUT
2	DQE.SYNC	
3-7	RESERVED	

RESERVED FIELD LENGTH = 2B

DQE.SOPO PRIORITY BASIS ONLY SWAPPING CONTROL FLAGS;
FIELD LENGTH = 1B.

BIT	LABEL	DESCRIPTION
0	DQE.GQPO	SWGQ STATE PRIORITY BASED SWAPPING
1-7	RESERVED	

16 DQE.IPUF IPU FLAG BYTE
FIELD LENGTH = 1B

BIT	LABEL	DESCRIPTION
0	DQE.IPUH	IPU INHIBIT FLAG
1	DQE.IPUB	IPU BIAS FLAG
2	DQE.IPUR	CPU ONLY
3	DQE.OSD	OS EXECUTION DIRECTIVE FLAG
4-7	RESERVED	

RESERVED FIELD LENGTH = 3B

17 DQE.UXRF UDT INDEX OF ROLLOUT (SWAP) FILE;
FIELD LENGTH = 1H;
SET WHEN THE ROLLOUT FILE IS ALLOCATED;
USED FOR IDENTIFICATION OF THE SWAP DEVICE.

DQE.TIFC TIMER FUNCTION CODE;
FIELD LENGTH = 1B;
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.

DQE.RILT REQUEST INTERRUPT (RI) LEVEL FOR TIMER;
FIELD LENGTH = 1B;
IDENTIFIES THE INTERRUPT LEVEL TO BE REQUESTED UPON TIMER EXPIRATION.

18 DQE.UTS1 USER TIMER SLOT WORD 1;
FIELD LENGTH = 1W;
CURRENT TIMER VALUE;
CONTAINS NEGATIVE NUMBER OF TIMER UNITS BEFORE TIME-OUT.

19 DQE.UTS2 USER TIMER SLOT WORD 2;
FIELD LENGTH = 1W;
RESET TIMER VALUE;
CONTAINS NEGATIVE NUMBER OF TIME UNITS;
USED TO RESET THE CURRENT TIMER VALUE WHEN IT EXPIRES.

20 DQE.NWIO NUMBER OF NO-WAIT I/O REQUESTS OUTSTANDING;
FIELD LENGTH = 1B.

DQE.TDA DISC ADDRESS OF SWAPPED TSA;
FIELD LENGTH = 1W (BYTE 0 = DQE.NWIO) CONTAINS DISC RELATIVE SECTOR NUMBER;
USED IN CONJUNCTION WITH DQE.UXRF TO IDENTIFY THE ROLLOUT FILE.

21 DQE.PRS PERIPHERAL REQUIREMENT SPECIFICATION;
FIELD LENGTH = 1W;
BITS 0-7 = RESERVED.
BITS 8-15 = DEVICE TYPE CODE.
BITS 16-23 = CHANNEL ADDRESS.
BITS 24-31 = SUBCHANNEL ADDRESS.
OR, CONTAINS FIRST WORD OF SWGQ ID.

22 DQE.PRM PERIPHERAL REQUIREMENTS MASK;
FIELD LENGTH = 1W;
X'00FF0000' = ANY DEVICE OF THIS TYPE CODE.
X'00FFFF00' = ANY DEVICE OF THE SPECIFIED TYPE CODE, ON THE SPECIFIED CHANNEL.
X'00FFFFFF' = THE SPECIFIC DEVICE AS DESCRIBED BY TYPE CODE, CHANNEL, AND SUBCHANNEL ADDRESS.
OR, CONTAINS SECOND WORD OF SWGQ ID.

23 DQE.CME NUMBER OF SWAPPABLE CLASS 'E' MAP BLOCKS CURRENTLY ALLOCATED;
FIELD LENGTH = 1B.

DQE.CMH NUMBER OF SWAPPABLE CLASS 'H' MAP BLOCKS CURRENTLY ALLOCATED;
FIELD LENGTH = 1B.

DQE.CMS NUMBER OF SWAPPABLE CLASS 'S' MAP BLOCKS CURRENTLY ALLOCATED;
FIELD LENGTH = 1B.

DQE.MST STATIC MEMORY TYPE SPECIFICATION;
FIELD LENGTH = 1B;
01 = CLASS 'E' MEMORY.
02 = CLASS 'H' MEMORY.
03 = CLASS 'S' MEMORY.
THIS FIELD IS USED TO SPECIFY THE TYPE OF MEMORY REQUIRED FOR INSWAP. THE AMOUNT OF MEMORY REQUIRED FOR INSWAP IS COMPUTED BY TAKING THE SUM OF DQE.CME, DQE.CMH, DQE.CMS.

24 DQE.PSSF PREEMPTIVE SYSTEM SERVICE HEAD CELL
STRING FORWARD ADDRESS;
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1W;
CONTAINS ADDRESS OF NEXT (TOP-TO-BOTTOM) ENTRY IN CHAIN.

25 DQE.PSSB PREEMPTIVE 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.

26 DQE.PSPR PREEMPTIVE SYSTEM SERVICE HEAD CELL
DUMMY PRIORITY (ALWAYS = 0);
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1B.

DQE.PSCT PREEMPTIVE 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

27 DQE.TISF TASK INTERRUPT HEAD CELL
STRING FORWARD ADDRESS;
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1W;
CONTAINS ADDRESS OF NEXT (TOP-TO-BOTTOM) ENTRY IN CHAIN.

28 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.

29 DQE.TIPR TASK INTERRUPT HEAD CELL
DUMMY PRIORITY (ALWAYS = 0);
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	LABEL	DESCRIPTION
0	DQE.RESP	TASK IS RESIDENT.
1	DQE.LKIM	TASK IS LOCKED IN MEMORY.
2	DQE.IO	TASK HAS UNBUFFERRED I/O IN PROGRESS.
3	DQE.OTSW	TASK IS OUTSWAPPED.
4	DQE.TLVS	TASK IS LEAVING SYSTEM
5	DQE.FCUS	TASK FORCED TO UNSWAPPABLE STATE DURING TERMINAL OUTPUT.
6	DQE.FCRS	TASK FORCED UNSWAPPABLE BECAUSE SWAP FILE HAS NOT BEEN ALLOCATED FOR IT.
7	RESERVED	

3 DQE.UBIO NUMBER OF UNBUFFERRED I/O REQUESTS CURRENTLY OUTSTANDING;
FIELD LENGTH = 1B.

30 DQE.RRSF RUN RECEIVER HEAD CELL
STRING FORWARD ADDRESS;
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1W;
CONTAINS ADDRESS OF NEXT (TOP-TO-BOTTOM) ENTRY IN CHAIN.

31 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.

32 DQE.RRPR RUN RECEIVER HEAD CELL
DUMMY PRIORITY (ALWAYS = 0);
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1B.

DQE.RRCT RUN RECEIVER HEAD CELL
NUMBER OF ENTRIES IN LIST;
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1B;

DQE.NSCT NUMBER OF SECTORS IN SWAP FILE;
FIELD LENGTH = 1H.

33 DQE.MRSF MESSAGE RECEIVER HEAD CELL
STRING FORWARD ADDRESS;
STANDARD HEAD CELL FORMAT;
FIELD LENGTH = 1W;
CONTAINS ADDRESS OF NEXT (TOP-TO-BOTTOM) ENTRY IN CHAIN.

34 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.

35 DQE.MRPR MESSAGE RECEIVER HEAD CELL
DUMMY PRIORITY (ALWAYS = 0);
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 MSG REQUESTS OUTSTANDING;
FIELD LENGTH = 1B.

36 DQE.RTI REQUESTED TASK INTERRUPT FLAGS;
FIELD LENGTH = 1H.

BIT	LABEL	DESCRIPTION
0	RESERVED	
1	DQE.EA1R	PRIORITY 1 END ACTION REQUEST. USED FOR PREEMPTIVE SYSTEM SERVICES.
2	DQE.DBBR	DEBUG BREAK REQUEST.
3	DQE.UBKR	USER BREAK REQUEST.
4	DQE.EA2R	END ACTION REQUEST. (PRIORITY 2).
5	DQE.MSIR	MESSAGE INTERRUPT REQUEST.
6-15	RESERVED	

DQE.ATI ACTIVE TASK INTERRUPT FLAGS;
FIELD LENGTH = 1H.

BIT	LABEL	DESCRIPTION
0	RESERVED	
1	DQE.AEA1	ACTIVE END ACTION PRIORITY 1.
2	DQE.ADM	ACTIVE DEBUG BREAK.
3	DQE.AUB	ACTIVE USER BREAK.
4	DQE.AEA	ACTIVE END ACTION PRIORITY 2.
5	DQE.AMI	ACTIVE MESSAGE INTERRUPT.
6-15	RESERVED	

37 DQE.SAIR SYSTEM ACTION TASK INTERRUPT REQUEST;
FIELD LENGTH = 1B.

BIT	LABEL	DESCRIPTION
0	DQE.DELR	REQUEST FOR DELETE OF THIS TASK
1	RESERVED	
2	DQE.HLDR	HOLD TASK REQUEST.
3	DQE.ABTR	ABORT TASK REQUEST.
4	DQE.EXTR	EXIT TASK REQUEST.
5	DQE.SUSR	SUSPEND TASK REQUEST.
6	DQE.RRRQ	RUN RECEIVER MODE REQUEST.
7	RESERVED	

DQE.TAD TSA ADDRESS (LOGICAL);
FIELD LENGTH = 1W (BYTE 0 CONTAINS DQE.SAIR)

38 DQE.ABC ABORT CODE;
FIELD LENGTH = 12B (3W).

41 DQE.CQC CURRENT QUANTUM COUNT;
FIELD LENGTH = 1W;
USED BY THE SCHEDULER TO ACCUMULATE ELAPSED EXECUTION TIME FOR THE TASK, FOR COMPARISON WITH THE LEVEL UNIQUE STAGE1 AND STAGE2 TIME DISTRIBUTION VALUES.

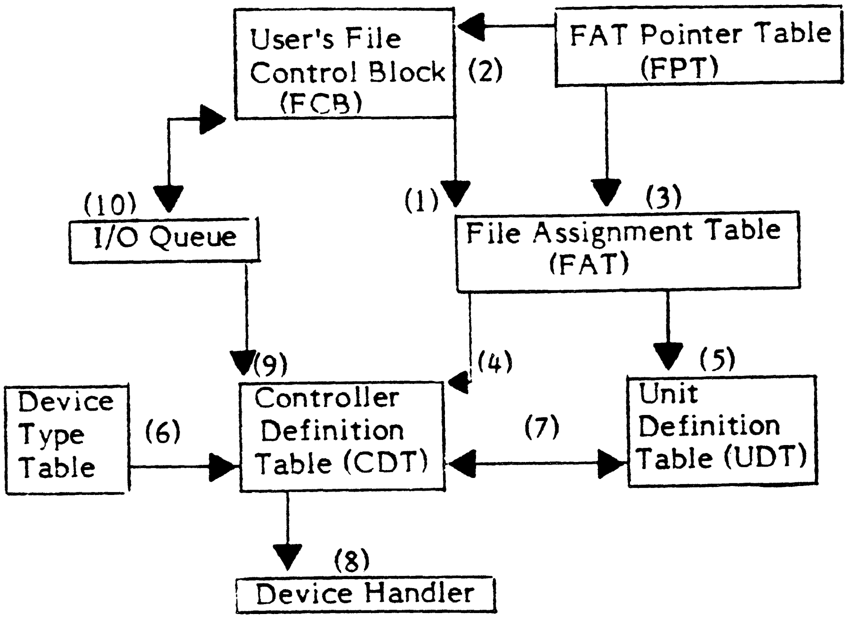
42-48 RESERVED
END

2.3.3 DQE Address Table (DAT)

The DQE Address Table (DAT) is a variable length table built by the system generation program. 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 (-1W) is contained in C.ADAT. The number of DAT entries is contained in C.NQUE and is equal to the number of DQE's.

2.4. Input Output

2.4.1 I/O Table Linkages


User's File Control Block (FCB)
FAT Pointer Table (FPT)
(2)
(10)
(1)
(3)
I/O Queue
File Assignment Table (FAT)
(9)
(4)
(5)
Device Type Table
(6)
Controller Definition Table (CDT)
(7)
Unit Definition Table (UDT)
(8)
Device Handler

Notes:

(1) & (2) Established at H.IOCS OPEN Time

(3), (4) & (5) Established at File Allocation Time

(6), (7) & (8) Established at SYSGEN Time

(9) & (10) Established at H.IOCS Op Code Processing Time

2.4.2 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 file or device is opened. This completes the logical connection from the task to the requested file or device 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.

Word	0 3	4 7	8 11	12 15	16 31
0	Reserved	Opcode (FCB.OPCD)	Logical file code (FCB.LFC)		
1	Quantity (FCB.TCW)			Data Address	
2	General control flags(FCB.GCFG)		Special flags (FCB.SCFG)	Random access address (FCB.CBRA)	
3	Status flags (FCB.SFLG)		Test status	DCC status	Device status
4	Record length (bytes) (FCB.RECL)				
5	Reserved		I/O queue address (FCB.IOQA)		
6	Special status (FCB.SPST)	Reserved	Wait I/O error return address (FCB.ERRT)		
7	Reserved		FAT address (FCB.FATA)		
8	Reserved		Expanded data address (FCB.ERWA)		
9	Expanded transfer quantity (bytes) (FCB.EQTY)				
10	Expanded random access address (FCB.ERAA)				
11	Extended I/O status word 1 (FCB.IST1)				
12	Extended I/O status word 2 (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	Reserved for I/O service expansion				

WORD 0

Bits 0-3 This field is always zero.

Bits 4-7 Operation code - A single hexadecimal digit specifies the type of function requested of the device handler. The allowable functions and their definitions are unique to each peripheral device.

Bits 8-31 Logical file Code - Any combination of three ASCII codes is allowed.

WORD 1

Note: Words 8 and 9 are used instead of Word 1 if Bit 6 of Word 2 is set.

Bits 0-11 Quantity - Three hexadecimal digits specify 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 12-31.

(or)

For GPMC devices which support chaining, number of Data/Command chain doublewords. If data/command chaining is desired (execute channel H.IOCS,10), this is used to indicate the number of data/command chain doublewords in the list.

Bits 12, 30,31 Format Code - These specify the addressing mode for data transfers. They are interpreted as follows:

Type of Transfer	F	C
Byte	1	xx
Halfword	0	y1
Word	0	00

xx = Byte Number (00,01,10 or 11)
y = 0, Left Halfword
y = 1, Right Halfword
00 = Word		

If a halfword or word transfer is specified for a device which accepts only bytes, IOCS adjusts the quantity accordingly. If a byte transfer is specified for a device which accepts only halfwords or words, IOCS will adjust the quantity accordingly if the number of bytes is an even multiple of the requested transfer mode and the data address is on the correct boundary. Otherwise, the request is treated as a specification error.

Bits 13-29 Data Address - The initial address data areas for read or write operations.

(or)

Data/Command Chain Address - If using execute channel entry point (H.IOCS,10).

WORD 2

Bits 0-7 General Control Specifications - 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
0	If this bit is set, IOCS will return to the user immediately after the I/O operation is queued. If the bit is reset, IOCS will exit to the calling program only when the requested operation has been completed.
1	If this bit is set, error processing will not be performed by either the device handler or IOCS. Normal error processing for disc 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 disc or magnetic tape is unsuccessful, an error status message is typed to the console. An error return address is not applicable if this bit is set, however the device status will be posted in the FCB (unless bit 3 is set).
2	When this bit is set, data formatting is inhibited. Otherwise, data formatting is performed by the appropriate device handler. (See Bit 8 for further explanation.)
3	If this bit is set, the device handlers perform no status checking and no status information is returned. Hence all I/O will appear to complete without error.
4	When this bit is set, file accessing will occur in the random mode. Otherwise, sequential accessing will be performed.
5	If set, a blocked file is specified (disc or tape assignments only).

6 Expanded FCB present (Words 8-15). This takes advantage of a larger I/O transfer quantity (in bytes), a 24-bit addressing field, and a 32-bit random access address. For Extended I/O operations, up to two interrupt status words are then returned after I/O complete. When this bit is set, IOCS assumes the FCB is 16 words long. The information in Words 8 and 9 is used instead of the data in Word 1. Also, the random access address in Word 10 is used instead of the data in word 2.

7 If this bit is set, a task will not be aborted even if an error condition occurs that would cause the task to be aborted.

Bits 8-12 Special Control Specification - this field contains device control specifications unique to certain devices. Interpretation and processing of these specifications are performed by the device handlers. A specification item is ignored unless the designated file is assigned as indicated below:

Bit Meaning

---

8 Normally, this bit is examined only when Bit 2 (data formatting inhibit) is set. The meaning is interpreted as shown in Section 2.4.2.1.

9 This bit has significance only for the following devices:

7-track tape: examined only if bit 2 set and Bit 8 not set	If set, odd parity If reset, even parity
ALIM (Read): examined only if Bit 2 reset	If set, ECHO mode If reset (and Bit 8 reset): Receive data If reset (and Bit 8 set): BLIND mode
ALIM (Write): examined only if Bit 2 reset	If set, device initilization If reset, Formatted Write
ADS (teletype/CRT):	If set the handler will issue a LCW to the ADS controller
FHD (read or write):	If set, FHD will read or write single sector If reset, FHD will read or write sectors sequentially

10 This bit has significance only for the following devices:

7-track tape:	If set, 556 BPI mode If reset, 800 BPI mode
ALIM (Read): Teletype/CRT	If set, inhibit conversion of lower case characters to upper case.

11 This bit has significance only for the following devices:

ADS (teletype/CRT):	If set, the handler will issue a TXB command to the ADS controller

12 This bit has significance only for the following devices:

ADS (teletype):	If set, external asynchronous interrupt (EAI) status will be posted in the FCB (FCB.IST1) only if an expanded FCB, refer to Bit 6 description

Note:

The expanded random access address in Word 10 (FCB.ERAA) is used instead of bits 12-31 in Word 2 if bit 6 of Word 2 is set.

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

2.4.2.1 Special Control Specifications

Device	BIT 2	BIT 8
CR	0=Read in Automatic mode select	Not interpreted
	1=Determine mode by Bit 8	0=ASCII read
CP	0=Punch in automatic mode select	Not interpreted
	1=Determine mode by Bit 8	0=ASCII punch 1=Binary punch
PT (Reader)	0=Read in formatted mode, skipping leader	Not interpreted
	1=Read unformatted	0=Do not skip leader 1=Skip leader
PT (Punch)	0=Punch in formatted mode	Not interpreted
	1=Punch unformatted	Not interpreted
LP	0=Interpret first character as carriage control	1=No carriage control and print buffer. No LF, just print.
	1=No carriage control Default=LF and print	
TY	0=Interpret first character as carriage control	Not interpreted
	1=No carriage control	Not interpreted
MT (Applicable to 7-Track only)	0=Read/write packed mode (binary)	Not interpreted
	1=Determine mode by Bit 8	0=Interchange (BCD) 1=Packed (binary)

DEVICE	BIT 2	BIT 8
DM,DF,FL	Not interpreted	Not interpreted
ALIM(READ)	0=Determine mode by Bits 8,9	0=Determine mode by Bit 9 1=BLIND mode
	1=Receive data	Not interpreted
ALIM(WRITE)	0=Determine mode by Bit 9	Not interpreted
	1=Unformatted write	Not interpreted

WORD 3

Bits 0-31 Status Word - 32 indicator bits are used by IOCS to indicate the status, error and abnormal conditions detected during the current or previous operation. The assignment of these bits is shown below:

Bit	Meaning
0	Operation in progress. (Request has been queued.) (Note: Reset after post I/O processing complete.)
1	Error condition found.
2	Invalid Blocking Buffer control pointers have been encountered during file blocking or deblocking.
3	Write protect violation.
4	Device inoperable.
5	Beginning-of-medium (BOM) (load point) or illegal volume number (multi-volume magnetic tape).
6	End-of-file.
7	End-of-medium (end of tape, end of disc file).

Non-Extended I/O Devices:

8-11	Specifies general testing status as received from an 8000 level Test Device instruction.
12-15	Specifies DCC testing status as received from a 4000 level Test Device instruction.
16-31	Specifies a device status as received from a 2000 level Test Device instruction. These bits are not applicable for the Paper Tape, Card Reader, and Teletypewriter. Bit meanings for 2000 level testing for non-extended I/O devices are shown in Section 2.4.2.2.

2.4.2.2 Device Status (2000 Level) Non-Extended I/O

FCB Word 3 Bits	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Line Printer LP	CD TERM	PROG VIO	DEV INOP	0	0	0	0	0	0	BOF	0	0	0	DEV BUSY	0	0
IOP Line Printer	0	0	0	0	0	0	0	0	0	0	0	0	CHAN END	DEV END	UNIT CK	BOF
MAG TAPE MT	CD TERM	PROG VIO	DEV INOP	VRC	0	REW IN PROG	CRC LRC	DATA LOST	0	EOT	BOT	EOF	0	DEV BUSY	FILE PROT VIO	ODD REC LGT
MOVING HEAD DISC DM (EXCEPT 1[illegible]MB)	CD TERM	PROG VIO	DEV INOP	CKSM ERR (HW)	CKSM ERR (SW)	FILE UN-SAFE	SEEK IN PROG	DATA LOST	0	SEC-TOR BIT 8	SEC-TOR BIT 4	SEC-TOR BIT 2	TCA ERR	SEC-TOR BIT 1	FILE PROT VIO	TRK ERR
FIXED HEAD DISC DF	CD TERM	PROG VIO	DEV INOP	CKSM ERR (HW)	CKSM ERR (SW)	0 SECTOR BIT 32	SLCT IN PROG SECTOR BIT 16	DATA LOST	0	SEC-TOR BIT 8	SEC-TOR BIT 4	SEC-TOR BIT 2	TCA ERR	SEC-TOR BIT 1	FILE PROT VIO	0
CARD READER/ PUNCH CD	CD TERM	PROG VIO	DEV INOP	ECHO CK.	0	PHOTO DIO ERR	0	0	0	0	0	0	0	DEV BUSY	0	0
CART. DISC DC	CD TERM	PROG VIO	DEV INOP	CKSM ERR (HW)	CKSM ERR (SW)	DEV. ABNOR-MAL	SEEK IN PROG	DATA LOST	0	OFFSET ACTIVE	ADDR ERR	DUAL TRK	TCA ERR	EOP	FILE PROT VIO	SEEK ERR
MOVING HEAD DISC DM (1[illegible]MB)	CD TERM	PROG VIO	DEV INOP	UNCOR DATA ERR	CKSM ERR (SW)	FAULT	SEEK IN PROG	CDR DATA ERR	0	0	ADDR ERR	0	TCA ERR	0	0	SEEK ERR
OTHER*																
HIGH SPEED DATA INTERFACE (GENERIC HANDLER)	CD TERM		ERROR STATUS FORMAT (See Word 3 description)							EXTERNAL TERM	IOCB ADDR ERR	ERR ON TI ADDR FETCH	DEV EOB	EPS ERR PRE-CLUDED REQUEST QUEUEING	(NONEXECUTE CHANNEL PROGRAM IOCB TYPE IN ERROR 00-DATA TRANSFER 01-DEV STATUS 10-COMMAND TRANSFER	

*TY, PT, CR not applicable

For Extended I/O Devices Only

Bit	Meaning
8	Zero
9	Zero
10	Last command exceeded time out value and was terminated.
11-15	Zero
16-23	Channel status (see Section 2.4.2.3)
24-31	Controller/device status (see Section 2.4.2.3)

2.4.2.3 Channel Status and Controller/Device Status for Extended I/O Devices

	Bit	Description
CHANNEL STATUS	16 -	Echo
	17 -	Post program controlled interrupt
	18 -	Incorrect length
	19 -	Channel program check
	20 -	Channel data check
	21 -	Channel control check
	22 -	Interface check
	23 -	Chaining check
CONTROLLER/ DEVICE STATUS	24 -	Busy
	25 -	Status modifier
	26 -	Controller end
	27 -	Attention
	28 -	Channel end
	29 -	Device end
	30 -	Unit check
	31 -	Unit exception

WORD 4

Bits 0-31 Record Length - This field is used 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 - This field is set by IOCS to point to the I/O queue for an I/O request initiated from this FCB.

WORD 6

Bit 0 No wait normal end action not taken

Bit 1 No wait error end action not taken

Bit 2 "Kill" command, I/O not issued

Bits 3-7 Reserved.

Bits 8-31 Wait I/O Error Return Address - 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

Bit 0-7 Reserved.

Bits 8-15 FAT Address - This field points to the File Assignment Table (FAT) entry associated with all I/O performed on behalf of this FCB. This field is supplied by IOCS.

Note: Words 8-15 are valid only if Bit 6 of Word 2 is set.

WORD 8

Bits 0-7 Reserved.

Bits 8-31 Expanded Data Address - Start address of data area for read or write operations. Must be a word address.

(or)

Expanded Data/Command Chain Address - Word address that points to the data or commmand chain list if using execute channel entry point (H.IOCS,10).

WORD 9

Bits 0-31 Expanded Quantity - Number of bytes of data to be transferred.

(or)

For GPMC devices which support data/command chaining: Expanded Number of Data/Command Chain Doublewords. If data/command chaining is desired (execute channel H.IOCS,10), this is used to indicate the number of data/command chain doublewords in the list.

WORD 10

Bits 0-31 Expanded Random Access Address - This field contains a block number (zero origin) relative to the beginning of the disc file. It is the start address for the current read or write operation.

(or)

For High Speed Data (HSD) Interface requests in non-Execute Channel Program format, this word defines a device command.

WORD 11

Bits 0-31 Status Word 1 - For extended I/O, these are the 32 bits returned by the SENSE command.

(or)

For communications adapter interface, external asynchronous interrupt (EAI) status if Bit 12 of Word 2 is set.

WORD 12

Bits 0-31 Status Word 2 - Second status word as returned from the Extended I/O hardware.

(or)

For High Speed Data (HSD) Interface applications, this word contains status sent from the user's device.

WORD 13

Bits 0-7 Reserved.

Bits 8-31 No-Wait I/O normal completion service address return. This user service must be terminated by calling H.IOCS,34 (no-wait I/O end action return).

(or)

For High Speed Data (HSD) Interface applications, this address plus 1 word is the location to which control is transferred on asynchronous notification.

WORD 14

Bits 0-7 Reserved.

Bits 8-31 No-Wait I/O error completion service address return. This user service must be terminated by calling H.IOCS,34 (no-wait I/O end action return).

WORD 15

Reserved for I/O service expansion.

2.4.3 Type Control Parameter Block (TCPB)

The Type Control Parameter Block (TCPB) is used to allow I/O to and from the OPCOM console by setting up task buffer areas for messages output by a task and optional reads back from the console.

Word	0	11	12	13	31
0	Output quantity (TCP.OQ)		1	Output data address (TCP.OTCW)	
1	Input quantity (TCP.IQ)		1	Input data address (TCP.ITCW)	
2	2	Reserved		Console Teletype Flag Byte (TCP.CONF)	3

Notes

1. This bit is set to one.
2. Bit 0 of this word is set if no-wait I/O.
3. Bit 31 of this word is set if operation in progress. (Note: Reset after post I/O processing complete.)
4. If no input is desired, word 1 of the TCPB must be zero.

2.4.4 Device Type Table (DTT)

The Device Type Table (DTT) is a system resident structure used to identify device types which are configured in the system and their associated controllers. The DTT is built by the SYSGEN process and its entries are linked to their associated Controller Definition Table (CDT) for identifying their controllers.

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

Notes

(1) For example, 01=any disc; 04=any magnetic tape; 08=any reader card; 0A=any line printer.

(2) Used by Job Control and Cataloger to validate ASSIGN3 statements with bits assigned as follows.

0 - If set, entry of device address not legal
1 - If set, entry of size or reel ID required
2 - If set, entry of reel ID required
3-7 - Reserved

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

2.4.5 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.

Word	0 7	8 15	16 23	24 31
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	Reserved
3	CDT index (CDT.INDX)		Device type code (CDT.DTC)	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 4	UDT address of first device on controller (CDT.UDTA)		
7	I/O status (CDT.IOST) See Note 5	TI address (CDT.TIAD) or SI if class 'F' (CDT.SIAD)		
8	UDT address unit 0* (CDT.UT0)			
9	UDT address Unit 1* (CDT.UT1)			
23	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. Bits in CDT.CLAS are assigned as follows.

X'00'	-	TLC line printer
X'01'	-	TLC card printer
X'02'	-	TLC typewriter
X'0D'	-	TCW type with bank bits
X'0E'	-	TCW type
X'0F'	-	Extended I/O

4. Bits in CDT.FLGS are assigned as follows.

0	-	Extended I/O device
1	-	I/O outstanding (set by handler, reset by IOCS)
2	-	GPMC device
3	-	Set if extended I/O initialization (INCH) has been performed for this channel
4	-	Set if D class (16MB GPMC)
5	-	I/O outstanding (set/reset by H.XMT)
6	-	If set, IOP controller (CDT.IOP)
7	-	If set, controller malfunction (CDT.MALF)

5. Bits in CDT.IOST are assigned as follows.

0	-	If set, IOQ linked to UDT (CDT.NIOQ)
1	-	Multiplexing controller
2-7	-	Reserved

6. CDT.SIZE = 24W

2.4.6 Unit Definition Table (UDT)

The Unit Definition Table (UDT) is a system resident structure used to identify 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 consequently its associated controller and handler.

Word	0 7	8 15	16 23	24 31
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 sub-address (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 sub-address (UDT.PSUB)	Sectors per block (UDT.SPB) or Number characters per line (UDT.CHAR) See Note (3)	Sectors per allocation unit (UDT.SPAU) or Number lines per screen (UDT.LINE) See Note (4)
4	Flags (UDT.FLGS) See Note (5)	Number of sectors per track on disc or global line counter if a terminal (UDT.SPT)	Maximum byte transfer (UDT.MBX)	
5	Total number of allocation units on disc or tab setting if a terminal (UDT.TAU)			
6	Sector size, on disc or a tab setting if a terminal (UDT.SSIZ)		Number of heads on disc or a tab setting if a terminal (UDT.NHDS)	
7	Serial number if tape or removable disc (UDT.SERN)			
8	Peripheral time out value (UDT.PTOV)			
9	Reserved	Address of context block (extended I/O) (UDT.CBLK) (UDT.XIOC)		
10	Bit flags See Note (6) (UDT.BIT2)			
11	Service interrupt handler address (UDT.SIHA)			
12	Device historical data address (UDT.HIST)			
13	Address of first IOQ linked to this device (UDT.FIOQ)			
14	Address of previous IOQ for this device (UDT.BIOQ)			
15	Link Priority (UDT.LPR1)	Link Count (UDT.IOCT)	Unit Status byte 2 (UDT.STA2) See Note (7)	Reserved

Notes

(1) Bits in UDT.STAT are assigned as follows.
0 - If set, on line
1 - If set, dual ported XIO disc
2 - If set, allocated
3 - If set, in use
4 - If set, system output unable to allocate
5 - If set, shared device
6 - If set, pre-mounted
7 - If set, terminal (TSM) device

(2) For example, 01 for any disc; 04 for any tape, etc.

(3) For discs, contains number of sectors per block (UDT.SPB). For terminals, contains number of characters per line (UDT.CHAR).

(4) For discs, contains number of sectors per allocation unit (UDT.SPAU). For terminals, contains number of lines per screen (UDT.LINE).

(5) Bits in UDT.FLGS are assigned as follows.
0 - If set, extended I/O device
1 - If set, I/O outstanding
2 - If set, removable disc pack
3 - If set, terminal user logged on
4 - If set, auto-selectable for batch SLO
5 - If set, auto-selectable for batch SBO
6 - If set, auto-selectable for real-time SLO
7 - If set, auto-selectable for real-time SBO

(6) Bits in UDT.BIT2 are assigned as follows.
0 -If set, port is private; else switched.
1 -If set, port is configured multi-drop.
2 -If set, port has graphic capability.
3 -If set, port is full duplex.
4-7 - Reserved

(7) Bits in UDT.STA2 are assigned as follows.
0 -If set, IOQ linked from this UDT.
1 -If set, IOP device (initialized by SYSGEN)
2 -If set, device malfunction
3 -If set, operator intervention applicable
4-5 - Reserved
6 -If set, cartridge module drive
7 -If set, moving head disc with fixed head option

2.4.7 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 equivalenced to the same FAT. The linkage to the FAT is performed at assignment. The linkage to the FCB is performed at opening. The FPT resides in the task's service area.

Word	0 7	8 31
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
   0 - If set, file dynamically allocated
   1 - If set, multiple FPT entries exist which point to the same FAT (i.e., "$ASSIGN4")
   2 - If set, FPT and corresponding FAT are allocated
   3 - If set, FPT open
   4 - If set, free to allocate
   5-7 - Reserved

2.4.8 File Assignment Table (FAT)

The File Assignment Table (FAT) is used to provide an association between a logical file code (lfc) and a file or device. It also coordinates access to the file or device referenced via lfc. The FAT is linked to the Unit Definition Table (UDT) and the Controller Definition Table (CDT) when the file or device is allocated.

Word	0 7	8 12	13 15	16 31
0	Status (DFT.STB) See Note 1	Access (DFT.ACF) See Note 2	Sys See Note 3	CDT index (DFT.CDTX)
1	Flags (DFT.FLGS) See Note 4	ASSIGN4 count (DFT.NAS)		UDT index (DFT.UDTX)
2	Reel ID (MTF.REEL) or starting disc address (DFT.SDA)			
3	Magnetic tape volume number (bits 0-7)(MTF.VOL) or current disc address (Bits 0-31)(DFT.CDA)			
4	Number of physical blocks in file (DFT.BIF)			
5	Open file count (DFT.OPCT)	Blocking Buffer address (DFT.BBA)		

Notes

1. Bits in DFT.STB are assigned as follows.
   0 - If set, file open
   1 - If set, file opened read/write
   2 - If set, permanent file
   3 - If set, blocking buffer output active
   4 - If set, file assigned magnetic tape
   5 - If set, file assigned to disc
   6 - If set, read only access
   7 - If set, TSM associated FAT

2. Bits in DFT.ACF are assigned as follows.
   If disc:
   0-1 Reserved
   2 If set, "$" read on SYC
   3-4 Reserved

   If tape:
   0 - If set, mount message has been inhibited or tape is shared
   1 - If set, multivolume tape
   2 - If set, mount message has been output
   3 - If set, tape at EOT
   4 - If set, tape at BOT

3. Bits 5-7 in DFT.ACF will contain one of the following values.
Value=0 - Not a system file
Value=1 - SYC file
Value=2 - SGO file
Value=3 - SLO file
Value-4 - SBO file

4. Bits in DFT.FLGS are assigned as follows.
0 - Blocking buffer present
1-3 - Reserved
4 - If set, FAT space available
5 - If set, TSM I/O (task is swappable)
6-7 - Reserved

2.4.9 I/O Queue (IOQ)

The I/O Queue (IOQ) is used to identify information required to process and queue a specific I/O operation. IOQ entries are allocated for processing I/O requests. They are linked to the File Control Block (FCB) and Controller Definition Table (CDT) at the start of operation code processing for an operator request. Also included is information for error processing, transaction identifier, control information, etc. The IOQ entry is variable in length and allows for command and data chaining for supporting extended I/O.

Word	0 7	8 15	16 23	24 31
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)	Sub-address (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 1 (IOQ.FCT1)			
9	Handler function word 2 (IOQ.FCT2) See Note 3			
10	Handler function word 3 (IOQ.FCT3) See Note 4			
11	Handler function word 4 (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 1 (IOQ.IOST)			
18	I/O returned status word 2 (IOQ.IST1) See Note 7			
19	I/O returned status word 3 (IOQ.IST2) See Note 8			
20	UDT address (IOQ.UDTA)			
21	Control information from word 2 of FCB (IOQ.CONT)			
22	Address of context block (extended I/O) (IOQ.CBLK)			
23	Mode bits (extended I/O) (IOQ.MODE) See Note 9 or Word address of set mode bits (IOQ.MOWD)	Reserved	Number of words extra in this queue entry (IOQ.XTRA)	
24	Device inop buffer address (for I/O error processing) (IOQ.INOP)			
25	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 post processing routine (S.IOCS1). When I/O completes, control will be passed to this service.

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

   0 - If set, I/O queue is active (Note: Reset by device handler when physical I/O transfer completes.)
   1 - If set, Sense command was issued on behalf of this I/O request (extended I/O)
   2 - If set, Error retry to be issued (rezero and retry entire IOCD list) (extended I/O)
   3 - If set, Sense command is to be issued on behalf of this I/O request (extended I/O)
   4 - If set, Read ECC needs to be issued (extended I/O)
   5 - If set, Read ECC was issued (extended I/O)
   6 - If set, Error retry to be issued (retry entire IOCD list) (extended I/O)
   7 - If set, ECC correction performed, continue processing IOCD list (extended I/O)

3. For extended I/O devices, IOQ.FCT2 contains the 24-bit virual address of the data (or) IOCL. (Bits 0-7=0)

4. For extended I/O devices, IOQ.FCT3 contains the adjusted byte transfer count in bits 0-31 (maximum is 512K bytes).

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

   0 - If set, multiplexed controller
   1 - If set, OPCOM console request
   2 - If set, TCW has been absolutized
   3 - If set, IOQ will be linked to the UDT
   4 - If set, deallocate OS buffer
   5 - If set, extended I/O
   6 - If set, error found
   7 - If set, system console queue
   8 - If set, data move required (OS to user buffer)
   9 - If set, rewind command in IOCD list for magnstic tape or Reserve command in IOCD list for disc (extended I/O)
   10 - If set, non-execute channel read command (extended I/O)
   11 - If set, non-execute channel write command (extended I/O)
   12 - If set, special handler post processing required (Handler EP6)
   13 - H.CT00 has been called with an FCB, not with a TCPB (i.e., not via H.IOCS,14)
   14 - If set, special class 'E' device I/O buffer in use
   15 - If set, terminal input
   16 - If set, terminal output
   17 - If set, task swappable during I/O
   18 - If set, release command in IOCD list for disc (extended I/O)
   19 - No-wait I/O (not TSM)
   20 - If set, I/O restart entry
   21 - If set, nondevice access I/O performed
   22 - "Kill" command issued for this I/O request
   23 - If set, execute channel by handler (extended I/O)
   24 - If set, user privileged

25 - D class controller (GPMC) only)
26 - Physical I/O performed on behalf of a user requesting blocked I/O
27 - Set by XIO disc handler when a reserve is issued to a dual port disc already reserved by an opposing CPU
28 - If set, priority override to be issued (extended I/O)
29-31 - Reserved

6. For extended I/O devices, IOQ.UTRN is a full word (Bits 0-31) 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 IOCD's.

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

9. Mode bits are peculiar to each device.

10. IOCD's are built and stored dynamically. Starting at this word in the I/O queue, as many IOCD's will be chained as required in case of discontinuity due to program loading. This cell contains the absolute data (or) IOCL address associated with the I/O request initially.

2.4.10 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

Word	0 11	12 31
0	Buffer status (1)	Next read/write address

Notes

(1) Bits in this field are assigned as follows.
0 - Reserved
1 - Set if buffer is empty
2-3 - Reserved
4 - Set if buffer is free to allocate
5-11 - Reserved

Record Control Bytes

0 7	8 15	16 23	24 31
Status bits last record (1)	Byte count last record	Status bits this record(2)	Byte count this record

Notes

(1) Bits in this field are assigned as follows.
0 - Set if end-of-file
1 - Set if beginning of block
2 - Set if end of block
3-7 - Reserved

(2) Bits in this field are assigned as follows.
0 - Set if end-of-file
1-7 - Reserved

(3) For the last record in a block, "status bits this record" and "byte count this record" are omitted.

2.4.11 IOP Channel Definition Table (CHT)

The Channel Definition Table (CHT) is a system resident structure applicable only to F-class 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 Controller Definition Tables (CDTs) linked to the channel, and defines other pertinent channel information.

Word	0 7	8 15	16 23	24 31
0	Register save area (CHT.REGS) (See Note 1)			
7				
8	Old PSD1/Old PSD2 (CHT.OPSD)			
9				
10	New PSD1/New PSD2 (CHT.NPSD)			
11				
12	IOCL Address (CHT.IOCL)			
13	Status Address (CHT.STAD)			
14	Flag Word (CHT.FLGS)			
15	Channel Interrupt Priority* (CHT.IPL)	Channel Address* (CHT.CHAN)	Channel Spurious Interrupt Count (CHT.SPUR)	
16	CDT Address Unit 0* (CHT.CDT0) (See Note 2)			
17	CDT Address Unit 1* (CHT.CDT1) (See Note 2)			
31	CDT Address Unit 15* (CHT.CDTF) (See Note 2)			
32	IOP Status Doubleword (CHT.STDW) (or)			
	Subaddress (CHT.SUBA)	Real IOCD Address (CHT.RIOA)		
33	Channel Status (CHT.CHST)	Cont/Device Status (CHT.CDST)	Residual Byte Count (CHT.RBC)	
34	Address of H.IOPXx Exit Entry Point (CHT.EXIT) (See Note 3)			
35	Address of H.IOPX Initialization Entry Point (CHT.INCH) (See Note 4)			
36 : 39	Reserved			

* Initialized by SYSGEN

Notes:

1. CHT.REGS must begin on a register file boundary.
2. These fields contain the addresses of the CDT entries for controllers connected to the corresponding IOP. Entries for unimplemented controllers will be set to zero.

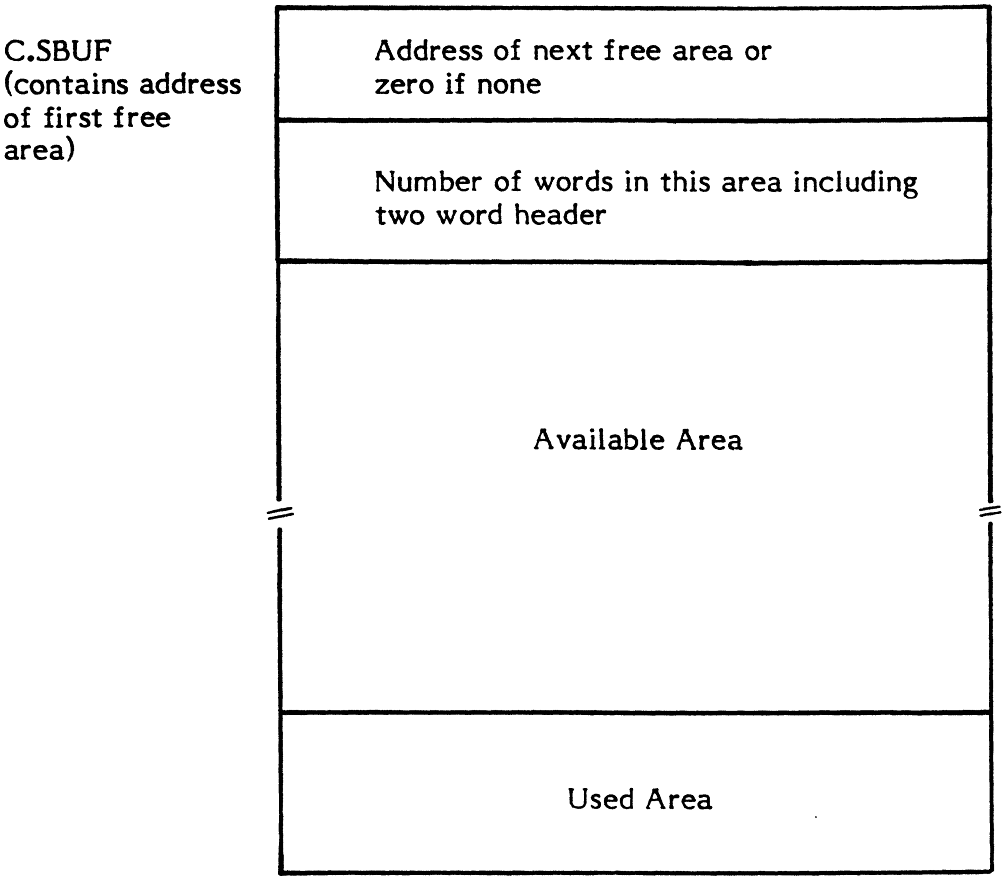
3. CHT.EXIT contains the address of the exit procedure in the corresponding IOP Interrupt Executive Program.
4. CHT.INCH contains a SIO instruction used to initialize the corresponding IOP channel.

2.5 Memory Management

2.5.1 Memory Pool Management

Memory Pool is an area of main memory beginning at the high-address end of resident MPX. Its size is specified at SYSGEN and it occupies an area up to the next map block boundary. Areas within Memory Pool are allocated in multiples of two words. Free areas are linked together; used areas are not linked and the entire area is useable. C.SBUF contains the address of the first free area within Memory Pool or equals zero if no free areas exist. C.POOL contains the starting address of Memory Pool which begins on a two word boundary.

Memory Pool


C.SBUF (contains address of first free area)
Address of next free area or zero if none
Number of words in this area including two word header
Available Area
Used Area

2.5.2 Memory Allocation Tables

The Memory Allocation Tables contain the current status of each 8KW map block of main memory that is present in a configuration. The address of this table is contained in C.MATA.

Each Memory Allocation Table entry consists of a flag byte representing the status of a configured map block. If C.MACH equals a 32/7x, there is one flag byte for every 8KW map block configured. If C.MACH equals a CONCEPT/32, there is one flag byte for every 2KW 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
MEM.CNT		MEM.SMN	
MEM.STAT	MEM.STAT	MEM.STAT	MEM.STAT
MEM.STAT	MEM.STAT	etc.	etc.

1. MEM.CNT - the number of map blocks configured in the system.

2. MEM.SMN - starting map number for memory table.

3. Flag bits in MEM.STAT are defined as follows.

MEM.ALL	0	If set, map block is allocated
MEM.SHR	1	If set, map block is shared
MEM.PRO	2	If set, multiprocessor shared
MEM.MAL	3	If set, malfunction exists
MEM.CON	4	If set, nonpresent
MEM.TYP	5	If set, semiconductor
MEM.CL1	6	Defined as follows
MEM.CL2	7	Defined as follows

Bit 6	Bit 7	Class
0	0	E
0	1	H
1	0	S
1	1	Undefined

2.5.3 Shared Memory Table (SMT)

Each entry in the Shared Memory Table (SMT) defines a shared memory area, i.e., 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.

Shared Memory Table

	0 7	8 15	16 23	24 31
0	SMT.NAME (Partition name)			
1				
2	SMT.TNUM (Ownername or task number)			
3				
4	SMT.FLAG		SMT.ACNT	SMT.UCNT
5	SMT.MAPN	SMT.MAPS	SMT.MTY	SMT.QUE
6	SMT.PAGE		SMT.PTOT	
7	SMT.SCTN		SMT.PSWD	
8	SMT.UDTI	SMT.SCTA		
9	SMT.IND	Reserved		
10	SMT.MIDL		SMT.MIDL+1H	
11	Map Image Descriptor List			
12				
13	For 32/7x provides up to 96 KW		For CONCEPT/32 provides up to 152 KW	
14				
15				

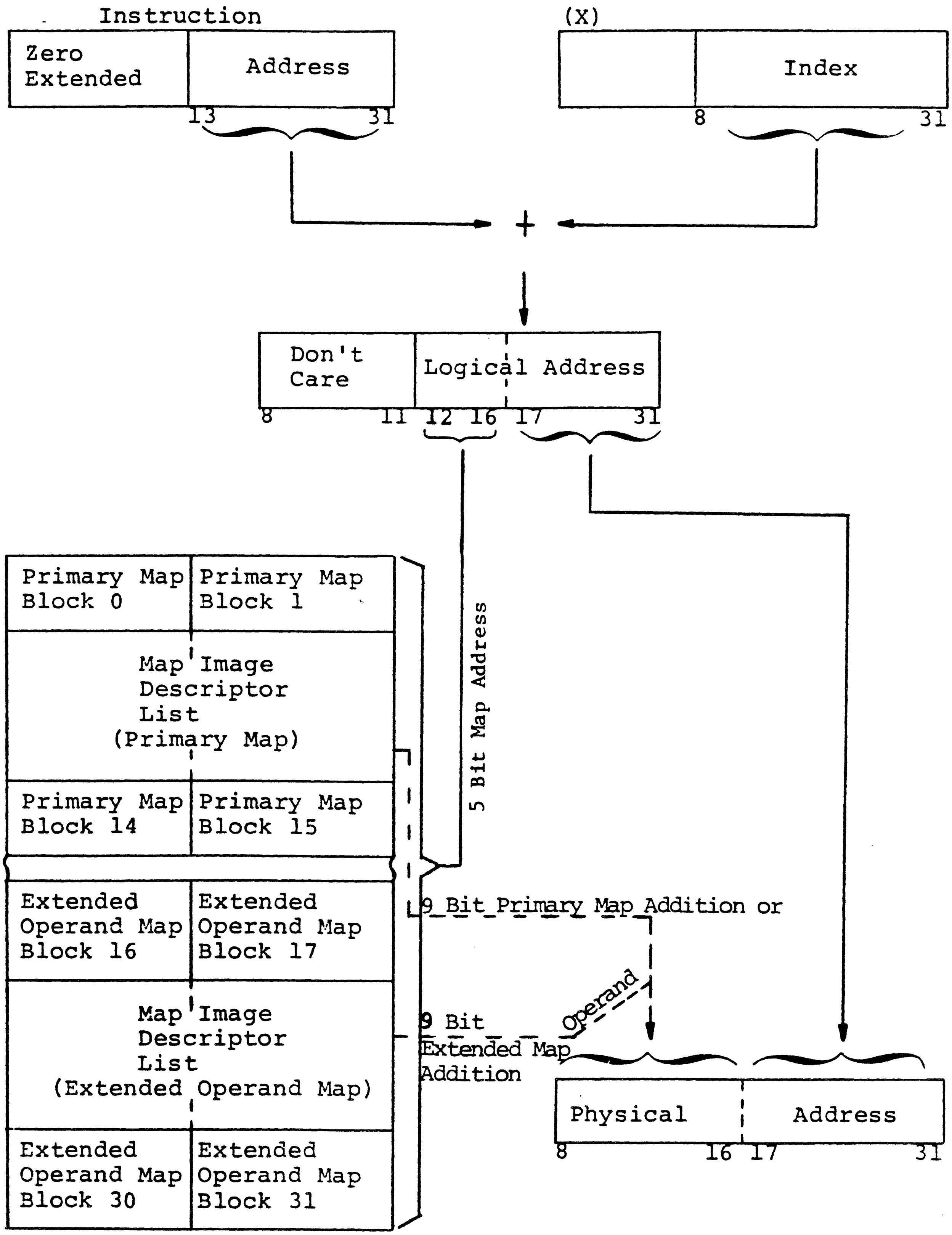
Bits in SMT.FLAG are assigned as follows.

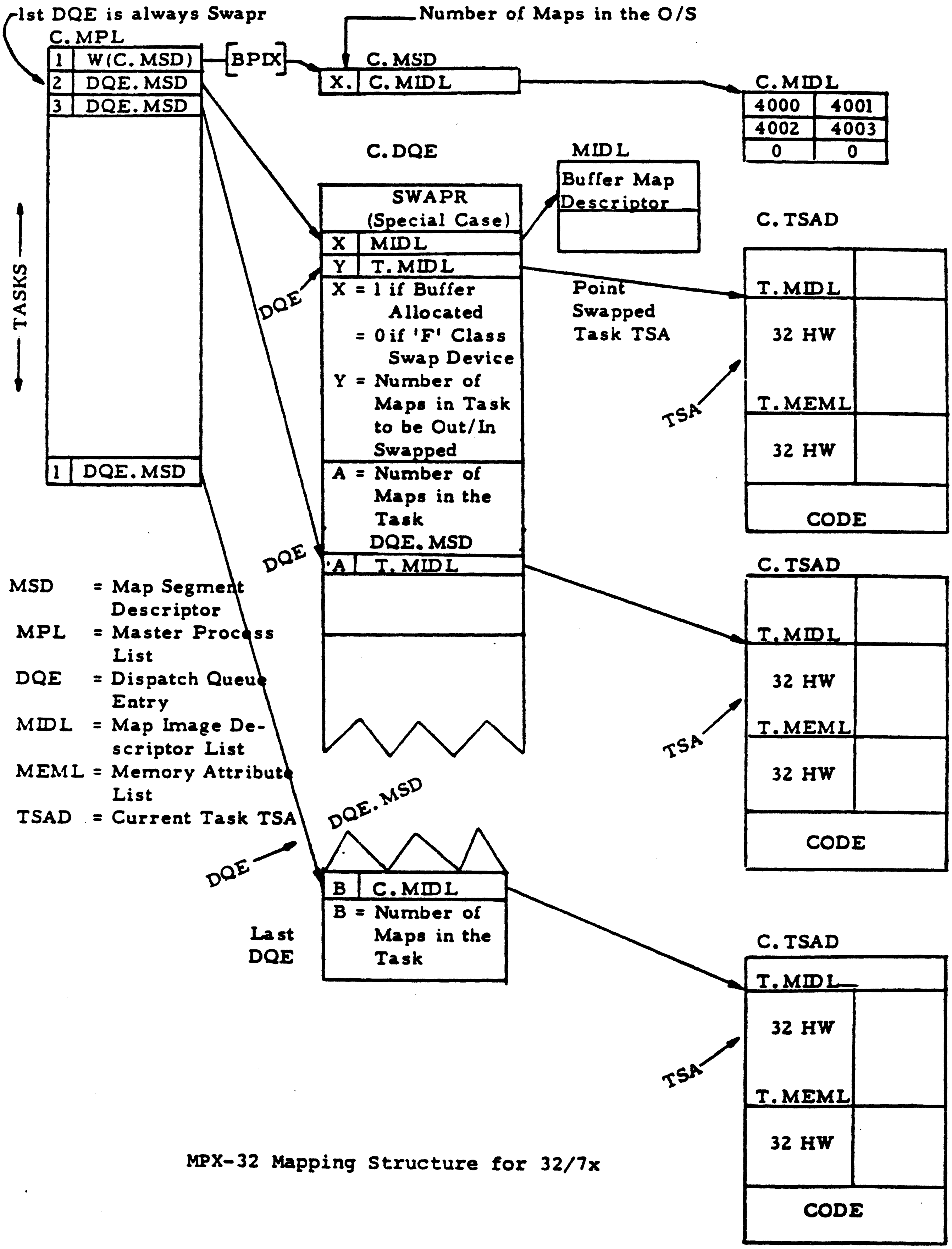
SMT.CSCT	0	Entry is a CSECT area
SMT.STCM	1	Entry defines a static common
SMT.DYCM	2	Entry defines a dynamic common
SMT.SWBL	3	Partition is swappable
SMT.OUTS	4	Partition is outswapped
SMT.BLDG	5	Shared memory table is in unstable state
SMT.RO	6	Partition is read only protected
SMT.PO	7	Partition has password limited access
SMT.PRSW	8	Set if previously outswapped (CSECTS only)
SMT.LOCK	9	Set if user requested no auto DEQUE

SMT.QUE - Number of tasks queued to this table
SMT.ACNT - Number of tasks sharing this memory area
SMT.UCNT - Number of tasks sharing this area that are not outswapped
SMT.MAPN - Number of map image descriptors

SMT.MAPS	-	Starting map register number
SMT.MTY	-	Memory type
SMT.IND	-	Shared memory table index
SMT.PAGE	-	Starting 512 word page number (static case only)
SMT.PTOT	-	Total number of pages (static case only)
SMT.SCTN	-	Number of sectors in swap file
SMT.PSWD	-	Password
SMT.UDTI	-	UDT index of swap file
SMT.SCTA	-	Sector address of swap file
SMT.MIDL	-	Beginning of map image descriptor list

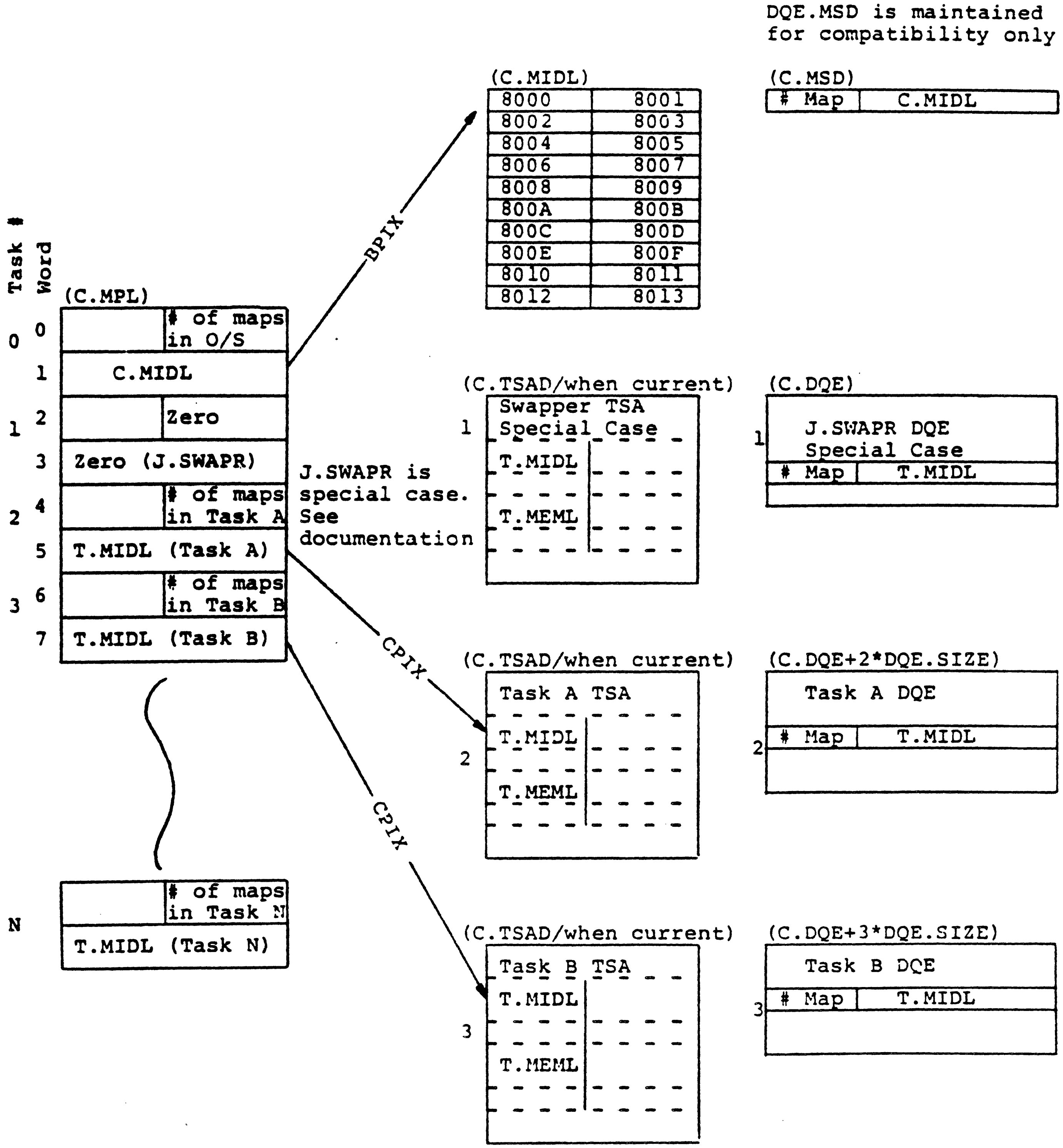
Address Generation Using the MPX 8KW Mapping Scheme


Instruction
Zero Extended
Address
13
31
(X)
Index
8
31
+
Don't Care
Logical Address
8
11
12
16
17
31
5 Bit Map Address
Primary Map Block 0
Primary Map Block 1
Map Image Descriptor List (Primary Map)
Primary Map Block 14
Primary Map Block 15
Extended Operand Map Block 16
Extended Operand Map Block 17
Map Image Descriptor List (Extended Operand Map)
Extended Operand Map Block 30
Extended Operand Map Block 31
9 Bit Primary Map Addition or
Operand
9 Bit Extended Map Addition
Physical
Address
8
16
17
31



MSD = Map Segment Descriptor
MPL = Master Process List
DQE = Dispatch Queue Entry
MIDL = Map Image Descriptor List
MEML = Memory Attribute List
TSAD = Current Task TSA

MPX-32 Mapping Structure for 32/7x



MPX-32 Map Structure for 32/27

2.6 Disc Management

The following variables contain the definition of the System Master Directory (SMD).

C.SMDUDT	UDT index of disc Containing SMD	
C.SMDD	Must be zero	Starting disc address (Starting Block Number)
	Must be zero	Length in 192-word blocks
C.SMDS	Number of entries in SMD	

Variables used by H.FISE for gating are as follows:

C.FGONR	DQE Address of FISE gate owner
C.FSFLGS	

Bit 0	=	FISE busy
Bit 1	=	gated externally
Bit 2	=	temporary allocation if set
	=	permanent allocation if reset
Bits 3-7	=	Reserved

2.6.1 SMD Entries

SMD.SIZE is equated to the SMD entry length.

Disc File SMD Entry

<table>
<tr><th>Word</th><th>0 7</th><th>8 15</th><th>16 31</th></tr>
<tr><td>0</td><td colspan="3" rowspan="2">File name (SMD.AFN)</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td><td>File type (SMD.FTYP)</td><td colspan="2">Start disc address (starting block number) (SMD.SBN)</td></tr>
<tr><td>3</td><td>File indicators (SMD.FIN)</td><td colspan="2">Length in 192-word blocks (SMD.BIF)</td></tr>
<tr><td>4</td><td colspan="3" rowspan="2">User name (SMD.AUN)</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td><td colspan="2">Compressed password (SMD.PWD)</td><td>UDT index (SMD.UDTX)</td></tr>
<tr><td>7</td><td colspan="3">Reserved (SMD.NU)</td></tr>
</table>

Memory Partition SMD Entry

<table>
<tr><th>Word</th><th>0 7</th><th>8 15</th><th>16 31</th></tr>
<tr><td>0</td><td colspan="3" rowspan="2">File Name (SMD.AFN)</td></tr>
<tr><td>1</td></tr>
<tr><td>2</td><td colspan="2">Starting logical page # (SMD.SLP)</td><td>Starting physical page # or zero (SMD.SPP)</td></tr>
<tr><td>3</td><td>File indicators (SMD.FIN)</td><td>Memory class (SMD.MC)</td><td>Length in pages (SMD.LIP)</td></tr>
<tr><td>4</td><td colspan="3" rowspan="2">Reserved</td></tr>
<tr><td>5</td></tr>
<tr><td>6</td><td colspan="2">Compressed password (SMD.PWD)</td><td>Reserved</td></tr>
<tr><td>7</td><td colspan="3">Reserved (SMD.NU)</td></tr>
</table>

Notes

1. File type (SMD.FTYP) specifies a two-character hexadecimal code which is output by the File Manager in ASCII. Default is 00.

ED	-	EDITOR SAVE
EE	-	EDITOR STORE
FE	-	EDITOR Workfile

FF	-	SYSGEN-created
BA	-	BASIC (unused)
CA	-	Cataloged Load Module
01-99	-	Available for Customer Use

2. Bits in SMD.FIN are assigned as follows.

0	-	If set, permanent file is active
1	-	If set, SYSGEN memory partition
2	-	If set, File Manager (do not save)
3	-	If set, fast file
4	-	If set, collision mapping
5	-	If set, non-SYSGEN memory partition
6	-	If set, password is required to write
7	-	If set, password is required to read/write

2.6.2 Disc Allocation Map Table

C.DAMAPT contains the address of the Disc Allocation Map Table. C.DALMAP contains the address of the Disc Allocation Bit Map Buffer. C.DAMCST contains the address of the Disc Allocation Map Checksum Table. Refer to Section 3.7.39 for further details.

Disc Allocation Table

	0 7	8 15	16 23	24 31
0	Number of words in bit map (DAT.NWDS)		UDT index of the disc described by this map (DAT.UDTI)	
1	Flags	Starting block number of allocation map (192W)(DAT.SBM)		
2	Number of 192W blocks in allocation map (DAT.NBM)			
3	Number of 192W free blocks in unit (DAT.BFU)			

This table contains an entry for each disc defined to the system as follows:

Word 0

Bytes 0-1	Number of words in map
Bytes 2-3	UDT index of the disc described by this map

Word 1

Bit 0	Resident flag - set if map currently in memory, reset if map on disc
Bits 1-7	Reserved
Bytes 1-3	Starting block number of map

Word 2

Byte 0 Reserved

Bytes 1-3 Length of map in 192W blocks

Word 3

Number of 192W blocks currently available for allocation on this disc

The end of the table is indicated by Word 0 being zero.

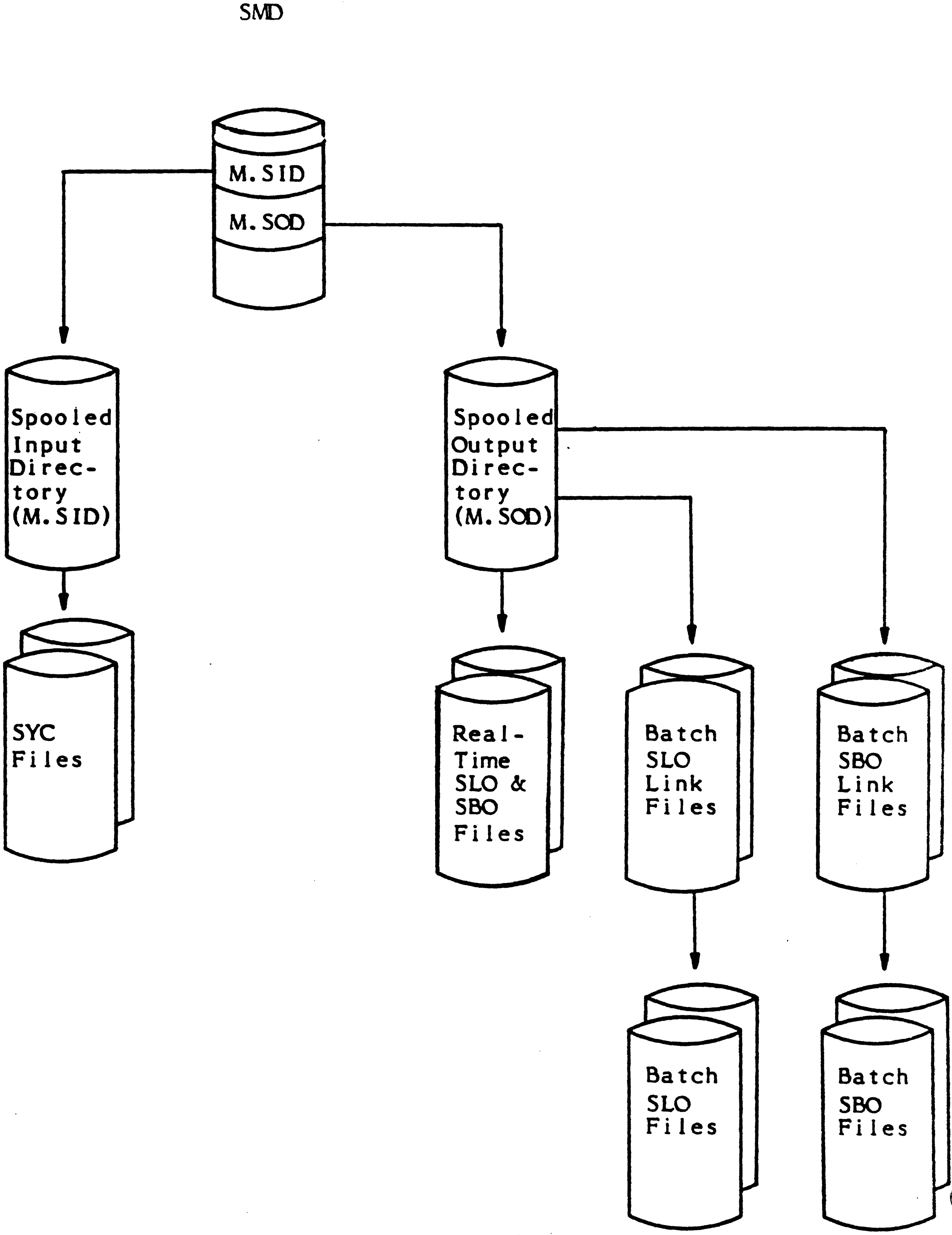
The Checksum Table contains one word for each Disc Allocation Table entry. SYSGEN initializes the Checksum Table so that each word has bit 0 set and bits 1-31 are zero.

2.7 Batch Processing

2.7.1 Spooled File Directories

Spooled file directories contain definitions of System Control (SYC), System Listed Output (SLO), and System Binary Output (SBO) files. The Spooled Input Directory (M.SID) contains definitions of SYC files. The Spooled Output Directory (M.SOD) contains definitions of real-time SLO and the SBO files and the definitions of batch SLO and SBO Link Files. Each job has an SLO Link File and and SBO Link File which contain definitions of the job's SLO and SBO files respectively. Entries in M.SID and M.SOD are linked by priority. Entries in SLO and SBO Link Files are sequential. A header entry occupies the first 12 words of the M.SID and M.SOD files. The header entry is followed by entries which contain actual file definitions. Unused entries must be zeroed.

Organization of Spooled Files



2.7.2 System Input (M.SID) and System Output (M.SOD) Directory Formats

	Word	0 15	16 31
Header Entry	0	Index to first entry (1)	
	1	Index to last entry (1)	
	2	Priority of first entry	Priority of last entry
	3	Last assigned job sequence number in binary (2)	
	4 // 11	Reserved	
File Definition Entry	0 1	Job name or real-time task's sequence number (SD.NAME)	
	2 3	M.SID: owner name from $JOB (SD.DEST) M.SOD: assigned destination task's pseudonym (SD.DEST)	
	4 5	M.SID: Reserved M.SOD: pseudonym of current destination processing task (SD.CURR)(3)	
	6	Job sequence number in ASCII (SD.JSEQ)	
	7 8 9	Disc space definition of SYC, real-time SLO/SBO, or batch SLO/SBO link file (SD.DEF)(4)	
	10	Priority (SD.PRI)	Flags (SD.FLAGS)(5)
	11	Index to next entry (SD.INDEX)(1)	

Notes

(1) Bits 0-7 contain the entry number of the M.SID/M.SOD entry relative to the beginning of the 192-word block that contains the entry. This value must be shifted left two places to obtain an index relative to the beginning of the 192-word block. Bits 8-31 contain the block number relative to the beginning of M.SID/M.SOD of the 192-word block that contains the entry.

(2) M.SID only. Unused in M.SOD.

(3) For M.SOD entries for jobs not completed, contains "BUILDING".

(4) The format of disc space definitions is as follows.

Word	0 7	8 15	16 31
0			UDT index
1		Starting disc address in 192-word blocks	
2		Length in 192-word blocks	

(5) Bits in SD.FLAGS are assigned as follows:

0	-	Set if M.SOD entry is for a batch job (SD.BATCH)
1	-	Set if M.SID entry is for sequential job (SD.SEQ)
2	-	Set if M.SOD entry is real-time and output is complete (SD.COMP)
3	-	Set if M.SOD entry is for SBO (SD.SBO)
4-15	-	Reserved

2.7.3 Link File Formats (Batch SLO and SBO)

Word	
0 1 2	Disc space definition of job's first SLO/SBO file [1]
0 1 2	Disc space definition of job's last SLO/SBO file [1]
0 1 2	Must be zero

Notes

1. The format of disc space definitions is as follows.

Word	0 7	8 15	16 31
0			UDT index
1		Starting disc address in 192-word blocks	
2		Length in 192-word blocks	

2.7.4 Job Table

The Job Table contains an entry for each currently active job. C.JOBA contains the memory address of the Job Table. C.JOBN contains the total number of entries in the Job Table.

Job Table Entry

Word	0 7 8 15 16 31
0 1	Job name from $JOB or contains zero if entry is unused (JOB.NAME)
2 3	Owner name from $JOB (JOB.OWNR)
4 5	SLO destination task pseudonym or file name (JOB.SLOD)
6 7	User name of SLO destination file (JOB.SLUS)
8 9	Password of SLO destination file (JOB.SLPW)
0 1	SBO destination task pseudonym or file name (JOB.SBOD)
2 3	User name of SBO destination file (JOB.SBUS)
4 5	Password of SBO destination file (JOB.SBPW)
6 7	User name from $USERNAME (JOB.UNAM)
8 9	PSD at batch task abort (JOB.ABAD)
0 1 2	Batch task abort code (JOB.ABT)
3	T.BIAS at batch task abort (JOB.BIAS)
4	Sequence # of task processing this job (JOB.TSKP)
5	Sequence # of current batch task if any (JOB.STEP)
6	Job sequence number in ASCII (JOB.SEQ)
7	Cumulative job CPU time (JOB.XTIM)
8	Job start time (JOB.STIM)
9	Job end time (JOB.NTIM)
0	Program option word (JOB.PGOW)
1 2 3	SGO file definition (JOB.SGO) [1]
4	SGO next write address (JOB.SGAD)
5 6 7	SYC file definition (JOB.SYC) [1]
8	SYC next read address (JOB.SYAD)

Word	0 7	8 15	16 31
9	M.SOD SLO index (JOB.SLIX) [2]		
0 1 2	SLO link file definition (JOB.SLLF) [1]		
3	Number of SLO links allowed (JOB.SLAL)		
4	Number of SLO links currently output (JOB.SLCR)		
5	M.SOD SBO index (JOB.SBIX) [2]		
6 7 8	SBO link file definition (JOB.SBLF) [1]		
9	Number of SBO links allowed (JOB.SBAL)		
0	Number of SBO links currently output (JOB.SBCR)		
1	Flags (JOB.FLGS) [3]		
2	Conditional job control flags (JOB.COND)		Compressed key from $USERNAME (JOB.KEY)
3	JOB.SCAN*	Job priority (JOB.PRI)	

*Last scan column for Job Control statements (JOB.SCAN)

Notes

1. The format of all file definition items is as follows.

Word	0 7	8 15	16 31
0			UDT index
1		Starting disc address in 192-word blocks	
2		Length in 192-word blocks	

2. M.SOD indexes are formatted as follows.

0 7	8 31
Entry number	192-word block number

The "entry number" is relative to the beginning of a 192-word block and must be shifted left two positions to obtain an index relative to the beginning of the block. The 192-word block relative to the beginning of M.SOD that contains the entry is contained in the "block number" field.

3. Bits in JOB.FLGS are assigned as follows
0 - Set if SLO destination is null (JOB.SLNO)
1 - Set if SLO destination is a permanent file (JOB.SLPF)
2 - Set if SBO destination is null (JOB.SBNO)
3 - Set if SBO destination is permanent file (JOB.SBPF)
4 - Set if operator REMOVE request for this job (JOB.RMOV)
5 - Set if SLO link file is allocated for job (JOB.JLL)
6 - Set if SBO link file is allocated for job (JOB.JBL)
7 - Set if SGO file is allocated for job (JOB.JGA)
8 - Set if Job Control task has completed processing of this job (JOB.DONE)
9 - Set if $JOB statement has been processed by Job Control task (JOB.JBCD)
10 - Set if job is sequential (JOB.SEQL)
11 - Set if previous batch task aborted or was deleted (JOB.ABDE)
12-31 - Reserved

2.8 Terminal Services

2.8.1 Terminal Line Buffer

The terminal Line Buffer is used to buffer terminal input and output. It is allocated from memory pool for each on-line 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. It is deallocated when the terminal is closed.

Word	0 7	8 15	16 23	24 31
0	Last argument [1] found by scanner			
1				
2	Buffer [2] length	Cursor [3] index	Field [4] delimiter	Field [5] size
3				
nW (length may vary)				

Notes

1. Doubleword containing last argument found by syntax scanner. It is left justified and blank filled.

2. Byte containing length of line buffer.

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.

Terminal I/O Buffer Pool Entry

Word	0 - 15	16 - 31
0	UDT index of terminal allocating this buffer or zero if free	I/O Buffer (C.TWID contains length in bytes)
	//	//

2.9 MPX-32 Load Module Format

See Chapter 6 for detailed description.

2.10 Resource Requirement Summary (RRS) Entries

ASSIGN1 fc=permanent file

0 - 7	8 - 15	16 - 31
Type (1)(RRS.FLGS)	Logical file code (fc) (RRS.LFC)	
Password (2)		Permanent file name (3)

ASSIGN2 fc=system file (SLO or SBO)

0 - 7	8 - 15 16 - 31
Type (1)(RRS.FLGS)	Logical file code (fc) (RRS.LFC)
Reserved	"SLO" or "SBO" in ASCII (RRS.SFC)
Reserved	Size in 192-word blocks (RRS.SNO)

ASSIGN2 fc=system file (SGO or SYC)

0 7	8 15 16 31
Type (1)(RRS.FLGS)	Logical file code (fc) (RRS.LFC)
Reserved	"SGO" or "SYC" in ASCII (RRS.SFC)
Reserved	

ASSIGN3 fc=device

0	1 7	8 15	16	17 23	24 31
Type (1) (RRS.FLGS)		Logical file code (fc) (RRS.LFC)			
4	Device type code (RRS.DT)	Volume number (RRS.VN)	5	Channel number (RRS.CN)	Subaddress (RRS.SA)
For disc, size in 192-word blocks. For magnetic tape, the reel identifier in ASCII. (RRS.REEL)					

ASSIGN4 fc1=fc2

0 7	8 31
Type (1)(RRS.FLGS)	Logical file code (fc1) (RRS.LFC)
Reserved	Logical file code (fc2) (RRS.LFC)
Reserved	

Notes

(1) Bits in RRS.FLGS are assigned as follows.

0 - Set if ASSIGN1 type
1 - Set if ASSIGN2 type for SLO or SBO
2 - Set if ASSIGN2 type for SGO or SYC
3 - Set if ASSIGN3 type
4 - Set if ASSIGN4 type
5 - Set if "unblocked" is specified for an ASSIGN1 or ASSIGN3 type
6 - Mount message inhibited
7 - System file allocation

(2) Compressed equivalent of eight-character password or zero if none.

(3) The characters in the name are in six-bit code where hexadecimal values 0 through 3F correspond to ASCII values 20 through 5F.

(4) Bit zero of this word is set if a value is present in the channel number field.

(5) Bit 16 of this word is set if a value is present in the subaddress field.

2.11 Job Accounting

The job accounting facility indicates elapsed time and CPU time for all jobs. It optionally collects job information on an accounting file used by TSM.

The accounting file is not blocked by MPX-32. However, J.ACCNT blocks the file in a special format which is 12 entries per 192 words of physical record space, with each of the 12 entries containing 16 words.

Default project names/numbers for accounting purposes can be established with the M.KEY utility (see MPX-32 Reference Manual Volume 2, Chapter 7).

For use of wild card characters, see the OPCOM LIST command in the MPX-32 Reference Manual Volume 1, Chapter 4.

	0	31
Word 0	Owner name (1-8 character ASCII)	
1		
2	Project (1-8 character ASCII)	
3		
4	Date (mm/dd/yy) (ASCII)	
5		
6	Log-on time (hh/mm/ss) (ASCII)	
7		
8	Elapsed time (hh:mm:ss) (ASCII)	
9		
10	CPU time (hh:mm:ss.tht) (ASCII)	
11		
12		
13	Raw CPU time (Binary)	
14	Origin (1-8 character ASCII)	
15		

Words 0-1 Owner name - The 1-8 character owner name associated with the job.

Words 2-3 Project - The 1-8 character alphanumeric project name/number associated with the job.

Words 4-5 Date - The numeric date associated with the job.

Words 6-7 Log-on time - The numeric time of day on the 24-hour clock the user signed on the system.

Words 8-9 Elapsed time - The total time (24-hour clock) the user was signed on the system.

Words 10-12 CPU time - The total CPU execution time for the job (formatted).

Word 13 Raw CPU time - The actual number of 38.4 microsecond intervals of CPU time for the job (unformatted equivalent of CPU time).

Word 14-15 Origin - The task pseudonym for the accounting session.

2.12 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 which are specified at SYSGEN. Entries in the table are identified by a two-character (ASCII) Timer ID specified via the Create Timer Entry service. Entries may be 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	1	2		4 7	8 16	17 31
Word 0				X	Function	XXXXXXXXXXXXXXX	Timer ID (2 ASCII Chars)
1	Function parameter 1						
2	Function parameter 2						
3	Current time value (negative time units)						
4	Reset time value (negative time units)						

Notes:

WORD 0 Bit 0 Timer Hold bit assigned as follows

= 0, timer not in hold state

= 1, timer in hold state

Bit 1 Free Entry bit assigned as follows

= 0, timer entry is free

= 1, timer entry is taken

Bit 2 Timer Override bit assigned as follows

= 0, no override requested

= 1, reissue activation request

Bit 3 reserved

Bits 4-7 function codes (4 bit numeric value) assigned as follows

= 1, Activate program

= 2, Resume program

= 3, Set bit in operating system or static memory partition

= 4, Reset bit in operating system or static memory partition

= 5, Request interrupt between X'12' to X'7F

Bits 8-16 Reserved

Bits 17-31 Timer ID, any two ASCII characters not matching any other ID in the timer table.

Words 1 and 2 Function Parameters

Function Parameters 1 and 2 for each function are assigned as follows.

Function 1 (Activate task)

Parameter 1,	DQE address of task to be activated upon timeout (the set timer service will pre-activate the task the user requests by 1 to 8 character taskname and then acquire the DQE address, the task will then remain in a suspended state until timeout).
Parameter 2,	Reserved.

Function 2 (Resume Task)

Parameter 1,	DQE address of task to be resumed upon timeout (the set timer service will acquire the DQE address from the user supplied 1 to 8 character taskname or 8 digit tasknumber).
Parameter 2,	Reserved.

Function 3 (Set bit in memory)

Parameter 1,	Address of word where bit is to be set.
Parameter 2,	Bit configuration that will be ored with the word specified in parameter 1 upon timeout.

Function 4 (Reset bit in memory)

Parameter 1,	Address of word where bit is to be reset.
Parameter 2,	Bit configuration that will be anded with the word specified in parameter 1 upon timeout.

Function 5 (Request Interrupt)

Parameter 1,	A request interrupt (RI) instruction for the user specified priority level to be executed upon timeout.
Parameter 2,	Reserved.

Word 3 Current time value.

Current time value is the negative timer units to elapse before the selected function is done. This word is incremented until its value is zero. At that time the selected function is done. If word 4 is zero at timeout this entry will be deleted. If word 4 is non-zero at timeout, the value in word 4 is loaded into word 3 and incremented.

Word 4 Reset time value.

Reset time value is the negative timer units to elapse before the selected function is repeated. This value is placed into word 3 when the timer expires and word 3 is then incremented. Word 4 is not changed until the timer is deleted, or the system is rebooted.

FUNCTION CODE	FUNCTION	FUNCTION PARAMETERS	PROGRAM NO. SET IN TIMER ENTRY?	PROGRAM DEPEN.TIMER BIT SET IN C.DQUE ENTRY	ABORT CASES
1	Activate Program	1 and 2: 1 to 8 ASCII character (left-justified) program name of program to be activated upon timeout.	No	No	None
2	Resume Program	1 and 2: 1 to 8 ASCII character (left-justified) program name of program to be resumed upon timeout.	No	No	None
3	Set Bit(s)	Parameter 1 contains the address to which the bit configuration specified in parameter 2 will be logically "ORed" upon timeout.	Set bit(s) out of program bounds: No	No	Request by non-privileged user is out of program bounds
			Set bits(s) within program bounds: Yes	Yes	
4	Reset Bit(s)	Parameter 1 contains the address to which the bit configuration specified in parameter 2 will be logically "ANDed" upon timeout.	Reset bit(s) out of program bounds: No	No	Request by non-privileged user is out of program bounds
			Reset bit(s) within program bounds: Yes	Yes	
5	Request Interrupt	Parameter 2 is null. Parameter 1 contains an RI for the priority level of the interrupt to be requested upon timeout.	No	No	Request by batch (background) program

3. MODULE DESCRIPTIONS

3.1 Executive (H.EXEC)

Entry Point Summary

Entry Point	Description
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
H.EXEC,5	WAIT FOR ANY NO-WAIT OPERATION COMPLETE
H.EXEC,6	WAIT FOR MEMORY POOL
H.EXEC,7	MEMORY REQUEST PROCESSING COMPLETE
H.EXEC,8	WAIT FOR MEMORY SCHEDULER EVENT
H.EXEC,9	REPORT MEMORY SCHEDULER EVENT
H.EXEC,10	REPORT MEMORY POOL AVAILABLE
H.EXEC,11	COMPLETION OF UNSWAPPABLE I/O REQUEST
H.EXEC,12	NO-WAIT I/O POST PROCESSING COMPLETE
H.EXEC,13	WAIT FOR PERIPHERAL RESOURCE
H.EXEC,14	WAIT FOR DISC FILE SPACE
H.EXEC,15	REPORT PERIPHERAL RESOURCE AVAILABLE
H.EXEC,16	REPORT DISC FILE SPACE AVAILABLE
H.EXEC,17	WAIT FOR FISE
H.EXEC,18	REPORT GATED FISE OPERATION COMPLETE
H.EXEC,19	RESUME EXECUTION OF SPECIFIED TASK
H.EXEC,20	SUSPEND EXECUTION OF CURRENT TASK
H.EXEC,21	SUSPEND EXECUTION OF SPECIFIED TASK
H.EXEC,22	GO TO SPECIFIED TASK CONTEXT (DEBUG)
H.EXEC,23	RUN USER BREAK RECEIVER (DEBUG)
H.EXEC,24	RESTART DEBUG (DEBUG)
H.EXEC,25	WAIT FOR ANY NO-WAIT OPERATION COMPLETE, MESSAGE INTERRUPT OR BREAK INTERRUPT
H.EXEC,26	CONTINUE SPECIFIED TASK
H.EXEC,27	GENERAL ENQUEUE
H.EXEC,28	REPORT RUN REQUEST POST PROCESSING COMPLETE
H.EXEC,29	REPORT WAIT MODE RUN REQUEST STARTING
H.EXEC,30	ENABLE DEBUG MODE BREAK
H.EXEC,31	HOLD CURRENT TASK
H.EXEC,32	HOLD SPECIFIED TASK
H.EXEC,33	DISABLE DEBUG MODE BREAK
H.EXEC,34	REPORT NO-WAIT MESSAGE POST PROCESSING COMPLETE
H.EXEC,35	REPORT WAIT MODE MESSAGE STARTING
H.EXEC,36	GENERAL DEQUE
H.EXEC,37	WAIT FOR MEMORY AVAILABLE
H.EXEC,38	INHIBIT ASYNCHRONOUS ABORT/DELETE
H.EXEC,39	ALLOW ASYNCHRONOUS ABORT/DELETE
H.EXEC,40	END ACTION WAIT

Subroutine Summary

Subroutine	Description
S.EXEC1	INTERACTIVE INPUT COMPLETE
S.EXEC2	TERMINAL OUTPUT COMPLETE
S.EXEC3	WAIT I/O COMPLETE
S.EXEC4	NO-WAIT I/O COMPLETE
S.EXEC5	EXIT FROM INTERRUPT
S.EXEC5A	EXIT FROM TRAP HANDLER WITH ABORT
S.EXEC6	NO-WAIT I/O POST PROCESSING COMPLETE
S.EXEC7	REPORT MEMORY POOL AVAILABLE
S.EXEC8	LINK ENTRY TO QUEUE BY PRIORITY
S.EXEC9	UNLINK ENTRY FROM QUEUE
S.EXEC10	LINK ENTRY TO BOTTOM OF QUEUE
S.EXEC11	LINK ENTRY TO TOP OF QUEUE
S.EXEC12	REPORT MEMORY SCHEDULER EVENT
S.EXEC13	BREAK SPECIFIED TASK
S.EXEC14	RESUME SPECIFIED TASK
S.EXEC15	SUSPEND EXECUTION OF CURRENT TASK
S.EXEC16	SUSPEND EXECUTION OF SPECIFIED TASK
S.EXEC17	ABORT CURRENT TASK
S.EXEC18	ABORT SPECIFIED TASK
S.EXEC19	ABORT TASK PROCESSING CONTROL SUBROUTINE
S.EXEC20	CPU SCHEDULER
S.EXEC21	PROCESS TASK INTERRUPT
S.EXEC22	WAIT FOR COMPLETION OF ALL NO-WAIT OPERATIONS
S.EXEC23	TERMINATE MESSAGES IN RECEIVER QUEUE
S.EXEC24	RESERVED
S.EXEC25	TERMINATE NEXT RUN REQUEST IN RECEIVER QUEUE
S.EXEC26	REMOVE TASK GATING
S.EXEC27	TRANSFER CONTROL TO ABORT RECEIVER
S.EXEC28	DELETE TASK PROCESSING CONTROL SUBROUTINE
S.EXEC29	EXIT TASK PROCESSING CONTROL SUBROUTINE
S.EXEC30	RESERVED
S.EXEC31	REPORT NO-WAIT RUN REQUEST POST PROCESSING COMPLETE
S.EXEC32	REPORT WAIT MODE RUN REQUEST COMPLETE
S.EXEC33	REPORT NO-WAIT MODE RUN REQUEST COMPLETE
S.EXEC34	RESERVED
S.EXEC35	REPORT NO-WAIT MODE MESSAGE POST PROCESSING COMPLETE
S.EXEC36	REPORT WAIT MODE MESSAGE COMPLETE
S.EXEC37	REPORT NO-WAIT MODE MESSAGE COMPLETE
S.EXEC38	INHIBIT SWAP OF CURRENT TASK
S.EXEC39	ENABLE SWAP OF CURRENT TASK
S.EXEC40	RESERVED
S.EXEC41	EXIT RUN RECEIVER
S.EXEC42	EXIT MESSAGE RECEIVER
S.EXEC43	REACTIVATE RUN RECEIVER TASK
S.EXEC44	CHANGE PRIORITY LEVEL OF CURRENT TASK
S.EXEC45	CHANGE PRIORITY LEVEL OF SPECIFIED TASK
S.EXEC46	RESERVED
S.EXEC47	RESERVED

S.EXEC48	CONVERT TASK NUMBER TO DQE ADDRESS
S.EXEC49	CONSTRUCT MRRQ
S.EXEC50	LINK MRRQ TO RUN RECEIVER OF DESTINATION TASK
S.EXEC51	LINK CURRENT TASK TO DESIGNATED WAIT STATE
S.EXEC52	MESSAGE OR RUN REQUEST POST PROCESSING SUBROUTINE
S.EXEC53	VALIDATE PSB
S.EXEC54	MOVE BYTE STRING
S.EXEC55	UNLINK TASK FROM DESIGNATED LIST AND LINK TO READY LIST
S.EXEC56	RESUME MEMORY SCHEDULER
S.EXEC57	LINK TASK TO READY LIST BY PRIORITY
S.EXEC58	LINK MRRQ TO MESSAGE RECEIVER OF DESTINATION TASK
S.EXEC59	RESERVED
S.EXEC60	VALIDATE PRB
S.EXEC61	TRANSFER PARAMETERS FROM MRRQ TO RECEIVER BUFFER
S.EXEC62	VALIDATE RXB
S.EXEC63	TRANSFER RETURN PARAMETERS FROM DESTINATION TASK TO MRRQ
S.EXEC64	NO-WAIT MODE MESSAGE POST PROCESSING SUBROUTINE
S.EXEC65	NO-WAIT MODE RUN REQUEST POST PROCESSNG SUBROUTINE
S.EXEC66	DEALLOCATE MRRQ
S.EXEC67	LINK ENTRY TO END ACTION QUEUE
S.EXEC68	CONSTRUCT AND VECTOR TO USER END ACTION PSD
S.EXEC69	COMMON NO-WAIT POST PROCESSING MERGE POINT
S.EXEC70	TERMINATE ALL RUN REQUESTS IN RECEIVER QUEUE OF CURRENT TASK
S.EXEC71	INSURE STARTUP OF DESTINATION RUN RECEIVER TASK
S.EXEC72	REPORT WAIT I/O STARTING
S.EXEC73	REPLACE CONTEXT ON TSA STACK
S.EXEC74	RESET STACK TO USER LEVEL
S.EXEC75	SITUATIONAL PRIORITY INCREMENT SUBROUTINE
S.EXEC76	UPDATE TASK EXECUTION ACCOUNTING VALUE
S.EXEC77	UPDATE DQE.CQC ON PREEMPTIVE CONTEXT SWITCH
S.EXEC78	MOVE CONTEXT FROM STACK TO T.CONTXT
S.EXEC79	PUSH CURRENT CONTEXT ONTO STACK FOR DEFERRED EA PULL
S.EXEC80	START THE IPU AND VERIFY

3.1.1 Entry Point 1 - Interactive Input Starting

Functional Description

H.EXEC,1 is called to report the beginning of processing for an interactive input request made by the currently executing task. The task will be removed from the associated ready-to-run list, and placed in the wait-for-interactive-input list. A return to the calling routine will be made upon completion of the input request.

Entry Conditions

Calling Sequence:

```
M.SHUT
UEI
M.CALL  H.EXEC,1
```

Registers:

R0 bit 0 1 indicates task is swappable during input processing

Exit Conditions

Return Sequence:

CPU scheduler (when I/O complete, with M.OPEN status)

Registers:

None

3.1.2 Entry Point 2 - Terminal Output Starting

Functional Description

H.EXEC,2 is called to report the beginning of processing for a terminal output request made by the currently executing task. The task will be removed from the associated ready-to-run list, and placed in the wait-for-terminal-output list. A return to the calling routine will be made upon completion of the output request.

Entry Conditions

Calling Sequence:

```
M.SHUT
UEI
M.CALL  H.EXEC,2
```

Registers:

R0 bit 0 1 indicates task is swappable during output processing

Exit Conditions

Return Sequence:

CPU scheduler (when I/O complete, with M.OPEN status)

Registers:

None

3.1.3 Entry Point 3 - Wait I/O Starting

Functional Description

H.EXEC,3 is called to report the beginning of processing for a wait I/O request made by the currently executing task. The task will be removed from the associated ready-to-run list, and placed in the wait-for-I/O list. A return to the calling routine will be made upon completion of the I/O request.

Entry Conditions

Calling Sequence:

```
M.SHUT
UEI
M.CALL  H.EXEC,3
```

Registers:

R0 bit 0   1 indicates task is swappable during I/O processing

Exit Conditions

Return Sequence:

CPU scheduler (when I/O complete, with M.OPEN status)

Registers:

None

3.1.4 Entry Point 4 - No-Wait I/O Starting

Functional Description

H.EXEC,4 is called to report the beginning of processing for a no-wait I/O request made by the currently executing task. A return to the calling routine is made after recording the no-wait I/O start event.

Entry Conditions

Calling Sequence:

```
M.SHUT
UEI
M.CALL  H.EXEC,4
```

Registers:

R0 bit 0   1 indicates task is swappable during I/O processing

Exit Conditions

Return Sequence:

```
M.OPEN
M.RTRN
```

Registers:

None

3.1.5 Entry Point 5 - Wait for Any No-Wait Operation Complete

Functional Description

H.EXEC,5 is functionally identical to H.EXEC,25 except that it does not check for outstanding message or break interrupt requests before placing the task on the ANYW queue. All queued end action requests will be processed before a return is made to the calling routine. The purpose for this special entry point is for IOCS usage when waiting for the completion of a particular no-wait I/O request.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,5
```

Registers:

R6 Zero if indefinite wait, otherwise contains negative number of timer units for timed wait.

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.6 Entry Point 6 - Wait for Memory Pool

Functional Description

H.EXEC,6 is called when the required memory pool space is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-memory-pool list. A return to the calling routine will be made when any memory pool space is deallocated. The calling routine may then make another attempt to allocate the required memory pool space.

Entry Conditions

Calling Sequence:

```
M.CALL H.EXEC,6
```

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.7 Entry Point 7 - Memory Request Processing Complete

Functional Description

H.EXEC,7 is called by the memory scheduler when processing for a memory request is complete. The DQE associated with the memory request will have been unlinked from the memory request queue by the memory scheduler. The completed memory request will be processed by H.EXEC,7 according to request type. The request type information is contained in the DQE. The task will then be linked into the appropriate ready-to-run list. A return to the memory scheduler will be made by issuing a M.RTRN.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,7
```

Registers:

R2 DQE address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.8 Entry Point 8 - Wait for Memory Scheduler Event

Functional Description

H.EXEC,8 is called by the memory scheduler when either no additional processing of outstanding memory requests is possible, or the memory request list is empty. H.EXEC,8 examines C.RRUN. If C.RRUN is not equal to zero, and the memory request queue is not empty, the memory scheduler will be re-executed. Otherwise, the memory scheduler will be removed from the ready-to-run list and placed in the wait-for-memory-event list. A return to the memory scheduler will occur when:

(a) A new memory request is queued, or

(b) The memory request queue is not empty and the status of allocated memory changes such that it either is deallocated or becomes more eligible for swapping.

Entry Conditions

Calling Sequence:

M.CALL H.EXEC,8

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.9 Entry Point 9 - Report Memory Scheduler Event

Functional Description

H.EXEC,9 is called when the status of allocated memory changes such that it is either deallocated, or becomes more eligible for swapping. The purpose of this routine is to insure the appropriate execution of the memory scheduler task. If the memory-request list is empty, no additional processing is required and a return is made to the user. If the memory-request list is not empty, C.RRUN is incremented, and the memory scheduler state is checked. If the memory scheduler is in the wait-for-memory-event list, it will be removed from that list, and placed in the ready-to-run list at the priority of the highest priority entry in the memory-request list. A return will then be made to the calling routine.

Entry Conditions

Calling Sequence:

```
        M.CALL  H.EXEC,9
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

3.1.10 Entry Point 10 - Report Memory Pool Available

Functional Description

H.EXEC,10 is called when memory pool space is deallocated. The purpose of this routine is to resume the execution of all tasks in the wait-for-memory-pool list. If the wait-for-memory-pool list is empty, no additional processing is required and a return is made to the calling routine. Otherwise, each entry in the list is removed and placed in its associated ready-to-run list. It is expected that when these tasks resume execution, they will reissue the request for the required memory pool space. When all entries have been flushed from the wait-for-memory-pool list, a return will be made to the calling routine.

Entry Conditions

Calling Sequence:

```
        M.CALL  H.EXEC,10
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

3.1.11 Entry Point 11 - Completion of Unswappable I/O Request

Functional Description

H.EXEC,11 is called by the IOCS post transfer processing logic, executing on behalf of the current task. The count of unswappable I/O transfers in the DQE is decremented. If no other swap inhibit reasons exist, a call will be made to H.EXEC,9 to report the memory scheduler event. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,11
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.12 Entry Point 12 - No-Wait I/O Post Processing Complete

Functional Description

H.EXEC,12 is called by the IOCS no-wait I/O post processing logic to exit from the task interrupt state. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level (the most recent) of pushdown in the TSA stack, then issue a M.RTRN to return to the point of task interrupt.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,12
```

Registers:

None

Exit Conditions

Return Sequence:

None

Registers:

None

3.1.13 Entry Point 13 - Wait for Peripheral Resource

Functional Description

H.EXEC,13 is called when the required peripheral resource is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-peripheral-resource list. A return to the calling routine will be made when the specified peripheral is deallocated by its current user. The calling routine may then make another attempt to allocate the device.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,13
```

Registers:

R6		Peripheral requirements specification
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address
R7		Requirements mask
	X'00FF0000'	Any device of this device type code
	X'00FFFF00'	Any device of the specified type code, on the specified channel
	X'00FFFFFF'	The specific device described by this type code, channel address, and subchannel address

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.14 Entry Point 14 - Wait for Disc File Space

Functional Description

H.EXEC,14 is called when the required disc file space is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-disc list. A return to the calling routine will be made when any disc file space is deallocated. The calling routine may then make another attempt to allocate the required disc file space.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,14
```

Registers:

R6		Disc device requirements specification
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address
R7		Disc device requirements mask
	X'00000000'	Any disc
	X'00FF0000'	Any disc of the specified type code
	X'00FFFF00'	Any disc of the specified type code on the specified channel
	X'00FFFFFF'	The specific disc device described by this type code, channel address, and subchannel address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.13 Entry Point 13 - Wait for Peripheral Resource

Functional Description

H.EXEC,13 is called when the required peripheral resource is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-peripheral-resource list. A return to the calling routine will be made when the specified peripheral is deallocated by its current user. The calling routine may then make another attempt to allocate the device.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,13
```

Registers:

R6		Peripheral requirements specification
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address
R7		Requirements mask
	X'00FF0000'	Any device of this device type code
	X'00FFFF00'	Any device of the specified type code, on the specified channel
	X'00FFFFFF'	The specific device described by this type code, channel address, and subchannel address

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.14 Entry Point 14 - Wait for Disc File Space

Functional Description

H.EXEC,14 is called when the required disc file space is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-disc list. A return to the calling routine will be made when any disc file space is deallocated. The calling routine may then make another attempt to allocate the required disc file space.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,14
```

Registers:

R6		Disc device requirements specification
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address
R7		Disc device requirements mask
	X'00000000'	Any disc
	X'00FF0000'	Any disc of the specified type code
	X'00FFFF00'	Any disc of the specified type code on the specified channel
	X'00FFFFFF'	The specific disc device described by this type code, channel address, and subchannel address

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.15 Entry Point 15 - Report Peripheral Resource Available

Functional Description

H.EXEC,15 is called when a peripheral device is deallocated. The purpose of this routine is to resume the execution of the tasks in the wait-for-peripheral-resource list, which have specified requirements that will be satisfied by the deallocated device. If no such tasks exist, no additional processing is required and a return is made to the calling routine. Otherwise, each such entry in the list is removed and placed in its associated ready-to-run list. It is expected that when these tasks resume execution, they will reissue the request for the required device. When all appropriate entries have been flushed from the wait-for-peripheral resource list, a return will be made to the calling routine.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,15
```

Registers:

R6		Peripheral resource definition
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.16 Entry Point 16 - Report Disc File Space Available

Functional Description

H.EXEC,16 is called when disc space is deallocated. The purpose of this routine is to resume the execution of the tasks in the wait-for-disc list which have specified requirements that may be satisfied by the deallocated disc file space. If no such tasks exist, no additional processing is required and a return is made to the calling routine. Otherwise, each such entry in the list is removed and placed in its associated ready-to-run list. It is expected that when these tasks resume execution, they will reissue the request for the required space. When all appropriate entries have been flushed from the list, a return will be made to the calling routine.

Entry Conditions

Calling Sequence:

```
          M.CALL  H.EXEC,16
```

Registers:

R6		Disc device resource definition
	bits 0-7	Reserved
	bits 8-15	Device type code
	bits 16-23	Channel address
	bits 24-31	Subchannel address

Exit Conditions

Return Sequence:

```
          M.RTRN
```

Registers:

None

3.1.17 Entry Point 17 - Wait for FISE

Functional Description

H.EXEC,17 is called by the File System Executive (FISE) on behalf of the currently executing task. FISE will call H.EXEC,17 when a task has requested a File System Executive service to be performed, and FISE is busy performing a gated operation on behalf of another task. The currently executing task will be removed from the associated ready-to-run list, and placed in the wait-for-FISE list. A return to the calling routine will be made when FISE is no longer gated, and a call to H.EXEC,18 has been made to flush the wait-for-FISE list.

Entry Conditions

Calling Sequence:

```
        M.CALL  H.EXEC,17
```

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler

Registers:

None

3.1.18 Entry Point 18 - Report Gated FISE Operation Complete

Functional Description

H.EXEC,18 is called by the File System Executive (FISE) upon completion of a gated FISE operation. The purpose of this routine is to resume the execution of the highest priority task in the wait-for-FISE list. If the wait-for-FISE list is empty, no additional processing is required and a return is made to the calling routine. Otherwise the first entry is removed and placed in its associated ready-to-run list. A return to the calling routine is issued after the list relinkage is complete.

Entry Conditions

Calling Sequence:

```
        M.CALL  H.EXEC,18
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

3.1.19 Entry Point 19 - Resume Execution of Specified Task

Functional Description

H.EXEC,19 is called to resume execution of the specified task. This routine will in turn call S.EXEC14 to accomplish the resume function. A return will then be made to the calling routine.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,19
```

Registers:

R2	DQE address of task to be resumed

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.20 Entry Point 20 - Suspend Execution of Current Task

Functional Description

H.EXEC,20 is called to suspend execution of the current task, either for an indefinite period, or for the specified number of time units. The specified time (if any) is stored as a one shot timer in the DQE along with a resume-program timer function code. S.EXEC15 is then called to suspend execution of the current task. A return will not be made until the timer expires or until the task is resumed.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,20
```

Registers:

R6	Zero if indefinite suspend, otherwise contains negative number of timer units for timed suspend

Exit Conditions

Return Sequence:

M.RTRN (on time out or resume)

Registers:

None

3.1.21 Entry Point 21 - Suspend Execution of Specified Task

Functional Description

H.EXEC,21 is called to suspend execution of the specified task, either for an indefinite period or for the specified number of time units. The specified time (if any) is stored as a one-shot timer in the DQE of the specified task, along with a resume-program timer function code. S.EXEC16 is then called to suspend execution of the specified task. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,21
```

Registers:

R2	DQE of task to be suspended
R6	Zero if indefinite suspend, otherwise contains negative number of timer units for timed suspend

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.22 Entry Point 22 - Go To Specified Task Context (DEBUG)

Functional Description

H.EXEC,22 is called by DEBUG to either begin or continue processing of the task being debugged. The execution context (registers and PSD) are contained in a parameter block associated with the call. The DEBUG mode is reset and control is passed to the specified user context, by pushing the context onto the TSA stack and invoking the CPU scheduler.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,22
```

Registers:

R1 Address of context block where:

word 0-7=registers 0-7;
word 8-9=PSD.

The context block must be on a file (8W) address boundary.

Exit Conditions

Control will be passed to the specified context. DEBUG will not be re-entered until a TRAP, break, or abort is encountered.

3.1.23 Entry Point 23 - Run User Break Receiver (DEBUG)

Functional Description

H.EXEC,23 is called by DEBUG to initiate execution of the user break receiver. The contents of T.CONTXT are pushed onto the TSA stack. The DEBUG mode is reset. The user break request flag is set, and control is passed to the CPU scheduler.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,23
```

Registers:

None

Exit Conditions:

Control will be passed to the user break receiver by the CPU scheduler. DEBUG will not be re-entered until a TRAP, break, break exit, or abort is encountered.

3.1.24 Entry Point 24 - Restart DEBUG (DEBUG)

Functional Description

H.EXEC,24 is called by DEBUG to restart DEBUG execution. All outstanding I/O requests are terminated; any outstanding messages are discarded; the stack is cleared, and control is passed to DEBUG entry point 2.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,24
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTNA
```

Registers:

None

3.1.25 Entry Point 25 - Wait for Any No-Wait Operation Complete, Message Interrupt or Break Interrupt

Functional Description

H.EXEC,25 is called to place the current task in a wait-state, waiting for the completion of any no-wait mode I/O request, no-wait mode message request, no-wait mode run request, or the receipt of a message or break interrupt. The wait state may be either indefinite in length, or may have an associated time-out value. A return will not be made until one of the wait conditions is satisfied, or until expiration of the time-out value.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,25
```

Registers:

R6	Zero if indefinite suspend, otherwise contains negative number of timer units for timed suspend

Exit Conditions

Return Sequence:

M.RTRN (on time out or satisfaction of wait condition)

Registers:

None

3.1.26 Entry Point 26 - Continue Specified Task

Functional Description

H.EXEC,26 is called to continue a task that is in the "hold" wait state. The DQE of the specified task is unlinked from the hold-state queue and linked to the ready-to-run queue. Note: if the task is not in the hold-state, the hold request flag in the DQE will be reset. A return to the calling routine will then be made.

Entry Conditions

Return Sequence:

M.RTRN

Registers:

R1	DQE address of task to be continued

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.27 Entry Point 27 - General Enqueue

Functional Description

H.EXEC,27 is called to place the current task in the general wait queue (C.SWGQ). The task will remain on the general wait queue until the optional watchdog timer expires, or unitl a corresponding general dequeue call (to H.EXEC,36) is made, whichever occurs first.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,27
```

Registers:

R4		Zero if indefinite wait, otherwise contains negative number of timer units for timed wait.
R5	Bit 0	Zero if normal (not-ready based) swapping, one if the task is to be swapped only by a higher priority task.
	Bits 1-23	Unused
	Bits 24-31	Function code (0-255)
R6,R7		Enqueue ID

Exit Conditions

Return Sequence:

M.RTRN (on timer expiration or Dequeue call with corresponding function code and ID - with M.OPEN in effect)

Note: Swap on priority restriction removed before M.RTRN

Registers:

R3	Zero if wait state terminated by corresponding Deque call, one if wait state time-out.

3.1.28 Entry Point 28 - Report Run Request Post Processing Complete

Functional Description

H.EXEC,28 is called by the run-request post processing logic to exit from the end action interrupt state. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level of pushdown in the TSA stack. A M.RTRN will then be issued to return to the point of task interrupt (or the point following the M.ANYW call).

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,28
```

Registers:

None

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.29 Entry Point 29 - Report Wait Mode Run Request Starting

Functional Description

H.EXEC,29 is called to report the beginning of processing for a wait mode run request issued by the currently executing task. The task will be removed from the associated ready-to-run list, and placed in the wait-for-run-complete list. A return to the calling routine will be made upon completion of the run request by the destination task.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,29
```

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler (when run request complete)

Registers:

None

3.1.30 Entry Point 30 - Enable DEBUG Mode Break

Functional Description

H.EXEC,30 is called by the DEBUG program to allow a break while the task is in DEBUG mode. It is used in conjunction with H.EXEC,33 (Disable DEBUG mode break).

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,30
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.31 Entry Point 31 - Hold Current Task

Functional Description

H.EXEC,31 is called to remove the current task from execution and place it in a hold state. The task will not continue execution until a continue request is issued to H.EXEC,26.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,31
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN (only after continue request)
```

Registers:

None

3.1.32 Entry Point 32 - Hold Specified Task

Functional Description

H.EXEC,32 is called to place the specified task in a hold state. The hold request system action interrupt flag is set in the DQE of the specified task. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,32
```

Registers:

R1	DQE address of task to be placed in hold state

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.33 Entry Point 33 - Disable DEBUG Mode Break

Functional Description

H.EXEC,33 is called by the DEBUG program to disable a break while the task is in DEBUG mode. Note: normally DEBUG mode break is not enabled. This routine is provided for use in conjunction with H.EXEC,30 (Enable DEBUG mode break).

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,33
```

Registers:

None

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.34 Entry Point 34 - Report No-Wait Message Post Processing Complete

Functional Description

H.EXEC,34 is called by the message request post processing logic to exit from the end action interrupt state. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level of pushdown in the TSA stack. A M.RTRN will then be issued to return to the point of task interrupt (or to the point following the M.ANYW call).

Entry Conditions

Calling Sequence:

```
M.CALL  H.EXEC,34
```

Registers:

None

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.35 Entry Point 35 - Report Wait Mode Message Starting

Functional Description

H.EXEC,35 is called to report the beginning of processing for a wait mode message request issued by the currently executing task. The task will be removed from the associated ready to run list, and placed in the wait-for-message-complete list. A return to the calling routine will be made upon completion of message processing by the destination task.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,35
```

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler (when message request complete)

Registers:

None

3.1.36 Entry Point 36 - General Dequeue

Functional Description

H.EXEC,36 is called to release the highest priority task queued for the specified function code and Enqueue ID. If none exist, the request is ignored.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,36
```

Registers:

R5	Function code (0-255)
R6,R7	Enqueue ID

Exit Conditions

Return Sequence:

M.RTRN (with M.SHUT in effect)

Registers:

R2	contains program number of dequed task, or zero if none dequed

3.1.37 Entry Point 37 - Wait for Memory Available

Functional Description

H.EXEC,37 is called when the required memory space is not available. The currently executing task will be removed from the associated ready-to-run list, and placed in the memory request list. A return to the calling routine will be made when the memory request has been satisfied.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,37
```

Registers:

R7 memory request definition word:

byte 0 = Memory request type:

- 0 = Inswap task
- 1 = Preactivation request
- 2 = Activation request
- 3 = Memory expansion request
- 4 = IOCS buffer request
- 5 = Shared memory request; bytes 2 and 3 will equal the address of the requested user SMT

byte 1 = Type of memory required:

- 1 = Class 'E' memory
- 2 = Class 'H' memory
- 3 = Class 'S' memory

byte 2 = Map register to be used (if byte 0 is not equal to 5):

Computed by subtracting the contents of C.MSD from the map register number (0-31).

byte 3 = Number of memory blocks required (if byte 0 is not equal to 5)

Exit Conditions

Return Sequence:

CPU scheduler (when memory is allocated)

Registers:

None

3.1.38 Entry Point 38 - Inhibit Asynchronous Abort/Delete

Functional Description

H.EXEC,38 is called to inhibit an asynchronously requested task abort or task delete. This entry point is used for gating purposes and is called when a program sequence is started that must be completed in order to maintain system integrity. Any asynchronous abort or delete requests received while abort/delete is inhibited will be deferred until the system critical sequence is complete, and a call is made to H.EXEC,39 to remove the inhibit status.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,38
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.1.39 Entry Point 39 - Allow Asynchronous Abort/Delete

Functional Description

H.EXEC,39 is called at the conclusion of a system critical program sequence, to remove the asynchronous abort/delete inhibit state previously invoked by a call to H.EXEC,38. Any deferred abort or delete requests will be processed.

Entry Conditions

Calling Sequence:

```
M.CALL   H.EXEC,39
```

Registers:

None

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.1.40 Entry Point 40 - End Action Wait

See Section 8.2.17 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.1.41 Subroutine S.EXEC1 - Interactive Input Complete

Functional Description

S.EXEC1 is called by the appropriate I/O handler from the interrupt service routine. Its purpose is to report the completion of processing for an interactive input request. The associated task is removed from the wait-for-interactive-input list and linked to the ready-to-run list (or to the memory-request list if an inswap is required).

Entry Conditions

Calling Sequence:

```
BL            S.EXEC1
```

Registers:

R1	DQE entry number

Exit Conditions

Return Sequence:

```
TRSW          R0
```

Registers:

None returned

None saved

3.1.42 Subroutine S.EXEC2 - Terminal Output Complete

Functional Description

S.EXEC2 is called by the appropriate I/O handler from the interrupt service routine. Its purpose is to report the completion of processing for a terminal output request. The associated task is removed from the wait-for-terminal-output list and linked to the ready-to-run list (or to the memory-request list if an inswap is required).

Entry Conditions

Calling Sequence:

BL	S.EXEC2

Registers:

R1	DQE entry number

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

None returned
None saved

3.1.43 Subroutine S.EXEC3 - Wait I/O Complete

Functional Description

S.EXEC3 is called by the appropriate I/O handler from the interrupt service routine. Its purpose is to report the completion of processing for a wait I/O request. The associated task is removed from the wait I/O list and linked to the ready-to-run list (or to the memory-request list if an inswap is required).

Entry Conditions

Calling Sequence:

BL	S.EXEC3

Registers:

R1	DQE entry number

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R6	Undisturbed

3.1.44 Subroutine S.EXEC4 - No-Wait I/O Complete

Functional Description

S.EXEC4 is called by the appropriate I/O handler from the interrupt service routine. Its purpose is to report the completion of processing for a no-wait I/O request. The associated task may be in the wait-for-any I/O list. If so, it will be removed from that list and linked to the ready-to-run list (or to the memory request list if an inswap is required).

The I/O queue entry will be linked to the DQE task interrupt list and will contain the no-wait I/O post processing service address. When the scheduler dispatches CPU control to this task, the specified routine will be entered as a preemptive system service. Preemptive system services take precedence over execution of the task, but do not take precedence over system services being executed on behalf of the task.

Entry Conditions

Calling Sequence:

BL	S.EXEC4

Registers:

R6	I/O queue entry address

Note: The first eight words of the I/O queue entry must be in the preemptive system service list entry header format.

R1	DQE entry number

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R6	Undisturbed

3.1.45 Subroutine S.EXEC5 - Exit from Interrupt

Functional Description

S.EXEC5 is called as an exit service by all interrupt service routines. Its purpose is to allow CPU scheduling based on events which may have occurred at an interrupt level. If lower level interrupts are active, processing will continue at the last (highest) interrupted level. If no interrupts are active, the CPU scheduler is entered.

Entry Conditions

Calling Sequence:

```
BEI
DAI/DACI (for associated level)
BL              S.EXEC5
```

Registers:

R6,R7 PSD from interrupted environment

X2 Address of register save block containing registers from interrupted environment

Exit Conditions

Return Sequence:

LPSD (or) CPU scheduler

Registers:

None

3.1.46 Subroutine S.EXEC5A - Exit From Trap Handler With Abort

Functional Description

S.EXEC5A is called as an exit service from the system error trap handlers: non-present memory, undefined instruction, privilege error, address exception (CONCEPT/32 only); cache memory parity (32/87 only), and map fault. Its purpose is to request that the current task be aborted and transfer execution back to the CPU scheduler, S.EXEC20. If lower levels of interrupt are active, a system KILL will be executed.

Entry Conditions

Calling Sequence:

```
BEI
BL              S.EXEC5A
```

Registers:

R2 address of register save area

R5 abort code

R6,R7 PSD from interrupt environment

Exit Conditions

Return Sequence:

CPU scheduler or M.KILL

Registers:

None

3.1.47 Subroutine S.EXEC6 - No-Wait I/O Post Processing Complete

Functional Description

S.EXEC6 is called to report the completion of no-wait I/O post processing. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level (the most recent) of pushdown in the TSA stack. A M.RTRN will then be issued to return to the point of task interrupt.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC6
```

Registers:

None

Exit Conditions

Return Sequence:

```
BU          S.EXEC20
```

Registers:

None

3.1.48 Subroutine S.EXEC7 - Report Memory Pool Available

Functional Description

S.EXEC7 is called when memory pool space is deallocated. The purpose of this subroutine is to resume the execution of all tasks in the wait-for-memory-pool list. If the wait-for-memory-pool list is empty, no additional processing is required and a return is made to the calling routine. Otherwise, each entry in the list is removed and placed in its associated ready-to-run list. It is expected that when these tasks resume execution, they will reissue the request for the required memory pool space. When all entries have been flushed from the wait-for-memory-pool list, a return will be made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC7

Registers:

None

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R5,R6	Saved
R1,R2,R3,R4,R7	Destroyed

3.1.49 Subroutine S.EXEC8 - Link Entry To Queue By Priority

Functional Description

S.EXEC8 is a register-reentrant subroutine, callable from either a software or interrupt priority level. Its purpose is to link an entry into the list associated with the designated head cell, by priority. This routine assumes that a standard head cell and entry header format are used. After the specified linkage is performed, a return is made to the calling program..

Entry Conditions

Calling Sequence:

(Gate as appropriate)

BL	S.EXEC8

Registers:

R1	Head cell address
R2	Address of entry to be linked

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2,R4,R6,R7	Saved
R1,R3,R5	Destroyed

3.1.50 Subroutine S.EXEC9 - Unlink Entry From Queue

Functional Description

S.EXEC9 is a register-reentrant subroutine, callable from either a software or interrupt priority level. Its purpose is to unlink the specified entry from the list associated with the designated head cell. This routine assumes that a standard head cell and entry header format are used. After the entry is unlinked, a return is made to the calling program.

Entry Conditions

Calling Sequence:

```
        (Gating as appropriate)

        BL              S.EXEC9
```

Registers:

R1	Head cell address
R2	Address of entry to be linked

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

R2,R4,R5,R6,R7	Saved
R1,R3	Destroyed

3.1.51 Subroutine S.EXEC10 - Link Entry To Bottom Of Queue

Functional Description

S.EXEC10 is a register-reentrant subroutine, callable from either a software or interrupt priority level. Its purpose is to link an entry to the bottom of the list associated with the specified head cell. This routine assumes that a standard head cell and entry header format are used. After the specified linkage is performed, a return is made to the calling program.

Entry Conditions

Calling Sequence:

(Gate as appropriate)

BL	S.EXEC10

Registers:

R1	Head cell address
R2	Address of entry to be linked

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2,R4,R5,R6,R7	Saved
R3	Destroyed

3.1.52 Subroutine S.EXEC11 - Link Entry To Top Of Queue

Functional Description

S.EXEC11 is a register-reentrant subroutine, callable from either a software or interrupt priority level. Its purpose is to link an entry to the top of the list associated with the specified head cell. This routine assumes that a standard head cell and entry header format are used. After the specified linkage is performed, a return is made to the calling program.

Entry Conditions

Calling Sequence:

(Gate as appropriate)

BL	S.EXEC11

Registers:

R1	Head cell address
R2	Address of entry to be linked

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2,R4,R5,R6,R7	Saved
R3	Destroyed

Standard Linked List Head Cell Format

WORD	0 … 7	8 … 15	16 … 23	24 … 31
0		STRING FORWARD ADDRESS (1)		
1		STRING BACKWARD ADDRESS (2)		
2	PRIORITY (3)	COUNT (4)	RESERVED	

Notes

(1) The string forward address is a 1W field which points to the first entry in the top-to-bottom chain. When the list is empty, it contains the address of the head cell.

(2) The string backward address is a 1W field which points to the first entry in the bottom-to-top chain. When the list is empty, it contains the address of the head cell.

(3) The head cell priority is a 1B field which contains a dummy head cell priority which is always zero.

(4) The count value is a 1B field which contains the number of entries in the list. This value is incremented/decremented as required by subroutines S.EXEC8 through S.EXEC11.

Standard Linked List Entry Header Format

WORD	0 7	8 15 16 23 24 31
0		STRING FORWARD ADDRESS (1)
1		STRING BACKWARD ADDRESS (2)
2	PRIORITY (3)	AVAILABLE FOR USE AS DEFINED BY ENTRY FORMAT

Notes

(1) The string forward address is a 1W field which points to the next entry in the top-to-bottom chain. If this is the last entry in the top-to-bottom chain, the string forward address will be the address of the head cell.

(2) The string backward address is a 1W field which points to the next entry in the bottom-to-top chain. If this is the last entry in the chain, the string backward address will be the address of the head cell.

(3) The priority field is a 1B field containing the priority of this entry. The acceptable range of this value is 1-255. Note that priority zero is reserved for use as a dummy priority by the head cell.

Misc. The last entry in the top-to-bottom chain is the first entry in the bottom-to-top chain. The last entry in the bottom-to-top chain is the first entry in the top-to-bottom chain.

3.1.53 Subroutine S.EXEC12 - Report Memory Scheduler Event

Functional Description

S.EXEC12 is called when the status of allocated memory changes such that it is either deallocated or becomes more eligible for swapping. The purpose of this subroutine is to insure the appropriate execution of the memory scheduler task. If the memory-request list is empty, no additional processing is required and a return is made to the user. If the memory-request list is not empty, C.RRUN is incremented, and the memory scheduler state is checked. If the memory scheduler is in the wait-for-memory-event list, it will be removed from that list and placed in the ready-to-run list at the priority of the highest priority entry in the memory-request list. A return will then be made to the calling routine.

Entry Conditions

Calling Sequence

```
BL          S.EXEC12
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW            R0
```

Registers:

R3,R6	Saved
R1,R2,R4,R5,R7	Destroyed

3.1.54 Subroutine S.EXEC13 - Break Specified Task

Functional Description

S.EXEC13 is called by the appropriate I/O handler from the interrupt service routine, or by the M.INT monitor service. It's purpose is to set the DEBUG-Break requested flag in the DQE if DEBUG is associated with the task and if the DEBUG mode active task interrupt flag is not already set. If DEBUG is not associated with the task, but a user break receiver has been established, the user-break requested flag will be set in the DQE. If DEBUG is not associated with the task, and no user break receiver has been established, the break is ignored. A return to the calling routine is made upon completion of processing.

Entry Conditions

Calling Sequence:

```
BL              S.EXEC13
```

Registers:

R2	DQE address of task to receive break
R3	Address of 2W scratchpad area (doubleword bounded)

Exit Conditions

Return Sequence:

```
TRSW            R0
```

Registers:

X3=X3-4W (scratchpad pointer)

R1,R2,R4	Saved
R5,R6,R7	Destroyed

3.1.55 Subroutine S.EXEC14 - Resume Specified Task

Functional Description

S.EXEC14 may be called either from an interrupt service routine or from a system service operating on behalf of a task. It's purpose is to resume the execution of a suspended task. If the specified task is not in a suspended state, no action will be taken. If the specified task is suspended and outswapped, it will be unlinked from the suspended list, and linked to the memory request list. Otherwise, it will be unlinked from the suspended list, and linked to the ready-to-run list at its current priority.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC14
```

Registers:

R2 DQE address of task to be resumed

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

None returned

R2	Saved
R1,R3,R4,R5,R6,R7	Destroyed

3.1.56 Subroutine S.EXEC15 - Suspend Execution of Current Task

Functional Description

S.EXEC15 is called to suspend execution of the current task. The DQE for the current task is unlinked from the ready-to-run list, and linked to the suspended list. Control is then transferred to the CPU scheduler to select the next task for execution.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC15
```

Registers:

R6 Zero if indefinite suspend, otherwise contains negative timer units

Exit Conditions

Branch to CPU Scheduler;

When resumed, the task will continue operation at the most recent context in the pushdown stack.

3.1.57 Subroutine S.EXEC16 - Suspend Execution Of Specified Task

Functional Description

S.EXEC16 is called to suspend execution of the specified task. The DQE is marked to indicate that an asynchronous suspend has been requested. The suspend request will be processed on behalf of the task being suspended, when the CPU scheduler selects that task for execution.

Entry Conditions

Calling Sequence:

BL	S.EXEC16

Registers:

R2	DQE address of task to be suspended
R6	Zero if indefinite suspend, otherwise contains negative timer units

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

None returned

R1,R2,R3	Saved
R4,R5,R6,R7	Destroyed

3.1.58 Subroutine S.EXEC17 - Abort Current Task

Functional Description

S.EXEC17 is called either from a system service, or from a system trap level. Its purpose is to store the abort code and set the abort requested bit in the DQE. It will then reset the TSA stack to a level routine.

If the task has an abort receiver established, the DQE.ABRA flag will be set and a return will be made to the calling routine. If no abort receiver is established, the DQE.ABRT flag will be set and the task will be marked as leaving the system (by setting the DQE.TLVS flag). A return will then be made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC17

Registers:

R5	Abort code characters 1-4
R6,R7	Abort code characters 5-12

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	DQE address of current task
R4,R7	Saved
R1,R3,R5,R6	Destroyed

3.1.59 Subroutine S.EXEC18 - Abort Specified Task

Functional Description

S.EXEC18 is called from a system service. Its purpose is to store the abort code and set the abort requested (asynchronous) bit in the DQE. It will then return to the calling routine. The abort requested bit will not be examined until the scheduler selects this task for execution.

If the specified task is in the SUSP, ANYW, or RUNW list, it will be unlinked from the wait list and linked to the ready to run list. If the task does not have an abort receiver established, DQE.TLVS and DQE.ABRT will be set. If an abort receiver has been established, only DQE.ABRA will be set.

Entry Conditions

Calling Sequence:

BL	S.EXEC18

Registers:

R1	DQE address of task to be aborted
R5	Abort code characters 1-4
R6,R7	Abort code characters 5-12

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	DQE address of task to be aborted
R4	Saved
R1,R3,R5,R6,R7	Destroyed

3.1.60 Subroutine S.EXEC19 - Abort Task Processing Control Subroutine

Functional Description

S.EXEC19 is an internal H.EXEC subroutine which is called only by S.EXEC21 to process an abort request on behalf of the currently executing task. The purpose of S.EXEC19 is to control the sequencing of abort processing by calling abort processing modules associated with the pertinent subsystems:

Module	Description
S.EXEC26	Remove context switch or FISE gating if any.
S.EXEC23	Terminate all messages in receiver queue.
S.EXEC22	Defer continued processing until all outstanding no-wait operations are complete.
S.EXEC25	Terminate active run request (if any).
S.EXEC43	Reactivate task if additional run requests queued.
S.EXEC27	Transfer control to abort receiver if any.
S.MONS2	Remove task associated timers.
H.MONS,38	Disconnect interrupt (if interrupt connected).
S.EXEC76	Update execution accounting value.
[H.TSM,4]	TSM task abort/delete processing (called only if TSM task).
[H.SOUT,26]	Batch task abort processing (called only if Batch task).

Module	Description
H.ALOC,3	Close and deallocate system critical files.
	Close and deallocate user I/O files/devices with blocking buffer purge and automatic EOF as appropriate.
	Deallocate all swap files and memory (excluding TSA).
	Deallocate TSA memory, DQE space, clear C.PRNO, transfer control to the CPU scheduler.

Entry Conditions

Calling Sequence:

BU	S.EXEC19

Registers:

None

Exit Conditions

Return Sequence:

Clear	C.PRNO
BU	S.EXEC20

Registers:

None

3.1.61 Subroutine S.EXEC20 - CPU Scheduler

Functional Description

S.EXEC20 is an internal H.EXEC subroutine. It is called by S.EXEC5 when all outstanding interrupts or traps have been exited, or by M.RTRN/M.RTNA with the return context on the TSA stack. Its purpose is to check for a ready-to-run task that is higher in priority than the currently executing task. This check is quickly made because the linkage of any task to the ready-to-run queue will cause a priority comparison between that task and the currently executing task. If the newly ready to run task is of higher priority, an indicator is set for S.EXEC20. S.EXEC20 will either return to the context of the current task, process a task interrupt on behalf of the current task, or select a higher priority task for execution.

Entry Conditions

Calling Sequence:

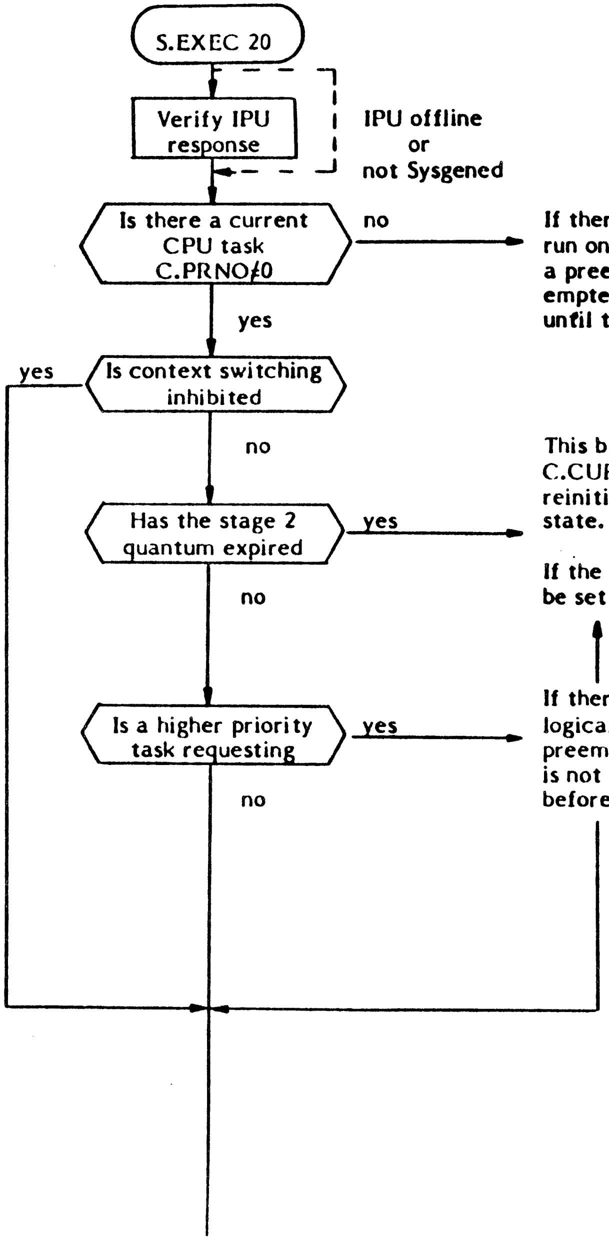
```
        BU              S.EXEC20
```

Registers:

None

Exit Conditions

Dispatch of CPU control


S.EXEC 20
Verify IPU response
IPU offline or not Sysgened
Is there a current CPU task C.PRNO≠0
no
yes
Is context switching inhibited
yes
no
Has the stage 2 quantum expired
yes
no
Is a higher priority task requesting
yes
no

If there are no current tasks, one of three actions will occur: (1) if there are R/T tasks ready to run one will be selected by SX20.5; (2) if there are no R/T tasks, C.CURR will be examined for a preempted T/D task and if found, it's context will be reestablished; (3) if there is no preempted task, SX20.5 will be allowed to search for a T/D task and select one or go into CPU wait until the next interrupt.

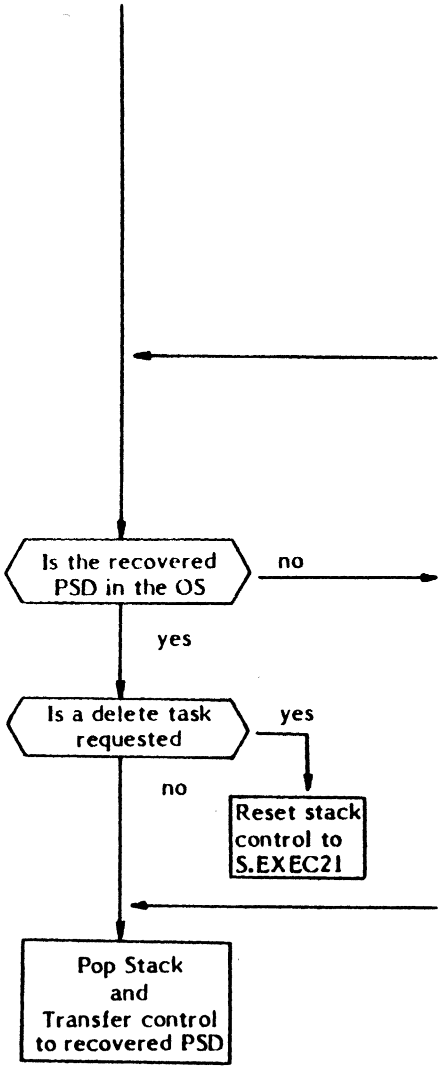
This block of logic does the preparation and relinking of the current CPU task from the C.CURR to its ready-to-run state. The interval time is collected and stored, the quantum flags reinitialized, and the priorities reset. A branch to S.EXEC55 links the current task to a ready state.

If the current CPU task is executing an OS service on behalf of the IPU, the IPU inhibit flag will be set to prevent it from being selected for IPU execution before it has finished in the OS.

If there is a higher priority task requesting and context switching is not inhibited, there are two logical paths. If there is a real time task requesting, the current time distributed task will be preempted and the real time task selected as the current CPU task. If the higher priority task is not real time, the current T/D task will be permitted to complete its allotted rotation time before task reselection occurs.

If there is an IPU online and it is idle, a branch link will be executed to a subroutine of H.CPU called SCHED. SCHED scans the ready queues and selects the highest priority ready-to-run task which is eligible for the IPU. The task will be linked to the IPU head cell and the IPU started. If the IPU is off line, an unconditional branch will direct execution around the IPU test and branch code in S.EXEC20.

The state chain queues are searched from the highest priority (SQRT) to the lowest (SQ64). When an entry is found (i.e., state chain where the head cell count > 1) the first entry in the chain is unlinked and relinked to the current head cell (C.CURR). The context of the communication region is established. The CPU is then remapped for the new current task; the map and protect registers are loaded into the CPU scratchpad. Finally, the interval is read, reset, loaded, and started.


Is the recovered PSD in the OS
no
yes
Is a delete task requested
yes
no
Reset stack control to S.EXEC21
Pop Stack and Transfer control to recovered PSD

If there is an online IPU executing a task, it will be tested for eligibility vis-a-vis software interrupts and system action requests. If the task is found to be ineligible for the IPU, a flag (EXEC.STR) will be set to indicate that the task should be removed from the IPU and linked to a ready state in the CPU.

The current CPU task is then tested for software interrupts and/or system action requests. If either one is found requesting, control is passed to S.EXEC21 to process the request.

If there are no software interrupts or system action requests, processing continues in S.EXEC20. The next step occurs only if there is an IPU online. The current CPU task will be tested for the IPU inhibited state by zeroing the IPU inhibit flag (DQE.IPUH). If the task is not inhibited, it will be tested for the IPU biased option. A biased task will replace the current IPU task if it is of higher priority. If the task is not biased and is not optioned CPU only, it will be an IPU candidate only if the IPU is idle.

If the current CPU task is optioned CPU only, is IPU inhibited, or for some other reason (no IPU in the system) the current CPU task is not an IPU candidate, the task's stack will be pushed and control passed to it via LPSD.

3.1.62 Subroutine S.EXEC21 - Process Task Interrupt

Functional Description

S.EXEC21 is called by S.EXEC20 to process any task interrupt requests, after the CPU scheduler has determined that the return address in the task context is not in the operating system area. It processes both system action interrupt requests in DQE.SAIR, and requested software task interrupts in DQE.RTI.

Entry Conditions

Calling Sequence:

```
        BU              S.EXEC21
```

Registers:

```
        R2              DQE address
```

Exit Conditions

Return Sequence:

Transfer control as appropriate for task interrupt, or return to interrupted context.

Task Interrupt Request Processing (S.EXEC21)

System Action Task Interrupts (DQE.SAIR)

PRIORITY (BIT)	LABEL	DESCRIPTION	PROCESSING ROUTINE
0	DQE.DELR	DELETE TASK REQUEST	S.EXEC28
1	RESERVED		
2	DQE.HLDR	HOLD TASK REQUEST	S.EXEC51
3	DQE.ABTR	ABORT TASK REQUEST	S.EXEC19
4	DQE.EXTR	EXIT TASK REQUEST	S.EXEC29
5	DQE.SUSR	SUSPEND TASK REQUEST	S.EXEC21
6	DQE.RRRQ	RUN REQUEST	S.EXEC21
7	RESERVED		

Requested Software Task Interrupts (DQE.RTI)

PRIORITY (BIT)	LABEL	DESCRIPTION
0	RESERVED	
1	DQE.EA1R	END ACTION REQUEST (PRIORITY 1)
2	DQE.DBBR	DEBUG BREAK REQUEST
3	DQE.UBKR	USER BREAK REQUEST
4	DQE.EA2R	END ACTION REQUEST (PRIORITY 2)
5	DQE.MSIR	MESSAGE INTERRUPT REQUEST
6-7	RESERVED	

3.1.63 Subroutine S.EXEC22 - Wait for Completion Of All No-Wait Operations

Functional Description

S.EXEC22 is called from either the task abort or task exit processing control subroutines (S.EXEC19 or S.EXEC29). Its purpose is to delay until all outstanding no-wait processing is complete. It accomplishes this by calling H.EXEC,25 until the number of outstanding no-wait requests is equal to zero. A return is then made to the calling routine.

Entry Conditions

Calling sequence:

```
        BL              S.EXEC22
```

Registers:

R2	Current task DQE address

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

R1,R2,R3,R4,R5,R7	Saved
R6	Destroyed

3.1.64 Subroutine S.EXEC23 - Terminate Messages In Receiver Queue

Functional Description

S.EXEC23 is called from one of the task termination processing control subroutines (S.EXEC19, S.EXEC28, or S.EXEC29). Its purpose is to unlink all messages from the receiver queue and to terminate these messages, relinking any waiting tasks to their respective ready-to-run queues. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC23
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

All registers destroyed

3.1.65 Subroutine S.EXEC24 - Reserved

3.1.66 Subroutine S.EXEC25 - Terminate Next Run Request In Receiver Queue

Functional Description

S.EXEC25 is called from one of the task termination processing control subroutines (S.EXEC19, S.EXEC28, or S.EXEC29). Its purpose is to unlink the next run request from the receiver queue, and to terminate the requests with abnormal status. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC25
```

Registers:

X3	Scratchpad address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5,R6,R7	Destroyed

3.1.67 Subroutine S.EXEC26 - Remove Task Gating

Functional Description

S.EXEC26 is called from one of the task termination processing subroutines (S.EXEC19, S.EXEC28, or S.EXEC29). Its purpose is to remove any outstanding gating mechanisms associated with the terminating task (e.g., context switch or FISE). A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC26

Registers:

R2	Current task DQE address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

All registers preserved

3.1.68 Subroutine S.EXEC27 - Transfer Control To Abort Receiver

Functional Description

S.EXEC27 is called from the abort task processing control subroutine (S.EXEC19). Its purpose is to transfer control to the user task abort receiver if one exists. Otherwise, a return will be made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC27

Registers:

R2	Current task DQE address

Exit Conditions

Return Sequence:

TRSW	R0 (or LPSD to abort receiver)

Registers:

All registers preserved

3.1.69 Subroutine S.EXEC28 - Delete Task Processing Control Subroutine

Functional Description

S.EXEC28 is an internal H.EXEC subroutine which is called only by S.EXEC21 to process a task delete request on behalf of the currently executing task. The purpose of S.EXEC28 is to control the sequencing of delete task processing modules associated with the pertinent subsystems:

Module	Description
S.MONS2	Remove task associated timers.
H.MONS,38	Disconnect interrupt (if connected).
S.EXEC26	Remove context switch or FISE gating if any.
H.IOCS,38	Terminate all outstanding I/O requests
S.EXEC23	Terminate all messages in receiver queue.
S.EXEC25	Terminate active run request (if any).
H.ALOC,3	Close and deallocate system critical files. Close and deallocate user I/O files/devices with minimal processing required (no attempt to preserve data integrity). Deallocate all swap files and memory (excluding TSA). Deallocate TSA memory, DQE space, clear C.PRNO, transfer control to CPU scheduler.
S.EXEC43	Reactivate task if additional run requests queued.
S.EXEC76	Update task execution accounting value.
[H.TSM,4]	TSM task abort/delete processing (called only if TSM task).
[H.SOUT,26]	Batch task delete processing (called only if Batch task).
S.ALOC2	Deallocate TSA and DQE.

Entry Conditions

Calling Sequence:

```
        BU          S.EXEC28
```

Registers:

None

Exit Conditions

Return Sequence:

```
        Clear       C.PRNO
        BU          S.EXEC20
```

Registers:

None

3.1.70 Subroutine S.EXEC29 - Exit Task Processing Control Subroutine

Functional Description

S.EXEC29 is an internal H.EXEC subroutine which is called only by S.EXEC21 to process an exit request on behalf of the currently executing task. The purpose of S.EXEC29 is to control the sequencing of exit processing by the exit processing modules associated with the pertinent subsystems:

Module	Description
S.MONS2	Remove task associated timers.
H.MONS,38	Disconnect interrupt (if connected).
S.EXEC26	Remove context switch or FISE gating if any.
S.EXEC22	Defer continued processing until all outstanding no-wait operations are complete.
S.EXEC25	Terminate active run request if any.
H.ALOC3	Close and deallocate system critical files.
	Close and deallocate user I/O files/devices with blocking buffer purge and automatic EOF as appropriate.
	Deallocate all swap files and memory (excluding TSA).
	Deallocate TSA memory, DQE space, clear CPRNO, transfer control to CPU scheduler.
S.EXEC43	Reactivate task if additional run requests queued.
S.EXEC76	Update task execution accounting value.
[H.TSM,3]	TSM task exit processing (called only if TSM task).
[H.SOUT,26]	Batch task exit processing (called only if Batch task).
S.ALOC2	Deallocate TSA and DQE.

Entry Conditions

Calling Sequence:

```
        BU              S.EXEC29
```

Registers:

None

Exit Conditions

Return Sequence:

```
        Clear           C.PRNO

        BU              S.EXEC20
```

Registers:

None

3.1.71 Subroutine S.EXEC30 - Reserved

3.1.72 Subroutine S.EXEC31 - Report No-Wait Run Request Post Processing Complete

Functional Description

S.EXEC31 is called to report the completion of no-wait run request post processing. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level (the most recent) of push down in the TSA. A M.RTRN will then be issued to return to the point of task interrupt.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC31
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN   (to previous context)
```

Registers:

None

3.1.73 Subroutine S.EXEC32 - Report Wait Mode Run Request Complete

Functional Description

S.EXEC32 is called by the appropriate run request exit processor on behalf of the requested task. Its purpose is to report completion of the wait mode run request to the requesting (waiting) task. The waiting task is removed from the wait list and placed in the ready-to-run list (or in the memory request list if an inswap is required).

Entry Conditions

Calling Sequence:

BL	S.EXEC32

Registers:

X1	DQE entry address of sending task
X2	MRRQ address
X3	Address of 22W scratchpad area

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5,R6,R7	Destroyed

3.1.74 Subroutine S.EXEC33 - Report No-Wait Mode Run Request Complete

Functional Description

S.EXEC33 is called to report the completion of run request processing. The call is made on behalf of the task which processed the run request. The requesting task may be in the wait-for-any-run-request-completion state. If so, it will be removed from that list and linked to the ready-to-run list (or to the memory request list if an inswap is required).

The run request queue entry will be linked to the DQE task interrupt list and will contain the no-wait mode run request post processing service address. When the scheduler dispatches control to the task, the specified routine will be entered as a preemptive system service.

Entry Conditions

Calling Sequence:

BL	S.EXEC33

Registers:

R1	DQE address of sending task
R2	MRRQ address
R3	Address of 22W scratchpad area

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5,R6,R7	Destroyed

3.1.75 Subroutine S.EXEC34 - Reserved

3.1.76 Subroutine S.EXEC35 - Report No-Wait Mode Message Post Processing Complete

Functional Description

S.EXEC35 is called to report the completion of no-wait mode message post processing. Its purpose is to clear the task interrupt processing lock, and to return to the point of task interrupt. It will discard one level (the most recent) of pushdown in the TSA stack. A M.RTRN will then be used to return to the point of task interrupt.

Entry Conditions

Calling Sequence:

BL	S.EXEC35

Registers:

None

Exit Conditions

Return Sequence:

CPU scheduler	(to previous context)

Registers:

None

3.1.77 Subroutine S.EXEC36 - Report Wait Mode Message Complete

Functional Description

S.EXEC36 is called by the appropriate message exit processor on behalf of the task that processed the message. Its purpose is to report completion of wait mode message processing to the waiting task. The waiting task is removed from the wait list and placed in the ready-to-run list (or in the memory request list if an inswap is required).

Entry Conditions

Calling Sequence:

```
BL          S.EXEC36
```

Registers:

X1	DQE entry address of sending task
X2	MRRQ address
X3	Address of 22W scratchpad area

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5,R6,R7	Destroyed

3.1.78 Subroutine S.EXEC37 - Report No-Wait Mode Message Complete

Functional Description

S.EXEC37 is called to report the completion of message processing. The call is made on behalf of the task which processed the message. The task which sent the message may be in the wait-for-any-message-completion queue. If so, it will be removed from that list and linked to the ready-to-run-list (or to the memory request list if an inswap is required).

The message queue entry will be linked to the DQE task interrupt list and will contain the no-wait mode message post processing service address. When the scheduler dispatches control to the task, the specified routine will be eentered as a preemptive system service.

Entry Conditions

Calling Sequence:

BL	S.EXEC37

Registers:

R1	DQE address of requesting task
R2	MRRQ address
R3	Address of 22W scratchpad area

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5,R6,R7	Destroyed

3.1.79 Subroutine S.EXEC38 - Inhibit Swap Of Current Task

Functional Description

S.EXEC38 is called to set the inhibit swap flag (DQE.LKIM) in the DQE of the current task. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC38
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R2	Destroyed
R1,R3,R4,R5,R6,R7	Saved

3.1.80 Subroutine S.EXEC39 - Enable Swap of Current Task

Functional Description

S.EXEC39 is called to reset the inhibit swap flag (DQE.LKIM) in the DQE of the current task. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC39
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R2	Destroyed
R1,R3,R4,R5,R6,R7	Saved

3.1.81 Subroutine S.EXEC40 - Reserved

3.1.82 Subroutine S.EXEC41 - Exit Run Receiver

Functional Description

S.EXEC41 is called to exit a run receiver when the M.XRUNR exit type is invoked. Its purpose is to process the exit according to the specifications contained in the Receiver Exit Block (RXB). The run receiver queue will be examined, and if not empty, the task executed again at the point following the M.XRUNR call on behalf of the next request. If the queue is empty, the exit options are examined. If option bit 0 is set, the task will be placed in a wait-state, waiting for the next run request to be received. If option bit 0 is reset, the task will exit the system.

Entry Conditions

Calling Sequence:

BL	S.EXEC41

Registers:

R1	Current task DQE address
R2	Receiver Exit Block (RXB) address
X3	Scratchpad address

Exit Conditions

Return Sequence:

No return (rerun task or exit)

3.1.83 Subroutine S.EXEC42 - Exit Message Receiver

Functional Description

S.EXEC42 is called to exit a message receiver when a M.XMSGR service has been called. If the message interrupt is not active, the task will be aborted. Its purpose is to reset the task interrupt lock and to process the exit according to the specifications in the Receiver Exit Block (RXB). The message receiver queue will be examined, and if not empty, the message interrupt will be invoked again on behalf of the next request. If the queue is empty, a return will be made to the point of interrupt (or following M.SUSP/M.ANYW) at the base execution level.

Entry Conditions

Calling Sequence:

BL	S.EXEC42

Registers:

R1	Current task DQE address
R2	Receiver Exit Block (RXB) address
X3	Scratchpad address

Exit Conditions

Return Sequence:

No return to caller (rerun message receiver or return to user base level context)

3.1.84 Subroutine S.EXEC43 - Reactivate Run Receiver Task

Functional Description

S.EXEC43 is called to examine the run receiver queue of a run receiver task that has used a standard M.EXIT call. S.EXEC43 is called by S.EXEC28, S.EXEC29, or S.EXEC19. If queued run requests exist, a call to H.ALOC1 is made to reactivate the task. If H.ALOC1 makes a denial return, all outstanding requests are terminated with abnormal status. If H.ALOC1 successfully starts the activation, S.EXEC43 will link any remaining queued requests to the DQE of the task being activated. A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC43

Registers:

None

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

All registers destroyed

3.1.85 Subroutine S.EXEC44 - Change Priority Level of Current Task

Functional Description

S.EXEC44 is called to change the priority level of the current task. The specified priority level is stored in DQE.CUP and DQE.BUP and as the priority level of the currently executing task. (No relink of this task is required since it is linked to the special state chain for the currently executing task.) A return is then made to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC44

Registers:

R6	Priority level

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	Current task DQE address
R1,R3,R4,R5,R6,R7	Saved

3.1.86 Subroutine S.EXEC45 - Change Priority Level of Specified Task

Functional Description

S.EXEC45 is called to change the priority level of the specified (not current) task. The specified task may either be in a ready-to-run state, or in a wait state. If the task is in a ready-to-run state, the specified priority is stored in DQE.CUP and in DQE.BUP. The task is then unlinked from its current ready to run list and relinked to the ready to run list associated with the new priority. If the task was in a wait state, it will be relinked according to its new priority into the same wait list. A return will then be made to the calling program.

Entry Conditions

Calling Sequence:

```
        BL            S.EXEC45
```

Registers:

R6	Priority level
R1	DQE address of specified task

Exit Conditions

Return Sequence:

```
        TRSW          R0
```

Registers:

All registers destroyed

3.1.87 Subroutine S.EXEC46 - Reserved

3.1.88 Subroutine S.EXEC47 - Reserved

3.1.89 Subroutine S.EXEC48 - Convert Task Number to DQE Address

Functional Description

S.EXEC48 is called to calculate the DQE address of the specified task.

Entry Conditions

Calling Sequence:

```
        BL            S.EXEC48
```

Registers:

R7	Task activation sequence number of specified task

Exit Conditions

Return Sequence:

```
        TRSW          R0
```

Registers:

R2	DQE address of specified task
R1,R3,R4,R5,R7	Saved
R6	Destroyed

3.1.90 Subroutine S.EXEC49 - Construct MRRQ

Functional Description

S.EXEC49 is called to construct an MRRQ entry for either a message or run request. Space for the MRRQ is allocated from memory pool. The MRRQ is constructed according to the contents of the Parameter Send Block (PSB) specified as a calling parameter.

Entry Conditions

Calling Sequence:

BL	S.EXEC49

Registers:

R2	Parameter Send Block (PSB) address
X3	Scratchpad address

Exit Conditions

Return Sequence:

TRSW	R0 (CC1 set indicates memory pool unavailable)

Registers:

R1	MRRQ address
X3=X3-4W	Scratchpad address
R2,R4	Saved
R5,R6,R7	Destroyed

3.1.91 Subroutine S.EXEC50 - Link MRRQ To Run Receiver Of Destination Task

Functional Description

S.EXEC50 is called to link the designated MRRQ entry to the run receiver queue of the specified task. If the target task is in a RUNW wait state, it will be unlinked from the wait list and linked to the ready-to-run list.

Entry Conditions

Calling Sequence:

BL	S.EXEC50

Registers:

R1	Target task DQE address
R2	MRRQ address
X3	Scratchpad address
R5	Zero indicates unlink if in PREA list
R7	Task activation sequence number of target task

Exit Conditions

Return Sequence:

```
TRSW        R0  (CC1 set indicates invalid target task)
```

Registers:

X3=X3-4W	Scratchpad address
R1,R2	Saved
R4,R5,R6,R7	Destroyed

3.1.92 Subroutine S.EXEC51 - Link Current Task To Designated Wait State

Functional Description

S.EXEC51 is called to place the currently executing task in the designated wait state. If the task is a time distribution task, the execution time accounting value is updated in the TSA. The current quantum value for next dispatch is updated in DQE.CQC for both real-time and time distribution tasks. When linkage to the wait state is complete, the memory scheduler is resumed if entries are queued in the memory request list.

Entry Conditions

Calling Sequence:

```
BEI         STATUS
BL          S.EXEC51
```

Registers:

R1	Address of wait state headcell
R2	Current task DQE address

Exit Conditions

Return Sequence:

No return to calling routine

M.RTRN to TSA stack context when task is ready to run

Registers:

None

3.1.93 Subroutine S.EXEC52 - Message Or Run Request Post Processing Subroutine

Functional Description

S.EXEC52 is called on behalf of the sending task when a message or run request has been processed by the destination task. It will transfer the return parameters to the return buffer designated in the PSB, update PSB status, and deallocate the MRRQ.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC52
```

Registers:

R2	MRRQ address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

X3=X3-4W	Scratchpad address
R5	Saved
R1,R2,R4,R6,R7	Destroyed

3.1.94 Subroutine S.EXEC53 - Validate PSB

Functional Description

S.EXEC53 is called to validate the parameters contained in the Parameter Send Block (PSB) associated with a message or run request. An immediate return is made if the most recent context on the TSA stack reflects a privileged caller. Otherwise, S.ALOC20 is called to verify the PSB address arguments, and general parameter validation is performed.

Entry Conditions

Calling Sequence:

```
          BL             S.EXEC53
```

Registers:

R2	PSB address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
          TRSW           R0  (CC1 set indicates validation error)
```

Registers:

X3=X3-4W	Scratchpad address
R6	Contains error code if CC1 set, otherwise destroyed
R2	Saved
R1,R4,R5,R7	Destroyed

3.1.95 Subroutine S.EXEC54 - Move Byte String

Functional Description

S.EXEC54 is a register reentrant routine which moves a byte string of the designated length from the origin address to the destination address.

Entry Conditions

Calling Sequence:

```
          BL             S.EXEC54
```

Registers:

R1	Origin (from) address
R2	Destination (to) address
R7	Negative number of bytes to be transferred

Exit Conditions

Return Sequence:

```
        TRSW          R0
```

Registers:

R3,R4,R5	Saved
R1,R2,R6,R7	Destroyed

3.1.96 Subroutine S.EXEC55 - Unlink Task From Designated List and Link To Ready List

Functional Description

S.EXEC55 is called to link a task to the ready-to-run queue. It will unlink the task from the designated list, then link the task to the ready list associated with its current priority. If the task is outswapped and cannot run, it will be linked to the memory request queue and the memory scheduler will be resumed.

Entry Conditions

Calling Sequence:

```
        BEI           STATUS
        BL            S.EXEC55
```

Registers:

R1	Designated list headcell address
R2	DQE address

Exit Conditions

Return Sequence:

```
        TRSW          R0
```

Registers:

R2,R4,R7	Saved
R1,R3,R5,R6	Destroyed

3.1.97 Subroutine S.EXEC56 - Resume Memory Scheduler

Functional Description

S.EXEC56 is called as a result of a memory scheduler event. An immediate return will be made if no memory requests are queued. Otherwise the memory scheduler is made ready to run at the priority of the highest priority memory request queued.

Entry Conditions

Calling Sequence:

```
BEI          STATUS
BL           S.EXEC65
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW         R0
```

Registers:

R4,R6,R7	Saved
R1,R2,R3,R5	Destroyed

3.1.98 Subroutine S.EXEC57 - Link Task To Ready List By Priority

Functional Description

S.EXEC57 is called to link the designated task to the ready to run list associated with its current priority.

Entry Conditions

Calling Sequence:

```
BEI          STATUS
BL           S.EXEC57
```

Registers:

R2	DQE address of specified task

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2,R4,R6,R7	Saved
R1,R3,R5	Destroyed

3.1.99 Subroutine S.EXEC58 - Link MRRQ To Message Receiver Of Destination Task

Functional Description

S.EXEC58 is called on behalf of the sending task to link an MRRQ entry to the message receiver queue of the destination task.

Entry Conditions

Calling Sequence:

BL	S.EXEC58

Registers:

R1	Destination task DQE address
R2	MRRQ address
X3	Scratchpad address
R7	Task activation sequence number of destination task

Exit Conditions

Return Sequence:

TRSW	R0 (CC1 set indicates invalid destination task)

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4	Saved
R5,R6,R7	Destroyed

3.1.100 Subroutine S.EXEC59 - Reserved

3.1.101 Subroutine S.EXEC60 - Validate PRB

Functional Description

S.EXEC60 is called to validate the Parameter Receive Block (PRB) of the destination task, when the destination task has made a request for the message or run request parameters. Validation is bypassed if the most recent pushdown on the TSA stack reflects a privileged caller. Otherwise general PRB validation is performed.

Entry Conditions

Calling Sequence:

BL	S.EXEC60

Registers:

R2	PRB address
X3	Scratchpad address

Exit Conditions

Return Sequence:

TRSW	R0 (CC1 set indicates validation error)

Registers:

R6	Error code if CC1 set, otherwise destroyed
R2	Saved
R1,R4,R5,R7	Destroyed
X3=X3-4W	Scratchpad address

3.1.102 Subroutine S.EXEC61 - Transfer Parameters From MRRQ To Receiver Buffer

Functional Description

S.EXEC61 is called on behalf of the destination task after a request for message or run request parameters has been made. The sent parameters are transferred by S.EXEC61 from the MRRQ entry to the receiver buffer specified in the PRB.

Entry Conditions

Calling Sequence:

```
        BL            S.EXEC61
```

Registers:

R1	MRRQ address
R2	PRB address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
        TRSW          R0
```

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5	Saved
R7	Destroyed
R6	Status code: 0=OK, 4=Receiver buffer length exceeded

3.1.103 Subroutine S.EXEC62 - Validate RXB

Functional Description

S.EXEC62 is called to validate the Receiver Exit Block (RXB) after the destination task has issued an exit from message or run request processing.

Entry Conditions

Calling Sequence:

```
        BL            S.EXEC62
```

Registers:

R2	RXB address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
        TRSW          R0 (CC1 set indicates validation error)
```

Registers:

R6	Error code if CC1 set, otherwise destroyed
X3=X3-4W	Scratchpad address
R2	Saved
R1,R4,R5,R7	Destroyed

3.1.104 Subroutine S.EXEC63 - Transfer Return Parameters From Destination Task To MRRQ

Functional Description

S.EXEC63 is called on behalf of the destination task to transfer return parameters to the MRRQ after the destination task has issued an exit from message or run request processing.

Entry Conditions

Calling Sequence:

BL	S.EXEC63

Registers:

R1	MRRQ address
R2	RXB address
X3	Scratchpad address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R5	Saved
R6,R7	Destroyed

3.1.105 Subroutine S.EXEC64 - No-Wait Mode Message Post Processing Subroutine

Functional Description

S.EXEC64 is invoked as a preemptive system service (end action priority 2) on behalf of the sending task. It in turn calls S.EXEC52 to accomplish post processing of the MRRQ. It will optionally vector to a user specified end action routine, or call H.EXEC,34 to report no-wait message post processing complete.

Entry Conditions

Calling Sequence:

Preemptive system service

Registers:

R2	MRRQ address

Exit Conditions

Return Sequence:

No return is made

S.EXEC64 will exit to the user end action routine or to H.EXEC,34

Registers:

R1	PSB address on entry to user end action routine

3.1.106 Subroutine S.EXEC65 - No-Wait Mode Run Request Post Processing Subroutine

Functional Description

S.EXEC65 is invoked as a preemptive system service (end action priority 2) on behalf of the sending task. It in turn calls S.EXEC52 to accomplish post processing of the MRRQ. It will optionally vector to a user specified end action routine, or call H.EXEC,28 to report no-wait run request post processing complete.

Entry Conditions

Calling Sequence:

Preemptive system service

Registers:

R2	MRRQ address

Exit Conditions

Return Sequence:

No return is made

S.EXEC65 will exit to the user end action routine or to H.EXEC,28

Registers:

R1	PSB address on entry to user end action routine

3.1.107 Subroutine S.EXEC66 - Deallocate MRRQ

Functional Description

S.EXEC66 is called to deallocate an MRRQ when processing associated with the MRRQ is complete. S.ALOC22 is called to return the MRRQ space to memory pool.

Entry Conditions

Calling Sequence:

```
         BL              S.EXEC66
```

Registers:

R2	MRRQ address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
         TRSW            R0
```

Registers:

X3=X3-4W	Scratchpad address
R5	Saved
R1,R2,R4,R6,R7	Destroyed

3.1.108 Subroutine S.EXEC67 - Link Entry To End Action Queue

Functional Description

S.EXEC67 is called on behalf of the destination task when destination task processing of a no-wait mode message or run request is complete. Its purpose is to link the MRRQ entry to the end action queue of the sending task. This will cause the appropriate post processing routine to be invoked as a preemptive system service on behalf of the sending task.

Entry Conditions

Calling Sequence:

```
          BEI            STATUS
          BL             S.EXEC67
```

Registers:

R1	DQE address of sending task
R2	MRRQ address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
          TRSW           R0
```

Registers:

X3=X3-4W	Scratchpad address
R1,R2,R4,R6	Saved
R5,R7	Destroyed

3.1.109 Subroutine S.EXEC68 - Construct And Vector Context To User End Action PSD

Functional Description

S.EXEC68 is called by the send request post processing logic to vector to a user specified end action routine. Control will be transferred with mapped, unblocked status.

Entry Conditions

Calling Sequence:

```
          BL             S.EXEC68
```

Registers:

R1	PSB address
R6	User end action routine address

Exit Conditions

Return Sequence:

No return

LPSD to user end action routine

Registers:

R2	PSB address

3.1.110 Subroutine S.EXEC69 - Common No-Wait Post Processing Merge Point

Functional Description

S.EXEC69 is called to remove the task interrupt processing lock and mark the task interrupt request if additional end action entries are queued. The most recent level of context in the TSA stack is discarded, and an M.RTRN is issued to pop to the previous context level.

Entry Conditions

Calling Sequence:

BL	S.EXEC69

Registers:

R1	Current task DQE address

Exit Conditions

Return Sequence:

No return to caller

M.RTRN to previous context level

Registers:

None

3.1.111 Subroutine S.EXEC70 - Terminate All Run Requests In Receiver Queue Of Current Task

Functional Description

S.EXEC70 is called when an error is encountered in attempting to reactivate a terminating task with additional run requests queued. S.EXEC70 will in turn call S.EXEC25 until the receiver queue is empty.

Entry Conditions

Calling Sequence:

```
BL          S.EXEC70
```

Registers:

X2	Current task DQE address
X3	Scratchpad address

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

X3=X3-4W	Scratchpad address
R2	Saved
R1,R4,R5,R6,R7	Destroyed

3.1.112 Subroutine S.EXEC71 - Insure Startup Of Destination Run Receiver Task

Functional Description

S.EXEC71 is called to unlink the destination task from the RUNW or PREA list (unless R5 is not zero), and to link the destination task to the ready list at the priority specified in R6.

Entry Conditions

Calling Sequence:

```
BEI         STATUS
BL          S.EXEC71
```

Registers:

R2	DQE address of destination task
R5	Zero if task is to be unlinked from PREA list
R6	Priority

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R2,R4	Saved
R1,R3,R5,R6,R7	Destroyed

3.1.113 Subroutine S.EXEC72 - Report Wait I/O Starting

Functional Description

S.EXEC72 is called to process a report of wait mode I/O starting. A check is made to see if the associated I/O has already completed. If so, an immediate return is made. Otherwise the swap inhibit flag will be set (unless R0 bit 1 is set) and the task will be linked to the designated wait list.

Entry Conditions

Calling Sequence:

```
BU        S.EXEC72
```

Registers:

R0	Bit 0 set indicates task is swappable during I/O
R1	Address of wait list headcell

Exit Conditions

Return Sequence:

No return to caller

CPU scheduler return to context on TSA stack when I/O complete

Registers:

None

3.1.114 Subroutine S.EXEC73 - Replace Context On TSA Stack

Functional Description

S.EXEC73 is called to replace the most recent context on the TSA stack with the designated context block.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC73
```

Registers:

R1	Address of 10W context block

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

R1,R3	Saved
R2,R4,R5,R6,R7	Destroyed

3.1.115 Subroutine S.EXEC74 - Reset Stack To User Level

Functional Description

S.EXEC74 is called on abnormal task termination to reset the stack to the point of last user call.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC74
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

R2,R4,R6,R7	Saved
R1,R3,R5	Destroyed

3.1.116 Subroutine S.EXEC75 - Situational Priority Increment Subroutine

Functional Description

S.EXEC75 is called to increment the priority of the specified task. Priority adjustment is bypassed if the specified task is a real time task.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC75
```

Registers:

R2	DQE address of target task
R4	Situational priority increment

Exit Conditions

Return Sequence:

```
        TRSW            R0
```

Registers:

R1,R2,R3,R4,R6,R7	Saved
R5	Destroyed

3.1.117 Subroutine S.EXEC76 - Update Task Execution Accounting Value

Functional Description

S.EXEC76 is called during task termination processing. The interval timer is read and the elapsed time added to T.ITAC in the TSA.

Entry Conditions

Calling Sequence:

```
        BL              S.EXEC76
```

Registers:

R2	Current task DQE address

Exit Conditions

Return Sequence:

```
         TRSW          R0
```

Registers:

R3	Current task TSA address
R1,R2,R4,R6,R7	Saved
R5	Destroyed

3.1.118 Subroutine S.EXEC77 - Update DQE.CQC On Preemptive Context Switch

Functional Description

S.EXEC77 is called by S.EXEC20 when a context switch is being performed as a result of preemption by a higher priority task. Its purpose is to update the current quantum value in the DQE (DQE.CQC) for use when the preempted task is re-dispatched.

Entry Conditions

Calling Sequence:

```
         BEI           STATUS
         BL            S.EXEC77
```

Registers:

R2	Current task DQE address

Exit Conditions

Return Sequence:

```
         TRSW          R0
```

Registers:

R1,R2,R4,R6	Saved
R3,R5,R7	Destroyed

3.1.119 Subroutine S.EXEC78 - Move Context From Stack to T.CONTXT

Functional Description

S.EXEC78 is called when the context of the PSD, R0-R7, and PC are copied from the TSA of the current task to the Debug context area. After filling the T.CONTXT area, it will return to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC78

Registers:

None

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1	Current pushdown address from T.REGP
R3	TSA Address of current task
R2,R4,R5	Unchanged
R6,R7	Destroyed

3.1.120 Subroutine S.EXEC79 - Push Current Context onto Stack for Deferred EA Pull

Functional Description

S.EXEC79 is called to format a PSD with the privileged mode set, the condition codes clear, the Extended Addressing Mode clear, the right halfword instruction clear, the arithmetic exception trap clear, and the PC pointing into H.EXEC25 in word one. Word two has the Map Mode set and CPIX of the specified task. The subroutine will then place the PSD and the registers (with the contents at the end of the subroutine) onto the stack. After placing the information onto the stack, the subroutine will return to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC79

Registers:

R2	DQE Address of specified task

Exit Conditions

Return Sequence:

R2,R3,R6,R7	Unchanged
R1,R4,R5	Destroyed

3.1.121 Subroutine S.EXEC80 - Start IPU and Verify

Functional Description

S.EXEC80, when called, will reset the IPU run flag and initialize and start the IPU verification timer. The subroutine will then return control to the calling routine.

Entry Conditions

Calling Sequence:

BL	S.EXEC80

Registers:

None

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2,R3,R5,R6,R7	Unchanged
R4	Destroyed

3.2 System Services (H.MONS)

ENTRY POINT	SVC NUMBER	DESCRIPTION
H.MONS,1	42	PHYSICAL DEVICE INQUIRY
H.MONS,2	43	PERMANENT FILE ADDRESS INQUIRY
H.MONS,3	44	MEMORY ADDRESS INQUIRY
H.MONS,4	45	CREATE TIMER ENTRY
H.MONS,5	46	TEST TIMER ENTRY
H.MONS,6	47	DELETE TIMER ENTRY
H.MONS,7	48	SET USER STATUS WORD
H.MONS,8	49	TEST USER STATUS WORD
H.MONS,9	4A*	CHANGE PRIORITY LEVEL
H.MONS,10	4B	CONNECT TASK TO INTERRUPT
H.MONS,11	4E	TIME OF DAY INQUIRY
H.MONS,12	4F	MEMORY DUMP REQUEST
H.MONS,13	50	LOAD OVERLAY SEGMENT
H.MONS,14	51	LOAD AND EXECUTE OVERLAY SEGMENT
H.MONS,15	52	ACTIVATE TASK
H.MONS,16	53	RESUME TASK EXECUTION
H.MONS,17	54	SUSPEND TASK EXECUTION
H.MONS,18	55	TERMINATE TASK EXECUTION
H.MONS,19	56	ABORT SPECIFIED TASK
H.MONS,20	57	ABORT SELF
H.MONS,21	40	ALLOCATE FILE OR PERIPHERAL DEVICE
H.MONS,22	41	DEALLOCATE FILE OR PERIPHERAL DEVICE
H.MONS,23	4D	ARITHMETIC EXCEPTION INQUIRY
H.MONS,24	4C	TASK OPTION WORD INQUIRY
H.MONS,25	58	PROGRAM HOLD REQUEST
H.MONS,26	60	SET USER ABORT RECEIVER ADDRESS
H.MONS,27	61	SUBMIT JOB FROM DISC FILE
H.MONS,28	62	ABORT WITH EXTENDED MESSAGE
H.MONS,29	63	LOAD AND EXECUTE INTERACTIVE DEBUGGER
H.MONS,30	N/A	DELETE INTERACTIVE DEBUGGER
H.MONS,31	5A	DELETE TASK
H.MONS,32	64	GET TASK NUMBER
H.MONS,33	73	PERMANENT FILE LOG
H.MONS,34	74	USERNAME SPECIFICATION
H.MONS,35	7A	GET MESSAGE PARAMETERS
H.MONS,36	7B	GET RUN PARAMETERS
H.MONS,37	7C	WAIT FOR ANY NO-WAIT OPERATION COMPLETE, MESSAGE INTERRUPT OR BREAK INTERRUPT
H.MONS,38	5D	DISCONNECT TASK FROM INTERRUPT
H.MONS,39	5E	EXIT FROM MESSAGE RECEIVER
H.MONS,40	5F*	PARAMETER TASK ACTIVATION
H.MONS,41	65	GET ADDRESS LIMITS
H.MONS,42	66	DEBUG LINK SERVICE
H.MONS,43	6B	RECEIVE MESSAGE LINK ADDRESS
H.MONS,44	6C	SEND MESSAGE TO SPECIFIED TASK

* Implies that this service is available to privileged users only.
N/A implies reserved for internal use by MPX-32.

ENTRY POINT	SVC NUMBER	DESCRIPTION
H.MONS,45	6D	SEND RUN REQUEST TO SPECIFIED TASK
H.MONS,46	6E	BREAK/TASK INTERRUPT LINK
H.MONS,47	6F	ACTIVATE TASK INTERRUPT
H.MONS,48	70	EXIT FROM TASK INTERRUPT LEVEL
H.MONS,49	7D	EXIT RUN RECEIVER
H.MONS,50	7E	EXIT FROM MESSAGE END ACTION ROUTINE
H.MONS,51	7F	EXIT FROM RUN REQUEST END ACTION ROUTINE
H.MONS,52	N/A	RTM CALM 'X55' EXIT
H.MONS,53	N/A	RTM CALM X'52' ACTIVATE
H.MONS,54	N/A	RTM CALM X'54' SUSPEND SELF
H.MONS,55	N/A	RTM CALM X'40' ALLOCATION FILE OR DEVICE
H.MONS,56	N/A	RTM CALM X'41' PHYSICAL DEVICE INQUIRY
H.MONS,57	2E	DISABLE MESSAGE TASK INTERRUPT
H.MONS,58	2F	ENABLE MESSAGE TASK INTERRUPT
H.MONS,59	N/A	GET PHYSICAL MEMORY CONTENTS
H.MONS,60	N/A	CHANGE PHYSICAL MEMORY CONTENTS
H.MONS,61	N/A	RTM CALM X'73' LOG FILE(S)
H.MONS,62	19	RESOURCEMARK LOCK
H.MONS,63	1A	RESOURCEMARK UNLOCK
H.MONS,64	N/A	REMOVE RSM LOCK ON TASK TERM
H.MONS,65	2D	TASK CPU EXECUTION TIME
H.MONS,66	1E	ACTIVATE PROGRAM AT GIVEN TIME OF DAY
H.MONS,67	1B	SET SYNCHRONOUS TASK INTERRUPT
H.MONS,68	1C	SET ASYNCHRONOUS TASK INTERRUPT
H.MONS,69		RESERVED
H.MONS,70	15	DATE AND TIME INQUIRY
H.MONS,71	14	GET DEVICE MNEMONIC OR TYPE CODE
H.MONS,72	13	ENABLE USER BREAK INTERRUPT
H.MONS,73	12	DISABLE USER BREAK INTERRUPT
H.MONS,99	N/A	SYSGEN INITIALIZATION

N/A implies reserved for internal use by MPX-32.

RTM System Services Under MPX-32

MPX-32 will accept call monitor (CALM) instructions which are syntactically and functionally equivalent to the RTM CALM's. These CALM's are implemented for compatibility purposes, only. The user is encouraged to use the new MPX-32 System Services, instead of CALM's, because they will run faster and support the new capabilities available in MPX-32. Mixing CALM's and SVC's in the same program element is discouraged.

Generally, RTM CALM's will operate under MPX-32 without any change in syntax or function. A few seldom-used CALM's have been deleted, and others may have additional restrictions applied to them. In general, however, the changes to the users source code should be minimal in the conversion from RTM to MPX-32.

On a CONCEPT/32 computer, a new SVC type 15 replaces CALM instructions. During reassembly of a program, the Assembler automatically converts CALM instructions to their equivalent SVC 15,X'nn' number if Option 20 is set.

Also, an address exception trap will be generated when a doubleword operation code is used with an incorrectly bounded operand, therefore coding changes will be required when a trap occurs.

Under MPX-32 the following RTM CALM implementation is slightly different from its RTM equivalent:

CALM X'73' Permanent File Log (The file definition is returned in sectors instead of allocation units).

The following RTM CALM's have been deleted in MPX-32.

CALM X'62' Unlink Dynamic Job Queue Entry (not required in MPX).
M.DDJS or CALL M:UNLKJ

CALM X'63' Activate with Core Append (replaced by memory expansion and contraction services of MPX).
M.ACAP (was not in RTM run-time)

CALM X'64' Retrieve Address of Appended Core (same as CALM X'63').
M.APAD (was not in RTM run-time)

CALM X'65' Initialize reentrant library pointers (MPX-32 does not support the RTM reentrant run-time library).

All Random Access Calls

CALM X'59' Random Access OPEN (MPX-32 does not support DRAH).
(CALL M:OPEN)

CALM X'5A' Random Access READ (same as CALM X'59').

CALM X'5B' Write Function (same as CALM X'59').
(CALL M:WRITE)

CALM X'5C' Define Function
(CALL M:DEFINE)

CALM X'5D' Find Function
(CALL M:DEFINE)

TSS CALM's

MPX-32 replaces TSS with TSM, a new online support package. Therefore, all TSS CALM's X'80' - X'84' have been deleted.

FORTRAN Run-Time

MPX-32 does not contain the FORTRAN run-time package. Certain performance differences will be associated with certain functions as have been highlighted in the monitor service section.

Summarizing

M:CONNECT is equivalent to M:CONRES.

M:UNLKJ is deleted.

M:IOEX

1. Files are left open after dispatch to abort receiver.

2. Normal exit does not result in dispatch of control to abort receiver.

3. Only privileged users may re-establish an abort receiver from an abort receiver (only one abort allowed for unprivileged user).

M:LOG - note SMD layout is different in MPX.

M:BLOCK

1. The user may specify blocking via the JCL or Catalog directives, rather than use this service.

2. Number of block files is specified by user via JCL and Catalog directives. Blocking buffers are maintained in user's TSA rather than in run-time package. Not limited to 5.

3. Number of FCB's has been expanded to 64.

3.2.1 Entry Point 1 - Physical Device Inquiry

See Section 7.8.16 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.2 Entry Point 2 - Permanent File Address Inquiry

See Section 7.8.8 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.3 Entry Point 3 - Memory Address Inquiry

See Section 8.3.1 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.4 Entry Point 4 - Create Timer Entry

See Section 8.2.33 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.5 Entry Point 5 - Test Timer Entry

See Section 8.2.43 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.6 Entry Point 6 - Delete Timer Entry

See Section 8.2.14 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.7 Entry Point 7 - Set User Status Word

See Section 8.2.32 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.8 Entry Point 8 - Test User Status Word

See Section 8.2.42 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.9 Entry Point 9 - Change Priority Level

See Section 8.2.29 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.10 Entry Point 10 - Connect Task To Interrupt

See Section 8.2.8 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.11 Entry Point 11 - Time-Of-Day Inquiry

See Section 8.2.40 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.12 Entry Point 12 - Memory Dump Request

See Section 8.3.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.13 Entry Point 13 - Load Overlay Segment

See Section 8.2.27 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.14 Entry Point 14 - Load and Execute Overlay Segment

See Section 8.2.27 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.15 Entry Point 15 - Activate Task

See Section 8.2.1 in the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.16 Entry Point 16 - Resume Task Execution

See Section 8.2.37 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.17 Entry Point 17 - Suspend Task Execution

See Section 8.2.38 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.18 Entry Point 18 - Terminate Task Execution

See Section 8.2.20 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.19 Entry Point 19 - Abort Specified Task

See Section 8.2.4 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.20 Entry Point 20 - Abort Self

See Section 8.2.4 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.21 Entry Point 21 - Allocate File Or Peripheral Device

See Section 7.8.1 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.22 Entry Point 22 - Deallocate File Or Peripheral Device

See Section 7.8.6 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.23 Entry Point 23 - Arithmetic Exception Inquiry

See Section 8.2.41 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.24 Entry Point 24 - Task Option Word Inquiry

See Section 8.2.28 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.25 Entry Point 25 - Program Hold Request

See Section 8.2.23 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.26 Entry Point 26 - Set User Abort Receiver Address

See Section 8.2.36 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.27 Entry Point 27 - Submit Job From Disc File

See Section 8.2.7 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.28 Entry Point 28 - Abort With Extended Message

See Section 8.2.4 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.29 Entry Point 29 - Load and Execute Interactive Debugger

See Section 8.2.10 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.30 Entry Point 30 - Delete Interactive Debugger

Functional Description

A call to this entry point will cause the Interactive Debugger to be disassociated with the calling task.

Entry Conditions

Calling Sequence:

```
        M.CALL        H.MONS,30
```

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

External References

System Macros:

```
        M.RTRN
```

Abort Cases:

None

Output Messages:

None

3.2.31 Entry Point 31 - Delete Task

See Section 8.2.11 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.32 Entry Point 32 - Get Task Number

See Sections 8.2.24 and 8.2.26 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.33 Entry Point 33 - Permanent File Log

See Section 7.8.15 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.34 Entry Point 34 - Username Specification

See Section 7.8.28 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.35 Entry Point 35 - Get Message Parameters

See Section 8.2.21 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.36 Entry Point 36 - Get Run Parameters

See Section 8.2.22 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.37 Entry Point 37 - Wait For Any No-Wait Operation Complete; Message Interrupt Or Break Interrupt

See Section 8.2.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.38 Entry Point 38 - Disconnect Task From Interrupt

See Section 8.2.13 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.39 Entry Point 39 - Exit From Message Receiver

See Section 8.2.47 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.40 Entry Point 40 - Parameter Task Activation

See Section 8.2.30 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.41 Entry Point 41 - Get Address Limits

See Section 8.3.7 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.42 Entry Point 42 - DEBUG Link Service

See Section 8.2.51 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.43 Entry Point 43 - Receive Message Link Address

See Section 8.2.31 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.44 Entry Point 44 - Send Message To Specified Task

See Section 8.2.34 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.45 Entry Point 45 - Send Run Request To Specified Task

See Section 8.2.35 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.46 Entry Point 46 - Break/Task Interrupt Link

See Section 8.2.5 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.47 Entry Point 47 - Activate Task Interrupt

See Section 8.2.25 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.48 Entry Point 48 - Exit From Task Interrupt Level

See Sections 8.2.6 and 8.2.45 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.49 Entry Point 49 - Exit Run Receiver

See Section 8.2.49 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.50 Entry Point 50 - Exit From Message End Action Routine

See Section 8.2.46 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.51 Entry Point 51 - Exit from Run Request End Action Routine

See Section 8.2.48 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.52 Entry Point 52 - Terminate Task Exectution

Functional Description

This entry point performs all normal termination functions required of exiting tasks. All devices and memory area are deallocated, related table space is erased, and the task's Dispatch Queue entry is cleared. If a timer or interrupt level is associated with the task, it will be reactivated, connected, and suspended. Resident tasks are merely suspended.

Entry Conditions

Calling Sequence:

```
CALM        X'55'
(or)
M.CALL      H.MONS,52
```

Registers:

None

Exit Conditions

Return Sequence:

Return to EXEC for sweep of Dispatch Queue

Registers:

None

External References

System Macros:

M.RTRN

Abort Cases:

None

Output Messages:

None

3.2.53 Entry Point 53 - Activate Task

Functional Description

This entry point is used to activate a task. The task assumes the owner name of the caller.

Entry Conditions

Calling Sequence:

```
CALM       X'52'
(or)
M.CALL     H.MONS,53
```

Registers:

R6,R7	1-8 ASCII character left-justified blank filled program name for which an activation request is to be queued.

Exit Conditions

Return Sequence:

M.RTRN

Note: If the task being activated is not in the system, an abort indication is not given.

Registers:

None

External References

System Macros

M.RTRN

Abort Codes:

None

Output Messages:

None

3.2.54 Entry Point 54 - Suspend Task Execution

Functional Description

This entry point results in the suspension of the caller or any other task of the same owner name for the specified number of time units or for an indefinite time period, as requested. A task suspended for a time interval results in a one-shot timer entry to resume the task upon time-out of the specified interval. A task suspended for an indefinite time interval must be resumed through the M.SUME system service (see MPX-32 Reference Manual Volume 1, Section 8.2.37).

Entry Conditions

Calling Sequence:

```
CALM      X'54'
(or)
M.CALL    H.MONS,54
```

Registers:

R7 zero if suspension for an indefinite time interval is requested

(or)

the negative number of time units to elapse before the caller is resumed

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

External References

System Macros:

M.RTRN M.IONN M.CALL
M.IOFF M.OPEN

Abort Cases:

None

Output Messages:

None

3.2.55 Entry Point 55 - Allocate File or Peripheral Device

Functional Description

This entry point dynamically allocates a peripheral device, a permanent disc file, a temporary disc file, or a SLO or SBO file, and creates a File Assignment Table (FAT) entry for the allocated unit and specified logical file code. This entry point may also be used to equate a new logical file code with an existing logical file code.

Entry Conditions

Calling Sequence:

```
CALM        X'40'
(or)
M.CALL      H.MONS,55
```

Registers:

R1	denial return address
R5 byte 0	function code as follows: 1=assign file code to a user or system permanent file 2=assign file code to a system file code 3=assign file code to a peripheral device 4=assign file code to a defined file code 5=assign file code to a system permanent file only
bytes 1,2,3	file code to be assigned

Exit Conditions

Return Sequence:

M.RTRN — condition code 1 is set in the program status doubleword if the calling task has read but not write access rights to the specified permanent file

Registers:

None

(or)

Return Sequence:

M.RTRNA 1 — for denial returns if the requested file or device cannot be allocated.

Registers:

None

(or)

Return Sequence:

M.RTRNA2 — condition code 2 is set in the program status doubleword if the calling task does not have read or write access rights to the specified permanent file.

Registers:

None

External References

Error Conditions:

None

Output Messages:

None

3.2.56 Entry Point 56 - Physical Device Inquiry

Functional Description

This entry point returns to the caller physical device information describing the unit to which a specified logical file code is assigned.

Entry Conditions

Calling Sequence:

```
        CALM        X'42'
        (or)
        M.CALL      H.MONS,56
```

Registers:

R5	three character logical file code for which physical device information is requested in bytes 1, 2, and 3

Exit Conditions

Return Sequence:

```
        M.RTRN      7
```

Registers:

R7	zero, if the specified logical file code is unassigned

(or)

Return Sequence:

```
        M.RTRN      5,6,7
```

Registers:

R5	disc=number of 192-word blocks in file magnetic tape=reel identifier all other devices =0
R6	bytes 0,1= maximum number of bytes transferrable to device bytes 2,3= device mnemonic (2 ASCII characters)
R7	bits 0-5 = device type code bits 6-15 = device address bits 16-23 = system file codes as follows: 0 = not a system file 1 = SYC file 2 = SGO file 3 = SLO file 4 = SBO file

bits 24-31 = disc = number of 192-word blocks per allocation unit
= magnetic tape = volume number (0 if single volume)
= all other devices = 0

Note: If the file is a SYC or SGO file that is not open, bits 13 through 15 of R7 are returned equal to 1 or 2. All other result bits are not applicable.

External References

System Macros:

M.RTRN

Abort Cases:

None

Output Messages:

None

3.2.57 Entry Point 57 - Disable Message Task Interrupt

See Section 8.2.15 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.58 Entry Point 58 - Enable Message Task Interrupt

See Secton 8.2.18 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.59 Entry Point 59 - Get Physical Memory Contents

Function Description

This entry point forces the specified physical address to an 8 word boundary and returns the memory contents of that 8 word block to the callers 8 word buffer area, which must be on an 8 word boundary.

Entry Conditions

Calling Sequence:

```
M.CALL      H.MONS,59
```

Registers:

R1	physical address of memory
R2	caller's buffer address, must be on an 8 word boundary

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.2.60 Entry Point 60 - Change Physical Memory Contents

Functional Description

This entry point stores a given value at the physical address specified by the caller.

Entry Conditions

Calling Sequence:

M.CALL	H.MONS,60

Registers:

R1	physical address to change
R7	value to be stored

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.2.61 Entry Point 61 - Permanent File Log

Functional Description

This entry point provides a log of currently existing permanent files.

Entry Conditions

Calling Sequence:

```
CALM        X'73'
(or)
M.CALL      H.MONS,61
```

Registers:

R4 contains a byte-scaled value which specifies the type of lot to be performed as follows:

0 specifies a single named system or user file
1 specifies all permanent files
2 specifies system files only
3 specifies user files
4 specifies a single named system file

If R4 contains zero and a user name is associated with the calling program, an attempt is made to locate the user file directory entry for the given file name. If unsuccessful, the system file directory entry is located, if any. If a user name is not associated with the calling program, the file is assumed to be a system file.

If R4 contains 3 and the calling program has an associated user name, that user's files are logged or all files are logged if the calling program has no associated user name.

R5 contains the address of an eight-word area where the file directory entry is to be stored

Exit Conditions

Return Sequence:

M.RTRN 4,5

Registers:

R4 If type ='N' or "O" (R4=0 or 4), R4 is destroyed. If type = "A", "S", or "U" (R4=1,2 or 3), this entry point is called repeatedly to obtain all the pertinent file definitions. The type parameter in R4 is specified in the first call only. R4 is returned containing the address of the next directory entry to be returned. The value returned in R4 must be unchanged upon the subsequent call to this service.

R5 Contains zero if type = "N" or "O" (R4=0) and the specified file could not be located or type = "A", "S", or "U" (R4=1,2 or 3) and all pertinent files have been logged. Otherwise, R5 is unchanged.

External References

System Macros:

M.CALL M.RTRN

Abort Cases:

MS28 A permanent file log has been requested, but the address specified for storage of the directory entry is not contained within the calling task's logical address space.

MS30 Task has attempted to obtain a permanent file log in a memory-only environment.

Output Messages:

None

3.2.62 Entry Point 62 - Resourcemark Lock

See Section 7.8.22 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.63 Entry Point 63 - Resourcemark Unlock

See Section 7.8.23 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.64 Entry Point 64 - Remove RSM Lock on Task Term

Functional Description

This entry point searches the Resourcemark Table for the calling tasks program number. If found, locks belonging to the task are cleared, the task is dequeued, and the lock is given to the next task waiting for that resource.

Entry Conditions

Calling Sequence:

```
        M.CALL        H.MONS,64
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN
```

External References:

H.EXEC,36

Abort Cases:

None

Output Messages:

None

3.2.65 Entry Point 65 - Task CPU Execution Time

See Section 8.2.50 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.66 Entry Point 66 - Activate Program at Given Time of Day

See Section 8.2.44 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.67 Entry Point 67 - Set Synchronous Task Interrupt

See Section 8.2.39 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.68 Entry Point 68 - Set Asynchronous Task Interrupt

See Section 8.2.3 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.69 Entry Point 69 - Reserved

3.2.70 Entry Point 70 - Date and Time Inquiry

See Section 8.2.9 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.71 Entry Point 71 - Get Device Mnemonic or Type Code

See Section 8.2.12 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.72 Entry Point 72 - Enable User Break Interrupt

See Section 8.2.19 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.73 Entry Point 73 - Disable User Break Interrupt

See Section 8.2.16 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.2.74 Entry Point 99 - SYSGEN Initialization

Functional Description

This entry point is for internal use only and is called during SYSGEN. H.MONS sets up its Entry Point Table, then returns to SYSGEN.

3.3 System Output Module (H.SOUT)

The System Output module consists of entry points for processing listed (SLO) and punched (SBO) spooled output files. Certain entry points are designed to be called exclusively by other H.SOUT entry points and are so noted.

System Input Directory (M.SID), System Output Directory (M.SOD) and link file indexes used as parameters throughout H.SOUT consist of a word with bits 0-7 containing the entry number of the entry relative to the beginning of the 192-word block that contains the entry. This value must be shifted left two places to obtain an index relative to the beginning of the 192-word block. Bits 8-31 contain the block number relative to the beginning of the M.SID, M.SOD or link files of the 192-word block that contains the entry.

Entry Point	Description
H.SOUT,1	LINK SLO OR SBO FILE TO SYSTEM OUTPUT QUEUE
H.SOUT,2	RESERVED
H.SOUT,3	PROCESS END OF JOB
H.SOUT,4	GET AND DELETE M.SID ENTRY
H.SOUT,5	GET NEXT SLO OR SBO FILE
H.SOUT,6	LOG M.SID OR M.SOD
H.SOUT,7	ALLOCATE SLO OR SBO FILE
H.SOUT,8	PROCESS DEPRINT COMMAND
H.SOUT,9	PROCESS DEPUNCH COMMAND
H.SOUT,10	ACTIVATE SYSTEM OUTPUT TASK
H.SOUT,11	DELETE JOB'S SLO OR SBO FILES
H.SOUT,12	DELETE REAL-TIME SLO OR SBO FILE
H.SOUT,13	SYSTEM OUTPUT TASK ABORT PROCESSING
H.SOUT,14	RELINK M.SID OR M.SOD ENTRY
H.SOUT,15	FIND FREE AND WRITE M.SID OR M.SOD ENTRY
H.SOUT,16	SET SYSTEM OUTPUT TASK'S USER STATUS WORD
H.SOUT,17	GET NEXT LINKED M.SID OR M.SOD ENTRY
H.SOUT,18	GATE AND READ HEADER
H.SOUT,19	BUILD SYSTEM FAT, FPT AND FCB
H.SOUT,20	LINK M.SID OR M.SOD ENTRY
H.SOUT,21	UNLINK M.SID OR M.SOD ENTRY
H.SOUT,22	SEARCH JOB TABLE
H.SOUT,23	READ SPECIFIED M.SID OR M.SOD ENTRY
H.SOUT,24	BUILD USER DISC FAT AND FPT ENTRIES FOR A DISC FILE
H.SOUT,25	CONVERT DEVICE ADDRESS TO HEXADECIMAL
H.SOUT,26	BATCH TASK EXIT REPORTING

3.3.1 Entry Point 1 - Link SLO or SBO File To System Output Queue

Functional Description

This entry point adds the definition of a completed SLO or SBO file to a system output queue and activates a System Output task to process the file.

Entry Conditions

Calling Sequence:

```
        M.CALL          H.SOUT,1
```

Registers:

R3	Address of FAT

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

External References

Abort Cases:

SM11	Unrecoverable I/O error to the allocation map denial received from H.FISE,4 on attempted deallocation of file space.

Output Messages:

None

3.3.2 Entry Point 2 - Reserved

3.3.3 Entry Point 3 - Process End of Job

Functional Description

This entry point initiates output to destination terminal devices of a completed job's accumulated SLO and SBO files.

Entry Conditions

Calling Sequence:

```
M.CALL      H.SOUT,3
```

Registers:

R7	ASCII job sequence number

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

SM04	A job with the specified job sequence number cannot be located in the Job Table.
SM11	See H.SOUT,1

Output Messages:

None

3.3.4 Entry Point 4 - Get and Delete M.SID Entry

Functional Description

This entry point unlinks and deletes an entry from the System Input Directory, M.SID.

Entry Conditions

Calling Sequence:

```
M.CALL      H.SOUT,4
```

Registers:

R3	memory address at which 12-word unlinked entry is to be stored 0 if unlinked entry is not to be returned

R7 job sequence number in ASCII

0 to get first M.SID entry

1 to get first M.SID entry that is not sequential

Exit Conditions

Return Sequence:

M.RTRN 7

Registers:

R7 M.SID index of entry found

0 if entry was not found

External References

Abort Cases:

None

Output Messages:

None

3.3.5 Entry Point 5 - Get Next SLO or SBO File

Functional Description

This entry point returns information on the next linked SLO or SBO file defined in the System Output Directory (M.SOD) or Link File associated with a M.SOD entry.

Entry Conditions

Calling Sequence:

M.CALL H.SOUT,5

Registers:

R1 address of FCB to be allocated to the file

0 if file is not to be allocated

R6,7 index of previous file processed with R6 containing the M.SOD index and R7 containing the job link file index if previous was batch

(or)

R6 M.SOD index and

R7	0 to obtain the first SLO or SBO file of a job
(or)	
R6	0 if no previous SLO or SBO file was processed

Exit Conditions

Return Sequence:

M.RTRN 1,3,4,5,6,7

Registers:

R1	bit 0	1 if no further files remain in M.SOD to be processed
	bit 1	1 if the previous file processed was a background file and no further SLO or SBO files remain to be processed in the job
	bit 2	1 if returned file is first file in a batch job
	bit 3	1 if returned file is batch
	bit 4	1 if returned file is SLO
	bit 5	1 if returned file was assigned to a specific device
R3		job sequence number in ASCII if batch file is returned
R4,5		real-time task sequence number or job name
R6		M.SOD index of entry found
R7		Link File index if SLO or SBO file is for a batch job

If R1 is not equal to zero in the call, the file code contained in the FCB specified in R1 is returned allocated to the SLO or SBO file.

External References

Abort Cases:

SM05	The previous entry specified in the call could not be located in M.SOD

Output Messages:

None

3.3.6 Entry Point 6 - Log M.SID or M.SOD

Functional Description

This entry point returns consecutive System Input Directory (M.SID) or System Output Directory (M.SOD) entries.

Entry Conditions

Calling Sequence:

```
M.CALL          H.SOUT,6
```

Registers:

R1 address at which to store 12-word M.SID or M.SOD entry

0 if M.SID or M.SOD entry is not to be returned

R5	bit 29=1		to return with FISE gated
	bit 30=1		to clear current destination in M.SID or M.SOD entry corresponding to the returned entry
	bit 31	=0	to log M.SID
		=1	to log M.SOD

R6 index of M.SID or M.SOD entry returned from previous call

0 if first call to this entry point

Exit Conditions

Return Sequence:

```
M.RTRN          6
```

Registers:

R6 M.SID or M.SOD index of returned entry

0 if no first or next M.SID or M.SOD entry

If R1 specifies an address in the call, the M.SID or M.SOD entry is stored at the specified address.

External References

Abort Cases:

None

Output Messages:

None

3.3.7 Entry Point 7 - Allocate SLO or SBO File

Functional Description

This entry point provides the System Output task a means of allocating a previously output SLO or SBO file.

Entry Conditions

Calling Sequence:

```
M.CALL       H.SOUT,7
```

Registers:

R1	address of FCB containing file code to be allocated to the SLO or SBO file
R6	M.SOD index of SLO or SBO file
R7	Link File index of SLO or SBO file if file is batch 0 if SLO or SBO file is real-time

Exit Conditions

Return Sequence:

```
M.RTRN

(or)

M.RTRN     6
```

Registers:

Unchanged if the specified file exists with the file code in the specified FCB allocated to the file

(or)

R6 0 if the specified file does not exist

External References

Abort Cases:

SM03	FAT/FPT space is not available

Output Messages:

None

3.3.8 Entry Point 8 - Process Deprint Command

Functional Description

This entry point processes the operator command DEPRINT.

Entry Conditions

Calling Sequence:

```
M.CALL        H.SOUT,8
```

Registers:

R5 bit 31 = 0 for deprint current only

= 1 for deprint all files

For deprint current:

R6,7 destination task pseudonym

For deprint all:

R6,7 real-time task sequence number in ASCII

(or)

R6 = 0 and

R7 ASCII job sequence number

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.3.9 Entry Point 9 - Process Depunch Command

Functional Description

This entry point processes the operator command DEPUNCH.

Entry Conditions

Calling Sequence

```
M.CALL        H.SOUT,9
```

Registers:

R5 bit 31 = 0 for depunch current only

= 1 for depunch all files

For depunch current:

R6,7	destination task pseudonym

For depunch all:

R6,7	real-time task sequence number in ASCII
	(or)
R6	0 and
R7	ASCII job sequence number

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.3.10 Entry Point 10 - Activate System Output Task

Functional Description

This entry point activates System Output tasks for other resident system functions.

Entry Conditions

Calling Sequence:

```
M.CALL          H.SOUT,10
```

Registers:

Option 1 for SYSASSIGN and REPRINT operator commands:

R5 bit 30 = 1

R6,7 left justified device type code and four-character device address in ASCII

Option 2 for H.SOUT,1 and H.SOUT,3:

```
R5 bit 29 = 1
   bit 28 = 0        if SLO
          = 1        if SBO
   bit 27 = 0        if batch
          = 1        if real-time
R6,7       destination task pseudonym
(or)
R6         0 if no destination task is assigned
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort cases:

SM12 Attempt to activate System Output task unsuccessful

Output Messages:

None

3.3.11 Entry Point 11 - Delete Job's SLO or SBO Files

Functional Description

This entry point deletes the disc space occupied by a job's SLO or SBO files and notifies any System Output task processing the files if that task is not the caller.

Entry Conditions

Calling Sequence:

```
M.CALL          H.SOUT,11
```

Registers:

R6	0 if SLO non-zero if SBO
R7	job sequence number in ASCII

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

SM11	See H.SOUT,1

Output Messages:

None

3.3.12 Entry Point 12 - Delete Real-Time SLO or SBO File

Functional Description

This entry point deletes the disc space occupied by a real-time SLO or SBO file and notifies any System Output task processing the file if that task is not the caller.

Entry Conditions

Calling Sequence:

M.CALL H.SOUT,12

Registers:

R5 0 if delete only if current destination matches calling task's name

R5 non-zero if delete unconditionally

R6 M.SOD index

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

External References

Abort Cases:

SM11 See H.SOUT,1

Output Messages:

None

3.3.13 Entry Point 13 - System Output Task Abort Processing

Functional Description

This entry point is called by System Output tasks upon task abort to ensure M.SOD bookkeeping.

Entry Conditions

Calling Sequence:

M.CALL H.SOUT,13

Registers:

None

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.3.14 Entry Point 14 - Relink M.SID or M.SOD Entry

Functional Description

This entry point relinks a job's M.SID or M.SOD entry as required by a specified priority.

Entry Conditions

Calling Sequence:

```
M.CALL          H.SOUT,14
```

Registers:

R4	bit 31 = 0	if M.SID entry
	= 1	if M.SOD entry
	bit 30 = 0	if current destination not to be cleared in M.SID or M.SOD entry
	= 1	if current destination to be cleared
	bit 28 = 0	for a SLO M.SOD entry
	= 1	for a SBO M.SOD entry
	bit 26 = 0	if assigned destination not to be changed
	= 1	if value from R0,1 to be stored in M.SID or M.SOD assigned destination

R5	new priority of M.SID or M.SOD entry
R7	job sequence number in ASCII of M.SID or M.SOD entry to be relinked

Exit Conditions

Return Sequence:

```
M.RTRN
(or)
M.RTRN      7
```

Registers:

Unchanged if the specified entry was found

(or)

R7   0 if specified entry was not found

External References

Abort Cases:

None

Output Messages:

None

3.3.15 Entry Point 15 - Find Free and Write M.SID or M.SOD Entry

Functional Description

This entry point adds a new entry to M.SID or M.SOD and links the entry at its priority.

Entry Conditions

Calling Sequence:

```
M.CALL      H.SOUT,15
```

Registers:

R3	address of 11-word M.SID or M.SOD entry
R5 bit 31	0 for M.SID entry 1 for M.SOD entry

Exit Conditions

Return Sequence:

```
M.RTRN    5
```

Registers:

R5 0 if free entry was not found

M.SID or M.SOD entry index if free entry was found with the new entry linked at the priority specified in the furnished entry

External References

Abort Cases:

None

Output Messages:

None

3.3.16 Entry Point 16 - Set System Output Task's User Status Word

Functional Description

This entry point stores the specified value in a System Output task's user status word for communication with the task.

Entry Conditions

Calling Sequence:

```
M.CALL    H.SOUT,16
```

Registers:

R5 value to be set in bits 0-31 of user status word

R6,7 pseudonym of System Output task

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 0 if task is not active

Unchanged if task is active

External References

Abort Cases:

None

Output Messages:

None

3.3.17 Entry Point 17 - Get Next Linked M.SID or M.SOD Entry

Functional Description

This entry point is for internal use by H.SOUT only. No gating is done.

Entry Conditions

Calling Sequence:

```
M.CALL        H.SOUT,17
```

The system FAT, FPT, and FCB are assumed to be built. A valid M.SID or M.SOD block is assumed to be contained in the system buffer.

Registers:

R3 memory address of M.SID or M.SOD entry returned from previous call to this entry point

0 on first call to this entry point

R4 index to first M.SID or M.SOD entry from header entry on first call to this entry point

Exit Conditions

Return Sequence:

```
M.RTRN        3,5
```

Registers:

R3 memory address of next linked M.SID or M.SOD entry. Entry is contained in system buffer

0 if none

R5 M.SID or M.SOD entry index, if any

External References

Abort Cases:

None

Output Messages:

None

3.3.18 Entry Point 18 - Gate and Read Header

Functional Description

This entry point is for internal use by H.SOUT only. No gating is done.

Entry Conditions

Calling Sequence:

```
M.CALL        H.SOUT,18
```

Registers:

R5 bit 31	0 for M.SID
	1 for M.SOD

Exit Conditions

Return Sequence:

```
M.RTRN        3,4
(or)
M.RTRN        3,4,5,6,7
```

Registers:

R3,4	0 if M.SID or M.SOD does not exist
	(or)
R3	non-zero if M.SID or M.SOD exists and
R4-7	first four words of M.SID or M.SOD header entry

If M.SID/M.SOD exists, the system buffer contains its first 192-word block.

Return is with FISE gated.

External References

Abort Cases:

None

Output Messages:

None

3.3.19 Entry Point 19 - Build System FAT, FPT, and FCB

Functional Description

This entry point is for internal use by H.SOUT only.

Entry Conditions

Calling Sequence:

```
M.CALL      H.SOUT,19
```

Registers:

R4 bit 30 1 if R5-7 contain the file's disc space definition

bit 31 0 if M.SID file

1 if M.SOD file

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

I/O error return addresses are inserted in FCBs to abort as follows on unrecoverable I/O errors encountered when these FCBs are used.

SM08 Unrecoverable I/O error on file whose disc space definition was furnished.

SM09 Unrecoverable I/O error on System Input Directory, M.SID.

SM10 Unrecoverable I/O error on System Output Directory, M.SOD.

External References

Abort Cases:

SM01 Call specified M.SID, but M.SID is not defined in the System Master Directory (SMD).

SM02 Call specified M.SOD, but M.SOD is not defined in the SMD.

SM13 Unrecoverable I/O error to the SMD received from H.FISE,1.

Output Messages:

None

3.3.20 Entry Point 20 - Link M.SID or M.SOD Entry

Functional Description

This entry point is for internal use by H.SOUT only. No gating is done.

Entry Conditions

Calling Sequence:

```
M.CALL        H.SOUT,19
```

Registers:

R4 bit 30 = 1 to clear current destination field in entry to be linked

bit 31 = 0 if M.SID entry

= 1 if M.SOD entry

bit 26 = 0 if assigned destination not to be changed

= 1 if value from R0,1 to be stored in assigned destination

R5 priority of entry to be linked

R7 M.SID or M.SOD entry index

The system FAT, FPT and FCB must be built for M.SID/M.SOD prior to calling this entry point.

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

SM06	Entry linkage has been destroyed on the System Input Directory, M.SID
SM07	Entry linkage has been destroyed on the System Output Directory, M.SOD

Output Messages:

None

3.3.21 Entry Point 21 - Unlink M.SID or M.SOD Entry

Functional Description

This entry point is for internal use by H.SOUT only. No gating is done.

Entry Conditions

Calling Sequence:

```
M.CALL        H.SOUT,21
```

Registers:

R3	memory address at which 12-word unlinked entry is to be stored
	0 if unlinked entry is not to be returned
R4 bit 25	1 if unlinked entry is to be cleared to zero
bit 31	0 if M.SID entry
	1 if M.SOD entry

For M.SID entries, the following additional information is contained in R4:

bit 28	0 to return sequential or non-sequential entry
	1 to return non-sequential entry only. If set, R6,7 must be zero

For M.SOD entries in calls where the M.SID or M.SOD index is not specified in R6, the following additional information is contained in R4:

bit 28	0 for a SLO entry
	1 for a SBO entry

R6,7 contain the identification of the entry to be unlinked in one of the following forms:

R6 M.SID or M.SOD index

(or)

R6 0 and

R7 job sequence number in ASCII

(or)

R6 0 and

R7 0 to unlink the first M.SID or M.SOD entry

The system FAT, FPT and FCB must be built for M.SID/M.SOD prior to calling this entry point.

Exit Conditions

Return Sequence:

```
M.RTRN        7
```

Registers:

R7 the entry's M.SID or M.SOD index if the specified entry was found

0 if the specified entry was not found

External References

Abort Cases:

None

Output Messages:

None

3.3.22 Entry Point 22 - Search Job Table

Functional Description

This entry point locates a specified Job Table entry.

Entry Conditions

Calling Sequence:

```
M.CALL         H.SOUT,22
```

Registers:

R7 job sequence number in ASCII

Exit Conditions

Return Sequence:

```
M.RTRN         3
```

Registers:

R3 memory address of Job Table entry

0 if Job Table entry was not found

External References

Abort Cases:

None

Output Messages:

None

3.3.23 Entry Point 23 - Read Specified M.SID or M.SOD Entry

Functional Description

This entry point for internal use by H.SOUT only. No gating is done.

Entry Conditions

Calling Sequence:

```
M.CALL         H.SOUT,23
```

Registers:

R0 bit 31	0 for M.SID entry
	1 for M.SOD entry
R1	M.SID or M.SOD entry index

(or)

0 to read first M.SID or M.SOD entry (header entry)

Exit Conditions

Return Sequence:

```
M.RTRN       1
```

Registers:

R1	memory address (in System Buffer) of entry
	0 if specified entry does not exist

External References

Abort Cases:

None

Output Messages:

None

3.3.24 Entry Point 24 - Build User Disc FAT and FPT Entries for a Disc File

Functional Description

This entry point simulates the allocation function by building FAT and FPT entries for a disc file.

Entry Conditions

Calling Sequence:

```
M.CALL       H.SOUT,24
```

Registers:

R4 bit 0	0	if permanent file
	1	if temporary file
bit 1	0	if blocked file
	1	if unblocked file

R4 file code in bytes 1-3

R5,6,7 disc space definition

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 non-zero if blocking buffer and FAT space is available

0 if blocking buffer or FAT space is unavailable

External References

Abort Cases:

None

Output Messages:

None

3.3.25 Entry Point 25 - Convert Device Address to Hexadecimal

Functional Description

This entry point is for internal use by H.SOUT only.

Entry Conditions

Calling Sequence:

```
M.CALL      H.SOUT,25
```

Registers:

R2 UDT entry address

Exit Conditions

Return Sequence:

```
M.RTRN      4,5
```

Registers:

R4 two-character device mnemonic in bytes 3 and 4 in ASCII

R5 four-character device address in ASCII

External References

Abort Cases:

None

Output Messages:

None

3.3.26 Entry Point 26 - Batch Task Exit Reporting

Functional Description

This entry point is used by the Executive for reporting batch task exits, aborts, and deletes.

Entry Conditions

Calling Sequence:

```
M.CALL          H.SOUT,26
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

Abort Cases:

None

Output Messages:

None

3.3.27 Entry Point 99 - SYSGEN Initialization

Functional Description

This entry point is for internal use only and is called during SYSGEN. H.SOUT sets up its Entry Point Table, then returns to SYSGEN.

3.4 Input/Output Control System (H.IOCS)

Entry Point Summary

ENTRY POINT	DESCRIPTION
H.IOCS,1	OPEN FILE
H.IOCS,2	REWIND FILE
H.IOCS,3	READ RECORD
H.IOCS,4	WRITE RECORD
H.IOCS,5	WRITE END OF FILE
H.IOCS,6	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,7	ADVANCE RECORD
H.IOCS,8	ADVANCE FILE
H.IOCS,9	BACKSPACE RECORD
H.IOCS,10	EXECUTE CHANNEL PROGRAM
H.IOCS,11	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,12	RESERVE CHANNEL
H.IOCS,13	RELEASE CHANNEL
H.IOCS,14	SYSTEM CONSOLE TYPE
H.IOCS,15	SUSPEND USER UNTIL I/O COMPLETE
H.IOCS,16	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,17	GET MEMORY POOL BUFFER
H.IOCS,18	RESERVED FOR INTERNAL USE BY. H.IOCS
H.IOCS,19	BACKSPACE FILE
H.IOCS,20	UPSPACE
H.IOCS,21	ERASE OR PUNCH TRAILER
H.IOCS,22	EJECT/PURGE ROUTINE
H.IOCS,23	CLOSE FILE
H.IOCS,24	RESERVE FHD PORT
H.IOCS,25	WAIT
H.IOCS,26	CONSOLE WAIT
H.IOCS,27	RELEASE FHD PORT
H.IOCS,28	ABSOLUTIZE TCW FOR CLASS 'E' DEVICES
H.IOCS,29	HANDLER ENTRY POINT 5 AND 2 INTERFACE
H.IOCS,30	ADJUST TCW FORMAT TO BYTES
H.IOCS,31	ADJUST TCW FORMAT TO HALFWORDS
H.IOCS,32	ADJUST TCW FORMAT TO WORDS
H.IOCS,33	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,34	NO-WAIT I/O END ACTION RETURN
H.IOCS,35	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,36	RESTART I/O
H.IOCS,37	VIRTUAL ADDRESS VALIDATE
H.IOCS,38	KILL ALL OUTSTANDING I/O
H.IOCS,39	SPECIAL DISCONTIGUOUS MEMORY CHECK
H.IOCS,40	BUILD IOCD'S FOR EXTENDED I/O READS AND WRITES
H.IOCS,41	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,42	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,43	RESERVED FOR INTERNAL USE BY H.IOCS
H.IOCS,99	SYSGEN INITIALIZATION

Subroutine Summary

SUBROUTINE	DESCRIPTION
S.IOCS1	POST I/O PROCESSING
S.IOCS2	PERFORM DEVICE TESTING
S.IOCS3	UNLINK I/O QUEUE FROM CDT
S.IOCS4	HALF-ASCII TO FULL ASCII CONVERSION
S.IOCS5	PERIPHERAL TIME OUT
S.IOCS6	BUFFER TO BUFFER MOVE ROUTINE (BYTE)
S.IOCS7	BUFFER TO BUFFER MOVE ROUTINE (WORD)
S.IOCS8	BUFFER TO BUFFER MOVE ROUTINE (DOUBLEWORD)
S.IOCS9	I/O HANDLER ABORT
S.IOCS10	DELETE I/O QUEUE AND OS BUFFER
S.IOCS11	GPMC DEVICE STATUS
S.IOCS12	STORE IOCD'S FOR EXTENDED I/O
S.IOCS13	ALLOCATE I/O QUEUE AND BUFFER SPACE
S.IOCS14	RESERVED
S.IOCS15	DELETE I/O QUEUE AND OS BUFFER
S.IOCS16	FIND FPT
S.IOCS17	LINK FAT
S.IOCS18	INITIALIZE BLOCKING BUFFER
S.IOCS19	GET SYC/SGO SPACE DEFINITION
S.IOCS20	GET DATA ADDRESS AND TRANSFER COUNT
S.IOCS21	READ LOGICAL BLOCKED RECORD
S.IOCS22	REPORT BLOCKED I/O ERROR
S.IOCS23	POST PROCESS NON-DEVICE ACCESS I/O
S.IOCS24	RESTORE FCB PARAMETERS FROM IOQ
S.IOCS25	SAVE FCB PARAMETERS IN SPAD
S.IOCS26	WRITE LOGICAL BLOCKED RECORD
S.IOCS27	PERFORM IMPLICIT OPEN
S.IOCS28	INITIALIZE IOQ ENTRY
S.IOCS29	REPORT I/O COMPLETE
S.IOCS30	ADVANCE LOGICAL BLOCKED RECORD
S.IOCS31	MARK UNITS OFFLINE
S.IOCS32	RESTORE FCB PARAMETERS FROM SPAD
S.IOCS33	UPDATE DISC FAT
S.IOCS34	ALLOCATE VARIABLE IOQ ENTRY

3.4.1 Entry Point 1 - Open File

See Section 7.8.9 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.2 Entry Point 2 - Rewind File

See Section 7.8.25 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.3 Entry Point 3 - Read Record

See Section 7.8.18 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.4 Entry Point 4 - Write Record

See Section 7.8.31 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.5 Entry Point 5 - Write End Of File

See Section 7.8.30 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.6 Entry Point 6 - Reserved for internal use by H.IOCS

Write blocked End of File - this routine called internally by Entry Point 5.

3.4.7 Entry Point 7 - Advance Record

See Section 7.8.12 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.8 Entry Point 8 - Advance File

See Section 7.8.12 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.9 Entry Point 9 - Backspace Record

See Section 7.8.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.10 Entry Point 10 - Execute Channel Program

See Section 7.8.34 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.11 Entry Point 11 - Reserved (unused)

3.4.12 Entry Point 12 - Reserve Channel

See Section 7.8.24 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.13 Entry Point 13 - Release Channel Reservation

See Section 7.8.21 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.14 Entry Point 14 - OPCOM Console Type

See Section 7.8.26 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.15 Entry Point 15 - Suspend User Until I/O Complete

Functional Description

Suspend checks the operation in progress bit in the FCB. If the bit is set, suspend calls the executive to wait until I/O completes. If the bit is reset, suspend performs a M.RTRN immediately.

Entry Conditions

Calling Sequence:

```
        M.CALL      H.IOCS,15
```

Registers:

R1	FCB address

NOTE

> The FCB address must be the address of the same FCB used to initiate the transfer on which the I/O suspend is being made.

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

Abort Cases:

None

Output Messages:

None

3.4.16 Entry Point 16 - Reserved (unused)

3.4.17 Entry Point 17 - Get Memory Pool Buffer

Functional Description

This entry point is used to obtain chunks of memory from the system memory pool. The maximum amount of core to allocate is 192 words. All core can be optionally zeroed before returning to the calling task.

If core is not available, the calling task will be suspended (via H.EXEC,6) until available.

Note that all core returned has the attribute that its virtual address is the same as its absolute address.

Entry Conditions

Calling Sequence:

```
M.CALL     H.IOCS,17
```

Registers:

R0	Bit 0, set if zeroing of core desired
R0	Bit 0, reset if no zeroing desired
R7	number of words to allocate

Exit Conditions

Return Sequence:

```
M.RTRN     R6,R7
```

Registers:

R6	start virtual (same as absolute) address
R7	actual number of words in buffer (may be more than requested amount)

Abort Cases:

None

Output Messages:

None

3.4.18 Entry Point 18 - Reserved (unused)

3.4.19 Entry Point 19 - Backspace File

See Section 7.8.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.20 Entry Point 20 - Upspace

See Section 7.8.27 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.21 Entry Point 21 - Erase Or Punch Trailer

Functional Description

The volume record is written if BOT on multi-volume magnetic tape; and ERASE/WRITE EOF performed if BOT on multi-volume magnetic tape.

Erase, punch trailer is not applicable to blocked or system files (i.e., SYC, SGO, SLO, SBO).

Entry Conditions

Calling Sequence:

```
        M.CALL      H.IOCS,21

        (or)

        SVC         1,X'3E'
```

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

Abort Cases:

IO12	File not opened in write mode.

IO13	Illegal operation on system file.

Output Messages:

MOUNT/DISMOUNT messages if EOT on multi-volume magnetic tape

3.4.22 Entry Point 22 - Eject/Purge Routine

Functional Description

This entry point performs the following functions:

If the file is blocked and output active, a purge is issued. Return is made to the user.

Writes volume record if BOT on multi-volume magnetic tape.

Performs ERASE/WRITE EOF if EOT on multi-volume magnetic tape.

Eject is not applicable to system files (i.e., SYC, SGO, SLO, SBO).

Entry Conditions

Calling Sequence:

```
M.CALL     H.IOCS,22
```

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

IO13	Illegal operation on system file.

Output Messages:

MOUNT/DISMOUNT messages if EOT on multi-volume magnetic tape

3.4.23 Entry Point 23 - Close File

See Section 7.8.3 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.24 Entry Point 24 - Reserve Dual Ported Disc/Reserve FHD Port

See Sections 7.8.20 and 7.8.36 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.25 Entry Point 25 - Wait I/O

See Section 7.8.29 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.26 Entry Point 26 - System Console Wait

See Section 7.8.5 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.27 Entry Point 27 - Release Dual Ported Disc/Release FHD Port

See Sections 7.8.19 and 7.8.35 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.28 Entry Point 28 - Absolutize TCW for Class 'E' Devices

Functional Description

Absolutize transfer control word for class 'E' devices.

Entry Conditions

Calling Sequence:

```
        M.CALL      H.IOCS,28
```

Registers:

R1	FCB address
R6	TCW associated with class 'E' device with address virtual

Exit Conditions

Return Sequence:

```
        M.RTRN      R6
```

Registers:

R6	TCW associated with class 'E' device with address absolutized

Abort Cases:

IO47	TCW address not within class 'E' memory.

Output Messages:

None

3.4.29 Entry Point 29 - Handler Entry Point 5 and 2 Interface

Functional Description

This entry point performs the following functions:

Places I/O request in a prioritized queue.

Branches to appropriate executive entry point to report type of I/O initiated.

For Wait I/O, branches to I/O post processing; for No-Wait I/O, returns immediately to the user.

Entry Conditions

Calling Sequence:

M.CALL	H.IOCS,29	used internally to H.IOCS by blocked I/O routines
(or)		
BU	H29	used by internal H.IOCS routines to complete normal I/O processing

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

See Return Sequence for S.IOCS1 (Section 3.4.45).

3.4.30 Entry Point 30 - Adjust TCW Format to Bytes

Functional Description

This entry point adjusts the transfer control word (TCW) to bytes. The adjusted quantity is clamped so as to not exceed the maximum quantity specified.

Entry Conditions

Calling Sequence:

```
M.CALL    H.IOCS,30
```

Registers:

R1	FCB address
R6	TCW
R7	maximum quantity

Exit Conditions

Return Sequence:

```
M.RTRN    4,6
```

Registers:

R4	adjusted quantity
R6	adjusted TCW

Abort Cases:

IO24	Boundary error.

Output Messages:

None

3.4.31 Entry Point 31 - Adjust TCW Format to Halfwords

Functional Description

This entry point adjusts the transfer control word (TCW) to halfwords. The adjusted quantity is clamped so as not to exceed the maximum quantity specified.

Entry Conditions

Calling Sequence:

```
M.CALL    H.IOCS,31
```

Registers:

R1	FCB address
R6	TCW
R7	maximum quantity

Exit Conditions

Return Sequence:

```
        M.RTRN      4,6
```

Registers:

R4	adjusted quantity
R6	adjusted TCW

Abort Cases:

IO24	Boundary error.

Output Messages:

None

3.4.32 Entry Point 32 - Adjust TCW Format to Words

Functional Description

This entry point adjusts the transfer control word (TCW) to words. The adjusted quantity is clamped so as not to exceed the maximum quantity specified.

Entry Conditions

Calling Sequence:

```
        M.CALL      H.IOCS,32
```

Registers:

R1	FCB address
R6	TCW
R7	maximum quantity

Exit Conditions

Return Sequence:

```
        M.RTRN      4,6
```

Registers:

R4	adjusted quantity
R6	adjusted TCW

Abort Cases:

IO24	Boundary error.

Output Messages:

None

3.4.33 Entry Point 33 - Reserved for internal use by H.IOCS

Read blocked record service, called internally by Entry Point 3 - read record.

3.4.34 Entry Point 34 - No Wait I/O End Action Return

See Section 7.8.32 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.4.35 Entry Point 35 - Reserved for internal use by H.IOCS.

Write blocked record service, called internally by Entry Point 4 - write record.

3.4.36 Entry Point 36 - Restart I/O

Functional Description

This entry point is used to restart I/O for devices where no-wait I/O incurred error or wait I/O retry aborted.

It is also used to restart I/O after an I/O channel is released back to the system.

Entry Conditions

Calling Sequence:

```
M.CALL    H.IOCS,36
```

Registers:

R0	I/O queue address (from CDT.FIOQ)

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

None

Output Messages:

None

3.4.37 Entry Point 37 - Virtual Address Validate

Functional Description

This entry point verifies that a given virtual start address through an optional transfer length is within a user's legal limits of program execution.

Entry Conditions

Calling Sequence:

```
        M.CALL      H.IOCS,37
```

Registers:

R6	bits 0-11	ignored
	bits 12-31	virtual start address
R7	bits 0-15	ignored
	bits 16-31	transfer length in bytes

Exit Conditions

Return Sequence:

```
        M.RTRN      6
```

Registers:

R6	virtual start address (same as on entry)
	(or)
	0 (transfer outside legal limits)

Abort Cases:

None

Output Messages:

None

3.4.38 Entry Point 38 - Kill All Outstanding I/O

Functional Description

This entry point is used to terminate all outstanding I/O for the current executing task.

Peripheral time-out is forced for pending I/O whereas queued I/O is removed from the CDT string. Appropriate status will be set to indicate either device time-out or "I/O killed" respectively.

Entry Conditions

Calling Sequence:

```
          M.CALL      H.IOCS,38
```

Registers:

None

Exit Conditions

Return Sequence:

```
          M.RTRN
```

Registers:

None

Abort Cases:

None

Output Messages:

None

3.4.39 Entry Point 39 - Discontiguous E-Memory Data Address Check

Functional Description

This entry point ensures that a given virtual data transfer is within E-memory and will not cross discontiguous memory blocks base on an optional transfer length.

Entry Conditions

Calling Sequence:

```
          M:CALL      H.IOCS,39
```

Registers:

R6	bit 0-11	ignored
	bits 12-31	virtual start address
R7	bit 0-15	ignored
	bits 16-31	transfer length in bytes

Exit Conditions

Return Sequence:

```
          M.RTRN      0
```

Registers:

R0	0, if transfer address are out of E-memory, or cross discontiguous memory blocks
	(or)
RC	not equal to 0, if transfer address are within contiguous E-memory

Note:	R0=1,	Operating system portion of E-memory (virtual = absolute)
	R0=2,	Not in operating system portion of E-memory (virtual = absolute)

Abort Cases:

None

Output Messages:

None

3.4.40 Entry Point 40 - Build IOCD's for Extended I/O Reads and Writes

Functional Description

Breaks down IOCD into two or more transfers and sets data chaining bit in IOCD if discontiguous.

Absolutes IOCD address.

Stores IOCD's into IOCD buffer within I/O queue and increments IOCD buffer address as required.

Entry Conditions

Calling Sequences:

```
M.CALL        H.IOCS,40
```

Registers:

R3	I/O queue address
R6	IOCD word 1 with command only (Bits 8-31 = 0).
R7	IOCD word 2 with flags only (Bits 8-31 = 0).

Note: The 20-bit virtual data start address is in cell IOQ.FCT2 and the adjusted transfer count is in cell IOQ.FCT3.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO46	Dynamic storage space for IOCD's within IOQ exhausted.

Output Messages:

None

3.4.41 Entry Point 41 - Reserved (unused)

3.4.42 Entry Point 42 - Reserved (unused)

3.4.43 Entry Point 43 - Reserved (unused)

3.4.44 Entry Point 99 - SYSGEN Initialization

Functional Description

Performs any required H.IOCS initialization at SYSGEN time.

If no extended I/O devices are configured, all associated H.IOCS entry points are overlaid. (Note: Because of this feature, only extended I/O routines are allowed at the end of H.IOCS.)

Entry Conditions

Calling Sequence:

Branch to H.IOCS initialization
Entry Point "H.IOCS. I"

Registers:

R5	HAT address of GPMC device handler if applicable (e.g., H.ASMP)
R6	UDT address, if applicable

R7	bits 0-7	hardware priority level, if applicable
	bits 8-31	CDT address, if applicable

Exit Conditions

Return Sequence:

```
M.XIR    H.IOCS        (Special SYSGEN Initialization Termination Macro)
```

NOTE

> If no extended I/O devices are configured, the initialization entry pointer is updated with a new pointer such that extended I/O related H.IOCS entry points are overlaid.

Registers:

Same as on entry.

Abort Cases:

None

Output Messages:

None

3.4.45 Subroutine S.IOCS1 - Post I/O Processing

Functional Description

This routine is entered via a branch and link directly after a wait I/O request completes or indirectly as a Task Interrupt service when a no-wait I/O request completes.

If wait I/O completed with errors, applicable error messages are output and I/O retry attempted unless error processing is inhibited. If error processing is not applicable, an abort message is output or the error return address is taken. Note that wait I/O would take the error return in FCB word 6.

For the no-wait I/O requests or wait I/O requests, appropriate data conversions and buffer transfers from system buffers to user buffers are performed. Any system buffer which had been allocated and the I/O queue itself are deallocated.

If no-wait I/O completed with errors, the no wait error end action address in word 14 of the FCB is honored if present. The user must return via H.IOCS,34 to exit his error end action service. If no error end action address is present, status information is posted in the FCB and a return is made to the user. Note that if an abort of this task had been issued, the end action routine will not be called and a status bit will be set to indicate this.

For wait I/O with errors for which retry is applicable yet the operator aborted, all system buffers and the I/O queue are deallocated and the I/O queue is unlinked from the CDT.

For no-wait I/O that completes without error, the no-wait normal end action address in word 13 of the FCB will be honored if present. The user must return via H.IOCS,34 to exit his normal end action service. Note that if an abort had been issued for this task, the end action routine will not be called and a status bit will be set to indicate this.

Wait I/O that completes without error always returns back one word beyond the original point of call.

Entry Conditions

Calling Sequence:

```
BL     S.IOCS1                (wait I/O)

(or)

LPSD  (I/O queue + 4W)        (no-wait I/O)
```

Registers:

```
R3     I/O queue address
```

Exit Conditions

Return Sequence:

```
M.RTRN                           (normal wait I/O completion; or error
                                 processing inhibited.
(or)

MRTNA  REGISTER                  (error return, wait I/O) (address from FCB
                                 word 6)
(or)

X1     FCB ADDRESS
BL     ERROR-END-ACTION          (no-wait, complete with error) (address
                                 from FCB.NWER)

                                 (Note: User exits via H.IOCS,34)
(or)

X1     FCB ADDRESS
BL     NORMAL-END-ACTION(no-wait, complete without error) (address
                                 from FCB.NWOK)

                                 (Note: User exits via H.IOCS,34)

M.CALL H.EXEC,12                 (normal no-wait I/O completion; or error
                                 processing inhibited; or no end-action
                                 addresses specified)
```

(Note: Post I/O processing will be reinitiated when I/O completes)

wait I/O retry:

(or)

M.CALL H.EXEC,1	wait I/O interactive input
H.EXEC,2	wait I/O terminal output
H.EXEC,3	wait I/O (not interactive input or terminal output)

(Note: Post I/O processing will be reinitiated when I/O completes)

Registers:

None unless noted.

Abort Cases:

IO21	Unrecoverable I/O error.

Output Messages:

*DTCHSA INOP: R,A?

where:

DT	device type code (e.g., LP)
CH	channel number
SA	subaddress

I/O ERR DEVICE: DTCHSA STATUS (XXCCDDDD)=ZZZZZZZZ LFC KKK

where:

DT	device type code
CH	channel number
SA	subaddress
ZZZZZZZZ	actual status returned
KKK	logical file code associated with I/O

3.4.46 Subroutine S.IOCS2 - Perform Device Testing

Functional Description

This routine performs device testing for applicable class 'E' devices. TD 8000, TD 4000 and TD 2000 test level commands are issued and status returned in the I/O queue.

Entry Conditions

Calling Sequence:

BL	S.IOCS2

Registers:

R1	I/O queue address
R2	CDT address
R3	address of test device instructions (TD 8000 followed by TD 4000 and TD 2000)

Exit Conditions

Return Sequence:

TRSW	R0 (or) TRSW R7

Registers:

R1,R2,R3	undisturbed
R6,R7	destroyed

Abort Cases:

None

Output Messages:

None

3.4.47 Subroutine S.IOCS3 - Unlink I/O Queue from CDT

Functional Description

This routine unlinks the just complete I/O queue entry from the CDT active I/O queue string.

Entry Conditions

Calling Sequence:

BL	S.IOCS3

Registers:

R2	I/O queue address
R3	CDT address

Exit Conditions

Return Sequence:

```
        TRSW      R0
```

Registers:

R1	CDT address
R2,R3	undisturbed
R4,R6,R7	destroyed

Abort Cases:

None

Output Messages:

None

3.4.48 Subroutine S.IOCS4 - Half ASCII to Full ASCII Conversion

Functional Description

This routine takes half ASCII input and translates each byte into full ASCII code. The full ASCII data is put directly back into the half ASCII buffer.

Entry Conditions

Calling Sequence:

```
        BL        S.IOCS4
```

Registers:

R2	address of half ASCII buffer
R4	negative number of bytes to convert

Exit Conditions

Return Sequence:

```
        TRSW      R0
```

Registers:

```
        R2,R3,R4,R6     destroyed
```

Abort Cases:

None

Output Messages:

None

3.4.49 Subroutine S.IOCS5 - Peripheral Time Out

Functional Description

This routine performs peripheral time out checking for all devices with I/O outstanding.

This routine is entered every timer unit and will branch to device handler entry point 4 (Lost Interrupt) for processing if the time limit is exceeded.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS5
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

```
        R1,R2,R3,R4,R5,R6,R7destroyed
```

Abort Cases:

None

Output Messages:

None

3.4.50 Subroutine S.IOCS6 - Buffer to Buffer Move Routine (Byte)

Functional Description

This routine moves the contents of one buffer to another buffer, one byte at a time.

Entry Conditions

Calling Sequence:

```
        BL        S.IOCS6
```

Registers:

R1 from buffer byte address

R2 to buffer byte address

R4 negative number of bytes to move

Exit Conditions

Return Sequence:

```
        TRSW      R0
```

Registers:

R1,R2,R4,R6 destroyed

Abort Cases:

None

Output Messages:

None

3.4.51 Subroutine S.IOCS7 - Buffer to Buffer Move Routine (Word)

Functional Description

This routine moves the contents of one buffer to another buffer, one word at a time.

Entry Conditions

Calling Sequence:

```
        BL        S.IOCS7
```

Registers:

R1	from buffer word address
R2	to buffer word address
R4	negative number of words to move

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2,R4,R6 destroyed

Abort Cases:

None

Output Messages:

None

3.4.52 Subroutine S.IOCS8 - Buffer to Buffer Move Routine (Doubleword)

Functional Description

This routine moves the contents of one buffer to another buffer, one doubleword at a time.

Entry Conditions

Calling Sequence:

```
BL        S.IOCS8
```

Registers:

R1	from buffer doubleword address
R2	to buffer doubleword address
R4	negative number of doublewords to move

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2,R4,R6	destroyed

Abort Cases:

None

Output Messages:

None

3.4.53 Subroutine S.IOCS9 - I/O Handler Abort

Functional Description

This routine is used by all I/O handlers which encounter fatal input parameter errors during op-code (entry point 5) processing.

Entry Conditions

Calling Sequence:

```
BL          S.IOCS9
```

Registers:

R1	File Control Block Address
R5	ASCII Abort Code

Exit Conditions

Return Sequence:

```
M.CALL      H.MONS,28(abort with extended message)
```

3.4.54 Subroutine S.IOCS10 - Delete I/O Queue and OS Buffer

Functional Description

This routine deallocates the I/O queue and any system memory pool core areas used during I/O.

Entry Conditions

Calling Sequence:

```
BL          S.IOCS10
```

Registers:

R3	I/O Queue address

Exit Conditions

Return Sequence:

TRSW	R6

Registers:

R1	FCB address

R0,R2,R3,R4,R5,R6,R7destroyed

Abort Cases:

None

Output Messages:

None

3.4.55 Subroutine S.IOCS11 - GPMC Device Status

Functional Description

This routine performs device status testing for GPMC devices. A test device command is issued and status is returned.

As of MPX-32 Release 1.4, this subroutine is no longer physically a part of IOCS. It is loaded as a separate module only if the user SYSGENs using pre-1.4 GPMC support (i.e., MUX=QGPMC). For details on GPMC support, see Chapter 13.

Entry Conditions

Calling Sequence:

BL	S.IOCS11

Registers:

R0	Return address
R2	IOQ entry address
R3	Device context block
R4	Status mask

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R0,R1,R2,R3,R4	same as on entry
R5	address of CDT
R6	specific TD - TIO command
R7	device status in right halfword

Abort Cases:

None

Output Messages:

None

3.4.56 Subroutine S.IOCS12 - Store IOCD's for Extended I/O

Functional Description

This routine dynamically stores IOCD's into the I/O queue as required during extended I/O request processing.

Entry Conditions

Calling Sequence:

BL	S.IOCS12

Registers:

R0	Return address
R3	I/O queue address
R6	IOCD most significant word
R7	IOCD least significant word

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R0,R1,R2,R3,R4,R6,R7same as on entry

R5	last dynamic IOCD location

Abort Cases:

IO38	Dynamic storage space for IOCD's within IOQ exhausted.

Output Messages:

None

3.4.57 Subroutine S.IOCS13 - Allocate I/O Queue and Buffer Space

Functional Description

This routine must be called by entry point 5 of device handlers processing operation codes for which I/O queue entries are required (i.e., those operations which result in a device access).

Entry Conditions

Calling Sequence:

```
BL          S.IOCS13
```

Registers:

R1	FCB address
R2	FAT address
R3	CDT address

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

Unchanged

Abort Cases:

IO33	Unprivileged user data buffer not in user's area.

Output Messages:

None.

3.4.58 Subroutine S.IOCS14 - Reserved

3.4.59 Subroutine S.IOCS15 - Delete I/O Queue and OS Buffer

Functional Description

This routine deallocates the I/O queue and any system memory pool core areas used during I/O. It is called from the handlers as a result of an OPCOM KILL request.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS15
```

Registers:

```
        R3          I/O queue address
```

Exit Conditions

Return Sequence:

```
        TRSW        R6
```

Registers:

```
        R1              IOQ address

        R0,R2,R3,R4,R5,R6,R7destroyed
```

Abort Cases:

None.

Output Messages:

None.

3.4.60 Subroutine S.IOCS16 - Find FPT

Functional Description

This routine is used to find a FPT entry in the user's TSA with a logical file code which matches that in the user's FCB.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS16
```

Registers:

R1	FCB address
R3	TSA address
R4	X'00FFFFFF' (mask value)
R5	lfc from FCB

Exit Conditions

Return Sequence:

TRSW R0	no matching lfc
R0+1W	match and FPT open
R0+2W	match and FPT closed

Registers:

R1	FCB address
R3	FPT address
R5	lfc from FPT
R2,R6,R7	destroyed

3.4.61 Subroutine S.IOCS17 - Link FAT

Functional Description

This routine is used to link a FAT to a FCB and a FCB to a FPT.

Entry Conditions

Calling Sequence:

BL	S.IOCS17

Registers:

R1	FCB address
R3	FPT address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R3	unchanged
R2	FAT address
R4	destroyed

3.4.62 Subroutine S.IOCS18 - Initialize Blocking Buffer

Functional Description

This routine is used to initialize a blocking buffer.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS18
```

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R3,R4	destroyed

3.4.63 Subroutine S.IOCS19 - Get SYC/SGO Space Definition

Functional Description

This routine is used to retrieve the address of a SYC or SGO file space definition.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS19
```

Registers:

R4	FAT system file code

Exit Conditions

Return Sequence:

```
          TRSW      R0
```

Registers:

R2	address of SYC/SGO file space definition
R5,R7	destroyed

3.4.64 Subroutine S.IOCS20 - Get Data Address and Transfer Count

Functional Description

This routine extracts the user's data address and transfer count from an 8 or 16 word FCB. The extracted transfer count is always in bytes and the extracted data address is always a pure address (no F and C bits). The transfer count is clamped to the maximum value for the device or transfer type.

Entry Conditions

Calling Sequence:

```
          BL        S.IOCS20
```

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

```
          TRSW      R0
```

Registers:

R3,R4	destroyed
R6	data address
R7	transfer count

3.4.65 Subroutine S.IOCS21 - Read Logical Blocked Record

Functional Description

This routine performs a read of a logical blocked record, i.e., it transfers a logical blocked record from a blocking buffer to a user's data area.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS21
```

Registers:

R1	FCB address
R2	next read/write address
R3	blocking buffer address

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R1	FCB address
R2,R3,R4,R5,R6,R7destroyed	

3.4.66 Subroutine S.IOCS22 - Report Blocked I/O Error

Functional Description

This routine is used to report an error on a blocked I/O operation.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS22
```

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R1	FCB address
R3,R4	destroyed

3.4.67 Subroutine S.IOCS23 - Post Process Non-Device Access I/O

Functional Description

This routine post processes non-device access I/O, i.e., logical blocked I/O requests and other I/O requests for which no device access occurs.

Entry Conditions

Calling Sequence:

BU	S.IOCS23

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

None

Registers:

None

3.4.68 Subroutine S.IOCS24 - Restore FCB Parameters from IOQ

Functional Description

This routine restores original FCB parameters following a physical I/O done on behalf of a user who requested blocked I/O.

Entry Conditions

Calling Sequence:

BL	S.IOCS24

Registers:

R1	FCB address
R3	I/O queue address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1	FCB address
R2	address of saved parameters
R3	I/O queue address

R4	users original opcode
R7	special status byte
R6	destroyed

3.4.69 Subroutine S.IOCS25 - Save FCB Parameters in Spad

Functional Description

This routine saves original FCB parameters and inserts new FCB parameters prior to a physical I/O to be done on behalf of a user who requested blocked I/O.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS25
```

Registers:

R1	FCB address
R3	blocking buffer address
R7	special status byte

Spad Cells Used: 1, 2, 3

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R2	address of saved parameters
R4,R6	destroyed

3.4.70 Subroutine S.IOCS26 - Write Logical Blocked Record

Functional Description

This routine performs a write of a logical blocked record, i.e., it transfers a logical blocked record from the users data area to a blocking buffer.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS26
```

Registers:

R1	FCB address
R2	blocking buffer address

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

```
        R1                  FCB address
        R2,R3,R4,R5,R6,R7destroyed
```

3.4.71 Subroutine S.IOCS27 - Perform Implicit Open

Functional Description

This routine performs an implicit open of a logical file code on behalf of the calling IOCS entry point. If the open service is called (i.e., file not already open), the open is performed in the WAIT mode. If the file opened is the null device, a return is made directly to the user.

Entry Conditions

Calling Sequence:

```
        BL          S.IOCS27
```

Registers:

```
        R1          FCB address
```

Spad Cell Used: 2

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

```
        R1                  FCB address
        R2                  FAT address
        R3,R4,R5,R6,R7      destroyed
```

3.4.72 Subroutine S.IOCS28 - Initialize IOQ Entry

Functional Description

This routine initializes the IOQ parameters from the FCB, CDT, UDT and FAT. It also sets the program number into the IOQ and sets FCB.IOQA.

Entry Conditions

Calling Sequence:

BL	S.IOCS28

Registers:

R1	FCB address (or TCPB address)
R2	FAT address (or zero)
R3	CDT address
R6	IOQ address
R7	number of words extra in this IOQ

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2,R3,R6	same as on entry (masked with X'FFFFF')
R4,R7	destroyed

3.4.73 Subroutine S.IOCS29 - Report I/O Complete

Functional Description

This routine is called by handlers to report I/O completion.

Entry Conditions

Calling Sequence:

BL	S.IOCS29

Registers:

R1	program number
R2	IOQ address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2,R6	IOQ address
R0,R1,R3,R4,R5,R7destroyed	

Note: IOQ.RTN is used to save the return address before calling S.EXEC1, S.EXEC2, S.EXEC3 or S.EXEC4.

3.4.74 Subroutine S.IOCS30 - Advance Logical Blocked Record

Functional Description

This routine performs an advance logical blocked record; no transfer is required, only next read/write address is updated.

Entry Conditions

Calling Sequence:

```
        BL        S.IOCS30
```

Registers:

R1	FCB address
R2	current logical record
R3	blocking buffer address

Exit Conditions

Return Sequence:

```
        TRSW      R0
```

Registers:

R1	FCB address
R6	destroyed

3.4.75 Subroutine S.IOCS31 - Mark Units Offline

Functional Description

This routine marks a controller, and all the units connected to the controller, offline.

Entry Conditions

Calling Sequence:

```
        BL        S.IOCS31
```

Registers:

R1	CDT address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1	CDT address
R3,R7	destroyed

3.4.76 Subroutine S.IOCS32 - Restore FCB Parameters from Spad

Functional Description

This routine restores original FCB parameters from spad subsequent to physical operations performed on behalf of a user who requested blocked I/O operations.

Entry Conditions

Calling Sequence:

BL	S.IOCS32

Registers:

R1	FCB address

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	address of saved parameters
R4,R6	destroyed

3.4.77 Subroutine S.IOCS33 - Update Disc FAT

Functional Description

This routine is called by disc handler programs to update the users FAT prior to performing a disc file operation which would move the disc file relative block address in the forward or backward direction. Checks are made to determine whether the operation would cause the current disc address to move outside the file boundaries.

Entry Conditions

Calling Sequence:

```
         BL          S.IOCS33
```

Registers:

R1	FCB address
R2	FAT address
R7	blocks spanned in operation (+ if forward, - if backward)

Exit Conditions

Return Sequence:

```
         TRSW        R0
```

Registers:

R6	0 if operation within file bounds. Not equal to 0 if operation causes EOM/EOF or BOM.
R7	logical disc address (block number) for current operation
R3,R4,R6	destroyed

3.4.78 Subroutine S.IOCS34 - Allocate Variable IOQ Entry

Functional Description

This routine is called by entry point 5 of device handlers processing opcodes for which I/O queue entries are required (i.e., opcodes resulting in a device access). It allows the handler to specify the amount of additional space it wants added to the end of the IOQ entry for creation of the actual IOCL chain.

Note: The IOQ may be extended by an additional 3 words if blocked I/O is being done.

For F-class devices, this routine:

- allocates and initializes an IOQ entry.

For D- and E-class devices, this routine:

- allocates and initializes an IOQ entry.
- allocates an OS I/O buffer if necessary.
- builds a TCW if necessary.

Entry Conditions

Calling Sequence:

BL	S.IOCS34

Registers:

R1	FCB address
R7	number of words to extend the IOQ by

Note: Enters S.IOCS13 for completion of IOQ building.

Spad Cells Used: 12, 13, 15-22

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	destroyed

3.5 Resource Allocator (H.ALOC)

Entry Point Summary

ENTRY POINT	SVC NUMBER	DESCRIPTION
H.ALOC,1	N/A	CONSTRUCT TSA AND DQE
H.ALOC,2	N/A	TASK ACTIVATION PROCESSING
H.ALOC,3	N/A	TASK EXIT PROCESSING
H.ALOC,4	N/A	ALLOCATE MEMORY
H.ALOC,5	N/A	DEALLOCATE MEMORY
H.ALOC,6	N/A	ALLOCATE FILE/DEVICE
H.ALOC,7	N/A	DEALLOCATE FILE/DEVICE
H.ALOC,8	X'69'	GET DYNAMIC EXTENDED INDEXED DATA SPACE
H.ALOC,9	X'6A'	FREE DYNAMIC EXTENDED INDEXED DATA SPACE
H.ALOC,10	X'67'	GET DYNAMIC TASK EXECUTION SPACE
H.ALOC,11	X'68'	FREE DYNAMIC TASK EXECUTION SPACE
H.ALOC,12	X'71'	SHARE MEMORY WITH ANOTHER TASK
H.ALOC,13	X'72'	GET SHARED MEMORY (INCLUDE)
H.ALOC,14	X'79'	FREE SHARED MEMORY (EXCLUDE)
H.ALOC,15	N/A	GET E CLASS I/O MAP BLOCK
H.ALOC,16	N/A	FREE E CLASS I/O MAP BLOCK
H.ALOC,17	N/A	ALLOCATE FILE BY SPACE DEFINITION
H.ALOC,18	N/A	SHARE CSECT MEMORY WITH ANOTHER TASK
H.ALOC,19	X'1F'	UNLOCK AND DEQUEUE SHARED MEMORY
H.ALOC,20	N/A	DEALLOCATE MEMORY DUE TO SWAPPING
H.ALOC,99	N/A	SYSGEN INITIALIZATION

N/A implies reserved for internal use by MPX-32

Subroutine Summary

SUBROUTINE	DESCRIPTION
S.ALOC1	READ AND VERIFY PREAMBLE
S.ALOC2	DEALLOCATE TSA AND DQE
S.ALOC3	WRITE PROTECTION IMAGE TO RAM (32/7x only)
S.ALOC4	MAGNETIC TAPE DISMOUNT MESSAGE
S.ALOC5	MAGNETIC TAPE MOUNT MESSAGE
S.ALOC6	DEALLOCATE ALL PERIPHERAL DEVICES
S.ALOC7	TEST FOR DEVICE ON SYSTEM
S.ALOC8	GET FIRST MATCHING UDT
S.ALOC9	GET NEXT MATCHING UDT
S.ALOC10	ALLOCATE DISC FILE BY SPACE DEFINITION
S.ALOC11	ALLOCATE BLOCKING BUFFER
S.ALOC12	LOCATE FPT/FAT ADDRESS FOR ALLOCATED LFC
S.ALOC13	LOCATE SHARED MEMORY TABLE ENTRY
S.ALOC14	ALLOCATE FPT/FAT
S.ALOC15	ALLOCATE SHARED MEMORY SWAP FILE
S.ALOC16	DELETE SWAP FILE SPACE
S.ALOC17	UPDATE MAP SEGMENT DESCRIPTOR COUNT IN DQE
S.ALOC18	GET SWAP FILE SPACE
S.ALOC19	REMAP USERS ADDRESS SPACE
S.ALOC20	VALIDATE BUFFER ADDRESS
S.ALOC21	ALLOCATE MEMORY POOL BUFFER
S.ALOC22	RELEASE MEMORY POOL BUFFER
S.ALOC23	COMPRESS FILE NAME
S.ALOC24	UNCOMPRESS FILE NAME
S.ALOC25	SET ANY BIT IN MEMORY
S.ALOC26	CLEAR ANY BIT IN MEMORY
S.ALOC27	TEST ANY BIT IN MEMORY
S.ALOC28	DEALLOCATE DEBUGGER MEMORY
S.ALOC29	LOAD DEBUG OVERLAY
S.ALOC30	CREATE A PROTECTION IMAGE
S.ALOC31	UPDATE TASK PROTECTION IMAGE DUE TO INCREASE IN EXECUTION SPACE (32/7x only)
S.ALOC32	UPDATE SHARED MEMORY PROTECTION IMAGE
S.ALOC33	UPDATE TASK PROTECTION IMAGE DUE TO DECREASE IN EXECUTION SPACE (32/7x only)

3.5.1 Entry Point 1 - Construct TSA and DQE

Functional Description

This entry point is called by H.MONS,15 and H.MONS,40 to initialize a primitive TSA and DQE for task activation. This is achieved in the following manner:

- Determine if task is already in execution. If not...
- Allocate a free DQE from the free list and attach it to the preactivation list.
- Build a primitive DQE.
- Merge in the load module information table.
- Check for privileged execution and allocate swap file.
- Allocate one map block of memory and logically locate it in the first invalid map block found in the parent's TSA in which to build the child's TSA.
- Construct a primitive TSA for the child.
- Update T.REGS and T.REGP to point to the second phase of activation, H.ALOC,2.
- Unmap the child from the parent's logical space but leave the child linked to the preactivation state.

Special Cases:

Any load module starting with the letters 'SYSG' will be treated as the SYSGEN task which requires special loading.

Entry Conditions

Calling Sequence:

```
M.CALL    H.ALOC,1
```

Registers:

R1	address of parameter block, or zero if none
R6,R7	name of specified load module

Exit Conditions

Return Sequence:

```
M.RTRN  R6,R7
```

Registers:

R6	0 if valid request, non-zero if invalid
R7	task DQE address of new or existing task

3.5.2 Entry Point 2 - Task Activation Processing

Functional Description

This entry point is entered on behalf of a new task being activated. It performs all necessary functions to complete the introduction of the new task to the system. This is accomplished by the following sequence:

- Get resource requirements from the load module.
- Merge any extra requirements from the parameter block.
- Verify all ASSIGN1's, ASSIGN2's, ASSIGN3's and ASSIGN4's.
- Determine complete TSA size and initialize remaining TSA data.
- Determine total task size and initialize bases.
- Create a protection image (32/7x only).
- Allocate all memory needed and distribute to proper locations in the task.
- Create a protection image (CONCEPT/32 only).
- Mark all FAT's and FPT's free to allocate.
- Allocate permanent files, static partitions and spooled files.
- Allocate temporary disc files and other peripherals.
- Move the preamble into scratchpad and call H.LODR,1 (see Section 8.1.4).
- Include task debugger if requested.
- Dispatch the task.

Special Cases:

The common error code return paths for the Allocator are found in this entry point.

Entry Conditions

Calling Sequence:

Entered by pop of TSA stack built by H.ALOC,1

Registers:

All zero

Exit Conditions

Return Sequence:

Dispatch to transfer address or to H.MONS,20 with abort code in R5

3.5.3 Entry Point 3 - Task Exit Processing

Functional Description

This entry point is called by S.EXEC18. The abort code, if any, will be output. The task clean-up includes the deallocation of all peripherals, disc space, memory and memory pool. Finally the TSA and DQE are deallocated and a return is made to the scheduler via S.EXEC20.

Entry Conditions

Calling Sequence:

```
M.CALL      H.ALOC,3
```

Registers:

None

Exit Conditions

Return Sequence:

```
BU          S.EXEC20 (CPU scheduler routine)
```

3.5.4 Entry Point 4 - Allocate Memory

Functional Description

This entry point is called by H.ALOC entry points 1, 2, 8, 10, and 15. It is also called by the Swapper. Its function is to allocate the memory required for the calling task. The memory is returned in the form of map image descriptors (MIDL) and memory attributes (MEML), one MIDL and one MEML per map block. Swappable map counts in the DQE are incremented as needed based on the MEML information. The entries in the memory allocation table are updated to reflect allocation.

Special Cases:

1. For CONCEPT/32 machines, the protection granules remain as they were before the call.

2. For 32/7x machines, the protection image is unchanged by this entry point.

Entry Conditions

Calling Sequence:

```
M.CALL      H.ALOC,4
```

Registers:

R1	address of MIDL (halfword bounded)

R5	right halfword = number of map blocks required	
	left halfword =	1 = E memory
		2= H memory
		3= S memory
R3	address of MEML (HW bounded)	

Exit Conditions

Return Sequence:

M.RTRN or M.RTRN R5

Registers:

If the request cannot be fully satisfied, no memory is allocated by this service.

CC1 is set if unable to allocate all required memory and R5 contains the number of map blocks which cannot be allocated now.

CC1 is reset if request is successful and R5 is unchanged.

Physical memory definitions in MIDL. Memory attributes returned in MEML.

DQE.CME, DQE.CMH, and DQE.CMS incremented for each swappable map.

3.5.5 Entry Point 5 - Deallocate Memory

Functional Description

This entry point is called by H.ALOC entry points 3, 9, 11, and 14. Its function is to deallocate memory as directed by the map image descriptor list (MIDL) and the memory attribute list (MEML) which are required inputs to this routine. Memory can be of mixed types and of mixed swappable characteristics. The swappable count in the DQE is updated according to the information in the MEML.

Special Cases:

1. For 32/7x machines, the protection image is not changed.
2. For CONCEPT/32 machines, the protection granules are set to show a protected map hole exists.

Entry Conditions

Calling Sequence:

M.CALL H.ALOC,5

Registers:

R1	address of MIDL (halfword bounded)
R5	number of map blocks to deallocate
R3	address of MEML (HW bounded)

Exit Conditions

Return Sequence:

M.RTRN

Registers:

DQE.CME, DQE.CMH, and DQE.CMS are decremented for each swappable map.

3.5.6 Entry Point 6 - Allocate File/Device

Functional Description

This entry point is called by H.ALOC,2 or by H.MONS,21. It performs all necessary steps to allocate a file based on a 3 word RRS entry. The FAT, FPT and if necessary, blocking buffer, are all allocated and initialized in this service.

Entry Conditions

Calling Sequence:

M.CALL H.ALOC,6

Registers:

R1	address of 3 word RRS entry

Exit Conditions

Return Sequence:

M.RTRN R1 or M.RTRN R1,R6,R7

Registers:

R1	0 if allocation was unsuccessful. Otherwise unchanged.

CC1	set if allocation denied
	R6 = scan mask
	R7 = device requirements mask

If allocation is denied for a permanent file, R7 will contain one of the following error codes:

1 if file is exclusively locked
2 if FLT is full

CC2	set if allocation error
	R6 = error code
	R7 = 0

Error Conditions:

R6=1 - permanent file non-existent
=2 - illegal file password specified
=3 - no FAT/FPT space available
=4 - no blocking buffer space available
=5 - shared memory table entry not found
=6 - invalid shared memory table password specified
=7 - dynamic common specified in ASSIGN1
=8 - unrecoverable I/O error to SMD
=9 - SGO assignment specified by terminal task
=10 - no 'UT' file code exists for terminal task
=11 - invalid RRS entry
=12 - LFC in ASSIGN4 non-existent
=13 - assigned device not on system
=14 - device in use by requesting task
=15 - SGO or SYC assignment by real-time task
=16 - common memory conflicts with allocated task
=17 - duplicate LFC allocation attempted

3.5.7 Entry Point 7 - Deallocate File/Device

Functional Description

This entry point is called by H.ALOC,2 and 3 and by H.MONS,41. The FAT associated with the lfc is deallocated unless an active ASSIGN4 is vectored through it. In that case, only the FPT is deallocated. For any deallocated peripheral a call is made to the scheduler via H.EXEC,15 to allow reallocation. File space is deallocated by calling H.FISE,4.

Entry Conditions

Calling Sequence:

```
     M.CALL    H.ALOC,7
```

Registers:

R5	right justified ASCII logical file code

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Error Conditions

CC1 = SET - unrecoverable I/O error to SMD

3.5.8 Entry Point 8 - Get Dynamic Extended Indexed Data Space

See Section 8.3.8 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.9 Entry Point 9 - Free Dynamic Extended Indexed Data Space

See Section ·8.3.5 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.10 Entry Point 10 - Get Dynamic Task Execution Space

See Section 8.3.9 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.11 Entry Point 11 - Free Dynamic Task Execution Space

See Section 8.3.6 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.12 Entry Point 12 - Share Memory With Another Task

See Section 8.3.11 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.13 Entry Point 13 - Get Shared Memory (INCLUDE)

See Section 8.3.10 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.14 Entry Point 14 - Free Shared Memory (EXCLUDE)

See Section 8.3.3 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.15 Entry Point 15 - Get 'E' Class I/O Map Block

Functional Description

This entry point is called by H.IOCS on behalf of a user who has requested a transfer of greater than 192 words to an 'E' class device, but has not specified a contiguous 'E' class memory buffer. An 'E' class map block will be allocated and mapped through logical address X'F0000'. It is understood that H.IOCS will use X'F0000' through X'F3FFF' to buffer no-wait requests, and will use X'F4000' through X'F7FFF' to buffer wait requests for I/O. If the map block is unavailable the user will be suspended by H.EXEC,5.

Special Cases:

> 'E' class devices are not supported by the CONCEPT/32 machines. Therefore, this routine is not needed for CONCEPT/32 support.

Entry Conditions

Calling Sequence:

```
M.CALL    H.ALOC,15
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN    R6
```

Registers:

R6	physical address of 8K buffer
CC1	set if map block already allocated

3.5.16 Entry Point 16 - Free 'E' Class I/O Map Block

Functional Description

This entry point is called by H.IOCS on behalf of a user whose 'E' class I/O has completed and been moved.

Special Cases:

> 'E' class devices are not supported by the CONCEPT/32 machines. Therefore, this routine is not needed for CONCEPT/32 support.

Entry Conditions

Calling Sequence:

```
M.CALL    H.ALOC,16
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.5.17 Entry Point 17 - Allocate Disc File By Space Definition

Functional Description

This entry point allocates a FAT/FPT for the space definition provided by the caller. If bit zero of R4 is set, the system FAT/FPT is allocated. If bit zero of R5 is set, a blocking buffer is allocated.

Entry Conditions

Calling Sequence:

```
M.CALL    H.ALOC,17
```

Registers:

R4	LFC (bit 0 set for system FAT/FPT)
R5	UDT index (bit 0 set for blocking buffer)
R6	sector address
R7	number of sectors

Exit Conditions

Return Sequence:

```
M.RTRN    R1, R2, R3, R5
```

Registers:

R1	UDT address
R2	FPT address
R3	FAT address
R5	blocking buffer address if required
CC1	set, no FAT/FPT space
CC2	set, no blocking buffer space

3.5.18 Entry Point 18 - Share CSECT Memory with Another Task

Functional Description

This entry point is called only by H.ALOC,2. It is a subset of H.ALOC,12.

3.5.19 Entry Point 19 - Unlock and Dequeue Shared Memory

See Section 8.3.12 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.5.20 Entry Point 20 - Deallocate Memory Due to Swapping

Functional Description

This entry point is called only by the Swapper. Its function is to deallocate memory as directed by the map image descriptor list (MIDL) and the memory attribute list (MEML) which are required inputs to this routine. Memory can be of mixed types but all map blocks are swappable. The swappable count in the DQE is updated according to the information in the MEML.

Special Cases:

1. For 32/7x machines, the protection image is not changed.
2. For CONCEPT/32 machines, the protection granules are not changed.

Entry Conditions

Calling Sequence:

```
        M.CALL        H.ALOC,20
```

Registers:

R1	address of MIDL (halfword bounded)

R3 address of MEML (halfword bounded)

R5 number of map blocks to deallocate

Note: DQE.CME, DQE.CMH and DQE.CMS are decremented for each swappable map. Protection registers are unchanged. MEML and MIDL reflect maps deallocated.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.5.21 Entry Point 99 - SYSGEN Initialization

Functional Description

This entry point performs any necessary SYSGEN initialization.

Entry Conditions

Calling Sequence:

```
BL        *HAT+18W
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW      R0
```

3.5.22 Subroutine S.ALOC1 - Read and Verify Preamble

Functional Description

This subroutine is used to read the preamble of a load module into the sysem buffer. The doubleword filename is compared with the name specified in the preamble. Error conditions are returned. This subroutine is used for activation and overlay functions.

Entry Condition

Calling Sequence:

```
        BL          S.ALOC1
```

Registers:

R6,R7 — left-justified filename of load module

Words 6 and 7 of scratchpad are assumed equal to R6 and R7 respectively

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R6,R7 — unchanged if valid load module, else indeterminate

R5 — zero if valid, else

2 if file not found
3 if file is password protected
4 if invalid preamble
6 if I/O error on SMD
7 if I/O error on file

R1 — address of preamble (T.BBUFA)

R2 — current register pointer (T.REGP)

R3 — address of TSA

Note: This routine uses word 29 of scratchpad.

3.5.23 Subroutine S.ALOC2 - Deallocate TSA and DQE

Functional Description

This subroutine deallocates all TSA map blocks and updates the memory allocation table. It also clears the tasks DQE and relinks it to the DQE free list.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC2
```

Registers:

None

Exit Conditions

Return Sequence:

Return to S.EXEC20

Registers:

None

3.5.24 Subroutine S.ALOC3 - Write Protection Image to RAM (32/7x only)

Functional Description

This subroutine is for use on a SYSTEMS 32/7x computer and it writes the protection image contained in T.PROT into the hardware protection registers. If the task is privileged, this subroutine is not executed.

Entry Conditions

Calling Sequence:

```
BL          S.ALOC3
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R0,R2	destroyed

Note: Words 16-23 of scratch pad are used.

3.5.25 Subroutine S.ALOC4 - Magnetic Tape Dismount Message

Functional Description

This subroutine is used to issue a dismount message to the operators console. No operator response is required.

Entry Conditions

Calling Sequence:

BL S.ALOC4

Registers:

R3 FAT address

Exit Conditions

Return Sequence

TRSW R0

Registers:

R1,R2,R4,R5,R6,R7 destroyed

R3 unchanged

3.5.26 Subroutine S.ALOC5 - Magnetic Tape Mount Message

Functional Description

This subroutine is used to issue a mount message in the following format to the operators console. A response is required by the operator.

$$\left\{\begin{array}{l}\text{TASK}\\ \text{jobno}\end{array}\right\}\text{ ,taskname,taskno MOUNT reel VOL volume ON devmnc DEV,R,A,H?}\left\{\begin{array}{l}\text{devmnc}\\ \text{R density}\\ \text{A}\\ \text{H}\end{array}\right\}$$

Entry Conditions

Calling Sequence:

BL S.ALOC5

Registers:

R1 FCB address

Exit Conditions

Return Sequence

TRSW R0

Registers:

R1 unchanged

R2,R3,R4,R5,R6,R7	destroyed
CC1	set if abort requested, otherwise CC1=0

Abort Conditions:

If abort response input from operator, the requesting task is aborted with an MS25.

3.5.27 Subroutine S.ALOC6 - Deallocate All Peripheral Devices

Functional Description

A scan is made of the FPT/FAT's and the non-disc devices are deallocated. Devices are made available and FPT/FAT and blocking buffer entries freed. Peripherals made available are reported to the executive via H.EXEC,13.

Entry Conditions

Calling Sequence:

```
BL        S.ALOC6
```

Registers:

None required

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R2,R3,R4,R5,R6	destroyed
R1,R7	unchanged

3.5.28 Subroutine S.ALOC7 - Test For Device On System

Functional Description

This subroutine builds a scan mask and requirements mask from a user provided ASSIGN3 type RRS. System tables are checked to see if the specified device is in the system.

Entry Conditions

Calling Sequence:

```
BL        S.ALOC7
```

Registers:

R1	address of 3 word RRS entry

Exit Conditions

Return Sequence:

```
        TRSW    R0
```

Registers:

R1,R2	unchanged
R3	address of first matching UDT
R4,R5	destroyed
R6	scan mask
R7	requirements mask
CC1	set if device not found

3.5.29 Subroutine S.ALOC8 - Get First Matching UDT

Functional Description

This subroutine sets up a scan of the UDT's for a specified mask and request. The first matching entry is returned. No allocation is made of the device. Other matching entries are obtained by calling S.ALOC9.

Entry Conditions

Calling Sequence:

```
        BL      S.ALOC8
```

Registers:

R4	scan mask
R7	requirements mask

Exit Conditions

Return Sequence:

```
        TRSW    R0
```

Registers:

R1,R2,R4,R7	unchanged
CC1	set, matching UDT not found
CC2	set if user has all of requested devices previously allocated

The following registers should not be modified if S.ALOC9 is to be called to get next matcing device:

R3	UDT address
R5	internal status register
R6	loop count

3.5.30 Subroutine S.ALOC9 - Get Next Matching UDT

Functional Description

This subroutine is called after calling S.ALOC8 to get first matching UDT entry. A forward scan is made for unshared devices first. A reverse scan is then made for a shared or unshared matching device. Registers 3, 5 and 6 should not be modified between calls. No device allocation is made.

Entry Conditions

Calling Sequence:

```
BL      S.ALOC9
```

Registers:

R3	current UDT address (from previous call)
R4	scan mask
R5	internal status register (from previous call)
R6	loop count (from previous call)
R7	requirements mask

Exit Conditions

Return Sequence:

```
TRSW    R0
```

Registers:

R1,R2,R4,R7	unchanged
CC1	set, matching UDT not found
CC2	set if user has all requested devices previously allocated
R3	UDT address
R5	internal status register, if required
R6	loop count, if required

3.5.31 Subroutine S.ALOC10 - Allocate Disc File By Space Definition

Functional Description

This subroutine allocates a FAT/FPT for the space definition provided by the caller. If bit zero of R4 is set, the system FAT/FPT is allocated. If bit zero of R5 is set, a blocking buffer is allocated.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC10
```

Registers:

R4	LFC (bit 0 set for system FAT/FPT)
R5	UDT index (bit 0 set for blocking buffer)
R6	sector address
R7	number of sectors

Exit Conditions

Return Sequence:

R1	UDT address
R2	FPT address
R3	FAT address
R5	blocking buffer address if required
R4,R6,R7	destroyed

Error Conditions:

CC1	set, no FAT/FPT space
CC2	set, no blocking buffer space

Note: Uses word 31 of scratch pad for storage.

3.5.32 Subroutine S.ALOC11 - Allocate Blocking Buffer

Functional Description

This subroutine allocates a free blocking buffer for the caller. The control word in the blocking buffer is cleared and the buffer empty bit is set. The buffer is marked allocated. The blocking buffer address is inserted in the FAT and the blocking buffer active bit is set in the status word. If the FAT address provided is the system FAT, the system blocking buffer is unconditionally allocated. CC1 is set if no blocking buffer is available.

Entry Conditions

Calling Sequence:

```
        BL      S.ALOC11
```

Registers:

R3	FAT address

Exit Conditions

Return Sequence:

```
        TRSW    R0
```

Registers:

R3	FAT address
R5	blocking buffer address (0 if no allocation)
R4	destroyed
CC1	set if no blocking buffer found (R5=0)

3.5.33 Subroutine S.ALOC12 - Locate FPT/FAT Address For Allocated LFC

Functional Description

This subroutine scans the allocated FPT entries for a matching LFC. The FPT and FAT addresses are returned for the matching LFC. If bit zero of register 5 is set on entry, the system FPT and FAT addresses are returned. No comparison check is made for the LFC in the system FPT.

Entry Conditions

Calling Sequence:

```
         BL          S.ALOC12
```

Registers:

R5	LFC to match (bit 0 set indicates get system FPT/FAT addresses)

Exit Conditions

Return Sequence:

```
         TRSW        R0
```

Registers:

R2	FPT address (0 if no match)
R3	FAT address (0 if no match)
R5	LFC with byte 0 cleared
R4	word address mask (X'FFFFFF')
CC1	set no match found (R2,R3 zeroed)

3.5.34 Subroutine S.ALOC13 - Locate Shared Memory Table Entry

Functional Description

This subroutine is used to find the first entry in the Shared Memory Table which contains the name and task number specified by the caller. It is also used to find a free entry.

Entry Conditions

Calling Sequence:

```
         BL          S.ALOC13
```

Registers:

R4,R5	owner name
R6,R7	name of shared memory partition or zero to locate a free entry
(or)	
R4	task activation number

R5	0
R6,R7	name of shared memory partition or zero to locate a free entry

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1	address of shared memory table entry or 0 if not found
R4	destroyed
R5,R6,R7	unchanged

3.5.35 Subroutine S.ALOC14 - Allocate FPT/FAT

Functional Description

This subroutine will locate an available FPT and associated FAT. The FPT is marked unavailable and the LFC and FAT address inserted. The associated FAT is marked unavailable and cleared. If bit zero of R5 is set on entry, the system FPT/FAT is unconditionally allocated.

Entry Conditions

Calling Sequence:

BL	S.ALOC14

Registers:

R5	logical file code to allocate (bit 0 set indicates system FPT/FAT be allocated)

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R2	FPT address (0 on no space available)
R3	FAT address (0 on no space available)
R5	LFC with byte 0 cleared
R4	destroyed

CC1	set, allocation denied
CC2	set, matching LFC exists (R2=FPT ADDR, R3=FAT ADDR)

3.5.36 Subroutine S.ALOC15 - Allocate Shared Memory Swap File

Functional Description

This subroutine allocates temporary disc space for use as a swap file. The shared memory table is updated to reflect the space information for the swap file. If a swap file is successfully allocated, the swappable bit in the shared memory table is set.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC15
```

Registers:

R1	address of shared memory table

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R1	address of shared memory table
R3	unused
R2,R4,R5,R6,R7	destroyed

SMT updated with swap file information

Error Conditions:

CC1	set, unrecoverable I/O error to SMD

3.5.37 Subroutine S.ALOC16 - Delete Swap File Space

Functional Description

This subroutine deallocates temporary disc space used as a swap file. The DQE is updated to reflect the new swap file status.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC16
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R0	error code, if any
R3	DQE address (C.CURR)
R4,R5,R6,R7	destroyed

Error Conditions

CC1	set, unrecoverable I/O error to SMD

3.5.38 Subroutine S.ALOC17 - Update Map Segment Descriptor Count In DQE

Functional Description

This subroutine updates byte 0 of DQE.MSD, which is the number of map image descriptors required to map the CPIX. The count is determined by examining TSA variables T.EBUF, T.DSOR, T.DSSZ, T.CSOR, T.CSSZ, T.MPOR, T.MPSZ, T.EAOR and T.EASZ.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC17
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R3	TSA address
R2	DQE address
R4	number of map blocks in CPIX

DQE.MSD (byte 0) is updated by contents of R4

3.5.39 Subroutine S.ALOC18 - Get Swap File Space

Functional Description

This subroutine allocates temporary disc space for use as a swap file. The device denoted by SYSGEN as the swap device is tested first for allocation of the file. If a failure occurs, allocation will be attempted on any device that has the same classification as the swap device. If a swap file cannot be allocated for the calling task, the task is marked unswappable and allowed to be dispatched. The DQE is updated to reflect the space definition of the swap file.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC18
```

Registers:

R5	number of map blocks that are to be swappable

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R0	error code, if any
R3	DQE address
R1,R2,R4,R5,R6,R7	destroyed

Error Conditions

CC1	set, unable to allocate file. DQE.FCRS is set to show task is forced to be unswappable.

3.5.40 Subroutine S.ALOC19 - Remap Users Address Space

Functional Description

This subroutine is used during allocation and deallocation of memory to add or delete physical maps from the users address space. The hardware map registers are reloaded with the new map images.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC19
```

Registers:

None

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R3,R4,R5	destroyed

3.5.41 Subroutine S.ALOC20 - Validate Buffer Address

Functional Description

This subroutine is used to verify a logical address provided by the user. The following inquiries are made:

- Is the starting address lower than the DSECT start address?
- Do the addresses specified cross a map block boundary?
- Are the addresses specified in a valid map block?
- Are the addresses specified in the extended address space?
- Are the addresses specified in a protected area?

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC20
```

Registers:

R6	20 bit logical starting address
R7	number of bytes to validate (buffer size). If zero, single address check is made.

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R6,R7	unchanged
R1,R2,R4,R5	destroyed

Status:

CC4	set, invalid address (i.e., in OS, below DSECT, or not mapped into users space)
CC3	set, locations specified are protected
CC2	set, buffer crosses map block boundary
CC1	set, address below DSECT

3.5.42 Subroutine S.ALOC21 - Allocate Memory Pool Buffer

Functional Description.

This subroutine is used by system services requiring temporary memory for I/O queue entries, I/O buffering, and messages. No attempt is made to track the use of such buffers by the memory management services. Thus use of this service must be restricted. Buffers are allocated on a double word boundary.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC21
```

Registers:

R7	number of words required

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R3	starting address of memory pool (doubleword bounded)
R7	number of words rounded to 2W increment
R1,R2	destroyed

Status:

CC1	set, no memory available and R3=0

Abort Cases:

None

3.5.43 Subroutine S.ALOC22 - Release Memory Pool Buffer

Functional Description

This subroutine will deallocate a previously allocated memory pool buffer.

Entry Conditions

Calling Sequence:

```
BL          S.ALOC22
```

Registers:

R3	starting address of memory pool
R7	number of words to deallocate

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R3	unchanged
R1,R2,R4,R7	destroyed

3.5.44 Subroutine S.ALOC23 - Compress File Name

Functional Description

This subroutine converts 8 bit ASCII coded characters to 6 bit ASCII coded characters. Hex 20 is subtracted from each byte to get new code.

Entry Conditions

Calling Sequence:

```
BL          S.ALOC23
```

Registers:

R6,R7	8 bit ASCII coded name blank filled

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R6,R7	6 bit ASCII coded name in lower 48 bits
CC1	set if character found X'20' or X'5F'
R2,R3,R4,R5	destroyed

3.5.45 Subroutine S.ALOC24 - Uncompress File Name

Functional Description

This subroutine converts 6 bit ASCII coded characters to 8 bit ASCII coded characters. A hex 20 is added to each 6 bit value to get new 8 bit value.

Entry Conditions

Calling Sequence:

```
BL          S.ALOC24
```

Registers:

R6,R7	6-bit ASCII coded name right-justified in lower 48 bits of R6,R7

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R6,R7	8 bit ASCII coded name
R2,R3,R4,R5	destroyed

3.5.46 Subroutine S.ALOC25 - Set Any Bit In Memory

Functional Description

This subroutine is used to set any bit in memory.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC25
```

Registers:

R2	base address of bit string
R4	relative bit number (0-2**20)

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R2	unchanged
R4,R5	destroyed

3.5.47 Subroutine S.ALOC26 - Clear Any Bit In Memory

Functional Description

This subroutine is used to clear any bit in memory.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC26
```

Registers:

R2	base address of bit string
R4	relative bit number (0-2**20)

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R2	unchanged
R4,R5	destroyed

3.5.48 Subroutine S.ALOC27 - Test Any Bit In Memory

Functional Description

This subroutine is used to test the status of any bit in memory.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC27
```

Registers:

R2	base address of bit string
R4	relative bit number (0-2**20)

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R2	unchanged
R4,R5	destroyed

Status:

CC1	set if bit tested is set

3.5.49 Subroutine S.ALOC28 - Deallocate Debugger Memory

Functional Description

This subroutine is used to deallocate the memory that was dynamically allocated for the loading of the task debugger into the calling task's space.

Entry Conditions

Calling Sequence:

```
        BL          S.ALOC28
```

Registers:

None

This routine is called by H.MONS,30

Exit Conditions

Return Sequence:

```
TRSW        R7  (R7 contains return address)
```

Registers:

None

3.5.50 Subroutine S.ALOC29 - Load Debug Overlay

Functional Description

This subroutine performs all memory management and set up requirements for loading the Debug overlay. The users context is copied to T.CONTXT prior to dispatching control to the Debug overlay. DQE.ADM, DQE.DBAT,T.DBHAT are all initialized. T.CSOR points to the start of the Debug overlay.

Entry Conditions

Calling Sequence:

```
BL          S.ALOC29
```

Registers:

None

This subroutine is called by H.TSM,6, H.MONS,29, or by H.ALOC,2.

Exit Conditions

Return Sequence:

```
TRSW        R0
```

Registers:

R7 transfer address of Debug overlay. Bit 0 is set to indicate privileged mode.

Note: Word 28 of scratch pad is used by this subroutine.

R5 zero if valid

1 if I/O error on SMD
2 if file not found

3 if I/O error on file
4 if invalid preamble
5 if insufficient room in user's address space

3.5.51 Subroutine S.ALOC30 - Create a Protection Image

Functional Description

This subroutine creates a protection image for the calling task. If the machine type is 32/7x, the protection image is built backwards in T.PROT. For the CONCEPT/32 machines, the protection granules in the MIDL entry are set. This subroutine protects every map block in the tasks logical space. This is allowed because at the point in time that this subroutine is called, the only memory that has been allocated to the task is the memory for the TSA, which needs to be protected.

Entry Conditions

Calling Sequence:

```
        BL        S.ALOC30
```

Registers:

```
        R3            TSA address
```

Exit Conditions

Return Sequence:

```
        TRSW      R0
```

Registers:

None (all registers are destroyed)

3.5.52 Subroutine S.ALOC31 - Update Task Protection Image Due to Increase in Execution Space (32/7x only)

Functional Description

This subroutine updates the calling tasks protection image due to an increase in the primary execution space. This subroutine is for 32/7x machines only. S.ALOC3 is called to write the protection image into RAM.

Entry Conditions

Calling Sequence:

```
        BL        S.ALOC31
```

Registers:

R3	relative map number

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2	destroyed
R3	starting address of last DSECT map

3.5.53 Subroutine S.ALOC32 - Update Shared Memory Protection Image

Functional Description

This subroutine updates the protection image of the calling task when a shared memory partition is included. For a 32/7x machine, the protection image is located in T.PROT. For a CONCEPT/32 machine, the protection granules in the MIDL are updated.

Entry Conditions

Calling Sequence:

```
BL        S.ALOC32
```

Registers:

R1	shared memory table address

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R2,R4,R5,R6,R7	destroyed

3.5.54 Subroutine S.ALOC33 - Update Task Protection Image Due to Decrease in Execution Space (32/7x only)

Functional Description

This subroutine is used to update the calling tasks protection image due to a decrease in the primary execution space. This subroutine is for 32/7x machines only. S.ALOC3 is called to write the protection image into RAM.

Entry Conditions

Calling Sequence:

BL	S.ALOC33

Registers:

R3	relative map number

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2	destroyed
R3	unchanged

3.6 Terminal Services (H.TSM)

The following table describes the Terminal Context Area (TCA) which J.TSM maintains for every TSM device (DTC=TY). This area is allocated and initialized in memory which J.TSM obtains dynamically using the memory expansion services when it starts.

Word	0 7	8 15	16 23	24 31
0	Terminal context area FCB (TCA.FCB)			
16	External permanent file FCB (TCA.EFCB)			
32	Macro parameters (TCA.MPAR)			
48	Number of characters per macro parameter (TCA.MSIZ)			
50	Number of parameters in macro call (TCA.CPAR)			
66	Number of characters per parameter (TCA.CSIZ)			
68	Target of current label scan (TCA.LABL)			
70	Header for macro buffer, used by M.TSCAN (TCA.MHDR)			
73	Macro buffer from file (TCA.MBUF)			
91	Second word of status flags (TCA.STS2) See Note 1	Number of parameters in macro (TCA.MNUM)	Number of parameters in macro call (TCA.CNUM)	Reserved
92	Used by IFP/IFA (TCA.EXM1)			
93	Used by IFT/IFF (TCA.EXM2)			
94	Current project number (TCA.PROJ)			
96	Accumulated CPU time per session (TCA.CPU)			

Word	0 7	8 15 16 23 24 31
97	Accumulated IPU time per session (TCA.IPU)	
98	Last abort code (TCA.ERR)	
99	Time-of-day session started (TCA.LOGN)	
100	Terminal attributes (TCA.TERM)	
101	Status flags (TCA.STAT) See Note 2	Address of terminal UDT (TCA.UDTA)
102	Task number of active TSM task (TCA.TNUM)	
103	Conditional job flags (TCA.JFLG) See Note 3	
104	Privileged access bits (TCA.ACES)	
106	SGO space definition (TCA.SGOS)	
110	Message receiver's context (TCA.RCVR)	
112	Canned message header (TCA.HDR1)	
116	Ownername of first message sender (TCA.ONR1)	
119	Date of first message (TCA.DAT1)	
122	Time of first message (TCA.TIM1)	
124	First message mailbox (TCA.MBX1)	
143	Canned message header (TCA.HDR2)	
147	Ownername of second message sender (TCA.ONR2)	
150	Date of second message (TCA.DAT2)	
153	Time of second message (TCA.TIM2)	

Word	0 7	8 15	16 23	24 31
155	Second message mailbox (TCA.MBX2)			
174	Execution mode flags (TCA.MODE)	Number of RRS entries (TCA.NRRS)	Number of protection granules to allocate (TCA.ALLO)	Memory class (TCA.MEMC)
175	Number of blocking buffers (TCA.NBUF)	Number of files (FAT/FPT) additional (TCA.NFIL)	Priority level of task (TCA.PRIO)	Reserved (TCA.NUSE)
176	Load module name (TCA.LMN)			
178	Pseudonym of task (TCA.SUDO)			
180	Ownername of task (TCA.ONRN)			
182	Username of task (TCA.USRN)			
184	User key for username (TCA.USRK)			
185	Option word for activation (TCA.PGOW)			
186	User status word (TCA.USW)			
187	Address of first RRS entry (TCA.FRRS)			
189	Address of I/O line buffer (TCA.LBFA)			
247	End of TCA table			

Notes:

1. Bits in TCA.STS2 are assigned as follows.

0 - If set, reading first line of command/macro file
1 - If set, processing macros
2 - If set, scanning for label
3 - If set, displaying processor input
4 - If set, next record already read
5 - If set, reading from message receiver
6 - If set, macro expansion inhibited
7 - Permanent file input mode

2. Bits in TCA.STAT are assigned as follows.

0 - If set, terminal in task mode (TCA.TASK)
1 - If set, terminal in command mode (TCA.COMM)
2 - If set, first mailbox has valid message
3 - If set, second mailbox has valid message
4 - If set, next mailbox flip-flop
5 - If set, terminal has valid ownername (TCA.OWNR)
6 - If set, terminal expecting message input
7 - If set, control file processed

3. Bits in TCA.JFLG are assigned as follows:

0 - Reserved
1-16 - Conditional job processing flags
17 - If set, previous task aborted
18 - If set, inhibit error expansions
19-32 - Reserved

ENTRY POINT	SVC NUMBER	DESCRIPTION
H.TSM,1	N/A	TERMINAL I/O INTERFACE
H.TSM,2	5B	SYNTAX SCANNER
H.TSM,3	N/A	USER TASK EXIT
H.TSM,4	N/A	USER TASK ABORT
H.TSM,5	59	SET USER TAB POSITIONS
H.TSM,6	5C	BREAK PROCESSING ENTRY
H.TSM,7	28	CONVERT ASCII DECIMAL TO BINARY
H.TSM,8	29	CONVERT ASCII HEXADECIMAL TO BINARY
H.TSM,9	2A	CONVERT BINARY TO ASCII DECIMAL
H.TSM,10	2B	CONVERT BINARY TO ASCII HEXADECIMAL
H.TSM,99	N/A	SYSGEN INITIALIZATION

3.6.1 Entry Point 1 - Terminal I/O Interface

Functional Description

This entry point is called by H.IOCS exclusively, when H.IOCS detects the on-line bit (7) in the caller's FAT. Return is back to the caller at the instruction following his original call to H.IOCS. H.IOCS performs general validation and set up of the user's FCB before calling this service. The IOCS op-code is examined by H.TSM; OPEN, READ, WRITE, CLOSE, REWIND, and WEOF result in further processing. All other operations result in the immediate return to the user. For READ and WRITE requests, TSM performs intermediate buffering operations, reissuing of the I/O request, and error checking.

READ Logic

First the program option word (T.PGOW) is tested to determine if prompt-before-read is in effect. If so, a prompt message is formed from the current load module name and written to the terminal. Next, the caller's TCW is clamped with the maximum transfer specified in UDT.CHAR. Then the READ is reissued to IOCS with a new FCB (T.BFCB) and a new input buffer (from T.LINBUF). Lower case input is allowed if the option was set in T.PGOW. The scheduler is alerted that terminal read is in progress, and the task is made roll-outable, even though it is in I/O wait.

After I/O is complete, including post I/O processing, the contents of the input buffer are moved back to the caller's buffer. All characters from the carriage return (if any) to the end of the buffer are blank-filled. The transfer count in the FCB is updated to reflect the actual nuumber of characters entered before the carriage return. Finally, the scheduler is informed that terminal input is complete and return is made back to the caller, or, if a BREAK was detected during the READ, H.MONS,47 is called, instead.

WRITE Logic

First the line count (T.LINNO) is checked to see if a screen size has been specified. If so, the current position of the cursor is checked against the maximum count. A prompt message is written if we are at bottom of screen. After user response, the count is reset to top of screen. Next, the caller's TCW is clamped with the maximum transfer specified in UDT.CHAR. Break detection is enabled and the scheduler is informed that terminal output is in progress. Finally, the write service is reissued to IOCS with a new FCB (from T.BFCB).

When I/O is complete, the scheduler is informed and a test for break is made. If BREAK occurred, this service calls H.MONS,47. Otherwise, return is made to the instruction following the original IOCS call.

OPEN Logic

The UDT is examined by H.TSM,1 to determine the width and height of the terminal device. The height (number of lines) is stored in T.LINNO (byte 0). Then a line buffer is allocated from the memory pool via H.ALOC,17. The size of the line buffer is determined by adding three words to UDT.CHAR and rounding up to an 8-word boundary. T.LINBUF is set to point to the start of the line buffer and the contents of UDT.CHAR is stored in byte 8 of the line buffer. Finally the OPEN request is issued to H.IOCS.

Note that multiple OPEN requests are ignored if T.LINBUF is not equal to zero.

CLOSE Logic

The line buffer pointed to by T.LINBUF is deallocated via S.ALOC,22. Then the CLOSE request is issued to H.IOCS.

REWIND Logic

The cursor position in the linebuffer is reset to the first input character. Then the REWIND request is issued to H.IOCS.

Entry Conditions

Calling Sequence:

```
        M.CALL     H.TSM,1
```

Registers:

R1 address of user's FCB

Output Conditions

Updated FCB

External References:

H.IOCS,1
H.IOCS,3
H.IOCS,4
H.IOCS,19
H.IOCS,23
H.EXEC,7
H.MONS,47
H.ALOC,17
S.ALOC,22

Terminal Messages:

ENTER CR FOR MORE
task (prompt)

3.6.2 Entry Point 2 - Syntax Scanner

See Section 5.6.1 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.3 Entry Point 3 - User Task Exit

Functional Description

This entry point is entered by the scheduler as part of the exit process for an on-line task. The terminal line buffer is deallocated and a break is sent to J.TSM.

Entry Conditions

Calling Sequence:

```
M.CALL      H.TSM,3
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.6.4 Entry Point 4 - User Task Abort

Functional Description

This entry point is entered by the scheduler when an on-line task aborts. The abort code and PSW address are written to the user's terminal. If the terminal is malfunctioning, the abort code is written to the system console device. Then this entry point merges with entry point 3.

Entry Conditions

Calling Sequence:

```
M.CALL      H.TSM,4
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.6.5 Entry Point 5 - Set User Tab Positions

Functional Description

This entry point is used by the Editor to pass the user's specified tab positions to the UDT. The user's tabs are examined by the terminal handler during formatted I/O processing and the cursor is adjusted as appropriate.

Entry Conditions

Calling Sequence

```
LD    R6,        TABS

SVC   1,X'59' (or) M.CALL  H.TSM,5.
```

where:

TABS is a double-word containing up to eight tab positions. These positions must be in ascending order with 0 being used to indicate no more tabs.

The caller must be a TSM task.

Exit Conditions

Return Sequence

M.RTRN

Registers:

None

3.6.6 Entry Point 6 - Break Processing Entry

See Section 5.6.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.7 Entry Point 7 - Convert ASCII Decimal To Binary

See Section 5.6.3.1 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.8 Entry Point 8 - Convert ASCII Hexadecimal To Binary

See Section 5.6.3.2 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.9 Entry Point 9 - Convert Binary To ASCII Decimal

See Section 5.6.3.3 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.10 Entry Point 10 - Convert Binary To ASCII Hexadecimal

See Section 5.6.3.4 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.6.11 Entry Point 99 - SYSGEN Initialization

There is currently no SYSGEN initialization required. Return is made via TRSW through Register 0.

3.7 File System Executive (H.FISE)

ENTRY POINT	SVC NUMBER	DESCRIPTION
H.FISE,1	N/A	GET SYSTEM SMD ENTRY
H.FISE,2	N/A	REWRITE SMD ENTRY
H.FISE,3	N/A	ALLOCATE TEMPORARY DISC SPACE
H.FISE,4	N/A	DEALLOCATE TEMPORARY DISC SPACE
H.FISE,5	N/A	ALLOCATE PERMANENT DISC SPACE
H.FISE,6	N/A	UNGATE FISE
H.FISE,7	N/A	DEALLOCATE PERMANENT DISC SPACE
H.FISE,8	N/A	ASCII COMPRESSION
H.FISE,9	N/A	GATE FISE
H.FISE,10	N/A	GET USER SMD ENTRY
H.FISE,11	N/A	PERMANENT FILE ALLOCATION CHECK
H.FISE,12	75	CREATE PERMANENT FILE
H.FISE,13	76	CHANGE TEMPORARY FILE TO PERMANENT
H.FISE,14	77	DELETE PERMANENT FILE
H.FISE,15	N/A	PERMANENT FILE LOG
H.FISE,16	N/A	VALIDATE USERNAME
H.FISE,17	N/A	READ ALLOCATION MAP INTO MEMORY
H.FISE,18	N/A	WRITE ALLOCATION MAP TO DISC
H.FISE,19	N/A	RESERVED FOR FUTURE USE
H.FISE,20	N/A	CREATE PERMANENT FILE
H.FISE,21	N/A	CHANGE TEMPORARY FILE TO PERMANENT
H.FISE,99	N/A	SYSGEN INITIALIZATION

N/A implies reserved for internal use by MPX-32

File Gating Services Available To Unprivileged Tasks

MACRO	SVC NUMBER	MODULE ENTRY PT.	DESCRIPTION
M.FXLS	X'21'	H.FISE,22	SET EXCLUSIVE FILE LOCK
M.FXLR	X'22'	H.FISE,23	RELEASE EXCLUSIVE FILE LOCK
M.FSLS	X'23'	H.FISE,24	SET SYNCHRONIZATION FILE LOCK
M.FSLR	X'24'	H.FISE,25	RELEASE SYNCHRONIZATION FILE LOCK

File Gating Services Available To Other Services and Privileged Tasks

MODULE ENTRY PT.	DESCRIPTION
H.FISE,26	SET EXCLUSIVE FILE LOCK
H.FISE,27	RELEASE EXCLUSIVE FILE LOCK
H.FISE,28	SET SYNCHRONIZATION FILE LOCK
H.FISE,29	RELEASE SYNCHRONIZATION FILE LOCK
H.FISE,30	RELEASE FILE ALLOCATION IN FLT
H.FISE,31	WAIT FOR RELEASE OF EXCLUSIVE FILE LOCK
H.FISE,32	WAIT FOR FLT ENTRY SPACE
H.FISE,33	RECORD FILE ALLOCATION IN FLT
H.FISE,34	SET EXCLUSIVE LOCK IF FILE IS NOT ALLOCATED
H.FISE,35	RELEASE EXCLUSIVE LOCK (SPACE DEFINITION)
H.FISE,36	RELEASE ALL EXCLUSIVE LOCKS ON TASK TERMINATION

SUBROUTINE	DESCRIPTION
S.FISE1	SEARCH SMD FOR ENTRY
S.FISE2	SETUP FAT FOR SMD I/O
S.FISE3	SETUP SYSTEM FPT AND FCB
S.FISE4	RECORD DISC FILE ALLOCATION IN FLT
S.FISE5	WAIT FOR FLT ENTRY SPACE
S.FISE6	WAIT FOR RELEASE OF FILE EXCLUSIVE LOCK
S.FISE7	RELEASE DISC FILE ALLOCATION IN FLT
S.FISE8	SEARCH FLT FOR MATCHING ENTRY
S.FISE9	CONVERT LFC TO CONCATENATED FILE ID
S.FISE10	CHECKSUM DISC ALLOCATION MAP

3.7.1 Entry Point 1 - Get System SMD Entry

Functional Description

This entry point is used to locate the System Master Directory (SMD) entry for a System permanent file or memory partition. The SMD consists of a prime number of entries and resides on disc. This number is computed by an algorithm that doubles the number of entries specified at SYSGEN and adds one to convert to an odd number. Each entry contains eight words which define the file and is located by mapping the file name into a directory entry number. After a circular right shift of 1 bit of the left half of the file name, the two words of the file name are exclusively ORed to obtain a bit difference. This difference is then divided by the number of directory entries. The remainder of the division is the directory entry number of the space definition of the file. If a file name other than that used to compute the entry number is defined in the directory entry (collision mapping), 1 is repeatedly added to the entry number, modulo the number of directory entries, until the desired name is located or an inactive entry is encountered. If an inactive entry is encountered, a return is made to the caller with the space definition of the file zeroed.

Entry Conditions

Calling Sequence:

```
        M.CALL    H.FISE,1
```

Registers:

R3 — contains the denial return address in case of an unrecoverable I/O error to the SMD

R6,R7 — contains the permanent file name consisting of one to eight ASCII characters, left-justified, and blank filled.

Exit Conditions

Return Sequence:

```
        M.RTRN    5,6,7
```

Registers:

R5 — contains the UDT index in bytes 2 and 3

R6,R7 — zero if the entry for the file could not be located

(or)

The space definition of the specified file as follows:

R6	byte 0	file type
	bytes 1,2,3	starting disc address or starting memory page
R7	byte 0	File indicators as follows: Bit 0 - Active Permanent File Bit 1 - SYSGEN Memory Partition Bit 2 - Not saved via SAVE DEVICE Bit 3 - FAST File Bit 4 - Collision Mapping Bit 5 - Non-SYSGEN Memory Partition Bit 6 - Write Protected (RO) Bit 7 - Password Protected (PO)
	bytes 1,2,3	number of 192 word blocks in file or number of memory pages (bytes 1,2 only)

(or)

Return Sequence:

M.RTNA	3	R3 is the denial return address in case of an unrecoverable I/O error to the SMD

Registers:

None

External References

System Macros:

M.RTNA
M.RTRN

System Subroutines:

S.FISE1
S.FISE2
S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.2 Entry Point 2 - Rewrite SMD Entry

Functional Description

This entry point is used to output to the SMD an eight word permanent file or memory partition definition. It is provided primarily for the creation of new entries and the deletion of existing entries, but it may be utilized for altering the content of any entry. The specified permanent file or memory partition name is mapped into the directory in the same manner as in Entry Point 1. If no collision occurs on the first mapping, the specified eight word space definition is entered in the SMD. If there is a collision mapping, the specified space definition is tested to determine whether a FAST or SLOW permanent file is specified. If a SLOW permanent file is specified, the backup algorithm is utilized to locate the existing entry for the file or an inactive entry. If a FAST file is specified, the entry mapped into is tested. If that entry is SLOW, it is replaced by the specified entry values and the backup algorithm is employed to locate an inactive entry for the use of the old entry values. For any entry which is re-written, the collision mapping indicator (bit 4 of word 3) is maintained intact. Thus, for entries which are deleted, entries mapping "through" deletions may still be located.

NOTE: This entry point may not be used to change the name of a permanent file or memory partition and may not be used to change the definition of a file from SLOW to FAST.

Entry Conditions

Calling Sequence:

M.CALL	H.FISE,2

Registers:

R1	contains the address of the eight word space definition block to replace the existing entry for the file or to occupy the first inactive entry mapped into.
R4	contains the denial return address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

(or)

Return Sequence:

M.RTNA	4,1	R4 is the denial return address

Registers:

R1	contains the reason for denial =0 if a FAST file entry was to be written and a collision mapping occured with an existing FAST file entry =1 if SMD is full =2 if an unrecoverable I/O error to the SMD

External References

System Macros:

M.CALL	H.IOCS,3; H.IOCS,4
M.RTRN	
M.RTNA	

System Subroutines:

S.FISE1
S.FISE2
S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.3 Entry Point 3 - Allocate Temporary Disc Space

Functional Description

This entry point is used to effect the allocation of temporary disc file space. When this entry point is called, the allocation map for the specified disc is read into a FISE buffer area if it is not already in memory. The first string of reset bits satisfying the length requested is located (temporary space is allocated from the low to high end of a disc, unless Unidirectional File Allocation has been declared via the Operator Communications MODE command, in which case all disc space is allocated from high to low). The bit string is set, the allocation map is rewritten, the space definition is computed and return is made to the caller. If the specified device does not contain available space of sufficient length to satisfy the request, a denial return is made to the caller with the space definition zeroed.

This entry point decrements the count of total disc space available on all units in C.NTBA in the communications region.

Entry Conditions

Calling Sequence:

M.CALL H.FISE,3

Registers:

R4 contains the denial return address in case of an unrecoverable I/O error to the allocation map

R5 contains the Unit Definition Table (UDT) index (entry number) of the entry which defines the disc for which the space allocation is requested, or zero to indicate any disc.

R7 contains the number of 192 word blocks requested for allocation.

Exit Conditions

Return Sequence:

M.RTRN 5,6,7

Registers:

R5 contains the UDT index of the disc where the space was allocated

R6,R7 contain the space definition of the allocated file space.

(or)

zero if the requested space was not available

(or)

Return Sequence:

M.RTNA 4 R4 is the denial return address in case of an unrecoverable I/O error to the allocation map

Registers:

None

External References

System Macros:

M.CALL H.FISE,17; H.FISE,18
M.RTRN
M.RTNA

Abort Cases:

None

Output Messages:

None

3.7.4 Entry Point 4 - Deallocate Temporary Disc Space

Functional Description

This entry point is called to release temporary disc file space, making it available for allocation. Although permanent disc space may be deallocated through this entry point, Entry Point 7 should be used for that purpose. The allocation map for the specified disc is read into a FISE buffer area if it is not already in memory; the bit string corresponding to the specified space definition is reset; the allocation map is rewritten on the disc, and a return is made to the caller.

This entry point increments the count of total disc space available (C.NTBA).

Entry Conditions

Calling Sequence:

```
M.CALL    H.FISE,4
```

Registers:

R4	contains the denial return address in case of an unrecoverable I/O error to the allocation map
R5	contains the UDT index of the disc on which the file resides
R6	contains the starting disc address
R7	contains the number of 192 word blocks to release

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

(or)

Return Sequence:

M.RTNA	4	R4 is the denial return address in case of an unrecoverable I/O error to the allocation map

Registers:

None

External References

System Macros:

M.CALL M.RTRN M.RTNA	H.FISE,17; H.FISE,18; H.EXEC,16

Abort Cases:

None

Output Messages:

None

3.7.5 Entry Point 5 - Allocate Permanent Disc Space

Functional Description

This entry point functions identically to Entry Point 3 with the exception that the allocation map is searched from the end toward the beginning for a bit string of sufficient length to satisfy the request. Allocating permanent file space at the end of the disc and temporary file space at the beginning reduces fragmentation of disc usage, and reduces the time required to allocate space. If Unidirectional File Allocation mode has been set via the Operator Communications MODE command, this algorithm is used to allocate temporary space also.

This entry point decrements the count of total disc space available on all units in C.NTBA.

Entry Conditions

Calling Sequence:

M.CALL H.FISE,5

Registers:

R4	contains the denial return address in case of an unrecoverable I/O error to the allocation map

R5	contains the Unit Definition Table (UDT) index (entry number) of the entry which defines the disc for which the space allocation is requested, or zero to indicate any disc.
R7	contains the number of 192 word blocks requested for allocation

Exit Conditions

Return Sequence:

M.RTRN	5,6,7

Registers:

R5	contains the UDT index of the disc where the space was allocated
R6,R7	contain the space definition of the allocated file space (or) zero if the requested space was not available

(or)

Return Sequence:

M.RTNA	4 R4 is the denial return address in case of an unrecoverable I/O error to the allocation map

Registers:

None

External References

System Macros:

M.CALL	H.FISE,17; H.FISE,18
M.RTRN	
M.RTNA	

Abort Cases:

None

Output Messages:

None

3.7.6 Entry Point 6 - Ungate FISE

Functional Description

This entry point makes FISE avaialable for use and allows any users that are suspended awaiting FISE to gain access to it. If applicable, the calling task is permitted to be swapped after FISE is ungated.

Entry Conditions

Calling Sequence:

```
M.CALL     H.FISE,6
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

System Macros:

```
M.CALL     H.EXEC,18
M.RTRN
```

System Subroutines:

S.EXEC39

Abort Cases:

None

Output Messages:

None

3.7.7 Entry Point 7 - Deallocate Permanent Disc Space

Functional Description

This entry point is called to release permanent disc file space, making it available for allocation. Although temporary space may be deallocated through this entry point, Entry Point 4 should be used for that purpose. The allocation map for the specified disc is read into a FISE buffer area if it is not already in memory; the bit string corresponding to the specified space definition is reset; the allocation map is rewritten on the disc and a return is made to the caller.

This entry point increments the count of total disc space available (C.NTBA).

Entry Conditions

Calling Sequence:

```
M.CALL     H.FISE,7
```

Registers:

R4 contains the denial return address in case of an unrecoverable I/O error to the allocation map

R5 contains the UDT index of the disc on which the file resides

R6 contains the starting disc address

R7 contains the number of 192 word blocks to release

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

(or)

Return Sequence:

M.RTNA 4 R4 is the denial return address in case of an unrecoverable I/O error to the allocation map

Registers:

None

External References

System Macros:

M.CALL	H.FISE,17; H.FISE,18; H.EXEC,16
M.RTRN	
M.RTNA	

Abort Cases:

None

Output Messages:

None

3.7.8 Entry Point 8 - ASCII Compression

Functional Description

This entry point performs compression on an ASCII character string to yield its halfword equivalent.

Entry Conditions

Calling Sequence:

M.CALL H.FISE,8

Registers:

R6,R7	contain the one to eight character ASCII string, left-justified, and blank filled

Exit Conditions

Return Sequence:

M.RTRN 7

Registers:

R7	contains the equivalent of the ASCII string in the right halfword

External References

System Macros:

M.RTRN

Abort Cases:

None

Output Messages:

None

3.7.9 Entry Point 9 - Gate FISE

Functional Description

This entry point gates FISE and prevents its use by any other task until Entry Point 6 is called. If FISE is currently gated when this call is made, the scheduler is notified and the calling task is placed in a "Waiting For FISE" queue until FISE is freed by a call to Entry Point 6. Once gated, the calling task will not be allowed to swap until FISE is ungated.

Entry Conditions

Calling Sequence:

```
M.CALL      H.FISE,9
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

External References

System Macros:

```
M.CALL      H.EXEC,17
M.RTRN
```

System Subroutines:

S.EXEC38

Abort Cases:

None

Output Messages:

None

3.7.10 Entry Point 10 - Get User SMD Entry

Functional Description

This entry point is used to locate the entry for a specified user permanent file in the System Master Directory. If unsuccessful, an attempt is made to locate a system file or memory partition of the specified name, however, Entry Point 1 should be used for this purpose.

Entry Conditions

Calling Sequence:

```
M.CALL    H.FISE,10
```

Registers:

R3 contains the denial return address in case of an unrecoverable I/O error to the SMD

R4,R5 contain the user name consisting of one to eight ASCII characters, left-justified, and blank filled

R6,R7 contain the permanent file name consisting of one to eight ASCII characters, left-justified, and blank filled

Exit Conditions

Return Sequence:

```
M.RTRN    5,6,7
```

Registers:

R5 contains the UDT index in bytes 2 and 3

R6,R7 zero if the entry for the file could not be located

(or)

The space definition of the specified file as follows:

R6	byte 0	file type
	bytes 1,2,3	starting disc address or starting memory page
R7	byte 0	File indicators as follows: Bit 0-Active Permanent File Bit 1-SYSGEN Memory Partition Bit 2-Not saved via SAVE DEVICE Bit 3-FAST File Bit 4-Collision Mapping Bit 5-Non-SYSGEN Memory Partition Bit 6-Write Protected (RO) Bit 7-Password Protected (PO)
	bytes 1,2,3	number of 192 word blocks in file or number of memory pages (bytes 1,2 only)
Additionally	CC1	0 If no file, or a user file, was found 1 If a System file was found

(or)

Return Sequence:

M.RTNA	3	R3 is the denial return address in case of an unrecoverable I/O error to the SMD

Registers:

None

External References

System Macros:

M.RTNA
M.RTRN

System Subroutines:

S.FISE1
S.FISE2
S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.11 Entry Point 11 - Permanent File Allocation Check

Functional Description

This entry point checks access rights to a permanent file being statically or dynamically allocated.

Entry Conditions

Calling Sequence:

```
M.CALL     H.FISE,11
```

Registers:

R2,R3 contain the one to eight character password, left-justified, and blank filled

R4,R5 contain the one to eight character user name, left-justified, and blank filled, or zero if allocation of a system file only is to be checked

R6,R7 contain the one to eight character permanent file or memory partition name, left-justified, and blank filled

(or)

R1 contains the address of a three word permanent file resource requirement summary entry

R4,R5 contain the one to eight character user name, left-justified, and blank filled, or zero if allocation of a system file only is to be checked

R6 contains zero

Exit Conditions

Return Sequence:

```
M.RTRN     5,6,7
```

Registers:

R5 UDT index of disc on which file resides

R6,R7 contain the space definition of the specified permanent file. If access to the file is restricted and a matching password was not furnished, bits 6 and 7 of R7 are returned as follows:

Bit 6 = 1 Write access is not allowed
Bit 7 = 1 No access is allowed
Bits 6,7 = 0 If read and write access is allowed

(or)

R6,R7 zero if the specified file could not be located

Additionally:

CC1 = 1 if an unrecoverable I/O error occurred

External References

System Macros:

```
M.CALL   H.FISE,8
M.RTRN
```

System Subroutines:

```
S.FISE1
S.FISE2
S.FISE3
```

Abort Cases:

None

Output Messages:

None

3.7.12 Entry Point 12 - Create Permanent File

See Section 7.8.4 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.13 Entry Point 13 - Change Temporary File To Permanent

See Section 7.8.17 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.14 Entry Point 14 - Delete Permanent File or Non-SYSGEN Memory Partition

See Section 7.8.7 of the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.15 Entry Point 15 - Permanent File Log

Functional Description

This entry point provides a log of currently existing permanent files and memory partitions.

Entry Conditions

Calling Sequence:

```
        M.CALL    H.FISE,15
```

Registers:

R4 contains a byte-scaled value which specifies the type of log to be performed as follows:

= 0 specifies a single named system or user file
= 1 specifies all permanent files
= 2 specifies system files only
= 3 specifies user files
= 4 specifies a single named system file

If R4 contains zero and a user name is associated with the calling task, an attempt is made to locate the user file directory entry for the given file name. If unsuccessful, the system file directory entry, if any, is located. If a user name is not associated with the calling task, the file is assumed to be a system file.

If R4 contains 3 and the calling task has an associated user name, that user's files are logged or all files are logged if the calling task has no associated user name.

R5 contains the address of an eight word area where the file directory entry is to be stored

R6,R7 contain the one to eight character file name if R4 contains 0 or 4

Exit Conditions

Return Sequence:

```
        M.RTRN    4,5
```

Registers:

R4 If R4 contains 0 or 4 in the service call, R4 is destroyed. If R4 contains 1,2,or 3 in the service call, this service is called repeatedly to obtain all the pertinent file descriptions. The parameter in R4 is specified in the first call only. R4 is returned containing the address of the next directory entry to be returned. The value returned in R4 must be unchanged upon the subsequent call to this service.

R5 contains zero if R4 contained 0 or 4 in the service call and the specified file could not be located, or R4 contained 1,2, or 3 and all pertinent files have been logged. Otherwise, R5 is unchanged.

Additionally, the eight word SMD entry, if any, is stored at the address specified in R5. The password field contains zero or one to indicate the absence or presence of a password respectively.

CC1 is set to 1 if there was an unrecoverable I/O error

External References

System Macros:

M.CALL	H.IOCS,3
M.RTRN	

System Subroutines:

S.FISE1
S.FISE2
S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.16 Entry Point 16 - Validate Username

Funtional Description

This entry point validates a user name and optional key against the user name file, M.KEY, if it exists.

Entry Conditions

Calling Sequence:

```
M.CALL    H.FISE,16
```

Registers:

R3 should contain zero

R4,R5 contain the one to eight character user key, left-justified and blank filled, or zero in R4 and R5 contains the compressed user key

R6,R7 contain the one to eight character user name, left-justified and blank filled. Each character must have an ASCII equivalent in the range 01 through 7F.

Exit Conditions

Return Sequence:

M.RTRN 3
(or)
M.RTRN 6,7

Registers:

R3	contains the address of an 8W area in T.BBUFA which contains the M.KEY entry, or is zero if no M.KEY file is present
R6,R7	unchanged if the user name is valid. Both R6 and R7 are zero if the user name contains invalid characters or is not contained in M.KEY or the user key is not correct.

Additionally, CC1 = 1 if an unrecoverable I/O error occured

External References

System Macros:

M.CALL	H.FISE,1; H.FISE,8; H.IOCS,3
M.RTRN	

System Subroutines:

S.FISE3

Abort Cases:

None

Output Messages:

None

NOTE: MPX-32 M.KEY file entries are twelve words long

Words 0-1	Username
2-3	OPCOM Flags
4-5	Tab Settings
6	Unused
7	Compressed key in right halfword Left halfword is zero
8-9	Key (ASCII)
10-11	Unused

3.7.17 Entry Point 17 - Read Allocation Map Into Memory

Functional Description

This entry point determines if the allocation bit map for the specified disc is resident in memory or not, and reads the map into memory if required.

Entry Conditions

Calling Sequence:

```
M.CALL      H.FISE,17
```

Registers:

R3	contains the UDT index of the disc whose allocation map is to be read

Exit Conditions

Return Sequence:

```
M.RTRN      3
```

Registers:

R3	unchanged or zero if the specified UDT index does not exist

Additionally, CC1 is set if an unrecoverable I/O error occurred when trying to read the map.

External References

System Macros:

```
M.CALL      H.IOCS,3
M.RTRN
```

System Subroutines:

S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.18 Entry Point 18 - Write Allocation Map to Disc

Functional Description

This entry point writes the currently resident allocation bit map back to the disc it describes.

Entry Conditions

Calling Sequence:

```
M.CALL     H.FISE,18
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Additionally, CC1 is set if an unrecoverable I/O error occurred when trying to write the map.

External References

System Macros:

```
M.CALL     H.IOCS,4
M.RTRN
```

System Subroutines:

S.FISE3

Abort Cases:

None

Output Messages:

None

3.7.19 Entry Point 19 - Undefined

Functional Description

This entry point is reserved for future expansion. Its current intended use is to implement a Contract File Space service. A call to this entry point causes an immediate return to the calling task; no action is taken prior to the return.

Entry Conditions

Calling Sequence:

```
        M.CALL     H.FISE,19
```

Registers:

None

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

External References

System Macros:

M.RTRN

Abort Cases:

None

Output Messages:

None

3.7.20 Entry Point 20 - Create Permanent File

Functional Description

This entry point allocates disc space for the specified permanent file and writes a corresponding entry into the SMD. Optionally, the allocated space is zeroed.

Entry Conditions

Calling Sequence:

```
        CALM      X'75'

        (or)

        M.CALL    H.FISE,20
```

Registers:

R1	bytes 1,2	contain restricted file code
	byte 3	contains file type code
R2		file size
R3	bit 0	set if a system file
	bit 1	set if pre-zero
	bit 10	set if not a save device
	bit 11	set if a fast file
	bit 14	set if read only
	bit 15	set if password only
	byte 2	contains device type code
	byte 3	contains optional device address (if present, bit 16 is also set)
R4,R5		password or R4=0
R6,R7		file name

Exit Conditions

Return Sequence:

```
        M.RTRN
```

Registers:

None

(or)

Return Sequence:

```
        M.RTRN 6,7
```

Registers:

R6	=1,	file already exists
	=2,	fast file collision mapping occurred
	=3,	restricted access but no password supplied
	=4,	disc space unavailable
	=5,	specified device not configured or available
	=6,	specified device is off-line
	=7,	SMD is full
	=8,	specified device type is not configured
	=9,	file name or password contain invalid characters
R7	zero	

External References

System Macros:

M.RTRN
M.CALL

Abort Cases:

FS01	Unrecoverable I/O error to SMD
FS02	Unrecoverable I/O error to disc allocation map

Output Messages:

None

3.7.21 Entry Point 21 - Change Temporary File to Permanent

Functional Description

This entry point changes the status of a temporary file allocated to the calling task to permanent. The file must be an open, temporary, SLO or SBO file.

Entry Conditions

Calling Sequence:

```
CALM      X'76'

(or)

M.CALL    H.FISE,21
```

Registers:

R1	byte 3	contains file type code
R2	bytes 0,1,2	contain logical file code

R3	bit 0	set if a system file
	bit 1	set if pre-zero
	bit 10	set if not a save device
	bit 11	set if a fast file
	bit 14	set if read only
	bit 15	set if password only
R4,R5		password or R4=0
R6,R7		file name

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

(or)

Return Sequence:

M.RTRN 6,7

Registers:

R6	=1,	file already exists
	=2,	fast file collision mapping occurred
	=3,	restricted access but no password supplied
	=4,	file not open, temporary, SLO or SBO file
	=7,	SMD is full
	=9,	file name or password contain invalid characters
R7	zero	

External References

System Macros:

M.RTRN
M.CALL

Abort Cases:

FS01	Unrecoverable I/O error to SMD
FS02	Unrecoverable I/O errror to disc allocation map

Output Messages:

None

3.7.22 Entry Point 22 - Set Exclusive File Lock

See Section 7.8.14 in the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.23 Entry Point 23 - Release Exclusive File Lock

See Section 7.8.13 in the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.24 Entry Point 24 - Set Synchronization File Lock

See Section 7.8.11 in the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.25 Entry Point 25 - Release Synchronization File Lock

See Section 7.8.10 in the MPX-32 Reference Manual for a detailed description of this entry point.

3.7.26 Entry Point 26 - Set Exclusive File Lock (M.CALL Only)

Functional Description

This entry point is provided for use by other system services. It is functionally identical to the M.FXLS (H.FISE,22) service except that it identifies the file by the concatenated UDT index and starting sector rather than by logical file code.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and file starting sector number
R4	timev (see H.FISE,22)
M.CALL	H.FISE,26

Exit Conditions

Return Sequence:

M.RTRN	R7

Registers:

R7 = 0, Request accepted, file is exclusively locked.

= 1, Request denied, file is allocted to multiple tasks, or is already exclusively locked and cannct be exclusively locked by this task.

= 2, Request denied, FLT space not available.
Note 1: applies to privileged requests to exclusively lock an unallocated file, and does not apply to SVC utilization through H.FISE,22.

= 3, Request denied, time-out occurred while waiting for FLT space. (See Note 1 above.)

= 4, Request denied, time-out occurred while waiting to become lock owner.

3.7.27 Entry Point 27 - Release Exclusive File Lock (M.CALL Only)

Functional Description

This entry point is provided for use by other system services. It is functionally identical to the M.FXLR (H.FISE,23) service except that it identifies the file by the concatenated UDT index and starting sector rather than by logical file code.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and file starting sector number
M.CALL	H.FISE,27

Exit Conditions

Return Sequence:

M.RTRN	R7

Registers:

R7 = 0, Request accepted, exclusive file lock released

= 1, Request denied, an exclusive file lock owned was not owned by this task

3.7.28 Entry Point 28 - Set Synchronization File Lock (M.CALL Only)

Functional Description

This entry point is provided for use by other system services. It is functionally identical to the M.FSLS (H.FISE,24) service except that it identifies the file by the concatenated UDT index and starting sector rather than by logical file code.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and starting sector number
R4	timev (see H.FISE,24)
M.CALL	H.FISE,28

Exit Conditions

Return Sequence:

M.RTRN	R7

Registers:

R7 = 0, Request accepted, synchronization lock set.

= 1, Request denied, synchronization lock already owned by another task.

= 2, Request denied, time-out occurred while waiting to become lock owner.

= 3, Request denied, matching FLT entry not found.

3.7.29 Entry Point 29 - Release Synchronization File Lock (M.CALL Only)

Functional Description

This entry point is provided for use by other system services. It is functionally identical to the M.FSLR (H.FISE,25) service except that it identifies the file by the concatenated UDT index and starting sector rather than by logical file code.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and starting sector number
M.CALL	H.FISE,29

Exit Conditions

Return Sequence:

M.RTRN	R7

Registers:

R7 = 0, Request accepted, synchronization lock released.

= 1, Request denied, synchronization lock was not set.

3.7.30 Entry Point 30 - Release File Allocation in FLT

Functional Description

This entry point is provided for use by system static and dynamic deallocation services. It searches the FLT for a matching file identifier. Once a matching FLT entry is found, any outstanding locks owned by the current task are released, and the allocation count is decremented. If the allocation count then equals zero, the FLT entry is marked available. The associated wait queues will also be pulled.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and file starting block number
M.CALL	H.FISE,30

Exit Conditions

Return Sequence:

M.RTRN

3.7.31 Entry Point 31 - Wait for Release of Exclusive File Lock

Functional Description

This entry point is provided for use by system static and dynamic allocation services. It searches the FLT for an existing exclusive lock on the specified file. If one exists, the task is placed in a wait state until the lock is released. Otherwise, an immediate return is issued.

Entry Conditions

Calling Sequence:

R6	Concatenated UDT index and file starting block number
M.CALL	H.FISE,31

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.7.32 Entry Point 32 - Wait for FLT Entry Space

Functional Description

This entry point is provided for system static and dynamic allocation services. It returns immediately if FLT entry space is available, otherwise the task is placed in a wait state until FLT space is available.

Entry Conditions

Calling Sequence:

```
M.CALL     H.FISE,32
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.7.33 Entry Point 33 - Record Disc File Allocation In FLT

Functional Description

This entry point is provided for use by the allocation system service to maintain system wide information on all currently allocated permanent disc files. It is used in conjunction with H.FISE,30 which is called by the deallocation system service.

Entry Conditions

Calling Sequence:

```
R6         File ID (Concatenated UDT index and starting sector)
M.CALL     H.FISE,30
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

3.7.34 Entry Point 34 - Exclusive Lock File If Unallocated

Functional Description

This entry point is used to insure that a file is not currently allocated by any task, and then to set an exclusive lock to prohibit subsequent allocations. It is used by the file-delete system service, and is used in conjunction with H.FISE,35.

Entry Conditions

Calling Sequence:

R5	UDT index
R6	Starting sector address
M.CALL	H.FISE,34

Exit Conditions

Return Sequence:

M.RTRN	R4

Registers:

R4	= 0, Request accepted.
	= 1, Request denied, file is allocated.
	= 2, Request denied, file is exclusively locked (but not allocated) by another task.
	= 3, Request denied, FLT space unavailable.

3.7.35 Entry Point 35 - Release Exclusive Lock (Unallocated File)

Functional Description

This entry point is identical to H.FISE,27 except that it accepts the UDT index and starting sector address as separate arguments, then concatenates them to form the file identification.

Entry Conditions

Calling Sequence:

R5	UDT index
R6	Starting sector address
M.CALL	H.FISE,35

Exit Conditions

Return Sequence:

M.RTRN

Note: Request is ignored if the calling task is not the lock owner.

Registers:

None

3.7.36 Entry Point 36 - Release Exclusive Locks For Unallocated Files On Task Termination

Functional Description

This entry point is called during task termination processing to release any file exclusive locks which are set for files which the task did not have allocated. Note that exclusive locks on allocated files are released when the file is deallocated.

Entry Conditions

Calling Sequence:

```
M.CALL      H.FISE,36
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

3.7.37 Entry Point 99 - SYSGEN Initialization

Functional Description

This entry point is for internal use only and is called during SYSGEN. H.FISE sets up its Entry Point Table then returns to SYSGEN.

3.7.38 System Master Directory (SMD)

Functional Description

Each permanent file, temporary file, or memory partition has associated with it a two word space definition which describes the location, length, etc. of the file or memory partition. The space definition of permanent files or memory partitions resides on the System Master Directory, and together with the permanent file name, user name and password, or memory partition name, comprise the SMD entry.

The space definition of temporary files are retained in the File Assignment Table area of the defining task's TSA until termination of the task or completion of System Output. At that time, these definitions are returned to the File System Executive for deallocation of the defined space. Deallocation of permanent file space is performed via Entry Point 14 or by the File Manager upon encountering a DELETE directive.

Memory partitions, similar to permanent disc files, have a two word space definition in the associated SMD entry. The SMD entry for a statically allocated memory partition is built by SYSGEN when a PARTITION directive is specified. This type of partition can be deleted only by omitting the definition in a subsequent SYSGEN warm or cold start. The SMD entry for a dynamically allocated memory partition is built by the File Manager in response to a CREATEM directive. This type partition is deleted by the File Manager upon encountering a DELETE directive, or alternatively via Entry Point 14.

System Master Directory (SMD) Entry

Permanent File Format:

Word 0,1 contain the one to eight ASCII character, left-justified, blank filled file name

Word 2,3 contain the space definition as follows:

Word 2

Byte 0 File Type -This field contains one or two hexadecimal digits to identify the origin of the file as follows:

ED - Editor Save File
EE - Editor Store File
FE - Editor Work File
FF - SYSGEN Created File
BA - BASIC File
CA - Cataloged Load Module

Bytes 1-3 This field contains the starting (base) disc address of the file expressed as the starting allocation unit number.

Word 3

Byte 0 File Indicators - Bits set in this field indicate the following:

Bit	
0	Active Permanent File
1	SYSGEN Memory Partition
2	File is not saved in response to the SAVE FILE File Manager directive

3 FAST File
4 Collision Mapping
5 Non-SYSGEN Memory Partition
6 Password required to write (read-only file)
7 Password required to read or write (password-only file)

Bytes 1-3 This field contains the number of 192 word blocks in the file

Word 4,5 contain the one to eight ASCII character, left-justified, blank filled, user name or zero if no user name is associated with the file

Word 6 contains the compressed password or zero in Bytes 0 and 1 and the UDT index of the disc on which the file resides in Bytes 2 and 3

Word 7 Zero

Memory Partition Format:

Word 0,1 contain the memory partition name; GLOBALxx or DATAPOOL

Word 2,3 contain the space definition as follows:

Word 2

Bytes 0-1 contain the partition's starting logical page number

Bytes 2-3 contain the partition's starting physical page number if the partition is created by SYSGEN or are zero for a non-SYSGEN partition

Word 3

Byte 0 File Indicators - Same as in Permanent File Format definition above

Byte 1 Memory Class - this field contains a 1,2, or 3 to indicate the class of memory (E, H, or S) this partition is to be allocated

Bytes 2-3 This field contains the number of pages allocated for the partition

Word 4,5 Zero (Memory partitions are always System files)

Word 6 contains the compressed password or zero in Bytes 0 and 1 and zero in Bytes 2 and 3

Word 7 Zero

3.7.39 Disc Allocation Maps

Functional Description

Each disc consists of a string of 192 word records. Disc space is allocated in blocks (called allocation units) of records. The number of records in an allocation unit is disc dependent. Each type of disc contains a different number of allocation units. Allocatable disc space is followed by a variable length record which serves as the space allocation map for the device. The map begins with the first bit of the record and consists of as many bits as there are allocation units on the disc. Each bit defines the state of the corresponding allocation unit and is set if the unit is allocated and reset if the unit is available for allocation. Following the allocation map is unused disc space (less than one allocation unit in length) which results from the disc sizes not being even multiples of the allocation unit. Additionally, an area of the disc may be reserved for diagnostic purposes, and therefore, not available for allocation (see Section 3.7.37.2).

A separate allocation bit map is maintained for each disc defined to the system at SYSGEN. The File System Executive builds an Allocation Map Table defining each allocation map during its SYSGEN Initialization phase. The allocation maps reside on the particular disc they define and are read into memory one at a time as required.

If control switch 0 is set, checksumming is performed on the bit maps whenever they are read from disc, written to disc, or changed during disc space allocation/deallocation. An Allocation Map Checksum Table, containing one entry for each disc, is defined at SYSGEN time. Detection of a checksum error causes execution of M.KILL with the abort code FS04 in R5.

The allocation bit maps vary in length as described above and as shown in the following table:

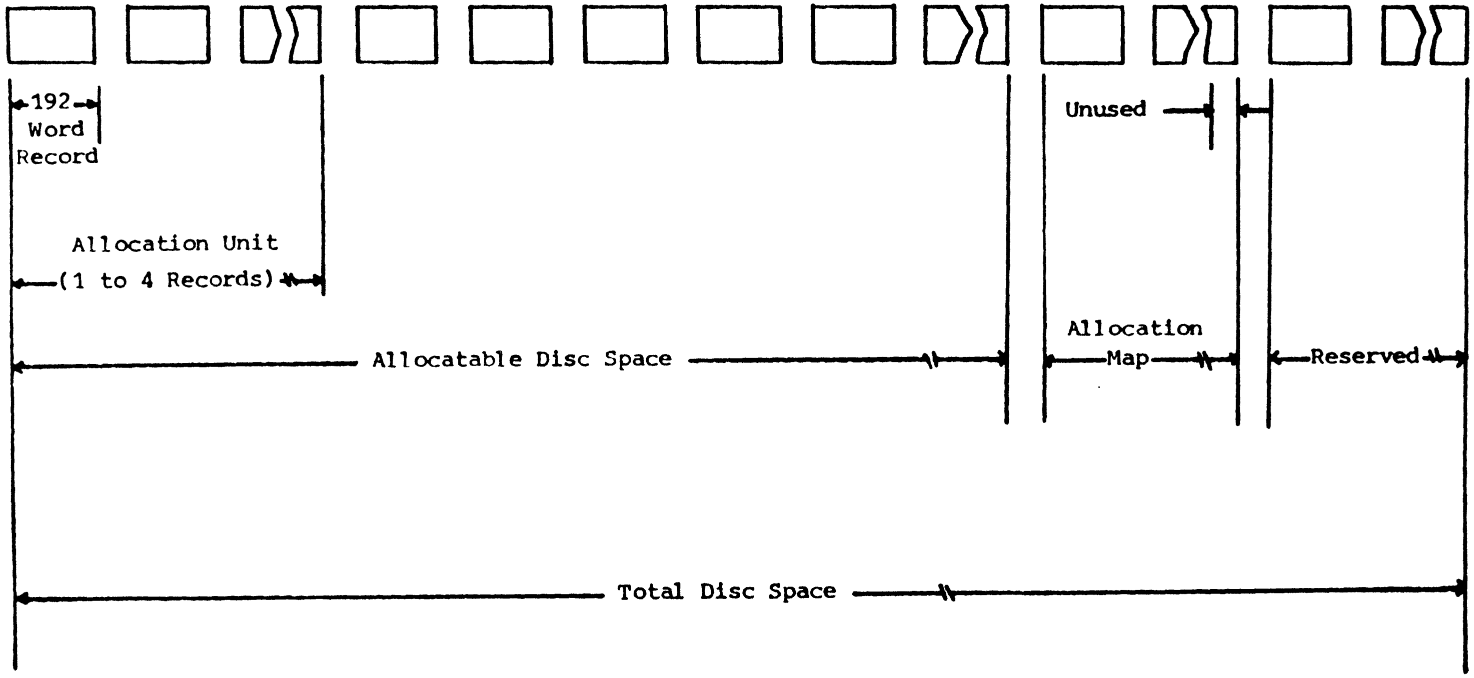
Unit Size	Type	Words/ Sector	Sectors/ Track	Number of Heads	Max Cylinders	Sectors/ Cylinder	Max Sectors	Sectors/ Allocation	Max Alloc. Units	Bit Map Size*	Max Byte Capacity
4MB	Fixed Head	192W	23	256	0	0	5888	1	5888	184	4.52MB
5MB	Fixed Head	192W	23	4	64	92	5888	1	5888	184	4.52MB
10MB	Cartridge Disc										
	5MB Removable Media	96W	16	2	400	32	12,800	1	12,800	200	4.91MB
	5MB Captive Media	96W	16	2	400	32	12,800	1	12,800	200	4.91MB
	Totals	-	-	4	-	64	25,600	-	25,600	400	9.82MB
40MB	Moving Head	192W	23	5	400	115	46,000	2	23,000	719	35.32MB
80MB	Moving Head	192W	23	5	800	115	92,000	2	46,000	1438	70.65MB
300MB	Moving Head	192W	23	19	800	437	349,600	4	87,400	2732	268.49MB
Extended I/O Devices											
1MB	Floppy Disc	64W	26	2	77	52	4002	3	1334	-	1.02MB
5MB	Fixed Head	192W	20	4	64	80	5120	1	5120	184	3.93MB
32MB	Cartridge Disc										
	16MB Removable Media	192W	20	1	800	20	16,000	2	8000	250	12.28MB
	16MB Captive Media	192W	20	1	800	20	16,000	2	8000	250	12.28MB
	Totals	-	-	2	-	40	32,000	-	16,000	500	24.56MB
40MB	Moving Head	192W	20	5	400	100	40,000	2	20,000	625	30.72MB
80MB	Moving Head	192W	20	5	800	100	80,000	2	40,000	1250	61.44MB
300MB	Moving Head	192W	20	19	800	380	304,000	4	76,000	2375	233.47MB

* Decimal Words

Note: The 10MB Cartridge Disc and 32MB Cartridge Disc contain removable and captive media within a single cabinet. Software treats the removable and captive medias as separate devices, each with its own device subaddress. Figures presented for these devices include each medias capacity and total drive capacity.

All discs with a Unit Size of 24MB or larger have reserved cylinders that are not useable by the operating system, i.e., the 80MB Moving Head Disc physically has 823 cylinders, however, only 800 are defined to the operating system.

3.7.39.2 Disc Format



3.7.40 MPX-32 Disc File Gating

MPX-32 provides both mutual exclusion and access synchronization methods of disc file gating. These functions are used by system services and are available to privileged and unprivileged tasks to ensure the access integrity of associated files.

3.7.40.1 General Method

MPX-32 maintains a memory resident File Lock Table (FLT). It contains an entry for each permanent disc file that is currently allocated. An FLT entry in turn contains an allocation count to reflect multiple tasks which concurrently have the associated file allocated. A file is uniquely identified in the FLT by the concatenation of its UDT index and starting sector number. The two-word FLT entry also contains space to reflect the program number of the task which has exclusively locked the file, or the program number of the task which has a synchronization lock set for the file.

3.7.40.2 Locking Services

Services are provided to set exclusive or synchronization locks for files that are allocated to a task. A task may use a logical file code (lfc) to identify the file when the lock request is made.

3.7.40.3 Exclusive Lock (FXL)

In order for an unprivileged task to exclusively lock a file, the file must be allocated to the requesting task, and will therefore have an entry in the FLT. When an FXL request is made it will be accepted if the file is allocated to the requesting task, and is not concurrently allocated to another task. Once a file has been exclusively locked, other tasks will be prevented from allocating it until the exclusive lock is released.

3.7.40.4 Synchronization Lock (FSL)

In order for a task to set a synchronous lock for a file, the file must first be allocated to the requesting task, and will therefore have an entry in the FLT. When an FSL request is made, it will be accepted if the file is allocated and an FSL for the file is not already owned by another task which has the file concurrently allocated.

3.7.40.5 Task Queueing

When a locking service is requested, the task may optionally specify an immediate denial return if the lock is unavailable, or the task may specify that it wishes to wait until the lock is available. If it specifies the wait option, it may also specify a watchdog timer value to ensure a return of control to the task if the lock does not become available within the time specified.

3.7.40.6 Cooperative Environment

It should be understood that use of the file locking services is voluntary. Participating tasks must utilize the file locking services in a cooperative environment to ensure properly synchronized access and to avoid deadlock situations.

3.7.41 System Subroutine S.FISE1 - Search SMD For Entry

Functional Description

This subroutine searches the SMD for the specified entry and returns the address of an 8W area containing the SMD entry if successful. If a username is specified, a search is made for the named userfile, else a system file is assumed. If the named userfile does not exist, the entry describing the system file of the same name is returned, if any.

```
***************************************************************************
*                               CAUTION                                    *
*            When this subroutine is called, it is assumed                 *
*            that calls have previously been made to gate FISE             *
*            and also that the caller will ungate FISE after               *
*            S.FISE1 returns                                               *
*                                                                          *
***************************************************************************
```

Entry Conditions

Calling Sequence:

```
        BL          S.FISE1
```

Registers:

R1 contains the FCB address - FCB, FAT, and FPT must be setup for I/O to the SMD

R4,R5 contain the one to eight ASCII character, left-justified, blank filled username, or zero if a system file only is to be found

R6,R7 contain the one to eight ASCII character, left-justified, blank filled filename of the entry to be found

Exit Conditions

Return Sequence:

```
        TRSW        R0
```

Registers:

R2 contains the address of an 8W area containing the SMD entry if the specified file exists

CC1 =0 if the specified file was found
=1 if no file was found

CC2 = 0 if no file was found or a user file was found during a user search
= 1 if a System file was found while looking for a user file and none existed

3.7.42 System Subroutine S.FISE2 - Setup FAT for SMD I/O

Functional Description

This subroutine sets up the system FAT to perform I/O to the SMD. It uses registers R1, R2, R3, and R5.

Entry Conditions

Calling Sequence:

BL S.FISE2

Registers:

R3 contains the UDT index of the disc on which the SMD resides

Exit Conditions

Return Sequence:

TRSW R0

Registers:

R1 contains the system FAT address

R5 contains the logical file code '(S)'

3.7.43 System Subroutine S.FISE3 - Setup System FPT and FCB

Functional Description

This subroutine sets up the system FPT and FCB for I/O. It uses registers R1, R2, R3, R4, and R5.

Entry Conditions

Calling Sequence:

BL	S.FISE3

Registers:

R1	contains the system FAT address
R4	contains the denial return address in case of an unrecoverable I/O error to the SMD
R5	contains the logical file code '(S)'

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R3	contains the system FCB address

Additionally, the system FPT and FCB are setup for a 192 word transfer to/from T.BBUFA

3.7.44 System Subroutine S.FISE4 - Record Disc File Allocation in FLT

Functional Description

This subroutine modifies the File Lock Table (FLT) to reflect the additional allocation.

Entry Conditions

Calling Sequence:

BL	S.FISE4

Registers:

R6	contains file ID (concatenated UDT index and starting sector number)

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1,R2,R3,R5	destroyed
R4,R6	saved
R7	0, request accepted 1, request denied (file is exclusively locked) 2, request denied (FLT space not available)

3.7.45 System Subroutine S.FISE5 - Wait for FLT Entry Space

Functional Description

This subroutine enqueues the caller until File Lock Table (FLT) space becomes available.

Entry Conditions

Calling Sequence:

```
BL        S.FISE5
```

Registers:

None

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2,R3	saved
R4,R5,R6,R7	destroyed

3.7.46 System Subroutine S.FISE6 - Wait for Release of File Exclusive Lock

Functional Description

This subroutine enqueues the caller until the file lock owner releases the file lock.

Entry Conditions

Calling Sequence:

```
BL        S.FISE6
```

Registers:

R6	contains file ID (concatenated UDT index and starting sector number)

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2,R3,R5,R7	destroyed
R4,R6	saved

3.7.47 System Subroutine S.FISE7 - Release Disc File Allocation in FLT

Functional Description

This subroutine decrements the use count in the file by one. The entry is cleared when the use count goes to zero.

NOTE

> Queued exclusive locks (QFXL) must be pulled if an exclusive lock (FXL) is owned by the releasing task or if the decremented allocation count is less than or equal to 1 and FXL is not set.

Entry Conditions

Calling Sequence:

```
BL        S.FISE7
```

Registers:

R6	contains file ID (concatenated UDT index and starting sector number)

Exit Conditions

Return Sequence:

```
TRSW      R0
```

Registers:

R1,R2,R3,R5,R7	destroyed
R4,R6	saved

3.7.48 System Subroutine S.FISE8 - Search FLT for Matching Entry

Functional Description

This subroutine searches the File Lock Table (FLT) for a matching concatenated file ID.

Entry Conditions

Calling Sequence:

```
        BL          S.FISE8
```

Registers:

R6	contains file ID (concatenated UDT index and starting sector number)

Exit Conditions

Return Sequence:

TRSW	R0	ignore (FLT not configured)
	R0+1W	no match, no FLT space
	R0+2W	no match, X1=free entry address
	R0+3W	match, X1=entry address

Registers:

R1	destroyed, or contains matching entry address or free entry address
R2,R5	destroyed
R3,R4,R6	saved
R7	2 if no match and no FLT space, otherwise = 0

3.7.49 System Subroutine S.FISE9 - Convert LFC to Concatenated File ID

Functional Description

This subroutine converts the lfc in register 5 to the concatenated file ID, consisting of the UDT index and starting sector address.

Entry Conditions

Calling Sequence:

```
        BL          S.FISE9
```

Registers:

R5	contains lfc

Exit Conditions

Return Sequence:

TRSW	R0	error return
	R0+1W	normal return

Registers:

R1,R2	destroyed
R3,R5	saved
R4	saved (destroyed if denial return)
R6	concatenated ID (if found)
R7	0, request accepted 5, request denied, (lfc not found) 6, request denied, (lfc not assigned to permanent disc file)

3.7.50 System Subroutine S.FISE10 - Checksum Disc Allocation Map

Functional Description

This subroutine computes the checksum of a specified disc's allocation map by sequentially summing halfwords. The checksum is compared to the current checksum for the disc in the Disc Allocation Map Checksum Table (DAMCST). Registers 2, 3, 4, 6, and 7 are used.

Entry Conditions

Calling Sequence:

BL	S.FISE10

Registers:

R1	points to the disc's entry in the Disc Allocation Map Table (DAMAPT)

Exit Conditions

Return Sequence:

TRSW	R0

Registers:

R1	unchanged

External References

System Macros:

M.KILL

Abort Cases:

FS04	checksum error detected

Output Messages:

None

3.8 Interrupt and Trap Processors

3.8.1 Power Fail Save - Auto Start and Interrupt/Trap Processor(H.IP00)

Functional Description

Processing performed at this level is considered to be dependent upon the installation and application. Therefore, MPX-32 provides a minimal routine for this level which may be easily overridden by a user-supplied routine. The processor executes a HALT instruction. Processing may be continued at the point of interrupt by the operator depressing the program STEP switch on the console to bypass the HALT "loop" and then depressing the RUN/HALT switch. Registers and PSD will remain intact but I/O in progress will be effected by the execution of the HALT instruction.

Entry Conditions

Calling Sequence:

Occurrence of interrupt or trap signal at priority level X'00'.

Exit Conditions

Return Sequence:

Depress program STEP switch on console.
Depress RUN/START switch on console.

NOTE

Depressing the RUN/START switch without first depressing STEP results in a branch that returns to the execution of the HALT instruction.

External References

System Macros:

None

Abort Cases:

None

Output Messages:

None

3.8.2 System Overide Interrupt/Trap Processor - 32/7x only (H.IP01)

Functional Description

Processing at this level is considered to be application and installation dependent, and, therefore, a minimal routine is provided which may be easily overridden by a user-supplied routine. The processor executes a HALT instruction. Processing may be continued at the point of interrupt with the registers and PSD intact.

Entry Conditions

Calling Sequence:

Occurrence of interrupt or trap signal at priority level X'01'.

Exit Conditions

Return Sequence:

Depress program STEP switch on console.

Depress RUN/START switch on console.

NOTE

Depressing the RUN/START switch without first depressing STEP results in a branch that returns to the execution of the HALT instruction.

External References

System Macros:

None

Abort Cases:

None

Output Messages:

None

3.8.3 System Auto Start Trap Processor - 32/27 only (H.IPAS)

Functional Description

Processing at this level is considered to be application and installation dependent, and, therefore, a minimal routine is provided which may be easily overridden by a user-supplied routine. The processor executes a HALT instruction. Processing may be continued at the point of interrupt with the registers and PSD intact.

Entry Conditions

Calling Sequence:

Occurrence of interrupt or trap signal at priority level X'01'.

This trap occurs during the power up sequence, provided a valid scratch pad image is resident in low memory (core memory or MOS memory with battery back-up only).

Exit Conditions

Returne Sequence:

Depress program STEP switch on console.

Depress RUN/START switch on console.

NOTE

Depressing the RUN/START switch without first depressing STEP results in a branch that returns to the execution of the HALT instruction.

External References

System Macros:

None

Abort Cases:

None

Output Messages:

None

3.8.4 Memory Parity Trap Processor (H.IP02)

Functional Description

A memory parity error is considered to be an indication of total hardware failure and is treated as a fatal system crash. Registers are loaded (for display) with the saved PSD and the instruction resulting in the trap, and the M.KILL macro is invoked. Processing may be continued at the point of interrupt with the registers and PSD intact, but I/O in progress when the HALT was executed will be terminated.

Entry Conditions

Calling Sequence:

Occurrence of interrupt signal at priority level X'12', or trap signal at level X'02'.

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1	PSD saved by the hardware when the trap occurred
R2	Physical address of the instruction causing the trap
R3	Instruction being executed when trap occurred
R4	CPU status word stored when trap occurred
R5	Abort code
R6	Address of register save block
R7	Trap

Exit Conditions

Return Sequence:

M.KILL

NOTE

Unless a user-supplied System Override Interrupt/Trap Processor is connected, processing may be continued by depressing the program STEP switch and then the RUN/START switch on the console.

External References

System Macros:

M.KILL

Abort Cases:

MP01	Memory error occurred

Output Messages:

None

3.8.5 Non-Present Memory Trap Processor (H.IP03)

Functional Description

This routine results in the task currently in execution being aborted. A register push-down and PSD save is made, the abort request is reported to the scheduler, and a standard exit is performed.

Entry Conditions

Calling Sequence:

Occurrence of interrupt signal at interrupt level X'24', or trap signal at level X'03'.

Exit Conditions

Return Sequence:

BL	S.EXEC5A

Registers:

R2	Register save area address
R5	Abort code
R6,R7	PSD

External References

System Macros:

None

Abort Cases:

NM01	Non-Present Memory Trap (all cases)

Output Messages:

None

3.8.6 Undefined Instruction Trap Processor (H.IP04)

Functional Description

This routine results in an abort of the task currently in execution.

Entry Conditions

Calling Sequence:

Occurrence of an interrupt signal at priority lvel X'25', or trap signal at level X'04'.

Exit Conditions

Return Sequence:

BU	S.EXEC5A

Registers:

R2	Register save area address
R5	Abort code
R6,R7	PSD

External References

System Macros:

None

Abort Cases:

UI01	Unimplemented instruction

Output Messages:

None

3.8.7 Privilege Violation Trap Processor (H.IP05)

Functional Description

This routine results in an immediate abort of the task currently in execution.

Entry Conditions

Calling Sequence:

Occurrence of an interrupt signal at priority level X'26', or a trap signal at level X'05'.

Exit Conditions

Return Sequence:

BL	S.EXEC5A

Registers:

R2	Register save area address
R5	Abort code
R6,R7	PSD

External References

System Macros:

None

Abort Cases:

PV01	Privilege violation (all cases)

Output Messages:

None

3.8.8 SVC Trap Processor (H.IP06)

Functional Description

This processor provides the SVC Secondary Vector Table and selects the appropriate processing routine based on SVC type.

Entry Conditions

- Occurrence of a trap signal at priority level X'06'.

Exit Conditions

SVC TYPE	DESCRIPTION
0	M.CALL Processor
1	SVC Type '1' Processor
2	SVC Type '2' Processor
3	M.OPEN Processor
4	M.RTRN/M.RTNA Processor
5-8	Reserved
9	Diagnostics
A-D	Event Trace
E	Available for customer use
F	'CALM' Replacement SVC

External References

None

3.8.9 M.CALL SVC Processor (H.SVC0)

Functional Description

This processor provides the means to enter any system module with a register push-down.

Entry Conditions

The requestor must be privileged.

Execution of the following code:

```
 M.CALL     mm,ee  (SVC type '0' instruction)
```

where:

mm	is the module number being called (bits 20-23 of SVC instruction)
ee	is the entry point number within the module to which control will be transferred (bits 24-31 of SVC instruction)

Exit Conditions

Return Sequence:

LPSD SVCO.T1	SVCO.T1 = Address of the entry point within the called module

Registers:

All registers remain intact

External References

Abort Cases:

SV01	Unprivileged task attempted use of M.CALL

3.8.10 Supervisor Call Trap Processor (H.SVC1)

Functional Description

This trap processor is entered whenever a SVC type '1' is executed for an I/O service, Monitor service, or dynamic DEBUG service. The processor determines the module and entry point to enter. A register push-down and PSD save is performed and control is transferred to the specified module.

Entry Conditions

Calling Sequence:

SVC type '1' instruction is executed.

Exit Conditions

Return Sequence:

LPSD SVCO.T1 SVCO.T1 = Address of MOD., E.P.

Registers:

Same as upon detection of trap signal.

External References

System Macros:

None

Abort Cases:

SV02	Invalid SVC number
SV03	Attempted use of privileged only service by unprivileged task

Output Messages:

None

3.8.11 M.OPEN SVC Processor (H.SVC3)

Functional Description

This routine removes the task context switch inhibit state set by the M.SHUT procedure.

Entry Conditions

Execution of the following code:

M.OPEN (SVC type '3' instruction)

Exit Conditions

Return Sequence:

```
BU          S.EXEC20
```

External References

None

3.8.12 M.RTRN/M.RTNA SVC Processor (H.SVC4)

Functional Desription

This processor provides the control transfer mechanism for system modules to return to the calling program. A register pop-up is performed with specified registers preserved returning control to the location specified or the location following the last CALM or M.CALL.

Entry Conditions

The requester must be privileged.

Execution of the following code:

```
SVC type '4' instruction
WAIT
DATAB    X'rr'
DATAB    X'aa'
```

where:

rr Bits 0-7 indicate the registers to be preserved through the pop-up. Each register is preserved if its corresponding bit is set.

aa May contain a bit (0-7) set indicating the corresponding register containing the address to which to return.

Exit Conditions

Return Sequence:

```
BU          S.EXEC.20
```

External References

None

3.8.13 Invalid SVC Type Processor (H.SVCN)

Functional Description

Entry to this processor results in an abort of the currently executing task.

Entry conditions

SVC type 'N' (where N is 5, 6, 7, 8 or E)

Exit Conditions

Return Sequence:

BU	S.EXEC20

External References

Abort Cases:

SV04	Invalid SVC type

3.8.14 Machine Check Trap Processor (H.IP07)

Functional Description

A machine check trap is treated as a fatal system crash. Registers are loaded for display, and the M.KILL macro is invoked. Processing may be continued at the point of interrupt with the registers and PSD intact, but I/O in progress when the HALT was executed will be terminated.

Entry Conditions

Occurrence of a trap signal at priority level X'07'.

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1	=	PSD saved by the hardware when the trap occurred.
R2	=	Physical address of instruction being executed when the trap occurred.
R3	=	Instruction being executed when the trap occurred.
R4	=	CPU status word stored when the trap occurred.
R5	=	Abort code
R6	=	Address of register save block.
R7	=	Trap.

Exit Conditions

Return Sequence:

M.KILL

NOTE

Unless a user-supplied System Override Interrupt/Trap Processor is connected, processing may be continued by depressing the program STEP switch and then the RUN/HALT switch on the CPU Control Panel.

External References

System Macros:

M.EQUS, M.KILL

Abort Cases:

MC01 Machine check trap

Output Messages:

None

3.8.15 System Check Trap Processor (H.IP08)

Functional Description

A system check trap is treated as a fatal system crash. Registers are loaded for display, and the M.KILL macro is invoked. Processing may be continued at the point of interrupt with the registers and PSD intact, but I/O in progress when the HALT was executed will be terminated.

Entry Conditions

Occurrence of a trap signal at priority level X'08'.

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1 = PSD saved by the hardware when the trap occurred.
R2 = Physical address of instruction being executed when the trap occurred.
R3 = Instruction being executed when the trap occurred.
R4 = CPU status word stored when the trap occurred.
R5 = Abort code.
R6 = Address of register save block.
R7 = Trap.

Exit Conditions

Return Sequence:

M.KILL

NOTE

Unless a user-supplies System Override Interrupt/Trap Processor is connected, processing may be continued by depressing the program STEP switch and then the RUN/HALT switch on the CPU Control Panel.

External References

System Macros:

M.EQUS, M.KILL

Abort Cases:

SC01	System check trap occurred at an address located within the operating system.
SC02	System check trap occurred within the current task's space.
SC03	System check trap occurred at a time when there were no tasks currently being executed (C.PRNO equals zero).
SC04	System check trap occurred within another trap (C.GINT does not equal 1).

Output Messages:

None

3.8.16 Map Fault Trap Processor (H.IP09)

Functional Description

This processor results in the task currently in execution being aborted. A register push down and PSD save is made, the abort request is reported to the scheduler, and a standard exit is performed.

Entry Conditions

Calling Sequence:

Occurrence of trap signal at priority level X'09'.

Exit Conditions

Return Sequence:

```
BL        S.EXEC5A
```

Registers:

R2	Register save area address
R5	Abort code
R6,R7	PSD

External References

System Macros:

M.EQUS

Abort Cases:

MF01	Map fault has occurred. Rerun task.

Output Messages:

None

3.8.17 Address Specification Trap - CONCEPT/32 only (H.IP0C)

Functional Description

This processor generates a trap when an address specification error occurs on a CONCEPT/32. If other tasks are not active, a normal return is taken from the interrupt. If other tasks are active, the processor will halt the system and display an abort code.

Entry Conditions

Occurrence of a trap signal at priority level X'0C'.

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1	=	PSD saved by the hardware when the trap occurred.
R2	=	Physical address of instruction being executed when the trap occurred.
R3	=	Instruction being executed when the trap occurred.
R4	=	CPU status word stored when the trap occurred.
R5	=	Abort code AD03 or AD04.
R6	=	Address of register save block.
R7	=	Trap.
(or)	If current task is being aborted	
R5	=	Abort code AD02

Exit Conditions

Return Sequence:

M.KILL (Abort code AD03 or AD04)
Normal interrupt return (Abort code AD02)

External References

System Macros:

M.TRACE, M.KILL, M.EQUS, M.TBLS

Abort Cases:

AD02	Address specification occurred within the current trask
AD03	Trap occurred while no tasks were in active state
AD04	Trap occurred in another interrupt trap routine

Output Messages:

None

3.8.18 Block Mode Timeout Trap Processor (H.IP0E)

Functional Description

A block mode timeout trap is treated as a fatal system crash. Registers are loaded for display, and the M.KILL macro is invoked. Processing may be continued at the point of interrupt with the registers and PSD intact, but I/O in progress when the HALT was executed will be terminated.

Entry Conditions

Occurrence of a trap signal at priority level X'0E'.

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1 = PSD saved by the hardware when the trap occurred.
R2 = Physical address of instruction being executed when the trap occurred.
R3 = Instruction being executed when the trap occurred.
R4 = CPU status word stored when the trap occurred.

Exit Conditions

Return Sequence:

M.KILL (Crash Code BT01)

NOTE

Unless a user-supplied System Override Interrupt/Trap Processor is connected, processing may be continued by depressing the program STEP switch and then the RUN/HALT switch on the CPU Control Panel.

External References

System Macros:

M.EQUS,M.KILL

Abort Cases:

None

Output Messages:

None

Crash Codes:

BT01	Block mode timeout trap.

3.8.19 Arithmetic Exception Interrupt/Trap Processor (H.IP0F)

Functional Description

This processor performs a set bit in memory instruction (SBM) on the arithmetic exception bit of the user currently in execution. The arithmetic exception bit may then be tested by requesting the Arithmetic Exception Inquiry monitor service.

Entry Conditions

Calling Sequence:

Occurrence of an interrupt signal at priority level X'29', or trap signal at priority level X'0F'.

Exit Conditions

Return Sequence:

```
LPSD  IP0F.OLD   IP0F.OLD=PSD saved by interrupt
```

Returns to destination registers are handled as follows:

Single Precision Floating Point

Underflow	All zeros
Positive Overflow	7FFFFFFF
Negative Overflow	80000001

Double Precision Floating Point

Underflow	All zeros
Positive Overflow	7FFFFFFFFFFFFFFF
Negative Overflow	80000001

External References

System Macros:

None

Abort Cases:

None

Output Messages:

None

3.8.20 Cache Memory Parity Error Trap Processor - 32/87 only (H.IP10)

Functional Description

This processor generates a trap when cache memory parity errors occur on a 32/87. If other tasks are not active, a normal return is taken from the interrupt. If other tasks are active, the processor will halt the system and display an abort code.

Entry Conditions

Occurrence of a trap signal at priority level X'10'

NOTE

Entry into this routine results in registers being loaded for console display as follows:

R0,R1	=	PSD saved by the hardware when the trap occurred.
R2	=	Physical address of instruction being executed when the trap occurred.
R3	=	Instruction being executed when the trap occurred.
R4	=	CPU status word stored when the trap occurred.
R5	=	Abort code CP03 or CP04.
R6	=	Address of register save block.
R7	=	Trap.
(or)		If current task is being aborted:
R5	=	Abort code CP02

Exit Conditions

Return Sequence:

M.KILL (Abort code CP03 or CP04)
Normal interrupt return (Abort code CP02)

External References

System Macros:

M.TRACE, M.KILL, M.EQUS, M.TBLS

Abort Cases:

CP02	Cache parity error occurred in task body
CP03	Trap occurred while no tasks were in active state
CP04	Trap occurred in another interrupt trap routine

Output Messages:

None

3.8.21 Console Interrupt Processor (H.IP13)

Functional Description

The console interrupt processor is directly connected to the console interrupt and is thereby activated upon its occurrence. This processor has one primary function, that of recognizing a request for console services, via the console interrupt, and issuing a break request for the associated task.

Entry Conditions

The processor is entered directly upon the occurrence of the console interrupt (priority level 13). Only registers used by this processor are saved.

Exit Conditions

All registers are restored to their content at the time of interrupt. Exit is made to the system via the standard interrupt exit sequence (BL S.EXEC5).

External References

System Macros:

M.EQUS

Abort Cases:

None

Output Messages:

None

3.8.22 Call Monitor (CALM) Interrupt Processor (H.IP27)

Functional Description

Entry to this routine results in an immediate abort of the task currently in execution.

Entry Conditions

Occurrence of an interrupt signal at priority level X'27'.

Exit Conditions

Return Sequence:

```
BL      S.EXEC5
```

Registers:

R2	Register save area address
R6,R7	PSD

External References

System Macros:

M.EQUS

Abort Cases:

CM01	Physical end-of file encountered on write to the compressed source output file. CALM instruction was not located.

Output Messages:

None

3.8.23 Real-Time Clock Interrupt Processor (H.IPCL)

Functional Description

The Real-Time Clock Interrupt Processor is directly connected to the specified real-time clock interrupt and therefore is activated upon its occurrence. This processor performs two primary functions: (1) maintains the clock interrupt counter (C.INTC) which is used for time of day calculations and (2) processes any active DQE timers.

Entry Conditions

The processor is entered directly upon the occurrence of the real-time clock interrupt to which it is connected.

Exit Conditions

Return Sequence:

```
BL      S.EXEC5
```

Registers:

R2	Address of register save area
R6,R7	PSD

External References

System Macros:

M.EQUS

Abort Cases:

None

Output Messages:

None

4. SYSTEM TASK DESCRIPTIONS

4.1 Swap Scheduler Task (J.SWAPR)

The Swap Scheduler Task is a memory management task. Its purpose is to process entries in the Memory Request Queue (MRQ) and provide memory allocation scheduling as appropriate to service individual requests for memory, i.e., make memory available to a task when there is no memory available.

4.1.1 Structure

J.SWAPR is a resident, privileged task residing in low memory, and is mapped into the address space of every current task in the system. It has its own minimal Task Service Area (TSA) and Dispatch Queue Entry (DQE) and it executes at the priority of the highest priority task in the MRQ.

```
*************************************************************************
*                                                                       *
*        DISPATCH QUEUE ENTRY MACRO                                     *
*                                                                       *
*************************************************************************
DQE      DEFM      SF,SB,COP,BOP,IOP,US,ON,LMN,USHF,MSD,CQC,SWIF,TAD
%TAN     SET       0
%NUM     SET       %TAN+1
         GEN       32/%SF            STRING FOREW ARD
         GEN       32/%SB            STRING BACKWARD
         GEN       8/%COP,8/%BOP,8/%IOP,8/%US
         GEN       8/%NUM,24/%TAN    QUE ENT #,TASK ACT #
         IFP       %ON,%L1
         GEN       64/%ON            OWNERNAME
         GOTO      %L1.1
%L1      ANOP
         GEN       64/C'MPX 1.4 '
%L1.1    ANOP
         IFP       %LMN,%L2
         GEN       64/%LMN           LOAD MODULE NAME
         GOTO      %L2.1
%L2      ANOP
         GEN       64/C'SYSTEM  '
%L2.1    ANOP
         REZ       3W
         GEN       32/%USHF
         GEN       32/%MSD
         DATAB     0,0,0,0
         DATAB     0,0,0,0
         DATAW     0
         GEN       3/1,29/%CQC
         REZ       7W
         DATAW     $,$-1W,0
         DATAW     $,$-1W
         GEN       16/0
         GEN       8/%SWIF,8/0
         DATAW     $,$-1W,0
         DATAW     $,$-1W,0
         DATAW     0
         GEN       8/0,24/%TAD
         DATAW     0,0,0,0
         ENDM
```

The Swapper's TSA contains a MOS memory initializer. This code is in the TSA, instead of the main task body, to save space and make the Swapper smaller. This can be accomplished because the MOS initializer is executed only at boot time and the code itself does not contain references to system services (SVC's or M.CALL's). This code is overwritten by the Swapper's main body initialization when building is completed.

```
*************************************************************
*                                                           *
*         TSA PROTOTYPE                                     *
*                                                           *
*************************************************************
J.SWAPR  EQU       $
TSA      REZ       TSA.SIZE         PSEUDO HAT ADDRESS
         ORG       TSA
         DATAW     1
         DATAW     $GINIT
         RES       6W
         GEN       1/1,7/0,24/SWAPPER
         IFT       C.3227,SWP.77
         GEN       1/1,15/0,1/1,12/1,3/0
         GOTO      SWP.CONT
SWP.77   ANOP
         GEN       1/1,15/0,1/1,13/1,2/0
SWP.CONT ANOP
         ORG       TSA+32W
         RES       8W
         GEN       1/1,7/0,24/SWAPPER
         IFT       C.3227,SWP.X77
         GEN       1/1,15/0,1/1,12/1,3/0
         GOTO      SWP.XXX1
SWP.X77  ANOP
         GEN       1/1,15/0,1/1,13/1,2/0
SWP.XXX1 ANOP

*.........................................................
*                                                        .
*                  INITIALIZE MOS MEMORY                 .
*                                                        .
*.........................................................
                              .
                              .
                              .

         ORG       TSA+T.REGP
*.........................................................
*                                                        .
*        END OF MOS INITIALIZER                          .
*.........................................................
```

```
          DATAW     TSA+32W             RUN AT ONE LEVEL OF PUSHDOWN
          ORG       TSA+T.PRNO
TSADQ     DATAW     0
          ORG       TSA+T.BBUFA
          DATAW     BBUFA
          ORG       TSA+T.FATA
          DATAW     FATA
          ORG       TSA+T.FPTA
          DATAW     FPTA
          ORG       TSA+T.BIAS
          DATAW     BMIDL
          ORG       TSA+T.TEND
          DATAW     BMIDL
          ORG       TSA+T.END
          DATAW     SW.END
          ORG       TSA+T.TRAD
          DATAW     SWAPPER
          ORG       TSA+T.BIT1
          DATAB     0,0,1,3
          ORG       TSA+T.DSOR
          DATAB     0,0,10,0
          ORG       TSA+T.MPUR
          DATAB     10,0,10,0
*         ORG       TSA+TSA.SIZE        FATA, FPTA, ETC MUST START AFTER
*                                       FIXED SIZE OF TSA.
*         FOR THE SWAPPER THIS IS NOT DONE SO THAT WE CAN SAVE SPACE
*         IN THE O/S.  THIS IS ALLOWED BECAUSE THE SWAPPER NEVER GOES
*         THROUGH THE TASK EXIT SEQUENCE.
*
*         FILE ASSIGNMENT TABLE AREA         (16 WORD / FAT)
*
FATA      DATAW     0,X'08000000',0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DATAW     0,X'08000000',0,0,0,0,0,0,0,0,0,0,0,0,0,0
          DATAW     0,X'08000000',0,0,0,0,0,0,0,0,0,0,0,0,0,0
*
*         FILE POINTER TABLE AREA            (3 WORDS / FPT)
*
FPTA      DATAW     0,X'08000000',0
          DATAW     0,X'08000000',0
          DATAW     0,X'08000000',0
*
*         BLOCKING BUFFER AREA               (  192 WORDS )
*
          BOUND     10
BBUFA     DATAW     X'08000000'
          RES       191W
```

The Swapper is set up to run by SYSGEN through its own SYSGEN initialization entry point. The initialization entry links the Swapper into the C.SQRT state where it is discovered by S.EXEC,20 after the first clock interrupt. The Swapper is then dispatched according to its priority (initially two) and suspended until resumed by H.EXEC,9 in response to a memory scheduler event.

```
**************************************************************
*
*         SYSGEN INITIALIZATION ENTRY
*
**************************************************************

          IFT       C.MEMO,MEMO.9                                   (MEMO)
H.SOUT.   EQU       $                DUMMY DEF FOR SYST. DEBUGGER (MEMO)
MEMO.9    ANOP                                                      (MEMO)
SGINIT    EQU       $
          M.EIR                      INIT EP MACRO
          LW        R1,C.DQUE        FETCH DISPATCH Q ADDRESS
          STW       R1,TSADQ         PLACE DQ ENTRY ADDR IN TSA
          LA        R7,C.SQRT         FETCH READY-TO-RUN QUEUE POINTER
          STW       R7,DQE1           STORE FOWARD POINTER
          STW       R1,C.SQRT        PUT J.SWAPR ON READY-TO-RUN QUEUE
          STW       R1,C.SQRT+1W
          LW        R7,C.SQRT+2W
          ADMW      R7,=X'00010000'  INCR NBR ON QUEUE
          STW       R7,C.SQRT+2W
          LA        R2,DQE1
          LI        R4,DQE.SIZE
          TRN       R4,R4            NEGATE SIZE OF DQE
MOVEDQE   LW        R7,0W,R2         MOVE "J.SWAPR" DQE ENTRY
          STW       R7,0W,R1
          ADI       R2,1W
          ADI       R1,1W
          BIW       R4,MOVEDQE
          M.XIR     TSA


DQE1      DQE       0,C.SQRT,2,2,2,3,,C'J.SWAPR',X
                    X'50000200',0,31250,X'80',J.SWAPR


SW.END    END
```

The Swapper's main initialization area is different from the MOS initializer. It cannot be placed inside the TSA because it includes SVC calls, and at this point, the Swapper is no longer running blocked (J.INIT among others are now active). The basic function of the main initialization area is to start up J.INIT, determine the TCW type of the swap device, and allocate an E-type memory buffer if needed. At this time, the Swapper also calculates the location within itself where inswap and outswapped candidates will be mapped. This location is saved as the User's TSA (UTSA) location.

Any MPX configuration containing an E-class swap device and a H-type or S-type memory are required to also have one dedicated E-type map block allocated to the Swapper. This map block is used exclusively for double buffering of H-type or S-type memory transfers to an E-class swap device (E-class disc).

Inswap and outswap are serial processes; they are always completed before the MRQ is reexamined. When sufficient memory is available to load an outswapped task into memory off the swap device, the inswap process is initiated. Dynamic memory requests are similar to inswap requests except there is not an associated disc file to read. Tasks linked to the MRQ may be queued for both expansion and inswap. There must be sufficient memory available to satisfy both requests before the inswap process is started.

4.1.2 Entry Conditions

When a memory scheduler event is reported by H.EXEC,9, the Swapper is linked to the ready-to-run state queue and resumed. The Swapper is concerned with five (5) types of memory scheduler events.

4.1.2.1 Dynamic Expansion of Address Space

A memory expansion request will occur whenever there is sufficient memory to satisfy a dynamic memory request on behalf of a task. The task is linked to the MRQ with the number of map blocks needed, the type of memory needed, the starting address of where to load new memory, and the expansion request code. The Swapper is then resumed.

4.1.2.2 Deallocation of Memory

When a task deallocates some or all of its allocated memory, and there are other tasks linked on the MRQ, the Swapper is resumed to satisfy the memory requests on behalf of those on the MRQ.

4.1.2.3 Request for Inswap (Memory Roll-In)

When a currently outswapped task becomes a candidate for execution (I/O completion occurs, device allocation occurs, etc.), the EXEC resumes the Swapper (the task is found on the MRQ and its inswap request is processed).

4.1.2.4 Change in Task Status

When a task which has been previously ineligible for swapping becomes eligible (completion of an unbuffered I/O operation, release of a lock-in-memory flag, expiration of a stage one time quantum, etc.), the Swapper is resumed.

4.1.2.5 Expansion Request for Inclusion of Shared Memory

When there is insufficient memory to satisfy the inclusion of a dynamic shared memory partition on the behalf of a task, the Swapper (having already been resumed prior to receiving the request) links the task to the MRQ in order to process the request.

4.1.3 Exit Conditions

When the Swapper finds the MRQ empty or when there are no outstanding requests to process, it suspends itself by unlinking from the ready-to-run queue and relinking to the wait-for-memory-event queue. This is accomplished by using entry point 8 in the EXEC.

4.1.4 Selection of Outswap Candidates

The Swapper initially attempts to allocate the memory requirement for the highest priority task on the MRQ. If there is insufficient memory 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 requestor is completely outswapped. Upon completion of the outswap process, the Swapper reexamines the memory request queue and continues to process memory requests.

Outswap candidates are chosen first from wait states then from run states. The order of each is shown on page 4-8.

```
NWS       DATAW    15        NUMBER OF WAIT STATES
WAITSTAT  DATAW    C.HOLD    WAITSTATE Q POINTERS IN OUTSWAP ORDER
          DATAW    C.SUSP
          DATAW    C.RUNW
          DATAW    C.SWDV
          DATAW    C.SWDC
          DATAW    C.SWTI
          DATAW    C.SWLO
          DATAW    C.SWIO
          DATAW    C.ANYW
          DATAW    C.SWSR
          DATAW    C.SWSM
          DATAW    C.MRQ
          DATAW    C.SWMP
          DATAW    C.SWGQ
          DATAW    C.SWFI


NRS       DATAW    13        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.CURR
          DATAW    C.RIPU
          DATAW    C.SQRT
```

Once an outswap candidate is chosen by the Swapper, two locations in its DQE are examined: DQE.SWIF and DQE.SOPO. Both locations are byte variables, each containing several different swapping limitations.

DQE.SWIF (swap inhibit flags) is checked first. If any bit in this flag byte is set, the task is unswappable and the Swapper searches for a new outswap candidate. If DQE.SWIF equals zero, DQE.SOPO (swap on priority only) is examined. If any bit in DQE.SOPO is set, the priority of the inswap requester is compared to the outswap task priority. If the inswap requester does not have a higher priority, the Swapper searches again for another outswap candidate. If DQE.SOPO equals zero, the outswap candidate is outswapped.

4.1.5 Outswap Process (Memory Roll-Out)

Outswap is a demand process which is initiated in response to an inswap or memory expansion request. The TSA of the outswap candidate is mapped into the Swapper. Protocol of outswap requires two (2) complete passes of the user's TSA tables (MEML and MIDL).

On the first pass, all swappable shared regions (if any) are built into a work area inside the Swapper in a logically contiguous format. A swap file is allocated and the outswap I/O process is performed. If there are no shared regions, I/O is not performed in this pass.

On the second pass, only non-shared maps are logically built into the Swapper's work area in their original TSA format. A second swap file is allocated and the outswap I/O process is performed.

At the end of each pass, memory is returned to the free list. When both passes of outswap are complete, the MRQ is reexamined to find the highest priority candidate to receive the memory from the free list.

4.1.6 Inswap Process (Memory Roll-In)

When sufficient memory is available, the Swapper allocates it to the highest priority task on the MRQ. If the request is for inswap, the Swapper allocates and opens the swap file associated to the task and reads the swapped image into the newly allocated memory. This is done in three (3) passes.

On the first pass, the entire TSA and DSECT are read off the file.

On the second pass, the MEML is scanned for shared regions. If any shared regions are found and they are not outswapped, the TSA is updated from the corresponding Shared Memory Table (SMT) entry.

On the third pass, the shared regions are interrogated to find out if they are outswapped. If they are outswapped, memory must be allocated and a second inswap performed before they can be loaded into the task's TSA tables. If sufficient memory cannot be allocated, the task is put on the MRQ asking for dynamic expansion to include a shared region.

4.1.7 Swap I/O Process

A single write request is given to H.IOCS for F-class and D-class swap devices (disc). Command and data chains are built in the handler to perform the specified transfer.

Multiple 4KW write requests are given to H.IOCS for E-class swap devices from the Swapper's internal buffer until all swappable map blocks have been written. The Swapper alternates between wait and no-wait requests and performs its own double-buffering of H-type and S-type map blocks through its dedicated E-class map block.

4.1.8 Other Considerations

The Swapper contains a MOS memory initializer which clears all memory parity errors at boot time. The initializer is located in the TSA area of the Swapper and is over-written after start-up is complete.

4.2 Terminal Service Manager Task (J.TSM)

4.2.1 Functional Description

The Terminal Service Manager Task (J.TSM) is a non-resident privileged system task whose function is to activate tasks which run in the on-line (interactive) environment. Its primary responsibilities include processing of initial log-on requests, command processing, parameter validation, and task activation. It also processes inter-terminal messages. J.TSM is scheduled to run whenever the user depresses the 'wake-up' character on his terminal, or whenever an on-line task exits or aborts.

4.2.2 Initial Activation

J.TSM is activated by J.INIT following a system restart. During this phase, J.TSM allocates one context area for each potential TSM terminal. (Each UDT belonging to TSM can be identified by bit 7 in UDT flags.) If necessary, J.TSM will expand its memory to contain the necessary context areas. Each context area is initialized with some fixed parameters. The DQE address of J.TSM is stored in C.TSMDQA. The number of TSM terminals is maintained in C.TSMTOT. C.TSMCNT equals the number of terminals in command mode.

4.2.3 Terminal Log-on

A terminal can be logged on by depressing the wake up character (specified in log-on file, M.LOGON). This generates a ring interrupt which is processed by EP3 of the ALIM or ADS handler. The handler resumes the task indicated in C.TSMDQA. When J.TSM is resumed, it will scan the terminal UDT's to determine which terminal(s) to allocate. The context areas are searched until the UDTI is matched. Then the terminal is allocated to J.TSM. If the terminal is already allocated, the ring is ignored. Otherwise, the terminal is opened and a greeting message is issued. Then the user is solicited for a valid ownername.

4.2.4 Command Processing

All write operations are performed in no-wait I/O with end action addresses. All read operations are performed in no-wait I/O with end-action addresses pointing to the command processing routine. There are two end-action routines. The first verifies that the ownername just entered is valid. This involves checking all the context areas and the M.KEY file. The second end action routine is the command processor. Generally, this involves converting the input into the form used by the M.PTSK service.

Upon entry into either end action receiver, register 1 will point to the current context area (equal to the FCB address). A test is made to determine if any messages need to be displayed. If so, they are written to the terminal. Two messages may be queued for each terminal. If the terminal is not logged on when the message is sent, it will be saved until the terminal is logged on. Messages sent to other on-line tasks are written by those tasks as a pre-emptive system service, not managed by J.TSM.

When the end-action routine processes a RUN command, the parameter block is passed to the M.PTSK service to activate an on-line task. Assignment to the terminal is transmitted in the parameter block by an assignment to UT. If the activation is unsuccessful, J.TSM will report the error status before soliciting the next command from the user. Following a successful activation, the context area is made inactive, and the end-action routine is exited. All previously entered commands remain effective until the CLEAR command is entered.

When the EXIT command is entered, a EOF is written to the terminal (enabling ring for ADS), the terminal is deallocated and the context area is cleared.

4.2.5 Resumption of Command Processing

When an on-line task exits or aborts, H.TSM,3 is called to inform J.TSM. The terminal is deallocated for the exiting task, and a break is sent to J.TSM. When resumed, J.TSM will reallocate the terminal. The context area is located by matching the UDTI, and the terminal is reallocated and opened. Then the user is solicited for the next command. Optionally, the previous assignments may remain in effect. After the user is prompted for input, the message exit service is called, and further processing is performed at the end action level.

4.2.6 Messages

There are two options for sending messages to terminals. The first form is to specify a single ownername. If the ownername is not in use, the message is discarded.

The second option is to send the message to all terminals. For all terminals the message is copied into the first or second mailbox. These messages are printed when the terminal becomes free, generally after a read is complete.

5. SYSTEM GENERATION TASK DESCRIPTION

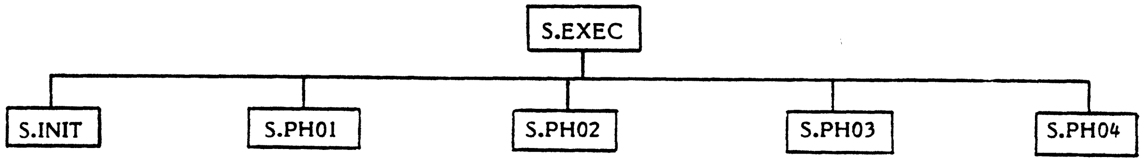
5.1 Task Structure and Functional Organization

The System Generation Task, SYSGEN, is a resident, privileged MPX-32 system task that operates within the framework of a standard MPX-32 system and can be executed either in batch or interactively. It consists of an executive segment and five overlays. Section 5.1.1 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. Section 5.1.2 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 absoulte format. This file may be subsequently restarted or utilized on a System Distribution Tape (SDT).

System generation via the SYSGEN utility is fully described in the MPX-32 Reference Manual Volume III. This chapter provides a functional description only.

5.1.1 SYSGEN Overlay Structure and Functions



S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
Opens statically allocated files: DIR,OBJ, and LO. Outputs title. Sets up map info and variables for remap of target system. Initializes default values in target communication region.	Reads directives from DIR. Writes directives to LO. Does initial pro-cesssing of directives.	Initializes tables (memory, ITLB, patch, job, timer, RMT, FLT, and activation). Constructs dispatch queue and DQE address tables and links them. Builds DTT, CDT, CHT and UDT tables Allocates and links the disc allocation table (DAT). Builds Checksum Table. Processes interrupt table entries and builds load module table entries. Builds target system scratch pad image. Obtains space for the SVC table, the GPMC table, and the module address table. Builds miscellaneous system parameters (MPL, MIDL, etc.)	Positions object file and allocates temporary disc space for loading process. Initializes loader variables and sets up pointers to the interrupt table. Scans object input file (OBJ) for match on program name in module record file. When match found, copies to temporary disc 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. Finalizes memory pool.	Builds memory tables to reflect target system. Constructs partition in shared memory table. Writes MPX image to file.

5.1.2 Directive Processing Breakdown

//HARDWARE	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/PARAMETERS					
MACHINE=type		Sets C.MGRAN, C.MACH, C.HIMAP and C.ADMASK			
IPU		Sets C.IPU in C.CONF			
MEMONLY		Sets C.MEMNLY			
/MEMORY					
SIZE=nn, TYPE=c, CLASS=x		Builds memory table prototype in scratch space	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 DEVICE=aa, DISC=devcode, SHR, DTC=tt, LINSIZ=x, PAGE=y, SPOOL=code, HANDLER=name, PHYSA=ccaa,OFF IOQ=mode		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 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 disc allocation table (DAT). Sets C.DAMAPT. Sets C.SMDD, C.SMDD + 1W, C.NTBA and C.SMDUDT. Builds Scratch pad entries. Builds Checksum Table. Sets C.DAMCST.	Loads handler's object and initializes.	String CDT's.

	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
/TRAPS					
PROGRAM=(name1,...name7)		Builds internal interrupt table in scratch space.	Builds load table with interrupt table. Builds scratch pad entries.	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	Verifies C.SIDV		
LOD=devmnc,IBP		Sets C.LODC and C.SIBP	Verifies C.LODC		
POD=devmnc		Sets C.PODC	Verifies C.PODC		
SWP=devmnc		Sets S.SUDT	Sets C.SUDT		
//SOFTWARE					
/PARAMETERS					
DISP=entries	Defaults C.NQUE to 10	Sets C.NQUE	Constructs dispatch queue and DQE address table and links them. Sets C.ADAT and C.DQUE		Sets C.SWAP.
POOL=words	Defaults C.POOL to 1000	Sets C.POOL		Builds memory pool. Sets C.SBUF.	
NTIM=number	Defaults C.NTIM to 60	Sets C.NTIM			

	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
MTIM=number	Defaults C.MTIM to 60	Sets C.MTIM			
ITIM= milliseconds	Defaults C.ITRS to 384	Sets C.ITRS	Used to re-compute C.IDLC, C.TDQ1, C.TDQ2, and C.TDQ3.		
ITLB=intlev		Builds entry in internal interrupt table for H.ICP. Sets C.NITI	Initializes indirectly connected interrupt table. Sets C.ITLB. Builds scratch pad entries.	H.ICP loaded from object file and initialized.	
PASSWORD= password		Sets C.PSWRD		Uses C.PSWRD for symbol table file.	Uses C.PSWRD for target system file.
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.	
TQFULL=time	Defaults C.IDCL to 26042, C.TDQ3 to 1200, and and C.TDQ2 to 800.	Sets C.TDQ3	Recomputes C.TDQ3, C.TDQ2 and C.IDLC.		
TQMIN=time	Defaults C.IDCL to 26042, C.TDQ1 to 400, and C.TDQ2 to 800.	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			

	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
PATCH=number		Sets C.PATCH	Sets C.MPAA, C.MPAC, C.MPAH, and zeros patch area.		
MODE=code		Sets C.SCBT, C.SIBP, C.SUFA and C.SIMM as specified.			
SVC=num	Defaults· C.SVTN to X '7F'.	Sets C.SVTN	Sets C.SVTA and zeros SVC table.		
RMTSIZE=num	Defaults C.RMTL to 32, and C.RMTM to 64.	Sets C.RMTM	Sets C.RMTA and zeros resource-mark table.		
FLTSIZE=num	Defaults C.FLTM to 200 and zeros C.FLTC.	Sets C.FLTM	Sets C.FLTA and zeros file lock table.		
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			
/MODULES					
MODULE=(name, module, entpoints)	Defaults C.MODN to 8.	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 initializes.	

/OVERRIDE	S.INIT	S.PH01	S.PH02	S.PH03	S.PH04
SYSMOD=name1, REPMOD=name2		Replaces module with given name in the system module table defined in SYSGEN.			
/PARTITIONS					
NAME=name, SIZE=np, STRTPG=sp, MAP=pm		Builds internal partition table in scratch space.			Uses partition table to initialize shared memory table and allocate more memory to the system.
/TABLES					
JOBS=number	Defaults C.JOBN to 1	Sets C.JOBN	Zeros job table. Sets C.JOBA		
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.		
/FILES					
SMD=devmc, ENTRIES=ents	Defaults C.SMDS to 3001 and C.SMDD + 1W to 126	Sets S.SMD, S.SPS, C.SMDS and C.SMDD + 1W	Create scratch pad entry for SPS. Sets C.SMDUDT, C.SMDD and C.SMDD + 1W		
SYCSIZE=blocks	Defaults C.SYCS to 100	Sets C.SYCS			
SGOSIZE=blocks	Defaults C.SGOS to 100	Sets C.SGOS			

5.2 SYSGEN Library (SG.LIB)

SYSGEN is cataloged with a library assignment to SG.LIB. SG.LIB contains the Device Type Table definition, the Device ID Table definition, and a special scanner used to parse SYSGEN directives.

5.2.1 DID and DTT Definitions

The Device Identification Table and the Device Type Table definitions in SG.LIB are defined using the Macro Assembler directive FORM. SYSGEN fills in the Device Type Table with information from the Controller Definition Table (CDT), and vice-versa. A diagram of the DTT is shown in Section 5.2.1.1.

SYSGEN uses the Device Identification Table to build Unit Definition Table (UDT) entries for discs. Information for all types of discs is contained in the DID table. See Section 5.2.1.2.

5.2.1.1 Device Type Table

```
*************************************************************
*                   DEVICE TYPE TABLE (DTT)                 *
*************************************************************
*
         BOUND      10
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'0D',0 DUMMY CT
    DTT  X'01',A(00),X'00',X'41',C'DC',16384,X'00',0 ANY DISC
    DTT  X'02',A(00),X'00',X'40',C'DM',16384,X'01',0 MOVING HEAD DISC
    DTT  X'03',A(00),X'00',X'40',C'DF',16384,X'01',0 FIXED HEAD DISC
    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'06',A(00),X'00',X'60',C'M7', 8192,X'04',0 7-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'09',A(00),X'00',X'00',C'CP', 0120,X'07',0 CARD PUNCH
    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'40',C'FL',16384,X'00',0 FLOPPY DISC
    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,X'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
```

5.2.1.2 Device ID Table

```
************************************************************************
*                      DEVICE ID TABLE                                 *
************************************************************************
*
         BOUND      1W
DID.TBL  EQU        $
*
*DEVICE ID NAME.................................................
*TOTAL ALLOC. UNITS...................................         :
*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 'E' DISC DEVICES
         DID    C'DE01', 1, 1, 23, 192, 256,  184,  5888, C'FE004    '
         DID    C'DE02', 2, 1, 16,  96,   2,  200, 12800, C'CE010    '
         DID    C'DE04', 1, 2, 23, 192,   5,  719, 23000, C'ME040    '
         DID    C'DE05', 1, 2, 23, 192,   5, 1438, 46000, C'ME080    '
         DID    C'DE06', 1, 4, 23, 192,  19, 2732, 87400, C'ME300    '
         DID    C'DE07', 1, 1, 23, 192,   4,  184,  5888, C'FE005    '
*        CLASS 'F' EXTENDED I/O DISC DEVICES
         DID    C'DF01', 3, 3, 26,  64,   2,     ,  1334, C'FL001    '
         DID    C'DF02', 1, 2, 20, 192,   5,  625, 20000, C'MH040    '
         DID    C'DF03', 1, 2, 20, 192,   5, 1250, 40000, C'MH080    '
         DID    C'DF04', 1, 4, 20, 192,  19, 2375, 76000, C'MH300    '
         DID    C'DF05', 1, 1, 20, 192,   4,  184,  5120, C'FH005    '
         DID    C'DF06', 1, 2, 20, 192,   1,  250,  8000, C'C0032    '
```

5.2.2 SYSGEN Scanner

Functional Description

The SYSGEN scanner parses SYSGEN directives by utilizing linked information tables built by calling system macros. The sytem macros SECTION, SUBSECT, DIRTV, KEYWD, and PARAM set up tables as shown in Section 5.2.2.1. 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 3, Chapter 7.)

The action addresses in Section 5.2.2.1 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 setup 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, i.e., section definition has null sub-section link)

Registers:

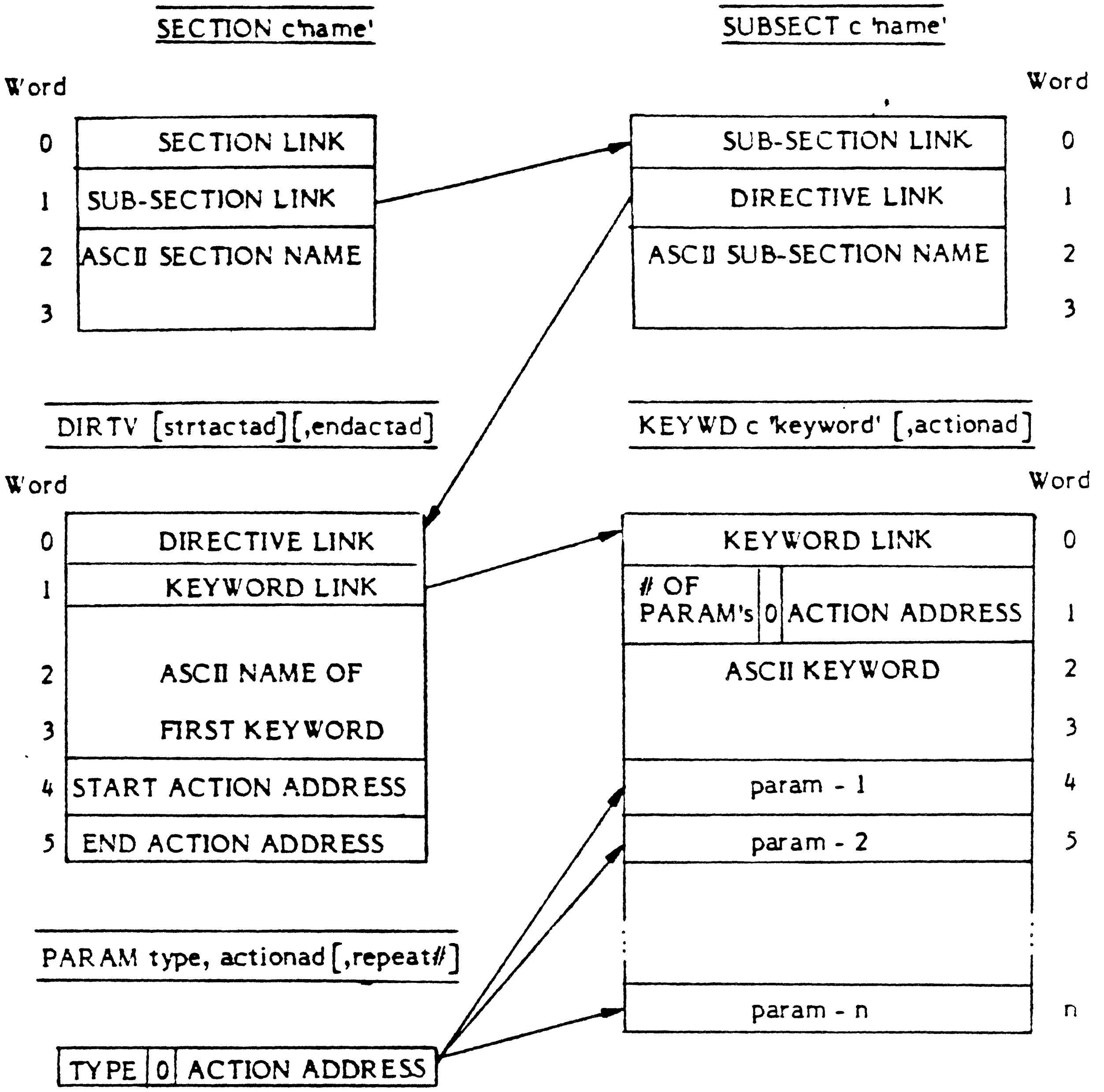
R1 Error message TCW if error detected.
R2 Current directive pointer.
R3 Negative length of remaining directive.

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	=	1st four bytes of string
	R7	=	2nd four bytes of string
	(or)		
	R7	=	Converted decimal number
	(or)		
	R7	=	Converted hexadecimal number
Outputs:	CC1	=	1 if error detected by action routine.

Example: See Section 5.2.2.2.

5.2.2.1 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

5.2.2.2 SYSGEN Scanner Example

```
        PROGRAM  EXAMPLE
        M.EQUS
        SDINIT                  TYPE TABLE EQUATES
START   M.READ   DIR.FCB        READ DIRECTIVE
        LA       R1,SDLIST      FETCH ADDR OF STMT DEF LIST
        LA       R2,DIR.BUF     ADDR OF DIR STMT BUFFER
        SBR      R2,12          SET 'F' BIT FOR SCANNER
        LI       R3,-72         # BYTES TO SCAN
        BL       SCANNER        SCAN DIRECTIVE
        BCT      2,EXIT         TERMINATION?
        BCF      1,START        ERROR? NO, GET NEXT STMT
        BU       ERROR          OUTPUT ERR MSG
************************************************************
*       STATEMENT DEFINITION LIST                          *
************************************************************
SDLIST  EQU      $
************************************************************
*       //HARDWARE                                         *
************************************************************
        SECTION  C'HARDWARE'
************************************************************
*       /SYSDEVS                                           *
************************************************************
        SUBSECT  C'SYSDEVS'
************************************************************
*       LOD=<DEVMNC>,IBP,DIR=<NBLKS>                       *
************************************************************
        DIRTV    A.LODS,A.LODX
        KEYWD    C'LOD'
        PARAM    STRING,A.LOD1
        KEYWD    C'IBP',A.LOD2
        KEYWD    C'DIR'
        PARAM    DNUMB,A.LOD3
*
A.LODS  EQU      $              START ACTION ROUTINE
        STW      RO,SCN.RTRN    SAVE RETURN ADDRESS
        SBM      0,S.LOD        INDICATE LOD DEFINED
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LOD1  STW      0,SCN.RTRN     SAVE RETURN ADDRESS
        BL       CONDEV         SCAN DEVICE MNEMONIC
        BCT      1,A.LOD1.1     NO ERROR DEVICE FOUND
        SBM      1,SCN.RTRN     SET ERROR INDICATOR CC1
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LOD1.1 STD     6,C:LODC       SET DEFAULT SYSTEM PUNCHED DEVICE
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LOD2  STW      0,SCN.RTRN     SAVE RETURN ADDRESS
        SBM      C:SIBP,C:BIT   INHIBIT BANNER PAGE
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LOD3  STW      0,SCN.RTRN     SAVE RETURN ADDRESS
        TRR      7,7            NO. OF DIRECTORY BLOCKS
        BNZ      A.LOD3.1       MUST BE NON-ZERO
        SBM      1,SCN.RTRN     SET ERROR INDICATOR
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LOD3.1 STW     7,S.SOD        SAVE NO. OF DIRECTORY BLOCKS
        BU       *SCN.RTRN      RETURN TO SCANNER
A.LODX  STW      RO,SCN.RTRN    END ACTION ROUTINE
        BU       SCN.RTRN

************************************************************
*       POD=<DEVMNC>                                       *
************************************************************
        DIRTV
        KEYWD    C'POD'
        PARAM    STRING,A.POD1
*
*       E T C E T E R A
************************************************************
*       //END                                              *
************************************************************
        SECTION  C'END'
************************************************************
*       END OF STATEMENT DEFINITION LIST                   *
************************************************************
        KWEND
        END      START
```

5.3 Table Building

5.3.1 System Tables

SYSGEN's main function is building the tables used by a 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 5.3.1.1 provides a list of all the tables with which SYSGEN interfaces, and where information about them can be found.

5.3.1.1 Tables Referenced in SYSGEN

Name of Table	SYSGEN Interaction	Where Documented
Activation Table	Built by SYSGEN	Ref. Man., Vol. 3, Sec. 9.1
Channel Definition Table - CHT	Partially built by SYSGEN	Tech. Man., Sec. 2.4.11
Controller Definiton Table - CDT	Built by SYSGEN	Tech. Man., Sec. 2.4.5
Device Identification Table - DID	Used by SYSGEN	Tech. Man., Sec. 5.2.1
Device Type Table - DTT	Used and filled in by SYSGEN	Tech. Man., Sec. 5.2.1
Disc Allocation Map Checksum Table - DAMCST	Allocated and zeroed by SYSGEN	Tech. Man., Sec. 2.6.2
Disc Allocation Map Table	Built by SYSGEN	Tech. Man., Sec. 2.6.2
DQE Address Table - DAT	Built by SYSGEN	Tech. Man., Sec. 2.3.3
DQE Table	Allocated, zeroed & linked by SYSGEN	Tech. Man., Sec. 2.3.2
File Lock Table - FLT	Allocated and zeroed by SYSGEN	Ref Man., Vol. 1, Sec. 2.9.1.6
GPMC Jump Table	Allocated and zeroed by SYSGEN	See H.MUX0
Indirectly Connected Task Linkage Table - ITLT	Allocated and initialized by SYSGEN	Tech. Man., Sec. 1.12
Job Table	Allocated and zeroed by SYSGEN	Tech. Man., Sec. 2.7.4
Map Tables	MPL, MSD & MIDL built by SYSGEN	Tech. Man., Sec. 2.5.3
Memory Allocation Table	Built by SYSGEN	Tech. Man., Sec. 2.5.2
Memory Pool	Allocated and zeroed by SYSGEN	Tech. Man., Sec. 2.5.1
Module Table	Allocated and zeroed by SYSGEN	Tech. Man., Sec. 1.13.14
Patch Area	Allocated and zeroed by SYSGEN	Ref. Man., Vol. 3, Sec. 9
Resourcemark Table - RMT	Allocated and zeroed by SYSGEN	Ref. Man., Vol. 1, Sec. 2.9.1.5
Scratch Pad	Partially built by SYSGEN	32/70 Tech. Man., Sec. 3-77
Shared Memory Table - SMT	Allocated and initialized by SYSGEN	Tech. Man., Sec. 2.5.3
SVC Table	Allocated and zeroed by SYSGEN	Tech. Man., Sec. 1.13.20
Timer Table	Allocated and zeroed by SYSGEN	Ref. Man., Vol. 1, Sec. 8.2.33
Unit Definition Table - UDT	Built by SYSGEN	Tech. Man., Sec. 2.4.6

5.3.2 Internal Tables

As SYSGEN processes directive input, it collects some information in its own internal tables for utilization later in the task. It has three tables which are limited to internal use: the Partition Table, the Module Table and the Interrupt/Trap Table. See Section 5.3.2.1.

The Partition Table is built with information provided from the Partition directives in Phase One (S.PH01). It is later used to initialize the Share 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 is also used to build the load map table, and it is additionally used to create interrupt entries in the scratch pad in Phase Two.

5.3.2.1 SYSGEN Internal Tables

Partition Table

Word		
0	String forward address (PAR.LINK)	
1	Start logic Page # (PAR.SPG)	# pages (PAR.NPG)
2	Physical Map block # (PAR.PBN)	
3	Reserved	
4	Partition Name	
5	(PAR.NAME)	

Module Table

Word		
0	String forward address (MOD.LINK)	
1	Module number (MOD.NO.)	# of entry points (MOD.NEPT)
2	Module name (MOD.NAME)	
3		
4	Reserved	
5	Reserved	Module type (MOD.LTYP)

Interrupt/Trap Table

Word				
0	String forward address (INT.LINK)			
1	Interrupt type (INT.TYP)	Priority level (INT.L1)	Controller class (INT.CLS)	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 # (INT.CHAN)	Sub address (INT.SUBA)	Module type (INT.LTYP)

Notes

1. Bits in INT.TYP are assigned as follows.

1	-	If set, indirect (TYP.NDIR)
2	-	If set, direct (TYP.DIR)
3	-	If set, service interrupt (TYP.SI)
4	-	If set, GPMC service interrupt (TYP.GPMC)
5	-	If set, extended I/O service interrupt (TYP.XIO)
6-7	-	Reserved

2 Values for INT.CLS are as follows.

Value=0 - TLC line printer
Value=1 - TLC card reader
Value=2 - TLC teletype
Value=3 - RTOM interval timer
Value=D - TCW class with blank bits
Value=E - TCW class
Value=F - Extended I/O

5.4 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 disc 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 via 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.

5.5 Special Considerations

5.5.1 MAPTGT/MAPHOST Routines

SYSGEN initially obtains two 8K blocks of memory in which to build the target system and acquires additional 8K 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. Mapping back and forth prevents SYSGEN to be run with the Debugger.

5.5.2 Special Case Activation

SYSGEN is a two 8K map block task. In order to allow SYSGEN to build target systems as large as 7 map blocks on the 32/7x and as large as 28 map blocks on the CONCEPT/32, the system allocator special cases it and loads it at address X'38000', which is the 8th block on the 32/7x and the 29th map block on the CONCEPT/32. This gives SYSGEN a maximum of 7 map blocks of memory on the 32/7x and a maximum of 28 map blocks on the CONCEPT/32 in which to map the target system.

6. BATCH TASK DESCRIPTIONS

6.1 CATALOGER

6.1.1 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 lfc DIR and LIB and from the system subroutine library assigned to lfc 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 system permanent file created by the Cataloger; its file name is the program name supplied in the CATALOG directive.

Exit Conditions:

Normal Exit:

CALM X'55'

Abnormal Exits:

CALM X'57' Abort

Abort Cases:

CT01	Physical end-of-file encountered on permanent file containing System Subroutine Library (MPXLIB) or assigned user library. Contents of the library altered subsequent to creation.
CT02	Load module file specified with Catalog cannot be allocated.
CT03	Unrecoverable I/O error encountered on the DATAPOOL dictionary file assigned to DPD.
CT04	Listed output space is depleted and additional SLO space cannot be allocated.
CT05	Unrecoverable I/O error on file or device assigned to SBM for SYMTAB output.
CT06	An error occurred, conditional Batch directives are still processed.

6.1.2 Processing Regions

The Cataloger consists of four distinct processing regions each of which is identified by a letter as follows:

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 with which they are associated.

6.1.2.1 X Region

The X region is composed of subroutines relating to MPX provided services.

6.1.2.2 M Region

The M region is primarily composed of 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, but following the Cataloger program logic. The utilization of this area is depicted in 6.1.2.3.1.

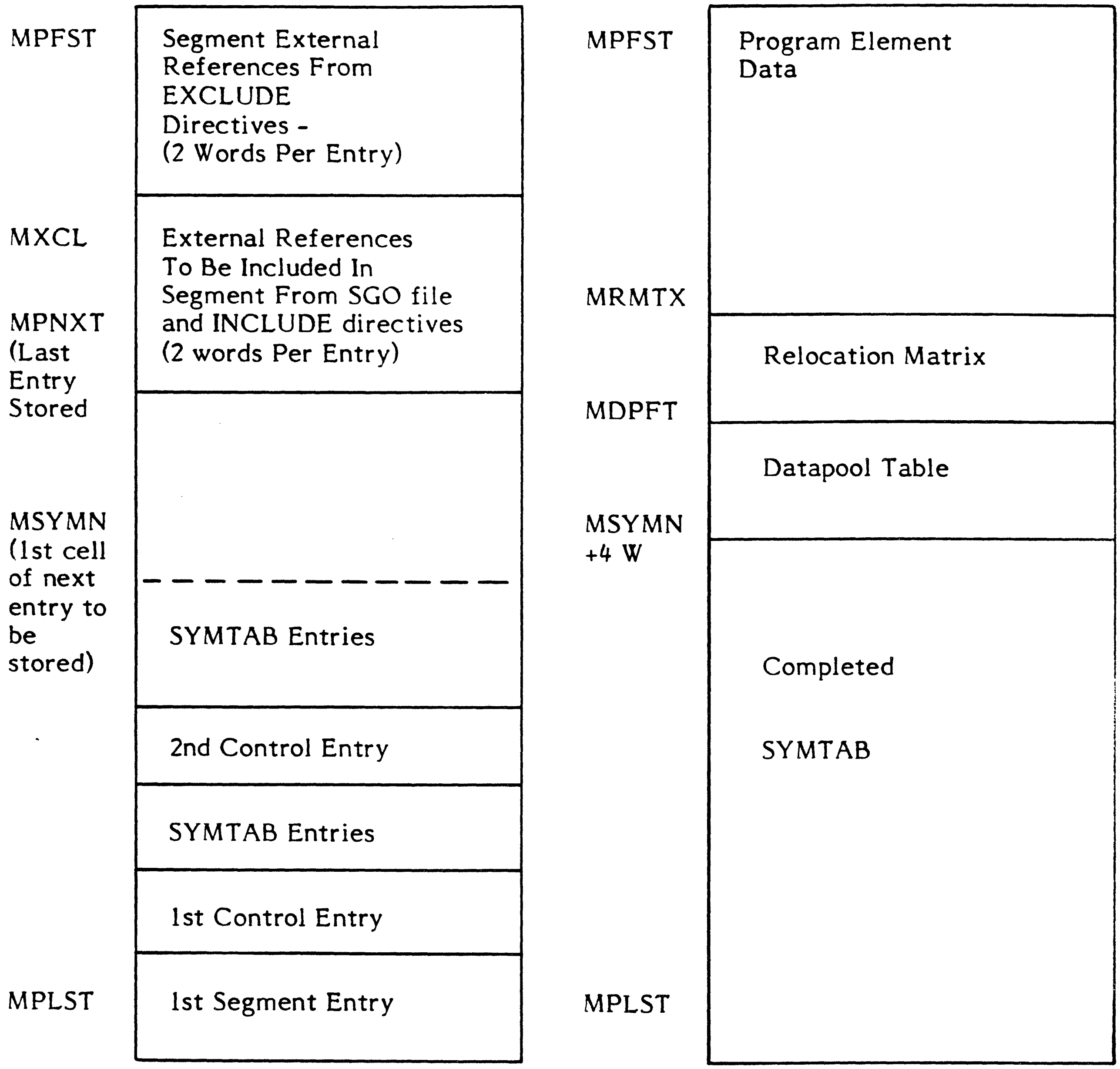
6.1.2.3 C Region

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 6.1.2.3.1. 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. It 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.

6.1.2.3.1 General Table Area


MPFST
Segment External References From EXCLUDE Directives - (2 Words Per Entry)
MXCL
External References To Be Included In Segment From SGO file and INCLUDE directives (2 words Per Entry)
MPNXT (Last Entry Stored
MSYMN (1st cell of next entry to be stored)
SYMTAB Entries
2nd Control Entry
SYMTAB Entries
1st Control Entry
MPLST
1st Segment Entry
MPFST
Program Element Data
MRMTX
Relocation Matrix
MDPFT
Datapool Table
MSYMN +4 W
Completed
SYMTAB
MPLST

Object Code Pass 1

Object Code Pass 2

6.1.2.3.2 SYMTAB Entries

<table>
<tr><th></th><th>0</th><th>6</th><th>8</th><th>16 31</th></tr>
<tr><td>Linkback Entries</td><td colspan="4">If the LINKBACK directive is used, the identities of specified overlay segments are saved in entries built toward low memory in 4-word blocks.</td></tr>
<tr><td>Segment (Module) Entry</td><td>Must be zero</td><td colspan="2">Overlay level</td><td>Sequence Number</td></tr>
<tr><td>0</td><td>ID
1 0 0 0 0 0</td><td>Flags</td><td>Option
Flags</td><td>Number of
Linkback Entries</td></tr>
<tr><td>1</td><td colspan="2">Overlay Level
(Main is zero)</td><td colspan="2">Sequence Number
(Main is zero)</td></tr>
<tr><td>2
3</td><td colspan="4">ASCII Segment Name, Left Justified</td></tr>
<tr><td>4</td><td colspan="2">Must be zero</td><td colspan="2">Transfer Address</td></tr>
<tr><td>5</td><td colspan="4">Origin of Segment TSA Relative*</td></tr>
<tr><td>6</td><td colspan="4">Number of Bytes in Module Common Delta</td></tr>
<tr><td>7</td><td colspan="4">Last (END) Address of Segment</td></tr>
</table>

*Bit zero of the word is set if the origin is the end of a specified segment; bit is set if the origin is in the end of the previous overlay level.

Flags	10	The CATALOG directives for the overlay was preceded by an LORIGIN or ORIGIN directive.
	01	The segment has a transfer address.
Option Flags	00000001	Suppress printing of module map
	00001000	Suppress output of load module to disc file
	00010000	Punch load module on cards
	00100000	Output Segment's SYMTAB

		0	6	8 31
Defined Entry Point	0	ID 0 1 0 0 0 0	Flags	
	1	Section Number		Module relative address
	2	ASCII Name, Left Justified		
	3			

		0	6	8 31
Common	0	ID 0 0 1 0 0 0*	Flags*	Size in Bytes (zero if allocated in another element)
	1	Section Number **		Module relative address
	2	ASCII Name, Left Justified		
	3			

* The ID is 001100 for the first COMMON entry for a given global COMMON block.

** Address (if allocated in another element, the address of the entry where allocated).

Flags:
- 10 Common block is program origined in this element
- 01 Common block is Cataloger allocated in this element

		0	6	8 31
Section Entry	0	ID 0 0 0 1 0 0	Flags	Size in bytes
	1	Section Number		Section Origin
	2	ASCII Name, Left Justified		
	3			

Program Name

<table>
<tr><th></th><th>0</th><th>6</th><th>8</th><th>16</th><th>31</th></tr>
<tr><td>0</td><td>ID
0 0 0 0 1 0</td><td>Flags</td><td>Sub.Lib
Log. Rec.
No.</td><td colspan="2">Sub Lib. Block No.
Containing 1st Record</td></tr>
<tr><td>1</td><td colspan="3">Section Number</td><td colspan="2">Module transfer address if
contained in this element (MTADD)</td></tr>
<tr><td>2
3</td><td colspan="5">ASCII Name, Left Justified</td></tr>
</table>

Flags:

10 Word 1 of entry contains Module Transfer Address
01 Program is from System Subroutine Library

Control Entry

<table>
<tr><th></th><th>0</th><th>6</th><th>8</th><th>31</th></tr>
<tr><td>0</td><td>ID
0 0 0 0 0 1</td><td>Flags*</td><td colspan="2">Total SYMTAB Entries
for this element</td></tr>
<tr><td>1</td><td colspan="2">Max. Bound
Required</td><td colspan="2">Number of words (Byte Scaled)
Allocated this Element for
Common and Bounding</td></tr>
<tr><td>2</td><td colspan="4">Number of Words (Byte Scaled) of Object Code
in this Element</td></tr>
<tr><td>3</td><td colspan="4">Module Relative Address of this Element, i.e.,
the Address of its Common if any</td></tr>
</table>

*Bit 8 set if section code

Flags:

10 Program element is from System Subroutine Library
01 Element is last in segment including (System Subroutine Library routines)

6.1.2.4 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 disc 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.

6.1.3 Load Module Structure

A load module consists of one or more program elements in a form which requires only load origin biasing at load time. A program element is the unit of program organization bounded by the assembler directives "PROGRAM" and "END". 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.

Each load module occupies a permanent file whose name is the name by which the module is known to the system.

The load module consists of up to 6 major segments. They are: the preamble, the resource requirement summary, CSECT data, CSECT relocation matrix, DSECT data, and DSECT relocation matrix. Either the CSECT segments or the DSECT segments may be omitted if they are empty. The resource requirement summary block is always present even if it is empty.

Load Module Preamble Equates

Byte	Word				
0	0	PR.NAME - Load module name			
8	2	PR.USER - User name			
10	4	PR.UKEY - Userkey			
14	5	PR TRAN- Module transfer address			
18	6	PR.CNTL	PR.FLAG	PR.NRRS	PR.PRIOR
1C	7	PR.PAGLEC	PR.PAGED	PR.PGSIZ	PR.MEMS
20	8	PR.PAGEG	PR.FILE	PR.BUFR	RESERVED
24	9	PR.OPTN - Program option word			
28	10	PR ORGC - Starting byte address of code section			
2C	11	PR.COMC - Common delta			
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			
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	30	PR.DATE - Date load module was created			

Control Information

Load Module Name (PR.NAME)

The left-justified and blank filled eight-character ASCII name of the load module. This name must match the file name and the file must be a system file.

User Name (PR.USER)

The left-justified and blank filled eight-character ASCII user name associated with the load module. This field is zero if the USERNAME directive was not used.

User Key (PR.UKEY)

The compressed (halfword) ASCII user key associated with the load module. This field is zero if the USERNAME directive was not used.

Module Transfer Address (PR.TRAN)

The module relative address of where the module is to start execution when loaded. Bit 7=1 indicates the transfer address is absolute.

Control (PR.CNTL)

A flag byte that defines the following software characteristics:

Bit 0 set = Reserved
Bit 1 set = Reserved
Bit 2 set = Overlay segment (PR.OVER)
Bit 3 set = Privileged task (PR.PRIV)
Bits 4-7 = Reserved

Flags (PR.FLAG)

A flag byte that defines the following software characteristics:

Bit 0 set = absolute CSECT load addresses (PR.ABSC)
Bit 1 set = resident (PR.RES) (program cannot be swapped to back store when in execution)
Bit 2 set = shared (PR.SHR) (a single copy can be shared by two or more users)
Bit 3 set = absolute DSECT load addresses (PR.ABSD)
Bit 4 = reserved
Bit 5 set = multicopy (PR.MULTI)
Bits 6-7 = reserved

RRS CNT (PR.NRRS)

The number of entries in the Resource Requirement Summary Table.

Priority (PR.PRIOR)

The base execution priority of the cataloged task/program.

CSECT pgs (PR.PAGEC)

The number of 512-word pages in CSECT.

DSECT pgs (PR.PAGED)

The number of 512-word pages in DSECT. This is derived from the ending address of the DSECT (PR.ENDD).

Block Size (PR.PGSIZ)

Defines the map block granularity required by the program. This granularity is the granularity specified in the ENVIRONMENT directive. It is the number of 512-word protection granules in a map block (10 hexadecimal for a 32/7x or 4 hexadecimal for a CONCEPT/32).

Mem Class (PR.MEMS)

Defines the class of memory required by the program. This memory class is the memory class specified in the ENVIRONMENT directive. The values currently defined are: 1 for E-class memory, 2 for H-class memory, and 3 for S-class memory. The default value is 3.

Common pgs (PR.PAGEG)

The number of 512-word pages in Global Common/Datapool.

Files (PR.FILE)

The number from the Files directive. The default value is 5 (the requirement for the Debugger).

Buffers (PR.BUFR)

The number from the Buffers directive. The default value is 3 (the requirement for the Debugger).

Program Option Word

A 32-bit program option word.

Description of CSECT and DSECT

CSECT Origin (PR.ORGC)

The address at which to begin loading the CSECT. Bit 7 indicates absolute origin.

CSECT Common Delta (PR.COMC)

Not used, contains 0 (zero).

CSECT Ending Address (PR.ENDC)

The address of the first free word after the CSECT.

CSECT Data Block Number (PR.SFAC)

The file relative block number of the CSECT data.

CSECT Data Byte Count (PR.BYTEC)

The number of bytes of CSECT data.

CSECT Data Checksum (PR.CHKC)

The checksum for the CSECT data. All halfwords of CSECT data are summed in a register.

CSECT Relocation Matrix Block Number (PR.SFACR)

The file relative block number of the CSECT relocation matrix.

CSECT Relocation Matrix Byte Count (PR.BYTCR)

The number of bytes, rounded up to a multiple of 4, of CSECT relocation matrix.

CSECT Relocation Matrix Checksum (PR.CHKCR)

The checksum for the CSECT relocation matrix. All halfwords of CSECT relocation matrix are summed in a register.

DSECT Origin (PR.ORGD)

The address at which to begin loading the DSECT. Bit 7 indicates absolute origin.

DSECT Common Delta (PR.COMD)

Not used, contains 0 (zero).

DSECT Ending Address (PR.ENDD)

The address of the first free word after the DSECT.

DSECT Data Block Number (PR.SFAD)

The file relative block number of the DSECT data.

DSECT Data Byte Count (PR.BYTED)

The number of bytes of DSECT data.

DSECT Data Checksum (PR.CHKD)

The checksum for the DSECT data. All halfwords of DSECT data are summed in a register.

DSECT Relocation Matrix Block Number (PR.SFADR)

The file relative block number of the DSECT relocation matrix.

DSECT Relocation Matrix Byte Count (PR.BYTDR)

The number of bytes, rounded up to a multiple of 4, of DSECT relocation matrix.

DSECT Relocation Matrix Checksum (PR.CHKDR)

The checksum for the DSECT relocation matrix. All halfwords of DSECT relocation matrix are summed in a register.

Global Symbol Block Number (PR.SYMG)

The relative sector number of the global symbol table. A zero indicates no symbols.

Local Symbol Block Number (PR.SYMP)

The relative sector number of the local symbol table. A zero indicates no symbols.

Date of Creation (PR.DATE)

A doubleword containing the date the load module was created. The format is identical to that for C.DATE.

Common blocks are allocated within a segment according to these rules:

1. A common block is allocated preceding the program element which contains a common origin referencing the block.

2. If a common block is not referenced by a common origin, it is allocated preceding the first program element which contains its definition.

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 disc. Therefore, these 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 disc 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 8-word boundaries. The main segment of a module with overlays is placed on a 8-word boundary. If necessary, the size of the transient area is increased so that it consists of an integral number of 8-word units.

References can be made from an overlay segment to symbols contained within the main segment. These symbols may be contained in subroutines called from the subroutine libraries. The symbols are established by using the assembler directives "DEF" and "COMMON". Within the overlay, the assembler directives "EXT" and "COMMON" allow the symbols to be referenced. The main segment may reference symbols which are "DEFs" in overlay segments as main overlays reference symbols in lower level overlays. Other linkage is dependent on the use of the LINKBACK directive.

6.1.4 Symbol Table Output Format

The Symbol Table (SYMTAB) is output by the Cataloger in a format similar to object records output by the assembler. The format of each record is as follows:

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 16. Data block types and their contents are as follows:

Type (Hex)	No. Bytes Data	Contents
0	16	The Symbol Table data output as four-word entries from high to low memory.
5	1 to 8	The name of the segment during whose cataloging the SYMTAB was output.
F	1	None - signals end of output.

A Type 5 block is output first. Type 0 blocks are output next and are terminated by a Type F block. All cards, except the last, contain six data blocks. The last may contain up to seven blocks (6 data and 1 end).

6.1.5 Load Module Format (Card)

When card output of a load module is specified, an image of the load module is built on a temporary file. This image is identical to the load module output to the module's permanent file. An integral number of 192-word records are punched to cards from the temporary file. Therefore, 192 words are output from the last record of the module whether the module occupies the entire record or not.

Cards containing the load module are formatted as follows:

Byte	1	2,3	4,5	6
Type (1)	Byte Count (2)	Checksum of Data	Sequence Number	Data Blocks (3)

(1) For all cards of the module except the last, the hexadecimal value FF is entered. The last card contains the value DF.

(2) The number of bytes of data on the card.

(3) The load module data. Each card contains 28 words of data except that the last card may contain fewer.

6.1.6 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 as a result of the inclusion of the object module into the Cataloger's input stream. The details of the object language follow.

6.1.6.1 Object Module Records

The object module consists of 1 or more variable length records up to 120 bytes. Each record contains 6 bytes of header information which describe that object record.

Byte 1 (Record Type) — This byte field contains either X'FF' or X'DF'. A value of X'DF' indicates the last record of the current object module.

Byte 2 (Byte Count) — This byte field contains the number of data bytes in the current record. It ranges from 2 to 114 (X'2', to X'72') and does not include the 6 bytes of header information.

Bytes 3,4 (Checksum) — This halfword field is the checksum of the data bytes within the record. It is computed by adding the data bytes and truncating the sum to a halfword.

Bytes 4,5 (Sequence) — This halfword field is the sequence number of the current object record. Its initial value is 1. If the field overflows, the count is reset to 1.

6.1.6.2 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 0) of each object command and contains two fields, the function code and the byte count. The function code is the left hand four bits and ranges from X'0' to X'F'. The byte count is the right hand four bits and is the count of data bytes for each command, exclusive of the control byte. A byte count of 0 is interpreted as X'10' data bytes.

A 'stringback' is a linked list of data that are terminated by a zero address. The word containing the zero address is absolute rather than relocatable. The Cataloger will make that word relocatable if necessary. The addresses in the list are module relative and are 19 bit word addresses. The Cataloger preserves bits 30 and 31 in stringing back the data.

The individual commands are detailed below.

6.1.6.2.1 Absolute Data

0	len	Absolute Data

len: Number of bytes of absolute data, 1-16.

6.1.6.2.2 Program Origin

1	3	Program Origin

19-bit origin, right justified.

Bit 0 must be set.

6.1.6.2.3 Absolute Data Repeat

2	len	repeat	Absolute Data

len-1: Number of bytes of absolute data, 1-15.

repeat: Number of times to repeat data, 1-255.

6.1.6.2.4 Transfer Address

3	3	Transfer Address

19-bit transfer address, right justified.

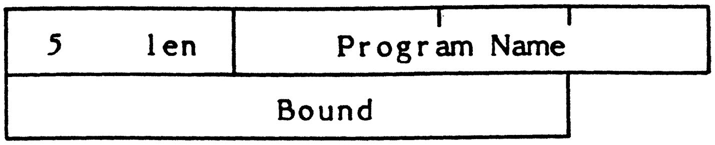
Bit 0 must be set.

6.1.6.2.5 Relocatable Data

4	len	Relocatable
Data		

len: Number of bytes of relocatable data, 4-16. Must be a multiple of 4.

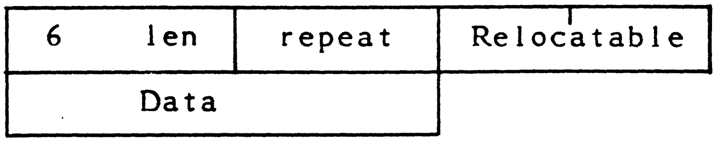
6.1.6.2.6 Program Name



Program Name: 1 to 8 character program name.

Bound: 3-byte field containing the minimum bounding requirement for the program. The nominal value is X'8', the maximum is X'20' or 8 words.

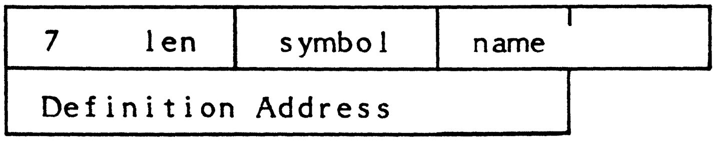
6.1.6.2.7 Relocatable Data Repeat



len-1: Number of bytes of relocatable data, 4-12. Must be a multiple of 4.

repeat: Number of times to repeat data, 1-255.

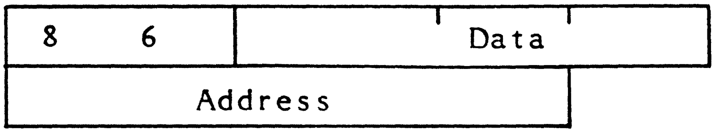
6.1.6.2.8 External Definition



Symbol Name: 1-8 character name of symbol being defined.

Definition Address: 19-bit address, right justified. Bit 0 must be set.

6.1.6.2.9 Forward Reference



Data: 19 bits of data, right justified. Bit 0, if set, indicates data is relocatable

Address: 19-bit address of stringback list. Data is put into each word in list. List is terminated by absolute 0 link. Bit 0 must be set.

6.1.6.2.10 External Reference

9 len	Symbol Name
Stringback Address	

Symbol Name: 1-8 character name of symbol being referenced.

Stringback Address: 19-bit address of stringback list. External address is put into each word in the list. List is terminated by absolute 0 link.

6.1.6.2.11 Common Definition

A len	Common Name
Block	Size

Common Name: 1-8 character name of common.

Block: 1-byte block number, assigned by compiler.

Size: 2-byte size of common in bytes.

6.1.6.2.12 Common Reference

B len	Block	Common
Reference		

Block: 1-byte common block number referenced by data.

Common Reference: 4-12 bytes of data that reference a common block. The base address of the common block is added to the low order 19 bits of each word.

6.1.6.2.13 DATAPOOL Reference

C len	Symbol Name
DATAPOOL Reference	

Symbol Name: 1-8 character name of symbol in DATAPOOL.

DATAPOOL Reference: 4-bytes of DATAPOOL reference. Symbol's value is added to low order 19 bits of DATAPOOL reference.

6.1.6.2.14 Escape to Extended Functions

D X	

Function code of X'D' indicates extended item. See Section 6.1.6.3.

6.1.6.2.15 Common Origin

E 3	Block	Origin

Block: 1-byte common block number.

Origin: 2-byte offset from beginning of common block.

6.1.6.2.16 Object Termination

F 1	0 0

This record terminates the object code for the current module.

6.1.6.3 Extended Object Commands

The extended object commands differ from the previous commands in the following respects:

- Byte 1 contains the function code ranging from 1 to B.
- Byte 2 contains the length of the item including byte 0.

Each command is detailed below.

6.1.6.3.1 Section Definition

D 0	0 1	1 0	bounding
sect. no.	section size (bytes)		
section name - 8 characters			

sect. no. 0 = DSECT
1 = CSECT

section name **DSECT*
**CSECT*

6.1.6.3.2 Section Origin

D 0	0 2	0 8	0 0
sect. no	origin		

origin: The offset within the section to establish as the new origin.

6.1.6.3.3 Section Relocatable Reference

D 0	0 3	length	0 0
sect. no.	rept. cnt.	relocatable data	
4 - 248 bytes.			

sect. no.: Section number the address is defined in.

rept. cnt.: The number of times to repeat the data. A value of 0 is equivalent to a value of 1.

relocatable data: Data in multiples of 4 bytes whose rightmost 19 bits is a relocatable address.

6.1.6.3.4 Section Transfer Address

D 0	0 4	0 8	0 0
sect. no.	transfer address		

transfer address: The offset within the section that is the transfer address.

6.1.6.3.5 Section External Definition

D 0	0 5	length	0 0
sect. no.	definition address		
symbol name 1 - 8 characters			

definition address: The offset within the section where the symbol is defined.

6.1.6.3.6 Section External Reference

D 0	0 6	length	0 0
sect. no.	string back address		
symbol name 1-8 characters			

stringback address: The offset within the section where the stringback list begins.

Note: If the address is zero and bit 0 of the address is set, a stringback is performed to address zero of the section.

A single stringback list may not include references to both CSECT and DSECT.

6.1.6.3.7 Section Forward Reference

D 0	0 7	0 C	0 0
sect. no.	definition address		
sect. no.	stringback address		

definition address: The offset within the section where the symbol is defined.

stringback address: The offset within the section where the stringback list begins.

6.1.6.3.8 Large Common Definition

D 0	0 8	length	0 0
common no.	common size (bytes)		
common name 1 - 8 characters			

common no.: Number assigned by compiler to common block.

6.1.6.3.9 Large Common Origin

D 0	0 9	0 8	0 0
common no.	common origin		

common origin: Offset from beginning of common.

6.1.6.3.10 Large Common Reference

D 0	0 A	length	0 0
common no.	rept. cnt.	relocatable data	
4 - 248 bytes			

common no: Common the address is defined in.

rept. cnt.: The number of times to repeat the data. A value of 0 is equivalent to a value of 1.

6.1.6.3.11 Debugger Information

D flg	0 B	length	flag
type	address		
size		symbol name	
8 characters			
		common name	
(if any) 8 characters			

flg: bit 4 = Symbol is in extended memory (address is that of a 24-bit pointer to the symbol).

bit 5 = Symbol is a formal parameter (address is that of a pointer to the symbol).

bit 6 = Symbol is in common. The common name follows the symbol's name.

bit 7 = Symbol is in DATAPOOL.

length: 18 if symbol is not in common
26 if symbol is in common

flag: bit 0 = Symbol is in CSECT

type:

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 bit address. Bits 29-31 indicate bit within byte while bits 9-28 indicate byte address.

size: Length of datum in bytes.

symbol name: 8 character, left-justified, blank filled variable name.

common name: 8 character, left-justified, blank filled common name.

6.1.6.3.12 Object Creation Date/Time

D 0	0 C	14	0 0
- - - - -	date		- - - - -
- - - - -	time		- - - - -

length: Number of bytes in the command.

date: An eight byte date in the ASCII standard format mm/dd/yy.

time: An eight byte time in the ASCII standard format hh:mm:ss.

Note: This object command is only output by FORTRAN 77+ release 4.1 or later.

6.1.6.3.13 Product Identification

D 0	0 C	length	0 0
product identification			

length: Number of bytes in the command.

product identification A user-supplied string of up to 32 bytes of text identifying the generated object code.

Note: This object command is only output by FORTRAN 77+ release 4.1 or later.

6.2 DEBUG

DEBUG functions essentially as an unsolicited overlay of a task being debugged. When attached to a user task it provides a set of commands through which the user can monitor and control the execution of the task. It is intended primarily as an interactive tool for the on-line terminal user operating under TSM, but may also be used to debug a batch task.

The design goals for DEBUG are the following:

(a) Provide an extensive and flexible set of interactive tools to aid the user in the development of his application software.

(b) Restrict the user's normal access to operating system facilities as little as possible.

(c) Provide a single debugging facility which functions in the on-line and batch environments.

(d) Provide the batch user as many of the on-line debugging facilities as possible.

(e) Provide a sound base for future enhancements, such as symbolic debugging and high-level language facilities.

6.2.1 The DEBUG Environment

The activation sequence for a task to be run under DEBUG is similar to the normal activation sequence except that:

(a) The size of the address space constructed is increased by the size of DEBUG.

(b) Control is given to DEBUG's startup entry point instead of the transfer address of the user task.

The address space constructed for DEBUG is pictured below:

```
*-------------------------------- *
*  GLOBAL COMMON/DATAPOOL         *
*       (IF ANY)                  *
*-------------------------------- *
*                                 *
*          USER'S CSECT           *
*                                 *
*-------------------------------- *
*                                 *
*          DEBUG                  *
*                                 *
*-------------------------------- *
*//////////////////////////////// *
*//////////////////////////////// *
*//////////////////////////////// *
*//////////////////////////////// *
*-------------------------------- *
*                                 *
*          USER'S DSECT           *
*                                 *
*-------------------------------- *
*          TSA                    *
*-------------------------------- *
*                                 *
*                                 *
*          MPX-32                 *
*                                 *
*                                 *
*-------------------------------- *
```

The DEBUG environment is established for a task by a call to H.MONS,29 (M.DEBUG service). The task may call H.MONS,29 at any time. The TSM command "DEBUG" and the Job Control command "$DEBUG" will cause H.MONS,29 to be called as part of the activation sequence for a user task.

Note that the combination of DEBUG and the user's code is a single task, with a single TSA and a single Dispatch Queue entry. When DEBUG gains control at its startup entry point, it makes dynamic assignments for its file codes according to whether it is running on-line or batch. Any dynamic assignments for these file codes made by the user task are prohibited. To minimize conflict with user file codes, all DEBUG file codes begin with the character "#".

When DEBUG gains control, whether at activation or upon the occurrence of a trap or abort, it runs privileged, allowing it to replace user instructions with traps. When DEBUG transfers control to the user's task it restores the privilege state (as cataloged) of the user's task.

6.2.2 Entry Points

DEBUG begins with a halfword address table (HAT) in the following format:

```
DEBUG     DATAW     5
          ACH       DEBUG.1
          ACH       DEBUG.2
          ACH       DEBUG.3
          ACH       DEBUG.4
          ACH       DEBUG.5
```

The entry points have the following functions:

Entry Point	Function
1	Startup
2	Restart
3	Trap/break receiver
4	Re-entry after BREAK command
5	User abort receiver

6.2.2.1 Entry Point 1 - Start-Up

Functional Description

File codes are assigned appropriately to the operating mode (on-line or batch), and files are opened. T.CONTXT is saved for a possible RESTART command. 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 as the result of a call to H.MONS,29 (M.DEBUG service), either 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 6.2.3, or M.EXIT in response to an immediate EXIT command, or H.MONS,50 (described in section 6.2.4).

6.2.2.2 Entry Point 2 - Restart

Functional Description

All files are closed and deallocated. The T.CONTXT image saved at entry point 1 is copied to T.CONTXT. All defaults are reset; e.g., PIF reverts to #IN, SS (single-step) is reset, etc. The trap table and base table are re-initialized. Control is then passed to entry point 1.

Entry Conditions

T.CONTXT contents are unpredictable and not used. T.REGP points to T.REGS+0W. This entry point is called by H.EXEC,24, which is called by DEBUG in response to an immediate RESTART command.

Exit Conditions

Exit is through any of the H.EXEC calls described in section 6.2.3, or M.EXIT in response to an immediate EXIT command, or H.MONS,50 (described in section 6.2.4).

6.2.2.3 Entry Point 3 - Trap/Break Receiver

Functional Description

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 DEBUG proceeds under control of the commands in the trap list.

For a break, the PIF reverts to #IN, a break report is issued on #OT, and DEBUG reads the next immediate command from the PIF.

Entry Conditions

This entry point is called when the user task executes a SVC 1,X'66' (DEBUG 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 6.2.3, or M.EXIT in response to an immediate EXIT command, or H.MONS,50 (described in section 6.2.4).

6.2.2.4 Entry Point 4 - M.BRKXIT Receiver

Functional Description

Execution of the user's M.BRKXIT is reported on #OT. DEBUG then reads the next immediate command from the PIF.

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 a DEBUG call to H.EXEC,23. T.CONTXT, T.REGS, and T.REGP are the same as they were immediately before DEBUG called H.EXEC,23.

Note that if, after DEBUG calls H.EXEC,23, 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 6.2.3, or M.EXIT in response to an immediate EXIT command, or H.MONS,50 (described in section 6.2.4).

6.2.2.5 Entry Point 5 - Abort Receiver

Functional Description

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. DEBUG then reads the next immediate command from the PIF.

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 DEBUG 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 6.2.3, or M.EXIT in response to an immediate EXIT command, or H.MONS,50 (described in section 6.2.4).

6.2.3 H.EXEC Calls

DEBUG uses three special entry points in H.EXEC for control transfers, as follows:

H.EXEC,22

H.EXEC,22 is called by DEBUG in response to an immediate GO or TRACK command, to begin or continue execution of the user task.

H.EXEC,23

H.EXEC,23 is called in response to an immediate BREAK command, to pass control to the user's break receiver. When the user task executes M.BRKXIT, control is passed to DEBUG Entry Point 4.

H.EXEC,24

H.EXEC,24 is called in response to an immediate RESTART command, to clear any push-down levels in T.REGS and perform related cleanup functions prior to passing control to DEBUG Entry Point 2.

6.2.4 H.MONS Calls

This section describes only those H.MONS calls which perform special DEBUG-related functions. DEBUG makes free use of many other H.MONS calls.

H.MONS,29

H.MONS,29 (M.DEBUG) is not called by DEBUG. It is called by a user task, or as part of the activation sequence, to establish the DEBUG environment for a user task.

H.MONS,30

H.MONS,30 is called in response to an immediate KILL command. Its function is to destroy the DEBUG environment previously established by H.MONS,29 (M.DEBUG), leaving the user task intact and transferring control to it at a specified context. In particular, H.MONS,30 makes DEBUG memory available for allocation by the user task.

H.MONS,42

H.MONS,42 (SVC 1,X'66') is the DEBUG trap instruction. It is stored in the user task, replacing the user's instruction, in response to the SET, GO, and TRACK commands. Execution of SVC 1,X'66' by the user task causes control to pass to DEBUG Entry Point 3 after T.CONTXT is loaded with the user context.

H.MONS,50

H.MONS,50 (SVC 1,X'7E') is called by the user program to exit. If DEBUG is associated with the task, Entry Point 5 is entered and a user exit message is generated.

6.2.5 File Code Usage

DEBUG 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 (on-line or batch). It assigns #02 and #03 in response to LOG, DUMP, FILE, and STORE commands. The following table lists the DEBUG file codes and their uses:

#IN Control input (commands)

On-line: ASSIGN4 #IN=UT

Batch: ASSIGN2 #IN=SYC (only if not already assigned)

#OT Primary output (displays, trap reports, diagnostics, etc.)

On-line: ASSIGN4 #OT=UT

Batch: ASSIGN2 #OT=SLO,10000 (only if not already assigned)

#01 Log File (log of all I/O on UT)

On-line: ASSIGN3 #01=DC,300

Batch: Not used

#02 SLO files for LOG and DUMP commands

On-line: ASSIGN2 #02=SLO,x (where "x" is determined by the size of the log or dump being produced)

Batch: Not used

#03 FILE and STORE files

On-line: ASSIGN1 #03=file (where "file" is as specified in FILE or STORE command)

Batch: Same as on-line

#04 Patch file (saved CM commands)

On-line: ASSIGN3 #04=DC,100

Batch: Not used

6.2.6 TSA References

The following TSA areas are referenced by DEBUG:

T.REGS together 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

6.2.7 Communication Region References

None so far.

6.2.8 Dispatch Queue Entry (DQE) References

The following areas of the DQE are referenced by DEBUG:

DQE.USHF to determine operating mode (on-line or batch)

DQE.ATI together with T.REGS and T.REGP to analyze user task interrupt status for abort, trap, and break reports

7. SYSTEM TRACE

System Trace is an MPX-32 debug facility designed to assist in determining the causes of system crashes. System events are recorded circularly in a trace table which may 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 (1 or 2)
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
24-30	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 implanted within 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 40000 to 7FFFF. The table is circular in that 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, i.e., 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 zero 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 4008. 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 non-zero 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 below. 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.

7.1 Trace Type 1 - Task Activation

Macro:

```
        M.TRAC      1

        TBM         0,C.TRACE
        BCT         1,$+2W
        SVC         X'A',1
```

Implanted in H.ALOC such that the address of the Task's Dispatch Queue entry is contained in register 7 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
```

7.2 Trace Type 2 - Task Termination

Macro:

```
        M.TRAC      2

        TBM         0,C.TRACE
        BCT         1,$+2W
        SVC         X'A',2
```

Implanted in H.EXEC such 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
```

7.3 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 such that register 2 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
```

7.4 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 such that register 2 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
```

7.5 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 such that register 3 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
```

7.6 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) such that register 1 contains the Task's Dispatch Queue entry number.

Trace Table Entry

Word 0	Type 06	
1	Interrupt Counter (C.INTC)	
2		
3		
4		
5		
6	DQE Entry # (DQE.NUM)	Task Activation Sequence Number (DQE.TAN)
7		

Printout:

```
C.INTC     END I/O TASK = xxxxxxxx
```

7.7 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

Word 0	Type 07		Level
1	Interrupt Counter (C.INTC)		
2			
3			
4			
5			
6			
7			

Printout:

```
     C.INTC     ENTERINT xxxx
```

7.8 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

Word 0	Type 08		Level
1	Interrupt Counter (C.INTC)		
2			
3			
4			
5			
6			
7			

Printout:

```
C.INTC      EXITINT xxxx
```

7.9 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

Word 0	Type 09	
1	Interrupt Counter (C.INTC)	
2 3		
4 5	PSD	
6	Task Activation Sequence Number (C.TSKN)	
7		

Printout:

```
C.INTC     M.SHUT TASK = xxxxxxxx  PSD = xxxxxxxx xxxxxxxx
```

7.10 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
```

7.11 Trace Type 11 - M.IOFF or BEI

Implemented via 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 5	PSD	
6 7	Task Activation Sequence Number (C.TSKN)	

Printout:

```
C.INTC      M.IOFF TASK = xxxxxxxx  PSD=xxxxxxxx  xxxxxxxx
```

7.12 Trace Type 12 - M.IONN or UEI

Implemented via 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

Word 0	Type 12	
1	Interrupt Counter (C.INTC)	
2		
3		
4	PSD	
5		
6	Task Activation Sequence Number (C.TSKN)	
7		

Printout:

```
C.INTC      M.IONN TASK = xxxxxxxx  PSD= xxxxxxxx  xxxxxxxx
```

7.13 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			
Word 0	Type 13		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     M.CALL TASK = xxxxxxxx  PSD=xxxxxxxx  xxxxxxxx

               MODULE = name,dd
```

where dd is the entry point number.

7.14 Trace Type 14 - SVC (1 or 2)

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      SVC1/2TASK = xxxxxxxx  PSD=xxxxxxxx  xxxxxxxx

                SVC=dddd
```

7.15 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

Word 0	Type 15	
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      M.RTRN/A TASK = xxxxxxxx  PSD = xxxxxxxx  xxxxxxxx
```

7.16 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
```

7.17 Trace Type 17 - Outswap Task

Macro:

```
    MTRAC     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
```

7.18 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 such 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
```

7.19 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 such 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	Schedule Flags (DQE.USHF)	

Printout:

```
C.INTC     RELINQ IPU TASK = xxxxxxxx DQE.LMN DQE.ON

           DQE.USHF = xxxxxxxx DQE.TAD = xxxxxxxx
```

7.20 Trace Type 20 - CALM

Macro:

```
    M.TRAC      20

    TBM         0,C.TRACE
    BCT         1,$+2W
    SVC         X'A'20
```

Implanted in H.IP27 such that R5 contains the CALM number.

Trace Table Entry

Word 0	Type 20	CALM Number
1	Interrupt Counter (C.INTC)	
2 3	PSD	
4 5	Ownername (DQE.ON)	
6	DQE Entry # (DQE.NUM)	Task Activation Sequence Number (DQE.TAN)
7	Stack Frame Address (T.REGP)	

Printout:

```
    C.INTC      CALM TASK = XXXXXXXX PSD = XXXXXXXX XXXXXXXX

                MODULE = XXXXXXXX     CALM = XXXXXXXX

                T.REGP = XXXXXXXX
```

7.21 Trace Type 21 - Mobile Event Trace 1

Implanted by the System Debugger via 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 5	PSD		
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
```

7.22 Trace Type 22 - Mobile Event Trace 2

Implanted by the System Debugger via 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
```

7.23 Trace Type 23 - SVC Type 15

Macro:

```
M.TRAC     23

TBM        O,C.TRACE
BCT        1,$+2W
SVC        X'A',23
```

Implanted in H.IP06.

Trace Table Entry

Word			
Word 0	Type 23		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     SVC15 TASK = xxxxxxxx PSD = xxxxxxxx xxxxxxxx

           SVC = dddd
```

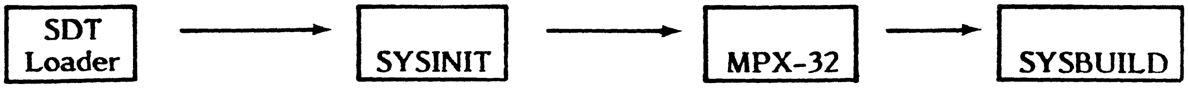
8. SYSTEM INITIALIZERS AND BUILDERS

This chapter describes the components of MPX-32 that are responsible for loading and initializing the system when it is booted. The components involved are:

- o The SDT loader contained within the File Manager
- o The BOOT7X or BOOT27
- o SYSBUILD, 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, all of the above components or a subset may be required.

Figures 8-1 - 8-3 describe the components which come into play when the system is booted from a Software Distribution Tape (SDT), booted from the CPU front panel, and booted from an online RESTART command.



SDT Loader

1. Reads system into memory
2. Reads BOOTxx into memory
3. Transfers control to H.LD

SYSINIT

1. Initializes disc/SMD (cold start only).
2. Puts disc image of bootstrap and system on SMD device
3. Initializes system.
4. Defines and links SYSBUILD as a task.

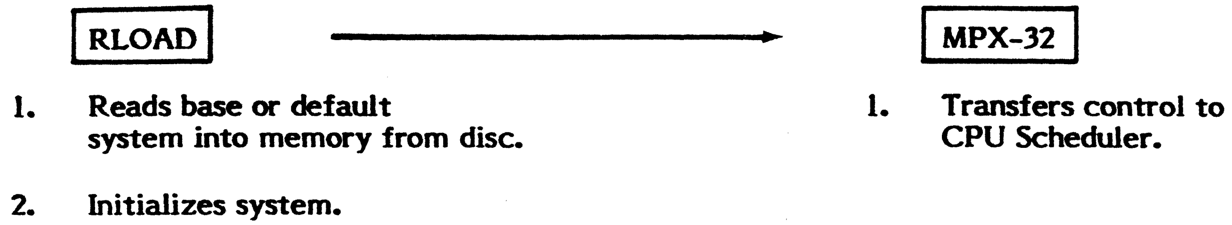
MPX-32

1. Transfers control to SYSBUILD as a ready-to run, real-time task

SYSBUILD

1. Forms disc files from remaining load modules named in SDT directive.
2. Positions tape to user files (Master SDT only).

Figure 8-1
Components and Functions in Boot from an SDT



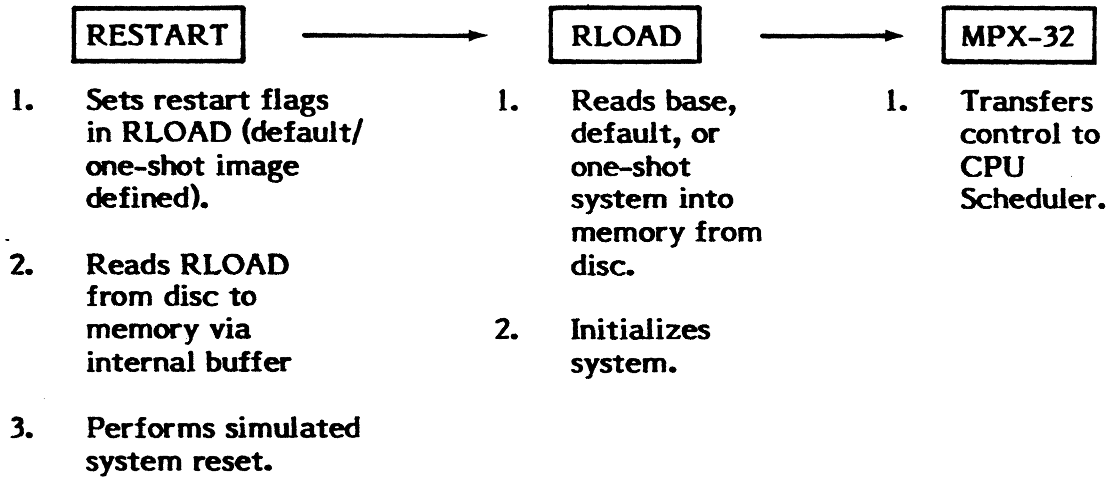
RLOAD

1. Reads base or default system into memory from disc.
2. Initializes system.

MPX-32

1. Transfers control to CPU Scheduler.

Figure 8-2
Components and Functions in Boot from CPU Front Panel



RESTART

1. Sets restart flags in RLOAD (default/ one-shot image defined).
2. Reads RLOAD from disc to memory via internal buffer
3. Performs simulated system reset.

RLOAD

1. Reads base, default, or one-shot system into memory from disc.
2. Initializes system.

MPX-32

1. Transfers control to CPU Scheduler.

Figure 8-3
Components and Functions in Boot from Online RESTART

8.1 SDT Loader

8.1.1 Functional Description

The SDT loader is a section of code written to a Software Distribution Tape (SDT) by the portion of the File Manager that processes the SDT directive. Its purpose is to read load modules from the SDT into memory until it encounters a transfer address. At that time, it transfers control to that address and execution continues within the context of the last module loaded.

8.1.2 SDT Loader Activation

The SDT loader is activated from the SDT by depressing the IPL button on the CPU front panel with the address of the input device displayed on the keyboard. The loader code must be the first piece of information contained on the SDT.

8.1.3 Required Input

The SDT loader requires no input. However, the starting load point may be altered by supplying a value other than zero in register 3 prior to depressing the IPL button. Starting load points default to X'800' for the SYSTEMS 32/7x and X'780' for the CONCEPT/32.

8.1.4 SDT Loader Processing

The SDT loader is written to tape as absolute code by supplying the SDT directive to the File Manager. The loader code is then followed by any other load modules whose names are supplied in the directive.

When the IPL button is depressed, the firmware reads the SDT loader from the IPL medium into memory, starting at absolute address zero. Control is then transferred to location zero, at which time the loader reads load modules into memory from tape or floppy disc sequentially starting at the load point specified in register 3 or the default load point if register 3 is zero.

Modules are 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 module is loaded into memory.

A checksum is performed while loading each module. The checksum is compared with the checksum value supplied in the preamble. If the checksum values do not agree, the loader halts execution.

If a relocation matrix is specified in the preamble, a bias factor (determined by the load point) is added to all relocatable items after the module has been loaded into memory.

After loading the last block of the load module, the loader checks the preamble data to determine if a transfer address is specified. If it is, control is transferred to that address and loading is complete. If no transfer address is specified, the next load module is read in from tape, and the process continues until the loader encounters a transfer address.

8.1.5 SDT Loader Results

At successful completion (no checksum errors), all load modules on the tape up to and including the first module with a transfer address are present in memory.

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 BOOTxx load module. From this point, control is transferred to the BOOTxx entry point as the first transfer address encountered during loading.

The system images created by SYSGEN do not contain a transfer address.

8.2 BOOTxx (RLOAD, SYSINIT, and SYSBUILD)

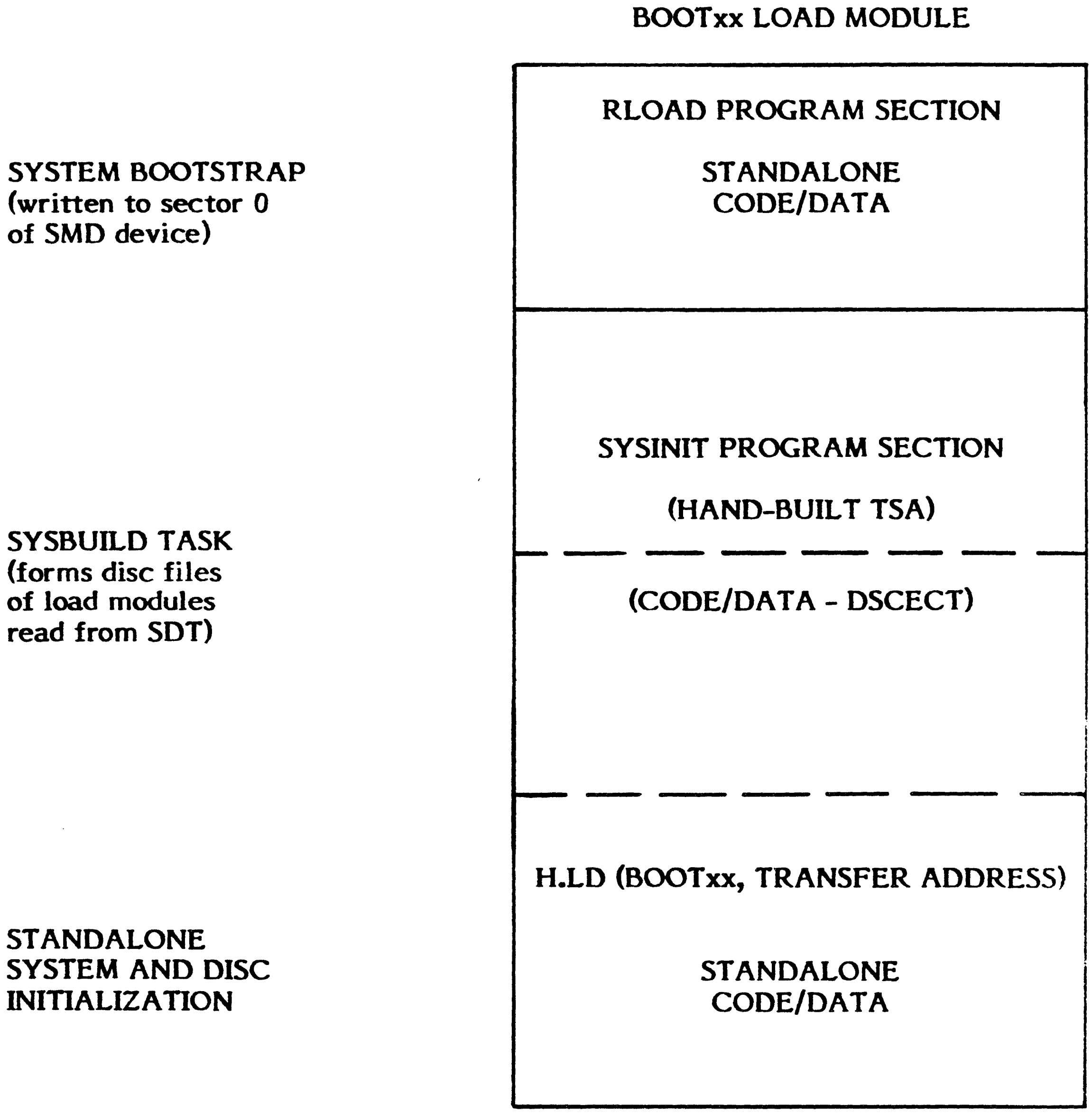
The cataloged load module, BOOTxx, is responsible for system boot from an SDT only. It consists of two program sections. The first section is named RLOAD. RLOAD contains the bootstrap code necessary to boot MPX-32 from a moving-head (XIO/non-XIO) or fixed-head (non-XIO) disc.

The second program section is named, SYSINIT. SYSINIT consists of code which is first executed as a standalone process and later as a task. Running standalone, SYSINIT creates a disc image of the bootstrap code and the MPX-32 load module on the System Master Directory (SMD) device. This allows subsequent boot operations to be directed from the SMD device rather than the IPL device.

SYSINIT then establishes the task, SYSBUILD, as a ready-to-run, real-time task. SYSBUILD creates disc files from load modules on the SDT under a fully operating version of MPX-32 (which does not yet include the File Manager). SYSBUILD is actually comprised of code contained within the SYSINIT program section and is executed as a task after the operating system becomes active. None of the programs or tasks contained within the BOOTxx load module require any input other than replies to console prompts, which will be discussed in context.

Section 8.2.1 shows the structure of BOOTxx.

8.2.1 BOOTxx Components


BOOTxx LOAD MODULE
SYSTEM BOOTSTRAP (written to sector 0 of SMD device)
SYSBUILD TASK (forms disc files of load modules read from SDT)
STANDALONE SYSTEM AND DISC INITIALIZATION
RLOAD PROGRAM SECTION
STANDALONE CODE/DATA
SYSINIT PROGRAM SECTION
(HAND-BUILT TSA)
(CODE/DATA - DSCECT)
H.LD (BOOTxx, TRANSFER ADDRESS)
STANDALONE CODE/DATA

8.3 The RLOAD Program Section

8.3.1 RLOAD Activation

The system bootstrap code, RLOAD, is activated after a system has been installed from an SDT. The user depresses the IPL button on the CPU front panel with the physical channel and subaddress of the SMD device displayed on the keyboard. The bootstrap code will have been previously placed at Sector 0 of the SMD device by SYSINIT and will be transferred from there into memory by the firmware during Initial Program Load (IPL).

8.3.2 RLOAD Processing

When the IPL button is depressed on the CPU front panel with the SMD device address displayed on the keyboard, the firmware reads the bootstrap code into low memory, starting at absolute address zero.

The amount of code read in by the firmware varies, depending on the device class (XIO or non-XIO). The bootstrap must be able to determine the class status of the IPL device prior to executing any of its code. This is accomplished by means of a Device Parameter Table (DPT) and overlay logic. The Device Parameter Table (Section 8.3.4) contains information about the specific system to be booted, and about the device from which that system is to be loaded.

The bootstrap code is always read from the SMD device. However, the operating system can be loaded from the same device or another disc on the same controller as the SMD device. To accommodate this flexibility, the Device Parameter Table can contain information specific to three different versions of the operating system: the base system read from the SDT, a default system supplied via the on-line RESTART task, or a one-shot test system supplied via on-line RESTART.

If it is loading from a non-XIO device, the bootstrap code overlays itself by reading from Sector 0 of the SMD device. This is necessary because only 96W are read into memory on IPL, from a cartridge disc, an amount insufficient to include all of the bootstrap code. Hence, overlay logic is provided within the first 96W of the bootstrap code to insure that all of RLOAD is resident before loading is attempted. This operation is not necessary for XIO devices.

Next the appropriate scratchpad entries are established to perform standalone I/O to the console and boot devices. The console is assumed to have physical address, X'7801' (TLC) or X'7EFC' (IOP), while the boot device information is obtained from the Device Parameter Table by testing the status of the Restart bit flags (base, default, or one-shot).

When the default bit is set, the bootstrap code displays a message at the console, giving the user the ability to select the default or base image before loading begins. The appropriate system image is then loaded into memory in 192W blocks.

The load module preamble is read first to obtain the starting sector address, byte count, and checksum value of the system image on the disc. The RRS block is ignored. Loading starts at X'800' for the 32/7x or X'780' for the CONCEPT/32. The last block of the system load module contains the trap and interrupt vectors established for handlers at SYSGEN. After the system is loaded, this block is moved down to low memory as an absolute module starting at X'0'.

A checksum is performed in a manner similar to that employed by the SDT Loader (see Section 8.1.4) and compared to the value obtained from the load module preamble, if the preamble value is non-zero. If the values do not correspond or if any other I/O error is encountered during loading, error retry logic is executed a set number of times.

With loading complete, the firmware scratchpad is initialized from information contained in the system area pointed to by C.SPAD. Finally, the service interrupt levels for all peripheral devices and designated RTOM interrupt levels are enabled along with the CPU mode. A wait state is then induced. When the interval timer fires, control is transferred to S.EXEC20 (the CPU Scheduler) within the context of a fully operational system.

8.3.3 RLOAD Results

When the bootstrap process is complete, the system indicated by the setting of the Restart flags is loaded into memory and initialized just as though a warm start from tape had been performed.

8.3.4 The Device Parameter Table (DPT)

Function:	Defines device information necessary to boot base, default, or one-shot versions of MPX-32.
Built By:	BOOTxx/RESTART
Managed By:	BOOTxx, RESTART, and RLOAD

<table>
<tr><th>Word #
(Decimal)</th><th>Offset
(Hex)</th><th>0 7</th><th>8 15</th><th>16 23</th><th>24 31</th></tr>
<tr><td>0</td><td>0</td><td colspan="2">R.NRST</td><td colspan="2">SG.NOS</td></tr>
<tr><td>1</td><td>4</td><td colspan="4">SG. IMAGE</td></tr>
<tr><td>2</td><td>8</td><td colspan="4">SG. IMAGE Cont.</td></tr>
<tr><td>3</td><td>C</td><td colspan="4">DF. IMAGE</td></tr>
<tr><td>4</td><td>10</td><td colspan="4">DF IMAGE Cont.</td></tr>
<tr><td>5</td><td>14</td><td colspan="4">IMAGE</td></tr>
<tr><td>6</td><td>18</td><td colspan="4">IMAGE Cont.</td></tr>
<tr><td>7</td><td>1C</td><td colspan="4">SG.STBLK</td></tr>
<tr><td>8</td><td>20</td><td colspan="4">DF.STBLK</td></tr>
<tr><td>9</td><td>24</td><td colspan="4">STBLK</td></tr>
<tr><td>10</td><td>28</td><td colspan="4">SG.PHYDV</td></tr>
<tr><td>11</td><td>2C</td><td colspan="4">SG.LOGDV</td></tr>
<tr><td>12</td><td>30</td><td colspan="4">DF.PHYDV</td></tr>
<tr><td>13</td><td>34</td><td colspan="4">DF.LOGDV</td></tr>
<tr><td>14</td><td>38</td><td colspan="4">IPLDV</td></tr>
<tr><td>15</td><td>3C</td><td>SG.SPT</td><td>DF.SPT</td><td>SPT</td><td>RSFLAGS</td></tr>
<tr><td>16</td><td>40</td><td colspan="4">NBLK</td></tr>
<tr><td>17</td><td>44</td><td>SG.SPB</td><td>DF.SPB</td><td>SPB</td><td>SG.CLAS</td></tr>
<tr><td>18</td><td>48</td><td>DF.CLAS</td><td>CLASS</td><td colspan="2">SG.SPC</td></tr>
<tr><td>19</td><td>4C</td><td colspan="2">DF.SPC</td><td colspan="2">SPC</td></tr>
<tr><td>20</td><td>50</td><td>SG.DEVT</td><td>DF.DEVT</td><td>DEVT</td><td>SG.NHDS</td></tr>
<tr><td>21</td><td>54</td><td>DF.NHDS</td><td>NHDS</td><td colspan="2"></td></tr>
</table>

Word(s)	Bit(s)	Name	Description	Set By	Reset By
0	0-15	R.NRST	Size of bootstrap in 192W blocks	BOOTxx	BOOTxx
	16-31	SG.NOS	Size of base system module in 192W blocks	BOOTxx	BOOTxx
1-2		SG.IMAGE	Base system name (MPX-32)	BOOTxx	BOOTxx
3-4		DF.IMAGE	Default system name	RESTART	RESTART
5-6		IMAGE	Current system name	RLOAD	RESTART
7		SG.STBLK	Starting disc block of base system	BOOTxx	BOOTxx
8		DF.STBLK	Starting disc block of default system	BOOTxx	BOOTxx
9		STBLK	Starting disc block of current system	RLOAD	RESTART
10		SG.PHYDV	Base physical channel/subaddress	BOOTxx	BOOTxx
11		SG.LOGDV	Base logical channel/subaddress	BOOTxx	BOOTxx
12		DF.PHYDV	Default physical channel/subaddress	RESTART	RESTART
13		DF.LOGDV	Default logical channel/subaddress	RESTART	RESTART
14		IPLDV	Current physical channel/subaddress	RLOAD	RESTART
15	0-7	SG.SPT	SMD device sector per track	BOOTxx	BOOTxx
	8-15	DF.SPT	Default device sectors per track	RESTART	RESTART
	16-23	SPT	Current device sectors per track	RLOAD	RESTART
		RSFLAGS	Restart bit flags	RESTART	RESTART
	24		One-Shot System		
	25		Default Image Present		
	26		Console is an IOP		
	27-31		Reserved		
16		NBLK	Size of current system in 192W blocks	RLOAD	RLOAD
17	0-7	SG.SPB	SMD device sectors per block	BOOTxx	BOOTxx
	8-15	DF.SPB	Default device sectors per block	RESTART	RESTART
	16-23	SPB	Current device sectors per block	RLOAD	RESTART
	24-31	SG.CLAS	SMD device class (XIO or non-XIO)	BOOTxx	BOOTxx
18	0-7	DF.CLAS	Default device class	RESTART	RESTART
	8-15	CLASS	Current device class	RLOAD	RESTART
	16-31	SG.SPC	SMD device sectors per cylinder	BOOTxx	BOOTxx
19	0-15	DF.SPC	Default device sectors per cylinder	RESTART	RESTART
	16-31	SPC	Current device sectors per cylinder	RLOAD	RESTART
20	0-7	SG.DEVT	SMD device type (MH or FH)	BOOTxx	BOOTxx
	8-15	DF.DEVT	Default device type	RESTART	RESTART
	16-23	DEVT	Current device type	RLOAD	RESTART
	24-31	SG.NHDS	No. of heads on SMD device	BOOTxx	BOOTxx
21	0-7	DF.NHDS	No. of heads on default device	RESTART	RESTART
	8-15	NHDS	No. of heads on current device	RLOAD	RESTART

8.4 The SYSINIT Program Section

8.4.1 SYSINIT Activation

The SYSINIT program is activated by the SDT loader when its entry point is encountered as a transfer address during a boot from the SDT. Execution continues within SYSINIT as standalone code, since certain operating system elements have not yet been initialized.

8.4.2 SYSINIT Processing

The SYSINIT program section is used only during system boot from an SDT. When control is transferred to its entry point by the SDT loader, the operating system is present in memory, but has yet to be completely initialized.

SYSINIT first loads the firmware scratchpad from information contained within the system code pointed to by C.SPAD. If the System Debugger has been included as part of the operating system, control is transferred to it. The remainder of the standalone portion of system initialization can then be traced by the user.

The appropriate information required to describe the SMD device is then placed in the Device Parameter Table (contained within the RLOAD bootstrap code). The information for the DPT is extracted from the SMD entry of the Unit Definition Table (UDT), which is part of the system code. This information is then used by the bootstrap code whenever a boot of the base system is requested.

The user is then given the option of performing a cold or warm start via a prompt at the console. If a cold start is selected, a new disc allocation bit map is written out to every disc identified in the disc allocation map table pointed to by C.DAMAPT using information contained in the associated UDT. After initializing all bit maps, a zeroed SMD is written out to the SMD device.

A cold start assumes that the SMD device was either void of any useful information or the user wished to remove all traces of the file structure of the previous system.

If a warm start is selected, the modifications to the disc allocation bit maps and the SMD are bypassed. From this point on, processing is the same for a cold or warm start.

The RLOAD bootstrap code is now written out to the SMD device in 192W blocks, starting at Sector 0. A psuedo load-module preamble is written out to the first block following the bootstrap code. The pseudo preamble contains the starting sector address of the system image, a byte count, and a zero checksum value. This is followed by a dummy RRS block to conform to the standard MPX-32 load module format.

Next, the system image, loaded from tape and presently residing in memory, is written out to the SMD device in 192W blocks, starting at the sector specified in the preamble. The tape image of the system thus becomes the base system loaded by the bootstrap code on subsequent system boots from the front panel (providing a default or one-shot system has not been selected). If any errors occur during I/O to the disc, error retry logic is executed a set number of times.

After the system image is written to the SMD device successfully, memory is allocated and defined for the SYSBUILD task. A Dispatch Queue Entry (DQE) is constructed and linked to the head of the real-time task queue. A disc allocation bit map is then written to the SMD device to protect the bootstrap and system images (residing on the low end of the disc) from possible overwrite by the SYSBUILD task.

When this is completed, the last block of the system image in memory is moved to the dedicated location in low memory. This block contains the trap and interrupt vectors defined by the handlers at SYSGEN. The service interrupt levels for all peripheral devices defined in the Controller Definition Table (CDT) are enabled along with designated RTOM interrupt levels. The CPU mode is set, and a wait state is induced.

Up to this point, the operating system has been resident but unable to function on its own. Hence, all I/O operations have been performed using standalone code. Now, when the Real-Time Clock fires, execution continues within the context of SYSBUILD as the first ready-to-run, real-time task under a functioning operating system.

8.4.3 SYSINIT Results

The system read in from the SDT is identified as the base system and an image of it is written to the SMD device along with the bootstrap code.

If a cold start has been selected, the SMD and disc allocation bit maps of all discs are destroyed and re-initialized. The necessary memory management and data structures are constructed so that SYSBUILD is created as a ready-to-run task. The resident system, read from the SDT, is initialized so that it can function on its own.

8.5 The SYSBUILD Task

8.5.1 SYSBUILD Activation

Execution commences within the context of the SYSBUILD task when the Real-Time Clock fires after having been enabled by SYSINIT. This occurs after all system initialization is complete, and the operating system is capable of functioning as an independent entity.

8.5.2 SYSBUILD Processing

SYSBUILD is code within the SYSINIT program that can be executed as a task in order to take advantage of the full functionality of the operating system, specifically, the services in the I/O Control System (IOCS).

First, SYSBUILD allocates and opens the device associated with the address displayed on the CPU front panel during IPL.

If the IPL device is a magnetic tape mechanism, the user receives the standard MOUNT message on the console and can either continue loading modules from tape or abort the SYSBUILD function. The MOUNT message is not displayed when IPL is performed from a floppy disc. If a resume indication is received in response to the MOUNT message, load modules are read from the tape and written to the SMD device under the control of IOCS until SYSBUILD detects an End-of-File mark (written by the File Manager).

This allows key load modules to be introduced to the system at a time when the File Manager is not present. Load modules required at this point are: J.INIT (for final system initialization), J.TSM and OPCOM (for user communication with the system), and FILEMGR (for restoring subsequent files). See Volume 3, Chapter 4 of the MPX-32 Reference Manual for a complete description of the SDT directive.

Upon detecting an EOF mark on the IPL medium, SYSBUILD rewinds the tape. The tape needed for subsequent RESTORE operations via the File Manager can then be mounted.

The rewind function only applies when starting up a system from a master SDT in which several versions of the operating system are present on the tape. After performing the rewind (if necessary), SYSBUILD exits normally. This allows the Swapper to activate J.INIT as the next task for final system initialization functions.

8.5.3 SYSBUILD Results

All load modules included in the File Manager SDT directive after BOOTxx are written to the SMD device so that they are available immediately when control is transferred to the operating system.

8.6 On-line Restart

8.6.1 Functional Description

The cataloged load module, RESTART, functions as a task and provides the user with the ability to reboot MPX-32 online, under TSM control.

RESTART allows the user to replace the existing system with the base system defined from the SDT or with a system of his own. If a system other than the base system is indicated, the user can specify that system as the default for subsequent bootstraps or as a one-time-only, test version.

RESTART is a privileged task and produces the same effect on the system as a boot from the CPU front panel.

8.6.2 RESTART Activation

RESTART is activated in a manner similar to any other TSM task. However, the user must be privileged in order to gain access to the utility. For a full description of the TSM RESTART command, see Volume 3, Chapter 5 of the MPX-32 Reference Manual.

8.6.3 Required Input

RESTART needs to know which system to use. If no name is specified in the RESTART command line, the base system created from the SDT (or the default system, if one is present) will be used. If a name is specified in the command line, it may be designated as the default system by providing an additional keyword after the system name.

The only operator input required by RESTART is a response to a prompt at the user's terminal. This prompt is issued by RESTART prior to loading a new system and has the following format:

```
DO YOU WANT TO BOOT (Y/N):
```

8.6.4 RESTART Processing

The RESTART task begins by verifying that the access flags of the user indicate a sufficient privilege to continue the boot procedure. If the user has sufficient privilege, RESTART reads the RLOAD bootstrap image from sector 0 of the SMD device into a 512W internal buffer. The internal buffer now contains the Device Parameter Table used by the bootstrap code to determine what system to load and where it resides.

Next, the RESTART command line is parsed using the standard TSM scanning routines. If no system name is specified in the command line, RESTART checks to see if the Default bit is set within the Device Parameter Table. If a default image is indicated, the logical and physical channel information for the device associated with that image are initialized for use by the bootstrap code. If a default image is not indicated, the channel information for the base system on the SMD device is used.

When a system name is supplied in the command line, RESTART checks to see if the specified name matches the name of the base system (MPX-32). (This name is established in the Device Parameter Table by BOOTxx before it writes the RLOAD bootstrap image out to disc.)

If the specified name does not match the base name, the file location is determined using the File System Execution (FISE). Then, using information contained in the UDT entry for this file, the appropriate information is inserted into the buffer image of the Device Parameter Table to identify the system as a one-shot system.

The first two blocks of the specified system image are read into a 384W internal buffer. Information in this buffer is compared to the corresponding information in the resident system image to insure that no changes will be made to the SMD when the new system is loaded. If the two SMD definitions are discrepant, the RESTART task is aborted with an error message.

Upon successfully verifying the integrity of the SMD, RESTART checks to determine if the DEFAULT option has been selected for this file. If so, the one-shot system data previously inserted in the buffer image is duplicated in the corresponding default parameter locations of the DPT to identify it as the default system. The Restart default bit flag is also set in the buffer image at this time. The Device Parameter Table portion of the internal buffer now reflects the device data necessary to honor the RESTART command.

RESTART writes the first block of the buffer back to sector 0 of the SMD device, thereby installing the new device data for subsequent bootstrap operations. If a one-shot only request has been made (file name with no DEFAULT option), the one-shot Restart bit flag is set in the internal buffer. In order to preclude the one-shot system from being selected on a system boot directed from the CPU front panel, the one-shot Restart bit is never set in the disc image of the bootstrap code.

The user is now given the option of continuing with the boot procedure or exiting via a prompt at the console. If the response to the prompt indicates that no further action is to be taken, RESTART exits normally. If the user has specified the DEFAULT option, default information has been written to the SMD device and it will be reflected in all subsequent boots from the CPU front panel, even though the online boot operation may have been bypassed.

If the user chooses to boot, modifications to the Device Parameter Table are completed and a simulated system reset is performed. In the simulated system reset, RESTART issues appropriate termination commands to all devices defined in the Controller Definition Table (CDT) found to be active with I/O operations. This is necessary to insure that these devices will be properly initialized by the RLOAD bootstrap operation.

RESTART's final action is to write the internal buffer containing the RLOAD bootstrap image into low memory and to branch to the portion of the code that starts beyond the overlay logic. From this point, system startup proceeds as if a boot from the CPU front panel had been initiated.

8.6.5 RESTART Results

The system specified in the command line is brought up as the new, resident operating system. All devices and I/O activity are terminated and re-initialized. If the DEFAULT option is selected, the Device Parameter Table within the bootstrap image on the SMD device is changed to reflect the presence and characteristics of the default system.

8.7 J.INIT and J.TINIT Tasks

See the MPX-32 Reference Manual, Volume 3, Chapter 9, for J.INIT documentation . J.TINIT is described in Volume 1, Chapter 5.

9. INTERNAL PROCESSING UNIT (IPU)

9.1 Overview

The IPU is a parallel CPU connected directly to the SelBUS. Scheduling for the IPU is accomplished by the MPX-32 executive and a new resident module, H.CPU. Execution of an IPU task is initiated and controlled by a small executive module, H.IPU. Synchronization between the IPU and CPU is maintained by the use of six (6) new traps supported by the 75A version of the CPU/IPU.

Task execution in the IPU is transparent to the user. User intervention is not required for the IPU to execute task level code.

If an IPU accounting interval timer is present and SYSGEN'd into the system, IPU execution time and idle time are tabulated by the resident handler H.IPUIT.

9.1.1 IPU-Memory Interface

The IPU can address all locations of physical memory. All memory allocation occurs before a task is queued for execution in the IPU. The Task Service Area (TSA) for the task is constructed before the task is passed to the IPU, and contains appropriate map block allocation. The IPU and CPU are thus coordinated in use of memory.

9.1.2 IPU-CPU Interface

New traps are implemented to coordinate IPU processing. Traps are:

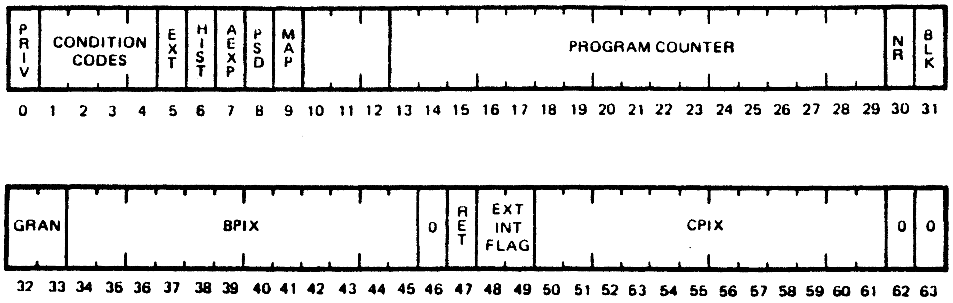
- Start IPU processing
- IPU Supervisor Call
- IPU errors (nonpresent memory, undefined instruction, privileged instruction, etc.)
- IPU Call Monitor
- Stop IPU processing
- Terminate IPU processing and reschedule IPU

9.1.3 System Services and I/O

When a SVC, CALM, or undefined instruction (CD, TC, etc.) is encountered during execution of a task by the IPU, execution is forced back to the CPU. If the task is IPU biased, execution will remain in the CPU only for the single SVC, CALM, or undefined instruction; the next instruction will be executed in the IPU. Any unbiased task will be allowed to execute as many instructions in the CPU as the scheduling algorithm and system resources will allow.

9.2 Data Structures

9.2.1 Program Status Doubleword (PSD)



Bit 0=0 Unprivileged mode
=1 Privileged mode

Bits 1-4 Condition codes as follows:
Bit 1 = CC1
Bit 2 = CC2
Bit 3 = CC3
Bit 4 = CC4

Bit 5=0 Extended mode (OFF) CEA
=1 Extended mode (ON) SEA

Bit 6=0 Last instruction executed was not a right halfword
=1 Last instruction executed was a right halfword

Bit 7=0 Arithmetic exception trap mask (OFF)
=1 Arithmetic exception trap mask (ON)

*Bit 8=0 Computer is in PSW mode (displayed PSD only) (PSW mode not used by IPU)
=1 Computer is in PSD mode (displayed PSD only)

*Bit 9=0 Unmapped (displayed PSD only)
=1 Mapped (displayed PSD only)

Bits 10-12 Reserved

Bits 13-29 Logical word address

Bit 30 Next instruction is a right halfword

*Bit 31 Blocked (displayed PSD only)

Bit 32 Map mode
0=Unmapped
1=Mapped

Bit 33 Reserved

*These bits are used for display only and are not present in the PSD stored in memory.

Bits 34-45 Provide a word index into the Master Process List (MPL) for the base process

Bit 46 Reserved

Bit 47 Retain current map contents

Bits 48-49 Interrupt control flags as follows:

Bits		
48	49	
0	0	operate with unblocked interrupts
0	1	operate with blocked interrupts
1	0	retain current blocking mode
1	1	retain current blocking mode

Bits 50-61 Provide a word index into the Master Process List (MPL) for the current process

Bits 62-63 Reserved

9.2.2 IPU Status Word

Bit	Definition
0	=0, Class 0, 1, 2, or E error* =1, Class F (Extended I/O) error*
1	=0, I/O processing error =1, Interrupt processing error*
2	Final bus transfer error
3	Bus number transfer error
4	I/O channel busy or busy status bit error*
5	Ready timeout error*
6	I/O DRT timeout error*
7	Retry count exhausted error*
8	Operand fetch parity error
9	Instruction fetch parity error
10	Operand nonpresent error
11	Instruction nonpresent error
12	Undefined PSD mode instruction error
13	Memory fetch DRT timeout error
14	Reset channel error*
15	Channel WCS not enabled error
16	Map not found (LEM, SEM, CEMA instructions only) or map register address overflow (map context switch)
17	Unexplained memory error
18	BRI I/O error*
19	Undefined instruction PSW mode only*
20	Map invalid access or map mode restriction error
21	IPU privileged violation or IPL I/O or memory error flag
22	IPU power fail occurred
23	IPU arithmetic exception
24	Enable arithmetic exception trap
25	Disable PSD mode traps
26	Block mode is active
27	IPU status or CPU power fail memory error (32/7x only)
28	Reserved
29	Reserved
30	Reserved
31	=0, CPU mode 55 =1, CPU mode 75
*	Not applicable to IPU

9.3 Options

There are two scheduling options related to IPU task execution -- IPU biased and CPU only. Either option may be specified at catalog time or at execution time.

9.3.1 IPU Bias

This option is specified as IPUB and causes every task level instruction to be executed by the IPU.

9.3.2 CPU Only

This option is specified as CPUO and causes the task to run only in the CPU.

9.4 Task Scheduling

The relationship between the CPU and IPU is symbiotic with the CPU in the dominant role. The IPU's function is to execute task level, unprivileged code in parallel with CPU execution.

There are two C. head cell addresses used by the operating system to control IPU execution: C.CIPU and C.RIPU. C.CIPU contains the Dispatch Queue Entry (DQE) address of the task currently executing in the IPU. There is only one task linked to C.CIPU at any time. C.RIPU contains the DQE addresses of tasks biased to the IPU waiting for IPU execution.

The CPU is responsible for all task scheduling, I/O, and System Services as well as for execution of its own scheduled tasks. Synchronization and communication between the CPU and IPU are accomplished through the use of six new trap vectors in low memory.

Tasks will be scheduled by the CPU for execution in the IPU dependent on their cataloged status as biased or unbiased tasks.

9.4.1 IPU-Biased Tasks

Tasks cataloged with a bias to the IPU will be queued on the IPU ready-to-run state queue, C.RIPU. Entries will be linked by priority, with the highest at the head. As long as tasks reside on the IPU ready queue they will monopolize IPU execution, ahead of higher priority unbiased tasks. Task replacement of biased tasks in the IPU will occur when a higher priority task is queued to the IPU ready state. If a biased task is linked to C.RIPU while a higher priority task is currently in the IPU, the higher priority task will continue to run in the IPU.

9.4.2 Unbiased Tasks

When there are no IPU biased tasks queued to the ready-to-run state, IPU task selection will proceed with the unbiased tasks. A search through the ready states, real time to priority 64, will begin looking for the first eligible task. Eligibility consists of the following:

- o Task is not restricted to CPU execution.
- o Task is not inhibited from execution.
- o There are no run requests or messages outstanding against the task.

If all three of the above are met, the candidate task is selected for IPU execution. If not, the next lower priority task is tested. If there are no tasks meeting the IPU eligibility requirements, the IPU remains idle.

9.4.3 CPU-Only Tasks

Tasks cataloged as CPU-only will not be selected for IPU execution. Typically these tasks are I/O bound.

9.5 IPU Execution

When a task is running in the IPU, it will continue until one of the following occurs:

1. The IPU encounters a System Service request (SVC or CALM).

2. The IPU encounters an exceptional or error condition (privileged instruction, undefined instruction).

3. The CPU executes a 'START IPU' instruction.

In each of the above cases, the IPU will trap via a dedicated vector to the IPU trap handler in order to process the event.

9.5.1 Execution of Time Distribution Tasks in the IPU

Tasks running in the IPU with Batch priorities (55-64) are not subject to priority migration or time distribution while executing in the IPU.

9.5.2 SVC, CALM, Undefined Instructions and Errors

When a SVC, CALM, undefined instruction, or error is encountered during execution of a task by the IPU, an internal trap occurs in the IPU. Processing for each trap is as follows:

SVC — IPU SVC processing consists of 15 indirect secondary vectors pointing to a single ICB. When a trap occurs, the OLD PSD is backed up 1 word to point to the SVC. The corrected PSD and the registers are pushed on the task's stack and the CPU is signaled to take control of the task and schedule another for the IPU.

CALM — The IPU CALM trap consists of a single ICB, PSD retard mechanism and a request for the CPU to take the task and schedule a replacement. The PSD retard mechanism requires the use of bit 6 as well as the F and C bits of the old PSD to determine if the 'calm' is in the left or right halfword.

ERRORS — Defined errors are map fault and privilege errors. These are processed by writing an extended abort code (either MF01.IPU or PV01.IPU) into the task's dispatch queue and setting the delete request bit.

Undefined instruction errors are special case errors in the IPU. Undefined instructions include all I/O class instructions and most of the other privileged instructions (refer to the appropriate IPU Technical Manual for more detailed information on the IPU instruction set). When an undefined instruction is encountered during execution of a task, the IPU will trap to error processing logic with the undefined instruction bit set in the status word. The error logic is special cased for undefined instruction to back-up the PSD, push the stack and pass the task back to the CPU for execution of the instruction.

Undefined errors in the IPU are all those errors other than map fault, privilege error or undefined instructions. They are reported to the user as 'IP01'.

Arithmetic exceptions in the IPU (as in the CPU) are handled by H.IP0F.

9.5.3 CPU Execution of IPU Tasks

An IPU-biased task sent to the CPU for execution of an exceptional condition (SVC, CALM, etc.) will execute in the CPU only until the PSD points back to the task execution area. At this point, the task will be relinked to the IPU ready state.

An unbiased task which has executed in the IPU but has been returned to the CPU will remain in the CPU until it has completed or has been reselected for IPU execution.

9.6 IPU Executive Module Descriptions

9.6.1 Entry Point 1 - IPU Executive

This entry point is used to request task scheduling, start task execution, or stop task execution on the receipt of traps from the 'START IPU'.

9.6.2 Entry Point 2 - Execute IPU Task

This entry point is used to remap the IPU to the task's map, perform a context switch, and dispatch control to the task.

9.6.3 Entry Point 3 - SVC Trap Handler

This entry point is used to correct the PSD to allow the CPU to re-execute an offending instruction. A call to the PUSH and HALT subroutines will be executed.

9.6.4 Entry Point 4 - CALM Trap Handler

This entry point is used to correct the PSD to allow the CPU to re-execute an offending instruction. A call to the PUSH and HALT subroutines will be executed .

9.6.5 Entry Point 5 - Exceptional Condition (ERROR) Trap Handler

This entry point is used to correct the PSD to allow the CPU to re-execute an offending instruction. A call to the PUSH and HALT subroutines will be executed.

9.6.6 Subroutine S.IPU1 - Perform Stack Push

This subroutine pushes the registers and 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.

9.6.7 Subroutine S.IPU2 - IPU Initialization

This subroutine initializes the IPU by storing the MPL address in the scratch pad and by setting the CPU (IPU) status word to 75 mode.

9.6.8 Subroutine S.IPU3 - Terminate IPU Execution

This subroutine contains both the trap which signals the CPU to trap and the privileged wait instruction which is the quiescent state of the IPU.

9.7 IPU Scheduler Module Descriptions

9.7.1 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 0, then the IPU is idle. Entry point 2 schedules the current task.

2. If the head cell count is greater than 0 and the Inhibit IPU flag is set, entry point 2 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 0, but the Inhibit IPU flag is reset, entry point 2 unlinks the task from the current state, relinks it to the IPU request queue, then schedules the new IPU task.

9.7.2 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.

9.7.3 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 elibible task is found, it leaves the IPU idle.

9.7.4 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.

9.7.5 Subroutine S.CPU2 - IPU Eligibility Test

This subroutine contains the tests for task eligibility to run in the IPU. The items checked are:

a. Is the task executing in the monitor (DQE.OSD)?

b. Is the task CPU only (DQE.IPUR)?

c. Is the task IPU inhibited (DQE.IPUH)?

d. Is a system action request pending against the task (DQE.SAIR)?

9.8 IPU Accounting Module Descriptions

9.8.1 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 1 second and the handler is exited.

9.8.2 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.

9.8.3 Subroutine S.IPUIT2 - Perform Accounting Prior to Starting the 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.

9.9 IPU SYSGEN Directive

Upon encountering the following directives, SYSGEN automatically includes H.IPU and H.CPU in the target system and sets C.IPU to indicate the IPU is configured:

```
//HARDWARE
/PARAMETERS
IPU
```

If an IPU accounting interval timer is present, the following is required (after the /INTERRUPTS directive):

```
PRIORITY=77,RTOM=(channel,subaddress),PROGRAM=H.IPUIT,INTV
```

However, if a Scientific Accelerator is also configured, priority 3F should be used for the IPU accounting interval timer.

10. MPX-32 HANDLER FOR HIGH SPEED DATA INTERFACE (HSD)

10.1 HSD Overview

The HSD handler is an optional software component of MPX-32 which provides general device support for user devices connected to an MPX-based SERIES 32 computer.

The handler design is based on the notion that the HSD hardware acts as a controller: it performs hand shaking with the CPU and performs all SelBUS operations needed to fetch and store either data or status relating to requested operations. Therefore, references to the HSD imply controller functions which are generic to I/O operations in general. The device attached to the HSD is the user's addition. Little is known or presumed about its nature. The general assumption is that the device can source and/or synchronize data with the possibility of presenting device specific status on request.

For additional information, refer to the High Speed Data Interface Technical Manual, publication number 325-329132-000.

10.2 Related Data Structures

The HSD handler provides a software interface between MPX-32 tasks and the HSD. The HSD, in turn, provides a hardware interface to a user device. The HSD is a class 'D' I/O device that uses an Input/Output Command List (IOCL) located in the processor memory to provide the commands for the operation. The IOCL is made up of one or more Input/Output Command Blocks (IOCB's). For each I/O operation processed by the HSD handler, an IOCL will be constructed. When the operation can be, it will be initiated by loading the address of the IOCL into the Transfer Interrupt (TI) location assigned to the device and executing the command device instruction START I/O for the HSD.

10.2.1 HSD I/O Command Block Structure

The HSD uses a four word IOCB with the following format:

Word	0 7	8 15	16 23 24 31
0	HSDCMD	UDDCMD	Transfer Count
1	Data Address or Device Command		
2	Unused by Hardware (Software)		
3	HSD/Device Status Word		

where:

HSDCMD is the HSD command and defines the operation of the controller with the following bit definitions:

Bit	
0	When set indicates input transfer, reset for output.
1	Command transfer, Word 1 of IOCB is sent to device.
2	Device status request, store into Word 3 of IOCB.
3	Continue on Error.
4	Interrupt when completed processing IOCB.
5	Transfer in channel; that is, branch to specified IOCB.
6	Command chain, that is, execute next IOCB.
7	Data chain, continue transfer with address and count specified in the next IOCB.

UDDCMD — is the user device dependent command byte and is passed by the HSD to the user device.

Transfer Count — is the count in bytes to transfer when the operation is not a command transfer or Transfer in Channel.

Data Address — is the physical address of data to be read or written (if data transfer operations) or the address of the IOCB to process next if a Transfer in Channel.

Device Command — is a 32-bit value sent to the device in the case that HSDCMD Bit 1 is set to A 1.

Unused Word — is made available for use by the software.

HSD/Device Status Word — is used to store operation status whenever an interrupt on end of block is requested or to store external device status returned in the case that HSDCMD Bit 2 is set.

10.2.2 IOCB Classes

The HSD I/O Command List (IOCL) is composed of one or more I/O Command Blocks (IOCB's). Each IOCB is one of the following general classes depending on the bits set in HSDCMD:

1. Device Command Transfer
2. Device Status Transfer
3. Transfer in Channel
4. Data Transfer Request
5. Data Chain Descriptor

10.2.2.1 Device Command Transfer

Device commands are operations with the command transfer bit set in the HSDCMD byte. The second word of the IOCB is sent to the device. This is generally used to provide device addressing or commanding. This is normally followed by a transfer request, but not necessarily.

10.2.2.2 Device Status Transfer

Under normal conditions, the status posted in the IOCB is controller status, i.e., generic information about the transfer. If specific information about the device is desired, it must be specifically requested by constructing an IOCB with bits 2 and 4 set in HSDCMD. This will cause the HSD to request the specific device status, and store it in the IOCB word 3. By convention, bit 0 of the returned status in word 3 is set to a 1 to flag device status as opposed to controller (HSD) status.

10.2.2.3 Transfer In Channel

This command is used to instruct the HSD to continue processing the IOCL at the address specified in the second word of the IOCB. This is a branch in the I/O Command List. A TIC is generally used to link discontiguous IOCL's together to form one logically contiguous command set or it is used to cause the device to reexecute the I/O Command List. In this case, the interrupt on end of block (word 3 of IOCB) is usually set to inform the software that the device has restarted the I/O command list.

10.2.2.4 Data Transfer Request

All IOCB's which do not have the command transfer, device status request, or transfer in channel bits set are data transfer requests. Write request is indicated by setting bit 0 of HSDCMD to A 1.

10.2.2.5 Data Chain Descriptor

This form of IOCB can only occur following a data transfer or another data chain descriptor IOCB. It has the effect of specifying a continuation of the current transfer operation. It is used to effect a "scatter write" or "gather read." One use is to account for the discontinuities in the logical-to-physical mapping of tasks in MPX-32.

10.3 HSD Request Processing

The handler will accept requests in two general formats. One looks very much like other MPX-32 I/O requests where the buffer address and transfer count as well as an optional device control word is included in the File Control Block (FCB). This is referred to as FCB format.

The other format permits the user to construct an I/O Command List (IOCL) and to have the FCB reference this list of commands to define the desired operation. This is referred to as STARTIO format.

The STARTIO format is indicated by I/O function EXCPM (Execute Channel Program) and the 15 remaining function codes are FCB format.

In START I/O format, there are two variations. The difference is whether the IOCL provided is a logical or physical IOCL. A physical IOCL is a final device I/O command list ready to be processed by the device. All addresses are physical and discontinuities in the logical to physical mapping have been resolved by breaking the request into physically contiguous segments with data chaining. A logical IOCL is an I/O command list representing the desired operation into the logical address space of the task. It must be transformed to a physical IOCL. This means that the addresses must be changed to physical, and logical address discontinuities must be accounted for by breaking the transfer into a series of data chained transfers.

For tasks which will perform the same operation repeatedly, the overhead of translation may not be acceptable. The physical IOCL can be generated once and used on each subsequent request. Note that tasks using a single physical IOCL must be privileged and non-swapable. Physical IOCL is indicated by the task setting the "Data Format Inhibit" bit in the FCB general control flags byte.

10.3.1 FCB Format Request

This form of request provides an interface to the user device compatable with standard devices and provides the flexibility to control almost any device. The FCB interface is designed to permit a device command and/or a data transfer to be initiated as the result of a user request. The standard MPX-32 FCB used to request I/O contains fields to define buffer address, byte count, and device address. See Volume 1, Chapter 7 of the MPX-32 Reference Manual for related FCB information.

For requests which involve data transfer, the buffer address and byte count describe the user buffer for the operation. The device address field in the FCB is used to supply the data to send to the device for device command transfers.

> Note: MPX-32 supports two variant forms of the FCB, normal and extended. Both forms are supported, however, the normal form has limited transfer count and device address fields which will restrict functionality.

MPX-32 supports 16 I/O operation codes. Section 10.3.1.1 defines the operation code number, its name, its corresponding IOCS entry point number for reference, its use by the HSD handler, and special considerations.

10.3.1.1 I/O Operation Codes

Op Code	Name	EP	Use	Notes
0	OPEN	1	device/handler init	only called once until close
1	RWND	2	device control function	(rewind device)
2	READ	3	output data transfer	may cause device command status
3	WRITE	4	input data transfer	may cause device command status
4	WEOF	5	device control function	(write end of file)
5	EXCPM	10	STARTIO format request	not used for FCB format
6	ADVR	7	device control function	(advance record)
7	ADVF	8	device control function	(advance file)
8	BKSR	9	device control function	(backspace record)
9	BKSF	19	device control function	(backspace file)
10	UPSP	20	device control function	(up space)
11	ERPT	21	device control function	(erace or punch trailer)
12	EJCT	22	device control function	(eject)
13	CLOSE	13	device/handler reset	
14	RSVP	24	device control function	(reserve port)
15	RLSP	27	device control function	(release port)

10.3.1.2 Device Open

This function is intended for user device specific processing as required. The standard handler will contain a null routine that can be expanded by the user as needed.

10.3.1.3 Device Close

This is the complementary function to open and will also consist of a null routine.

10.3.1.4 Device Control Functions

All device control functions will dispatch to a common routine in the device driver. An internal table, which could be written at open time, is used to set the UDDCMD field of the IOCB. The device address field from the FCB is used as the device command word (IOCB word 1) and will be sent to the device.

Users have the option of adding specific processing to the standard handler on a per operation code basis. The first FCB special flag bit (bit 8 of word 2 of FCB) is used to indicate if device specific status should be requested after the device command transfer. This results in a second IOCB being added to the IOCL. The device status will be returned in extended I/O status word 2 of the FCB. This requires that the request be made using the expanded FCB.

10.3.1.5 Data Transfer Initiate Requests

Input and output requests are processed in the same fashion. The only difference is the direction bit in the HSDCMD. The following special flag bits are used to control the operation:

SCFG 0	Request Device Status After Transfer
SCFG 1	Send Device Command Prior to Data Transfer
SCFG 2	Disable Timeout on this Request
SCFG 3	Set UDDCMD from Least Significant Byte of Word 2

Request Device Status After Transfer - This bit indicates that an IOCB should be added to the IOCL to retrieve device specific status after the data transfer has completed.

Send Device Command Prior to Data Transfer - This bit indicates that an IOCB should prefix the data transfer to transmit a device command word to the device. If the FCB is the expanded form, the value sent is the 32-bit expanded random access address. Otherwise, the value sent is the least significant 20 bits of word 2 of the IOCB: the random access address field.

Disable Timeout on this Request - This flag indicates that the operation will take an indeterminable period of time and the handler should wait an indefinite period of time for the I/O to complete. This generally only has meaning on read operations.

Set UDDCMD from Least Significant Byte of Word 2 - This bit indicates the UDDCMD byte in the data transfer operation should be set from the least significant byte of the random access address field of the FCB. This provides the ability to pass additional control information to the device without modifying the device driver.

10.3.2 STARTIO Format Requests

The STARTIO format is intended to permit tasks to take full advantage of the hardware capability available from their device/HSD interface.

A STARTIO format request is indicated by the operation code EXCPM. The data format inhibit flag bit indicates request with physical IOCL. For physical IOCL requests, the handler simply queues the request for processing. For logical requests, the IOCL must be made physical.

This involves three transformations. First, all addresses in data transfer IOCB's must be converted to physical. In addition, each time a transfer crosses a physical boundary, (8KW for a 32/7x and 2KW for a CONCEPT/32), the request must be broken at the boundary, and data chaining must be specified to account for the physical discontinuity of the task address space. Finally, Transfer in Channel request addresses must be updated to reflect the final address of the target IOCB. This must account for expansions in the IOCL due to intervening data transfers that have crossed map blocks.

The HSD device handler in MPX-32 supports two additional functions as part of the start I/O protocol. They are:

1. Subtract one and branch non-zero IOCB (SOBNZ).
2. Asynchronous status presentation.

These functions are initiated when the HSD handler receives a non-terminal interrupt (an interrupt when Transfer Interrupt (TI) status does not indicate End of Line (EOL)). This indicates that the interrupt on End of Block bit was set on an IOCB that was not last in the list. The IOCB could be a TIC or a command with the command chaining bit set.

10.3.2.1 Subtract One and Branch Non-Zero

This function is a special version of the Transfer in Channel IOCB. It provides the ability for the task to specify that a series of IOCB's may be executed a specific number of times.

Word	0 7	8 15	16 23	24 31
0	HSDCMD	UDDCMD	Not Used	
1	Address of Next IOCB			
2	Initial Count		Current Count	
3	Normal Next IOCB Address			

The HSDCMD specifies Transfer In Channel (TIC) and interrupt on end of block. When processing starts, the initial count and current count values are equal to the number minus 1 of times to execute the IOCB loop. The contents of word 1 and 3 are equal of the address of the IOCB to execute in the loop.

Each time the HSD processes the TIC, it will interrupt. The handler will detect the interrupt to be a SOBNZ function and subtract one from the current count value. When this causes the count to go to zero, word 1 is modified to contain the IOCB following the SOBNZ. On the next pass, the interrupt will indicate that processing has passed out of the loop. The handler resets word 1 to the loop IOCB from word 3 and resets the current count from the left halfword of word 2.

10.3.2.2 Asynchronous Status Presentation and Notification

Asynchronous status presentation and notification is a software analog to the hardware capability to receive an interrupt and HSD status on the completion of each block, or the ability to explicitly request status from the device at the completion of any operation.

> Asynchronous Status Presentation - When a logical IOCL is converted to physical and asynchronous status is required. The address of the logical IOCB is saved in word 2 of the physical IOCB. When the HSD handler receives a non-terminating interrupt, the IOCB indicated by the TI location indicates status should be posted in the user space. This is done by copying word 3 of the physical IOCB to word 3 of the logical IOCB and by adding to the right halfword of word 2 of the logical IOCB to indicate that the status changed.
>
> Asynchronous Notification - Asynchronous notification is a logical extension to the I/O end action routines in MPX-32. They are delivered with the same priority as I/O end action routines so that notification routines will not interrupt end action routines and vice/versa.
>
> Whenever the HSD updates status in the task's logical address space, it checks to see if the previously requested notification has been delivered. If yes, the request for notification interrupt is requeued to the Task Interrupt Dispatch Queue. If it has not been delivered, nothing further is done. Whenever the task receives the interrupt. It must check for all possible status changes until it finds one that has not changed. On the next interrupt, the task can be assured that the status will have changed, since the hardware executes the IOCL in a well defined order. The task should know where to look for status changes.

10.4 HSD I/O Request Processing Details

When an I/O request is made by a task in MPX-32, IOCS performs some initial validation. If validation is successful, IOCS transfers control to the associated device driver at entry point five with the address of the user's FCB as input. Byte zero of the FCB is set to contain the requested operation code in the range of 0 to 15. The handler performs an indexed jump based on this value. In the standard driver, this will result in codes 0 to 13 going to the FCB format request routine, 14 to the logical IOCL routine, and 15 going to the physical IOCL routine.

10.4.1 FCB Format Request Processing

Processing of the FCB request is done in two phases. First, the required size of the IOCL is computed. This is added to the size of the I/O Queue Entry (IOQ) and the IOQ is allocated from the system's memory pool. The I/O queue portion is then initialized by the IOCS routine INIT.IOQ. The IOCL is then constructed according to the tables based on the tasks operation code. The request is queued to the CDT and initiated in its turn. When it completes, the final status is sent into the FCB and the request is terminated.

10.4.1.1 FCB Request IOCL Size Computation

The IOCL for FCB format request has from one to three sections in it. They are any combination of:

1. Device Command Transfer IOCB
2. Data Transfer IOCL
3. Device Status Request IOCB

The device command and status IOCB's are always one word. Space is allocated if the related function is required. The data transfer IOCL is one IOCB to initiate the request with one additional IOCB for each map block crossed by the transfer. The total size is the sum of the required pieces.

10.4.1.2 FCB Request IOCL Construction

The following procedure defines the construction of the IOCL:

```
1.  Set IOCBPTR to first IOCB.

2.  If device command transfer required, then:

    o  set HSDCMD to CMDXFER
    o  set device address from FCB to word 1 of IOCB

    If data or device status transfer required, then:

    o  set command chain in HSDCMD
    o  advance IOCBPTR
    o  ENDIF
    ENDIF

       or

    If data transfer, then:

    o  set HSDCMD to read
    o  if transfer is write, then set output bit in HSDCMD

    W:  set XFERADR in IOCB to transfer address
    o  set XFERCNT in IOCB to number of bytes within map block
    o  add XFERCNT to transfer address
    o  subtract XFERCNT from transfer count

    If transfer count greater that zero, then:

    o  set data chain in HSDCMD
```

- o advance IOCBPTR
- o go to W
- o ENDIF

3. If device status request, then:

   - o set command chain in HSDCMD
   - o advance IOCBPTR
   - o ENDIF

4. If device status transfer, then:

   - o set HSDCMD to XFRDEVSTAT

5. Set interrupt on End of Block in HSDCMD (for last IOCB)

10.4.2 Logical IOCL STARTIO Format Request

Users of this form of request will generate a logical IOCL. This will contain some number of IOCB's that define their operation. The IOCL will perform any one the following operations:

1. Command Device
2. Transfer In Channel
3. Data Transfer
4. Data Chain

The logical IOCL must be converted to a physical IOCL since the device deals in terms of physical addresses which can not be generated by a non-privileged user. Also, transfers which cross map block boundaries must be separated using data chaining to allow for physical memory discontinuities. The entire IOCL is copied to the system memory pool to perform the conversion. This insures the physical IOCL itself is physically contiguous.

10.4.3 Conversion of Logical to Physical IOCL

The conversion process varies for each type IOCB in the logical IOCL. Command device IOCB's need no conversion other than the move to the physical IOCL.

Transfer In Channel IOCB's must be moved and the physical address of the target IOCB must be found. This results in a two pass conversion of the IOCL. During pass 1, the TIC's are simply copied to the physical IOCL. Also on pass 1, word 3 of each IOCB in the physical IOCL is loaded with the logical address of its counterpart in the logical IOCL. This serves as a logical to physical address translation table. At the end of pass 1, when the IOCL has been expanded to its maximum size, pass 2 is initiated. Pass 2 searches the physical IOCL for TIC's and then searches the physical IOCL for the IOCL which corresponds to the logical IOCB referenced in the TIC. The address is updated.

Data transfer and data chained entries are handled the same way as TIC's. Each time the requested transfer crosses a map block, the IOCB is divided and data chaining is specified. This results in the physical IOCL becoming larger than its logical counterpart which necessitates the use of the translation table for conversion. (Same as previously described for TIC's.)

The size of the physical IOCL is equal to the size of the logical IOCL plus the additional space required for the additional data chain entries needed. Therefore, the size is computed by adding the number of TIC, device command, and device status IOCB's plus the number needed to map each data transfer initiate or data chain IOCB.

10.4.4 Physcial IOCL Processing

Service of this request is a subset of the logical IOCL processing in that the IOCL needs no conversion. Tasks must be privileged and resident to use this form of the request. An I/O queue and a notification packet is allocated for processing this request. When asynchronous status is presented by the device, all that is done is to bump the right halfword of word 2 of the IOCB and request a task interrupt.

10.5 Common Request Handling

For all requests, an I/O queue is allocated and set to describe the request. This includes the IOCL address to account for the variation in location of the IOCL depending on the request type.

10.6 Product Relationships

The HSD handler becomes an integral part of the MPX-32 system when included at SYSGEN time. The handler is designed to operate with MPX-32 release 1.2 and above. It will require minimum 1.2 level software to operate.

10.7 Device Considerations

Development of the generic HSD handler has uncovered several characteristics of the HSD which would require a significant increase in the overhead of using the device if programmed around. They are described in this section.

1. The device will not interrupt on END-OF-LIST unless explicitly told to do so.

   It is software's responsibility to insure that the IOE bit is set whenever an IOCB does not have COMMAND or DATA CHAINING set in order to insure that an interrupt is generated when the device goes idle.

2. In the case of non-present memory or FIFO overflow when the IOCB has CONTINUE ON ERROR set, the device will post status but only interrupt if the INTERRUPT ON END OF BLOCK bit is set. It was felt that the device should interrupt when it posts error status.

   When the HSD is operated in MODE 2, only NON-PRESENT MEMORY errors are effected by CONTINUE ON ERROR. Therefore, it is recommended that MODE 2 be used and that CONTINUE ON ERROR be avoided since NON-PRESENT

MEMORY errors are serious and their cause should be determined and eliminated.

3. If the device stops a transfer due to DEVICE-END-OF-BLOCK and is executing an IOCB in a data chain sequence that is not the last IOCB in that sequence, it will stop processing the IOCL. The device will not interrupt unless the IOE bit was set in the IOCB currently being processed, or the device is operating in MODE 2. INTERRUPT ON END OF BLOCK for an IOCB of this nature is not the normal case. This behavior varies depending on whether it is the last data chaining IOCB or not. Also, if COMMAND CHAIN and no INTERRUPT ON END OF BLOCK is set in the IOCB for which DEVICE-END-OF-BLOCK is posted, no residual byte count is given. In other words, if there is no interrupt, no residual byte count is posted.

   As indicated, this problem does not exist in MODE 2, therefore, it is again recommended that the device be operated in MODE 2 to prevent this behavior from causing a problem.

4. When EXTERNAL MODE is active and the software issues a CD STARTIO, the device rejects the command with a privilege violation, but no indication is given to the software. The operation will timeout, resulting in a HALT I/O being issued which will kill the external operation if it is still going.

   This problem only becomes apparent when the device is operated in a combination of normal (internal) and external control mode. This is probably rare, however, there is no clean work around. One approach is to have the user device force an error at the end of each external mode transfer. This will cause an interrupt each time and if the handler had started a transfer, it would know to restart the operation. Note that any device timeout must be set long enough to insure that the external transfer can complete or else it may get aborted in the device timeout routine.

5. If the interrupt bit is set on a TIC, no SI STATUS is posted, the interrupt is delivered, and the TIC address is lost, causing the device behavior to be unpredictable.

   Interrupts on TIC IOCBs must be avoided. When the software needs to know that the HSD has executed a TIC, it should set the INTERRUPT ON END OF BLOCK on the previous IOCB. By the time the software finds out what is going on, the HSD will have long since executed the TIC and will be off processing the IOCL at the point indicated by the TRANSFER IN CHANNEL IOCB. The speed of the HSD guarantees this. Also, if the TIC is being used to effect a channel program loop and manages to complete the IOCB preceeding the TIC before the software has deactivated the interrupt, the HSD will stall and wait until the level becomes inactive. Therefore, if the software wants to perform a loop counting operation and modify the branch address, it can do so with repeatable behavior as long as the interrupt service routines run with the interrupt level active until after the adjustment, or the decision not to adjust the TIC destination address, is made.

11. MEMORY-ONLY MPX-32

11.1 System Modifications

A synopsis of modifications made to MPX-32 to create memory-only MPX-32 (MPX-32/M) follows.

11.1.1 System Resource Allocator (H.ALOC)

The resource allocation module is modified primarily in the areas of task preactivation and activation (entry points 1 and 2). Details of these changes are discussed in the MPX-32 Reference Manual, Volume 3, Section 10.5.

All references to establishing a swapping mechanism for the task are removed. Any ASSIGN1 (lfc to file) or ASSIGN2 (lfc to system input/output file) references in the Resource Requirement Summary (RRS) block of the load module are ignored and all code associated with the processing of these assignments is deleted. All code associated with the allocation of files of any kind is removed.

Each task is assigned the pseudonym TSM.xxxx where xxxx represents the logical address of the system console or zero if a console is not configured. This allows all tasks to communicate with the operator's console via the TSM mechanism without requiring any changes to the resident module H.TSM. In lieu of this, the task pseudonym entry in the activation parameter block is ignored during an activation from M.PTSK.

In addition to the normal abort conditions, an abort indication will be issued by the memory-only system allocator for the following reasons:

(1) If attempting to allocate an unshared resource that was not available during task activation.

(2) If insufficient memory to perform task loading.

(3) If inability to resume SYSBUILD during system installation.

(4) If inability to deallocate IPL device after dynamic task activation.

(5) If attempt to share memory via a dynamic memory partition.

11.1.2 File System Executive (H.FISE)

The file system executive is reduced extensively from its MPX-32 counterpart. All entry points are defined with the exception of entry points 8 (ASCII compress) and 22 (SYSGEN initialization). All entry points return an unrecoverable I/O error or null return indication. If a call is made to any of the non-applicable entry points, the caller will be able to process the returned conditions as though the call had been made under MPX-32.

11.1.3 System Loader (H.LODR)

The system loader has been converted to a sequential loading device. Information obtained from the preamble buffer is interpreted to determine the starting locations of the various load module sections, as with random access loading. However, if the current position in the load module does not match the starting position of a particular section, the appropriate number of records are advanced to position the file location to the correct point before the information in that section is processed.

11.1.4 System Spooled Output (H.SOUT)

The concepts of system input/output files and batch job stream do not exist in MPX-32/M and all references to it have been eliminated from the resident system code.

11.1.5 System Swap Scheduler (J.SWAPR)

The system swap scheduler is modified to preclude the inswap or outswap of any task. All references to the swap device are removed. The routine that searches for an outswap candidate unconditionally suspends itself through entry point 8 of H.EXEC. Therefore, when a memory scheduling event occurs, outswapping will not take place (hence, no inswapping).

The swapper task continues to perform the functions of MOS memory initialization and other memory related operations in the MPX-32/M environment.

12. E-CLASS DEVICE HANDLERS

This chapter defines a prototype Class E device handler.

12.1 Entry Point 1 - Queue Drive Interrupt Service Routine

Functional Description

This entry point accomplishes device access on behalf of queued I/O service requests. Individual queue entries are processed in the order of the requestor's software priority. Processing of an I/O queue by this entry point continues until the queue is exhausted or until the associated channel is reserved by a user wishing to issue I/O requests directly. Upon entry, this routine will typically perform the following functions:

1. Post-access processing associated with the device access which has just been completed. Post access processing most typically will include device testing, automatic error retry, status information update in the IOQ, minimal format conversions, unlinking of the queue entry for which processing has been completed, and finding the next highest priority queue entry in the I/O queue for processing.

2. Pre-access processing associated with the next queued device access request. Pre-access processing most typically will include minimal format conversions and set up of multiple command device instructions which might be necessary to complete servicing for this single request.

Issuance of the Command Device instructions necessary to service a particular request may, of course, include the storing of the associated TCW (transfer count and data address) in the appropriate dedicated TI (transfer interrupt) location.

When the queue has been emptied, processing is discontinued, and the service interrupt is connected to Entry Point 3 (Spurious Interrupt Service Routine).

Entry Conditions

This entry point is entered at the service interrupt level, as the result of the completion of a command issued to a device.

The new PSD (program status doubleword) is set so that the machine state will be unmapped, interrupts unblocked, and the interrupt level that is being serviced active. The current map at interrupt time is retained. Note that all referenced addresses must be absolute.

The global interrupt count (C.GINT) must be incremented and all registers saved.

Exit Conditions

After issuing the next command, or after determining that the I/O queue is empty and no more commands are to be issued, this entry point returns to the point of interrupt after performing the following functions:

1. Report I/O complete via the appropriate executive routine (S.EXEC1, S.EXEC2, S.EXEC3, or S.EXEC4).

2. Restore the state of the machine to mapped, block all interrupts and issue a deactivate request on the interrupt level being serviced.

3. Exit via executive routine (S.EXEC5).

12.2 Entry Point 2 - Queue Start Interrupt Service Routine

Functional Description

This entry point initiates processing of the I/O queue for this channel. Upon entry, this routine will set the interrupt linkage such that subsequent interrupts at this level will cause execution of Device Handler Entry Point 1 (Queue Drive Interrupt Service Routine). With the exception of post-access processing, (none exists since a command is not completing), and the method of exiting, this entry point is functionally identical to the Oueue Drive entry point. This routine will therefore, typically merge with entry point 1 at the pre-access processing phase.

Entry Conditions

This entry point is entered via a "BL" from IOCS with interrupts blocked. IOCS will cause execution of this entry point whenever a request is queued for this channel and the queue is not actively being driven.

Calling Sequence:

```
M.IOFF                    (Block Interrupts)
BL     *2W,R2             (From IOCS to EP2)
BL W(-4W),R2              GPMC devices
M.IONN                    (Unblock interrupts)
```

Registers:

R1 = CDT Address
R2 = Handler HAT address (H.MUXO's HAT if GPMC) (WDO of 6 word interrupt block if extended I/O)

Exit Conditions

This entry point exits directly back to H.IOCS via a TRSW after restoring the saved registers.

12.3 Entry Point 3 - Spurious Interrupt Service Routine

Functional Description

This entry point is connected to the appropriate SI level whenever the I/O queue is not being driven (the channel is not in use). The function of this entry point is to prevent a spurious type interrupt from causing illegal execution of the normal handler entry points. Normally, this entry point will keep a tally of all such spurious interrupts received. This tally can be used as a check to insure proper operation of the I/O channel.

Entry Conditions

This routine is entered from the associated SI interrupt level. The global interrupt count (C.GINT) must be incremented and all registers saved.

Exit Conditions

Exit from this entry point is via executive routine S.EXEC5.

12.4 Entry Point 4 - Lost (Timed Out) Interrupt Processor

Functional Description

The function of this entry point is to take corrective measures appropriate to the device when an expected service interrupt fails to occur. A running tally of all such time-outs is normally kept for subsequent reference. Typical corrective action is to reset the status of the channel by executing a CD "Terminate" command to the channel. This will cause entry into entry point 1 where normal status testing and error retry can occur.

Entry Conditions

Whenever a command is issued to a device (normally in Entry Point 1 or Entry Point 2) a timer associated with the channel for that device may be set. Every timer unit the real time clock interrupt handler (H.IPCL) calls S.IOCS5 to see if a timer has expired. If the timer has expired, S.IOCS5 forces execution of entry point 4 of the Device Handler. When an interrupt occurs, a handler utilizing this time-out feature must reset the timer so that it will not expire. Execution of this entry point of the handler takes place at the priority level of the clock connected to H.IPCL.

Calling Sequence:

```
M.IOFF              (Block Interrupts)
BL     *4W,R2       (From S.IOCS5 to handler EP4)
M.IONN              (Unblock Interrupts)
```

Registers:

R1 = UDT address
R2 = HAT address of handler
R3 = CDT address

Exit Conditions

After taking the appropriate corrective measures,the Device Handler Entry Point 4 must return to S.IOCS5. This can be accomplished by execution of the TRSW instruction.

12.5 Entry Point 5 - Opcode Processing Reentrant Service

Functional Description

In requesting a particular operation to be performed, the user program will invoke the appropriate IOCS entry point (open, read, write, etc.). The IOCS entry point will accomplish processing of this request which is common to all devices (e.g. general parameter validation, etc.). Ultimately, however, the IOCS entry point must pass control to the appropriate device handler for request processing which is unique to each device. This is accomplished by the IOCS entry point storing an operation (function) code in the first byte of the FCB and then entering entry point 5 of the Device Handler. Entry point 5 will then examine the saved OP code to determine the requested operation. User control specifications pertinent to the request are of course, contained in the FCB. Entry point 5 may wish to examine these specifications in order to handle the service request. For service requests which do not require device access, (e.g. rewind of a disc file) a service complete return can be made to IOCS. For requests which require device access, the handler should call subroutine S.IOCS13 to allocate I/O queue space and buffer space if necessary. The handler should then make a "Request Must be Queued" return to IOCS. If this type of return is taken, IOCS will expect to find in registers 5, 6, 7, the special information which must be stored in I/O queue entry words. These three words are called the Handler Function Words, and may contain whatever information is desired to be passed from Device Handler entry point 5 to Device Handler entry point(s) 1 and/or 2. These function words would typically contain (1) any special handling flags required, (2) the transfer quantity and data address (TCW), and (3) the appropriate command device instruction.

Entry Conditions

Calling Sequence:

```
        BL     *5W,R3        (IOCS to handler EP5)
```

Registers:

R1 = FCB address
R2 = FAT address
R3 = HAT address

Exit Conditions

Entry point 5 must select one of the following four returns by branching unconditionally to the external labels listed below.

Return 1 (Illegal OP code-BU ILOPCODE) is selected when the OP code passed to the device handler is clearly in error (e.g. a read operation requested from the line printer device handler).

Return 2 (Servicing Complete-BU SERVCOMP) is selected when the request has been serviced, with no device access required (e.g. rewind of disc file). This return may also be taken to ignore requests for operations which have no meaning for this device, but which the user program may have included for the sake of device interchangeability.

Return 3 (Post Transfer Processing Required-BU POSTPROS) is selected when device access is required and a request must be queued, but in addition, lengthy post transfer processing, which should not be done at the interrupt service routine level, must be performed. Lengthy format conversion or input data manipulation are examples of this type of post transfer processing. This will cause IOCS to impose "wait" I/O upon the user, and when the requested device access is complete, to enter Device Handler Entry Point 6 to accomplish the post transfer processing at the user's software priority level.

Return 4 (Queue Request-BU IOLINK) is selected when normal device access is required to service the request.

(Note: Registers 5, 6, 7 must contain the handler function words as described above).

Registers:

R1 = FCB address

12.6 Entry Point 6 - Post Transfer Processing Service

Functional Description

This entry point is used if the device requires lengthy post transfer processing after input. It would typically be used for lengthy format conversion or input data manipulation.

Entry Conditions

This entry point is entered at the software priority level of the user on whose behalf the original I/O request was issued.

Calling Sequence:

```
BL     *6W,X3          (IOCS to handler EP6)
```

Registers:

R0 = Return address
R1 = FCB address
R2 = FAT adress
R3 = HAT address of handler

Exit Conditions

This entry point exits to IOCS by executing a TRSW R0 instruction.

Registers:

R1 = FCB address

12.7 Entry Point 7 - Error Processing

Functional Description

This entry point is entered at the software priority level of the user when IOCS determines that an error or abnormal condition was encountered during device access.

Entry Conditions

Calling Sequence:

```
        BL    *7W,R2        (IOCS to handler EP7)
```

Registers:

R1 = FCB address
R2 = HAT address of handler
R3 = IOQ address

Exit Conditions

Entry point 7 must select one of the two returns listed below. The return is selected by adding 0, or 1 words to the return address contained in register zero, and then executing a TRSW RO instruction to effect the return.

Return 1 (Operator Intervention Not Applicable) is selected when operator intervention, followed by operator specified retry of the device access is not applicable to this particular error. It should be noted here that compatible treatment of error status between entry points 1 and 7 is necessary. In order to effect operator specified retry, the I/O queue entry on whose behalf the error was encountered must be left strung to the Controller Definition Table (CDT), and not unlinked as it would be if it was determined by entry point 1 that operator specified retry did not apply. If this return is taken, the IOQ must have been unlinked from the CDT during entry point 1 processing.

Return 2 (Allow Operator Intervention) is selected when it is desired that an error notification message be printed by IOCS on the operator's console, and the operator given the opportunity to specify retry or abort of the I/O operation. If this return is taken, the IOQ must remain linked to the CDT during entry point 1 processing.

Registers:

R1 = FCB address
R3 = IOQ address

12.8 Entry Point 8 - Device Handler Initialization I/O Handlers

Functional Description

This entry point, which may be entered only by SYSGEN, provides the capability of handler self-initialization of all DCC/device dependent parameters, i.e., Command Device and Test Device instructions. Device handler initialization, which is normally entry point 8, must be the last entry point of a handler, and code connected to this entry point should physically be located at the end of the handler.

Communications between the system (including SYSGEN) and the handler is via the handler's Halfword Address Table (HAT). This protocol makes the following options available to the handler:

1. In all cases, SYSGEN resumes loading of subsequent modules at the address indicated by the entry point 8 HAT pointer. This enables the handler to allocate or release space to the system. Generally, this option is used to overlay the handler initialization once it has been executed.

2. If a copy of a reentrant handler has already been loaded (to service another DCC), the handler initialization may link its HAT to that logic. Space occupied by the unneeded logic may be returned to the system by updating the address in its entry point 8 HAT pointer.

Entry Conditions

Calling Sequence:

```
        BL      *8W,R2          (SYSGEN to handler EP8)
```

Registers:

```
        R0 = Return address
        R2 = HAT address of handler
        R5 = HAT address of GPMC device handler if applicable (e.g., H.ASMP)
        R6 = UDT address
        R7 = bits 0-7       hardware priority level
             bits 8-31      CDT address
```

Exit Conditions

Return Sequence:

```
        M.XIR     HAT       (Special SYSGEN Initialization Termination Macro)
```

Note: "HAT" is a label equated to 0 relative within the handler.

Registers:

Same as on entry

Common Handler Subroutines

A set of resident, reentrant subroutines are provided for use by the I/O Device Handlers. This set of subroutines is designed for use by entry point 1 or 2 of a device handler or by the I/O post processing routine S.IOCS1.

12.9 Subroutine S.IOCS2 - Common Test Device

Functional Description

The purpose of this routine is to accomplish device testing and to update the resulting status information in the user's I/O queue entry. It is entered at the device SI priority level.

Entry Conditions

Calling Sequence:

```
BL    S.IOCS2
```

Registers:

```
R1 = IOQ address
R2 = CDT address
R3 = address of test device parameter block
```

Note: This test device parameter block consists of 3 test device instructions for the appropriate device as follows:

```
TD8000
TD4000
TD2000
```

In the case of a device to which 2000 level status does not apply, two NOP instructions may replace the TD2000.

Exit Conditions

Return Sequence:

```
TRSW R0*
```

Registers:

```
R1,R2,R3,R4,R5     = same as on entry
R0,R6,R7           = destroyed
```

12.10 Subroutine S.IOCS3 - Common Queue Entry Unlink

Functional Description

The purpose of this routine is to unlink the current I/O queue entry from the Controller Definition Table (CDT) link list. It is entered by entry point 1 of the device handler or by the I/O post processing routine S.IOCS1 at the device SI priority level or with interrupts blocked respectively.

Entry Conditions

Calling Sequence:

```
BL     S.IOCS3
```

Registers:

R2	=	IOQ address
R3	=	CDT address

Exit Conditions

Return Sequence:

```
TRSW     R0
```

Registers:

R1	=	CDT address
R2,R3	=	same as on entry
R0,R4,R5,R6,R7	=	destroyed

12.11 Subroutine S.IOCS4 - Half ASCII to ASCII Conversion

Functional Description

The purpose of this routine is to convert half-ASCII to ASCII coded data. It is called by the special I/O post processing routine S.IOCS1 and runs at the requestor's software priority. Note that the data is converted in place within the buffer ,i.e., data to be converted buffer = destination buffer.

Entry Conditions

Calling Sequence:

```
BL     S.IOCS4
```

Registers:

R2	=	address of data to be converted
R2	=	destination address for converted data
R4	=	negative number of bytes to convert

Exit Conditions

Return Sequence:

```
TRSW     R0
```

Registers:

R0,R1,R5,R7 = same as on entry
R2,R3,R4,R6 = destroyed

12.12 E-CLASS Line Printer Handler Coding Example

```
*LINE PRINTER HANDLER             RELEASE 1.5 MPX     HEADER  H.LP00
*  DATE  6/02/82     TIME : 17:44
         PROGRAM   H.LP00
**********************************************************************
*                                                                    *
*        LINE PRINTER HANDLER                                        *
*                                                                    *
*                                                                    *
*                                                                    *
**********************************************************************
         M.EQUS
         ENDM
         M.IOQ.
         ENDM
         M.COT.
         ENDM
         M.UDT.
         ENDM
         M.FCB.
         ENDM
         M.ICB.
         ENDM
         M.DQE.
         ENDM
         EXT       S.EXEC3
         EXT       S.EXEC4
         EXT       S.EXEC5
         EXT       S.IOCS2
         EXT       S.IOCS3
         EXT       S.IOCS10
         EXT       S.IOCS15
         EXT       S.IOCS13
         EXT       ILOPCODE
         EXT       SERVCOMP
         EXT       IOLINK
*
*
*          HANDLER ADDRESS TABLE
*
HAT      EQU       $
         DEF       H.LP00,LP00.8
H.LP00   EQU       $
         DATAW     8               NUMBER OF HANDLER ENTRY POINTS
         ACH       LP00.1          QUEUE DRIVE INTERRUPT SERVICE ROUTINE
         ACH       LP00.2          QUEUE START INTERRUPT SERVICE ROUTINE
SPINT    ACH       LP00.3          SPURIOUS INTERRUPT SERVICE ROUTINE
         ACH       LP00.4          LOST (TIMED OUT) INTERRUPT PROCESSOR
         ACH       LP00.5          OPCODE PROCESSING REENTRANT SERVICE
         ACH       LP00.6          POST TRANSFER PROCESSING SERVICE
         ACH       LP00.7          ERROR PROCESSING ENTRY POINT
         ACH       LP00.8          INIT E. P.
*
*
*
*
*        LINE PRINTER
*        COMMON PARAMETERS :   E. P. 8 INIT
*
PA1      DATAW      0                     CDT ADDRESS
PA2      TD         X'7A',X'8000'         TD 8000
PA3      TD         X'7A',X'4000'         TD 4000
PA4      TD         X'7A',X'2000'         TD 2000
PA5      CD         X'7A',X'0'            CD (BASIC)
PA6      DATAW      -4                    LOST INTERRUPT TIME OUT VALUE
PA7      DI         X'21'                 DISABLE INTERRUPT INSTRUCTION
PA8      EI         X'21'                 ENABLE INTERRUPT INSTRUCTION
PA9      RI         X'21'                 REQUEST INTERRUPT INSTRUCTION
PA11     CD         X'7A',X'1000'         CD TERMINATE
PA12     DATAW      0                     SPURIOUS INTERRUPT COUNTER
PA13     DATAW      0                     LOST INTERRUPT COUNTER
PA14     DATAW      0                     TI DEDICATED LOCATION
PA15     DATAW      0                     SI DEDICATED LOCATION
*
*        LINE PRINTER
*       SPECIAL PARAMETERS :   E. P. 8 INIT
*
HLPCDT   EQU        $-1W
SPA1     DATAW      X'FFD68800'           - CD EJECT AND PRINT          1
SPA2     DATAW      X'FFD68000'           + CD NO SPACE AND PRINT       2
SPA3     DATAW      X'FFD68800'           1 CD EJECT AND PRINT          3
SPA4     DATAW      X'FFD60100'           0 CD SPACE (DOUBLE SPACE)     4
SPA5     DATAW      X'FFD68100'             CD SPACE AND PRINT          5
*
SPA6     DATAW      X'FFD60800'           CD EJECT (OP-CODE)
* CD TABLE FOR FORMAT CONTROL ONLY (QTY TO PRINT EQU ZERO)
SPA7     DATAW      X'FFD60800'           - CD EJECT
SPA8     DATAW      X'FFD60800'           + CD EJECT
SPA9     DATAW      X'FFD60800'           1 CD EJECT
SPA10    DATAW      X'FFD60100'           0 CD SPACE
SPA11    DATAW      X'FFD60100'           SPACE CD SPACE
SPA12    DATAW      X'FFD61000'           CD TERMINATE
HLPFCTBL DATAB      C' ',C'0',C'1',C'+',C'-'

*
*        INTERNAL SAVE CELLS, FLAGS, AND MASKS
*
         BOUND      8W
STACK    RES        8W                    REGISTER SAVE AREA
ACTIVE   RES        1W                    ACTIVE QUEUE ADDRESS
COMQEADR RES        1W                    COMPLETING QUEUE ADDRESS
FLAGS    DATAB      0                     FLAGS
ENTRYPT  EQU        2                     IF SET - ENTERED AT EP2
POSTPROC EQU        3                     IF SET - POST PROCESSING REQUIRED
CDTERMFL EQU        7                     IF SET - CD TERMINATE WAS ISSUED
MASK2B   DATAW      X'FFFF'               TWO BYTE MASK
MASK3B   DATAW      X'FFFFFF'             THREE BYTE MASK
MASKADR  DATAW      X'FFFFF'
TERMSTAT DATAW      X'40208000'           DCC ERR, CD TERMINATE
         BOUND      1D                    SCRBLK MUST BE DOUBLEWORD BOUNDED
SCRBLK   REZ        22W                   22 WORD SCRATCHPAD FOR S.EXEC
```

```
*****************************************************************
*                                                               *
*       LINE PRINTER HANDLER                                    *
*            ENTRY POINT 1                                      *
*            QUEUE DRIVE INTERRUPT SERVICE ROUTINE              *
*                                                               *
*****************************************************************
        BOUND     1D
LP00.1  REZ       2W                  OLPSD SAVE AREA

        GEN       12/X'800',20/W(EP1)  EP1 EXECUTABLE CODE ADDR
        DATAW     X'00010000'         UNMAPPED, B47 = KEEP CURR MAP
*                                     B48, B49 = 0 - UNBLK INT & LEAVE ACT
IOCDA   DATAW     0                   ADDR OF 2ND WORD OF IOCD (32/27)
TCW     DATAW     0                   DEDICATED TCW LOCATION (32/27)
EP1     ABM       31,C.GINT           ADD 1 TO GLOBOL INTERRUPT CNTR
        STF       R0,STACK            SAVE REGISTERS
        M.TRAC    7,21                FOR DEBUG
        IFT       C.TRACF,102E9
        TBM       7,C.TRACE
        BCT       1,$+2W
        IFT       7 .EQ. 7,102EA
        SVC       X'B',X'21'
        GOTO      102E9
102EA       IFT       7 .EQ. 8,102EB
        SVC       X'C',X'21'
        GOTO      102E9
102EB       SVC       X'A',7
102E9       ANOP
        ENDM
        LW        R1,ACTIVE           GET QUEUE ENTRY ADDRESS
        STW       R1,CUMQEADR         SAVE AS COMPLETING QUEUE ADDR
        TRR       R1,R2               R2 = ACTIVE IOQ ENTRY ADDRESS

        TBM       22,IOQ.FLGS,X1      KILL COMMAND ISSUED?
        BS        UNLINK              YES, BRANCH
        TBM       3,IOQ.CONT,X1       IS TESTING INHIBITED?
        BCT       SET,GOON            YES, BRANCH
        BU        HLPTDS              NO, TEST AND PLACE STATUS IN IOQ
GOON    LW        R2,ACTIVE           GET ACTIVE QUEUE ENTRY ADDRESS
        ZBM       CDTERMFL,FLAGS      CLEAR CD TERMINATE FLAG
        ZBM       0,IOQ.FCT1,X2       CLEAR DOUBLE FLAG
        BCT       SET,DODUB           BRANCH IF SET
GONODB  TBM       1,IOQ.FCT1,X2       BOM FLAG?
        BCF       SET,NOBOM           NO, BRANCH
        SBM       5,IOQ.IOST,X2       SET BOM FLAG IN IOQ WRD 14
NOBOM   ZBM       0,IOQ.STAT,X2       CLEAR IN PROGRESS BIT IN IOQ
        LW        R7,IOQ.FCT2,X2      FETCH TCW FROM IOQ
        SRL       R7,20               KEEP QUANTITY ONLY

        STH       R7,IOQ.UTRN,X2      STORE QUANTITY TRANSFERED INTO IOQ
UNLINK  EQU       $
        LW        R3,PA1              FETCH CDT ADDRESS
        SBM       POSTPROC,FLAGS       INDICATE POST PROCESSING REQUIRED
*       UNLINK IOQ ENTRY FROM IOQ CHAIN
*       ENTRY:
```

```
*            X2 = IOQ ADDRESS
*            X3 = CDT ADDRESS
*          EXIT:
*            X2 & X3 = UNCHANGED
*            R1,R4,R6 = DESTROYED

          BL        S.IOCS3           UNLINK I/O QUEUE FROM CDT
MGEP2     EQU       $
          LB        R2,CDT.IOCT,X3    ANY MORE I/O QUEUE ENTRIES?
          BZ        QEMPT             NO, BRANCH
          LW        R4,MASK3B         FETCH 3 BYTE MASK
          LMW       R2,CDT.FIOQ,X3    GET NEXT QUEUE ENTRY ADDR FROM CDT
          STW       R2,ACTIVE         SAVE QUEUE ENTRY ADDRESS
          SBM       0,IOQ.STAT,X2     SET IN PROGRESS STATUS IN QUEUE ENTRY
NORMCD    LW        R7,IOQ.FCT2,X2    GET TCW FROM QUEUE ENTRY
          LW        R3,PA1            GET CDT ADDRESS
          STW       R7,*CDT.TIAD,X3   STORE TCW IN DED. TI LOC. VIA CDT
          SBM       1,CDT.FLGS,X3     SET I/O OUTSTANDING IN CDT
          LB        R7,C.MACH         FETCH CPU TYPE
          CI        R7,2              32/7X?
          BLT       SET.TIM           IF SO ... SET TIME OUT VALUE
          LA        R7,TCW            FETCH ADDR OF DEDICATED TCW LOCATION
          STW       R7,*IOCDA         STORE TCW ADDR IN IOCB
SET.TIM   LMW       R3,IOQ.UDTA,X2    FETCH UDT ADDRESS
          LW        R7,PA6            FETCH TIME OUT VALUE
          STW       R7,UDT.PTOV,X3    STORE TIME OUT INTO UDT
          SBM       1,UDT.FLGS,X3     SET I/O OUTSTDG BIT IN UDT
          EXM       IOQ.FCT3,X2       EXECUTE CD IN QUEUE ENTRY


EXIT      ZBM       ENTRYPT,FLAGS     CLEAR ENTRY POINT FLAG
          BCT       SET,EXIT2         BRANCH IF ENTRY FROM EP2
EXIT1     ZBM       POSTPROC,FLAGS    SHOULD POST PROCESS EXIT BE TAKEN?
          BCF       SET,MERGE         NO, BRANCH
          LW        R2,COMQEADR       FETCH COMPLETING IOQ ADDRESS
          TRR       R2,R6             R6 = IOQ ADDR

          LB        R1,IOQ.PRGN,X2    FETCH PROGRAM NUMBER
          LA        R3,SCRBLK         FETCH 22W SCRATCH BLOCK ADDR
*                                     R1 = PROGRAM NUMBER
*                                     R2 = IOQ ADDRESS
*                                     R3 = 22 WRD SCRATCH BLOCK ADDR
*                                     R6 = IOQ ADDRESS
          TBM       22,IOQ.FLGS,X2    KILL COMMAND ISUED?
          BNS       CK80              NO, BRANCH
          TRR       R2,R3             R3 = COMPLETING IOQ ENTRY ADDRESS

          BL        S.IOCS15          DELETE THE IOQ ENTRY
          BU        MERGE             BRANCH
CK80      EQU       $
          TBM       0,IOQ.CONT,X2     WAIT I/O COMPLETE?
          BS        NOWAIT            NO, BRANCH
          BL        S.EXEC3           REPORT EVENT - WAIT I/O COMPLETE
          BU        MERGE             BRANCH
NOWAIT    EQU       $
          BL        S.EXEC4           REPORT EVENT - NO WAIT I/O CMPLT
```

```
MERGE    EQU      $
         LA       R2,LP00.1          FETCH EP1 ADDRESS
CALLEXEC EQU      $
         LD       R6,ICB.OLD,X2      FETCH OLD PSD (EP1 OR EP3)
         LA       R2,STACK           FETCH ADDR OF REG SAVE BLOCK
         M.TRAC   8,21               FOR DEBUG
         IFT      C.TRACF,102EC
         TBM      8,C.TRACE
         BCT      1,$+2W
         IFT      8 .EQ. 7,102ED
         SVC      X'B',X'21'
         GOTO     102EC
102ED        IFT      8 .EQ. 8,102EE
         SVC      X'C',X'21'
         GOTO     102EC
102EE        SVC      X'A',8
102EC        ANOP
         ENDM
         LPSD     RESTRMAP           RESTORE CURRENT MAP
*
         BOUND    1D
RESTRMAP GEN      12/X'800',20/W(DOBEI)  NEW PSD
         DATAW    X'80010000'        B33 = 8K GRAN, B47 = KEEP CURR MAP
*
DOBEI    BEI                         BLOCK EXTERNAL INTERRUPTS

DAI1     DATAW    0                  DAI BUILT BY EP8 & STORED HERE
         BL       S.EXECS            EXIT TO S.EXEC, NO RETURN


EXIT2    LF       R0,STACK           RESTORE REGISTERS
         TRSW     R0                 RETURN

*
*
*
*
QEMPT    EQU      $    -

         LW       R7,SPINT           GET SPURIOUS INTERRUPT ADDRESS
         STW      R7,*PA15           STORE IN SI DED. LOC.
         LW       R2,COMQEADR        FETCH COMPLETING QUEUE ADDRESS
         LW       R2,IOQ.UDTA,X2     FETCH UDT ADDRESS
         ZMW      UDT.PTOV,X2        ZERO TIME OUT VALUE
         BU       EXIT
*
*
*
*    QUEUE ENTRY ADDRESS IN R2
DODUB    EQU      $
         LH       R1,IOQ.FCT1+1H,X2     GET QUANTITY
         BCT      LE,NORMCD          BRANCH IF NO DATA TO PRINT
         LW       R7,IOQ.FCT2,X2     FETCH TCW
         STW      R7,*PA14           STORE INTO TI DEDICATED LOCATION
         LB       R7,C.MACH          FETCH CPU TYPE
         CI       R7,2               32/7X7
         BLT       SET.LTIM        IF SO ... SET LOST INTRPT T/O
         LA        R7,TCW          FETCH ADDR OF DEDICATED TCW LOCATION
         STW       R7,*IOCDA       UPDATE IOCB WITH TCW ADDRESS
SET.LTIM LW        R2,IOQ.UDTA,X2  FETCH UDT ADDRESS
         LW        R7,PA6          FETCH LOST INT TIMEOUT VALUE
         STW       R7,UDT.PTOV,X2  STORE TIME OUT INTO UDT
         EXM       SPA5            CD UPSPACE AND PRINT
         BU        EXIT


HLPTOS   EQU       $
         LW        R2,PA1          GET CDT ADDRESS
         LA        R3,PA2          ADDRESS OF TO BLOCK
         ZBM       CDTERMFL,FLAGS  CLEAR CD TERMINATE FLAG
         BCF       SET,NOCDTERM    NOT SET ... PROCEED NORMALLY
         LW        R4,TERMSTAT     BITS 1,10,16
         STW       R4,IOQ.IOST,X1  STORE STATUS INTO IOQ WRD 14
         BU        HLPERR          CONTINUE PROCESSING
NOCDTERM EQU       $
         BL        S.IOCS2         PERFORM DEVICE TESTING
         LW        R1,ACTIVE       GET QUEUE ENTRY ADDRESS
         LW        R4,MASK38       STATUS MASK
         LMW       R6,IOQ.IOST,X1  GET STATUS, ANY ERRORS?
         BCT       ZR,GOON         NO, BRANCH
         TBR       R6,10           YES, ANY OCC ERRORS?

         BCT       SET,HLPERR      YES, BRANCH
         TBR       R6,18           NO, IS DEVICE INOP

         BCF       SET,NOTINOP     NO, BRANCH
         SBM       4,IOQ.IOST,X1   YES, SET GENERAL STATUS FLAG
         BU        HLPERR
NOTINOP  TBM       25,IOQ.IOST,X1  BOF STATUS??
         BCF       SET,HLPERR      NO, BRANCH
         ZBM       1,IOQ.IOST,X1   RESET ERROR CONDITION FOUND BIT
         BU        GOON


HLPERR   EQU       $
         LW        R2,ACTIVE       QUEUE ENTRY ADDRESS
         SBM       6,IOQ.FLGS,X2   SET ERROR FOUND FLAG IN IOQ
         TBM       1,IOQ.CONT,X2   ERROR PROCESSING INHIBITED?
         BCT       SET,NOBOM       YES, BRANCH
         TBM       19,IOQ.FLGS,X2  CHECK FOR NO-WAIT I/O
         BCT       SET,NOBOM       YES, GO UNLINK IOQ
         ZBM       0,IOQ.STAT,X2   CLEAR 'IN PROGRESS' INDICATOR IN QUEU
         SBM       POSTPROC,FLAGS  INDICATE POST PROCESSING EXIT REQUIRE
         BU        QEMPT
```

```
************************************************************************
*                                                                      *
*       LINE PRINTER HANDLER                                           *
*            ENTRY POINT 2                                             *
*            QUEUE START INTERRUPT SERVICE ROUTINE                     *
*                                                                      *
************************************************************************
LP00.2   STF        R0,STACK           SAVE REGISTERS
         SBM        ENTRYPT,FLAGS      SET QUEUE START FLAG
         LW         R3,PA1             GET CDT ADDRESS
         LW         R6,HAT+1W          GET QUEUE DRIVE ADDRESS
         STW        R6,*PA15           STORE IN SI DED. LOC.
         BU         MGEP2
```

```
*************************************************************************
*                                                                       *
*        LINE PRINTER HANDLER                                           *
*             ENTRY POINT 3                                             *
*            SPURIOUS INTERRUPT  SERVICE ROUTINE                        *
*                                                                       *
*************************************************************************
         BOUND     1D
LP00.3   REZ       2W                OLDPSD SAVE AREA

         GEN       12/X'800',20/W(EP3)  EP 3 EXECUTABLE CODE ADDR
         DATAW     X'00010000'       UNMAPPED, B47 = KEEP CURR MAP
*                                    B48, B49 = 0 - UNBLK INT & LEAVE ACT
         REZ       2W

EP3      ABM       31,C.GINT         ADD 1 TO GLOBOL INTERRUPT COUNT
         STF       R0,STACK          SAVE REGISTERS
         M.TRAC    7,21              FOR DEBUG
         IFT       C.TRACF,102EF
         TBM       7,C.TRACE
         BCT       1,$+2W
         IFT       7 .EQ. 7,102F0
         SVC       X'B',X'21'
         GOTO      102EF
102F0        IFT       7 .EQ. 8,102F1
         SVC       X'C',X'21'
         GOTO      102EF
102F1        SVC       X'A',7
102EF        ANOP
         ENDM
         ABM       31,PA12           INCREMENT SPURIOUS INTERRUPT COUNTER
         LA        R2,LP00.3         FETCH EP3 ADDRESS
         BU        CALLEXEC          BRANCH
```

```
***************************************************************
*                                                             *
*       LINE PRINTER HANDLER                                  *
*            ENTRY POINT 4                                    *
*            LOST INTERRUPT PROCESSING                        *
*                                                             *
***************************************************************
LP00.4    EQU       $
          ABM       31,PA13           INCREMENT LOST INTERRUPT COUNTER
          EXM       $PA12             ISSUE CD TERMINATE
          SBM       CDTERMFL,FLAGS          SET CD TERM. FLAG ISSUED BIT
          TRSW      R0                RETURN
```

```
*************************************************************************
*                                                                       *
*        LINE PRINTER HANDLER                                           *
*             ENTRY POINT 5                                             *
*             OPCODE PROCESSING REENTRANT SERVICE                       *
*          ENTER:  R1=FCB ADDRESS                                       *
*                  R2=FAT ADDRESS                                       *
*                                                                       *
*************************************************************************
LP00.5   EQU      $

         LI       R4,X'1'           FUNCTION CODE FOR BFT
         EXM      PA2               EXECUTE 'TD8000'
         BFT      DEVINOP           BRA ON ALL CC'S SET
         ZR       R5                CLEAR FUNCTION WD1

         LB       R3,FCB.OPCD,X1    GET OP CODE FROM FCB
         SLL      R3,2

         BU       *OTAB,X3          VECTOR TO OPCODE PROCESSOR


* HANDLER OPCODE VECTOR TABLE
         BOUND    1W
OTAB     EQU      $
         ACH      OPEN
         ACH      RWND              1
         ACH      READ              2
         ACH      WRIT              3
         ACH      WEOF              4
         ACH      UNUSED            5
         ACH      ADVR              6
         ACH      ADVF              7
         ACH      BKSR              8
         ACH      BKSF              9
         ACH      UPSP              10
         ACH      ERPT              11
         ACH      EJCT              12
         ACH      CLSE
         ACH      UNUSED            14
         ACH      UNUSED            15


RWND     EQU      $
         SBR      R5,1              SET BOM FLAG IN HANDLER FUNCTION WD.
EJCT     EQU      $

         LW       R3,PA1            GET CUT ADDR
         BL       S.IOCS13          GET IOQ
         LW       R7,SPA6           CD EJECT
MBQR     BL       IOLINK            MUST BE QUEUED RETURN

READ     EQU      $
UNUSED   EQU      $
ADVR     EQU      $
ADVF     EQU      $
```

```
BKSR      EQU       $
BKSF      EQU       $
          BL        ILOPCODE          ILLEGAL OR UNSPECIFIED RETURN

OPEN      EQU       $
CLSE      EQU       $
WEOF      EQU       $
ERPT      EQU       $
RCR       BL        SERVCOMP          REQUEST SERVICING COMPLETE RETURN
          TRSW      R0
*
*
*
UPSP      EQU       $

          LW        R3,PA1            GET CDT ADDR
          BL        S.IOCS13          GET IOQ
          LW        R7,SPA4           CD SPACE
          BU        MBQR

WRIT      EQU       $
          LW        R3,PA1            GET CDT ADDR
          BL        S.IOCS13          GET IOQ
          LW        R2,FCB.IOQA,X1    FETCH IOQ ADDRESS FROM FCB
          LW        R2,IOQ.UDTA,X2    FETCH UDT ADDRESS FROM IOQ
          LW        R4,MASK2B        FETCH 2 BYTE MASK
          LMH       R7,UDT.MBX,X2     FETCH MAX TRANSFER COUNT FROM UDT
          LW        R6,FCB.TCW,X1     GET TCW FROM FCB
          M.CALL    H.IOCS,30         ADJUST AND CLAMP QUANTITY
          SVC       0,H.IOCS*256+30
          ENDM
          M.CALL    H.IOCS,28         MAKE TCW ABSOLUTE
          SVC       0,H.IOCS*256+28
          ENDM
          LW        R7,SPA5           CD SPACE AND PRINT (DEFAULT NO FC)
          TRR       R4,R5             SET QUANTITY IN HFW1

          TBM       2,FCB.GCFG,X1     DATA FORMATTING INHIBITED?
          BCT       SET,MBQR          YES
          LW        R2,FCB.IOQA,X1    GET IOQ ADDR
          LW        R3,IOQ.TBUF,X2    GET ADDR OF BUFFER IN USER'S SPACE
          SEA                         SET EXTENDED ADDRESSING

          LB        R4,0,R3           GET FC CHAR
          CEA                         CLEAR EXTENDED ADDRESSING

          SUI       R5,1              DECREMENT QUANTITY
          ANMW      R6,MASKADR        KEEP ADDRESS ONLY
          SLL       R5,20
          ORR       R5,R6             INSERT NEW QUANTITY
          ABR       R6,31             INCREMENT ADDR
          SRL       R5,20
          LI        R2,-5B            COUNTER
          CAMB      R4,HLPFCTBL+5B,R2
          BCT       EQ,FCFND          FC CHAR FOUND
          BIB       R2,$-2W           CONTINUE SEARCH
```

```
         LI       R2,-5B            ASSUME SPACE IF NOT FOUND
FCFND    TRN      R2,R2             CONVERT TO POSITIVE
         SLL      R2,2              CONVERT TO WORD DISPLACEMENT
         CI       R2,4W             IS THIS A DOUBLE SPACE AND PRINT?
         BCF      EQ,$+2W           NO
         SBR      R5,0              YES, SET DOUBLE FLAG

         LI       R4,X'FFF'
         TRRM     R5,R4             ANY DATA TO BE PRINTED

         BCF      ZR,HLPOUT         YES
         ADI      R2,6W             ADJUST POINTER FOR NO DATA
HLPOUT   LW       R7,HLPCDT,R2      GET APPROPRIATE CD INSTRUCTION
         BU       MBUR              MUST BE QUEUED RETURN
DEVINOP  LI       R4,X'48F0'        BUILD STATUS TO PLACE IN FCB
*                                   SET ERROR COND., DEVICE INOP AND
*                                   TEST STATUS BITS IN THE FCB.
         SLL      R4,16             MOVE TO FIRST HALF OF STATUS WRD

         STW      R4,FCB.SFLG,X1    PLACE IN FCB
         BU       SERVCOMP          SERVICE COMPLETE RETURN TO IOCS
         LPOOL                      DUMP LITERALS HERE
```

```
************************************************************************
*                                                                      *
*        LINE PRINTER HANDLER                                          *
*             ENTRY POINT 6                                            *
*             POST TRANSFER PROCESSING                                 *
*                                                                      *
************************************************************************
LP00.6   EQU      $
         TRSW     R0                NOT APPLICABLE - RETURN


************************************************************************
*                                                                      *
*        LINE PRINTER HANDLER                                          *
*             ENTRY POINT 7                                            *
*             ERROR PROCESSING                                         *
*                                                                      *
************************************************************************
LP00.7   EQU      $

         ADI      R0,1W             YES - OPER. INTERVENT. APPLIC.
         TRSW     R0
```

```
****************************************************************
*                                                              *
*       LINE PRINTER HANDLER                                   *
*            ENTRY POINT 8                                     *
*            HANDLER INITIALIZATION                            *
*                                                              *
****************************************************************
        M.INIT    LPO,,SPA1,SPA2,SPA3,SPA4,SPA5,SPA6,SPA7,+
                  SPA8,SPA9,SPA10,SPA11,SPA12
LP00.8    M.EIR
        ENDM
        STW       7,PA1
        LW        1,PA1
        LB        5,CDT.CHAN,1
        SRC       5,13

        STW       5,102F7
        M.INITX PA2,102F8
        LW        6,PA2
        ANMW      6,102F8
        ORR       5,6

        STW       6,PA2
        ENDM
        M.INITX PA3,102F8
        LW        6,PA3
        ANMW      6,102F8
        ORR       5,6

        STW       6,PA3
        ENDM
        M.INITX PA4,102F8
        LW        6,PA4
        ANMW      6,102F8
        ORR       5,6

        STW       6,PA4
        ENDM
        M.INITX PA5,102F8
        LW        6,PA5
        ANMW      6,102F8
        ORR       5,6

        STW       6,PA5
        ENDM
        LNW       5,PA6
        BCT       ZR,102F9
        MPMW      4,C.MTIM
        DVMW      4,C.NTIM
        BCF       6,102F9
        LI        5,1
102F9       TRN        5,5

        STW       5,PA6
        LB        5,CDT.IPL,1
        SRC       5,13
```

```
M.INITX PA7,102F8

LW          6,PA7
ANMW        6,102F8
ORR         5,6

STW         6,PA7
ENDM
M.INITX PA8,102F8
LW          6,PA8
ANMW        6,102F8
ORR         5,6

STW         6,PA8
ENDM
M.INITX PA9,102F8
LW          6,PA9
ANMW        6,102F8
ORR         5,6

STW         6,PA9
ENDM
LW          5,102F7
M.INITX PA11,102F8
LW          6,PA11
ANMW        6,102F8
ORR         5,6

STW         6,PA11
ENDM
LB          2,CDT.IPL,1
TRR         2,3
SLL         5,19
ORMW        5,102FA
STW         5,UAI1
LB          5,C.MACH
CI          5,2
BLT         102FB
LW          3,MAT+1W
ZBR         3,31

LA          4,5W,3
STW         4,PA14
STW         4,CDT.TIAD,1
LI          3,X'F1'
SLL         3,2

ADMW        3,C.SPAD
TRR         2,5
SLL         5,2
ADMW        5,0,3
SUI         2,4
SLL         2,3

ADI         2,4
ADMW        2,1W,3
```

```
          STW       4,0,2
          LW        3,HAT+1W
          ZBR       3,S1

          STW       2,4W,3
          BU        102FC
102FB         ADI        2,X'80'
          SLL       2,2

          ADMW      2,C.SPAD
          LH        5,1H,2
          SUI       5,X'40'
          STW       5,PA14
          STW       5,CDT.TIAD,1
          LI        7,X'40'
          LB        6,CDT.IPL,1
          CI        6,X'23'
          BCT       6,102FD
          SLLD      6,2
          TRR       6,7
102FD         ADR        7,5

102FC         STW        5,PA15
          LA        6,HAT
          STW       6,CDT.SIHA,1
          LW        6,HAT+3W
          STW       6,*PA15
          IFP       102F2,102FE
          LW        6,102FF
          STW       6,PA4
102FE         ANOP
          LW        5,102F7
          IFP       SPA1,10300
          M.INITX SPA1,102F8
          LW        6,SPA1
          ANMW      6,102F8
          ORR       5,6

          STW       6,SPA1
          ENDM
10300         ANOP
          IFP       SPA2,10301
          M.INITX SPA2,102F8
          LW        6,SPA2
          ANMW      6,102F8
          ORR       5,6

          STW       6,SPA2
          ENDM
10301         ANOP
          IFP       SPA3,10302
          M.INITX SPA3,102F8
          LW        6,SPA3
          ANMW      6,102F8
          ORR       5,6
          STW        6,SPA3
          ENDM
10302          ANOP
          IFT        G'LPO' .EQ. G'CRU',10303
          LW         4,PA14
          STW        4,SPA4
          GOTO       10304
10303          ANOP
          IFT        G'LPU' .EQ. G'CPU',10305
          LW         4,PA14
          STW        4,SPA4
          GOTO       10306
10305          ANOP
          IFP        SPA4,10304
          M.INITX SPA4,102F8
          LW         6,SPA4
          ANMW       6,102F8
          ORR        5,6

          STW        6,SPA4
          ENDM
10304          ANOP
          IFP        SPA5,10307
          M.INITX SPA5,102F8
          LW         6,SPA5
          ANMW       6,102F8
          ORR        5,6

          STW        6,SPA5
          ENDM
10307          ANOP
          IFP        SPA6,10308
          M.INITX SPA6,102F8
          LW         6,SPA6
          ANMW       6,102F8
          ORR        5,6

          STW        6,SPA6
          ENDM
10308          ANOP
          IFP        SPA7,10309
          M.INITX SPA7,102F8
          LW         6,SPA7
          ANMW       6,102F8
          ORR        5,6

          STW        6,SPA7
          ENDM
10309          ANOP
          IFP        SPA8,1030A
          M.INITX SPA8,102F8
          LW         6,SPA8
          ANMW       6,102F8
          ORR        5,6

          STW        6,SPA8
```

```
         ENDM
1030A        ANOP
         IFP        SPA9,1030B
         M.INITX SPA9,102F8
         LW         6,SPA9
         ANMW       6,102F8
         ORR        5,6

         STW        6,SPA9
         ENDM
1030B        ANOP
         IFP        SPA10,1030C
         M.INITX SPA10,102F8
         LW         6,SPA10
         ANMW       6,102F8
         ORR        5,6

         STW        6,SPA10
         ENDM
1030C        ANOP
         IFP        SPA11,1030D
         M.INITX SPA11,102F8
         LW         6,SPA11
         ANMW       6,102F8
         ORR        5,6

         STW        6,SPA11
         ENDM
1030D        ANOP
         IFP        SPA12,1030E
         M.INITX SPA12,102F8
         LW         6,SPA12
         ANMW       6,102F8
         ORR        5,6

         STW        6,SPA12
         ENDM
1030E        ANOP
         IFP        102F4,1030F
         M.INITX    102F4,102F8
1030F        ANOP
         IFP        102F5,10310
         M.INITX    102F5,102F8
10310        ANOP
         IFP        102F6,10306
         M.INITX    102F6,102F8
10306        ANOP
         BU         10311
102F7        DATAW      0
102F8        DATAW      X'FC07FFFF'
102FF        DATAW      X'00020002'
102FA        DATAW      X'FC040000'
10311        EQU        *
         ENDM
         M.XIR      HAT
         ENDM
         END
```

13. GENERAL PURPOSE MULTIPLEXER (GPMC) SUPPORT

13.1 Overview

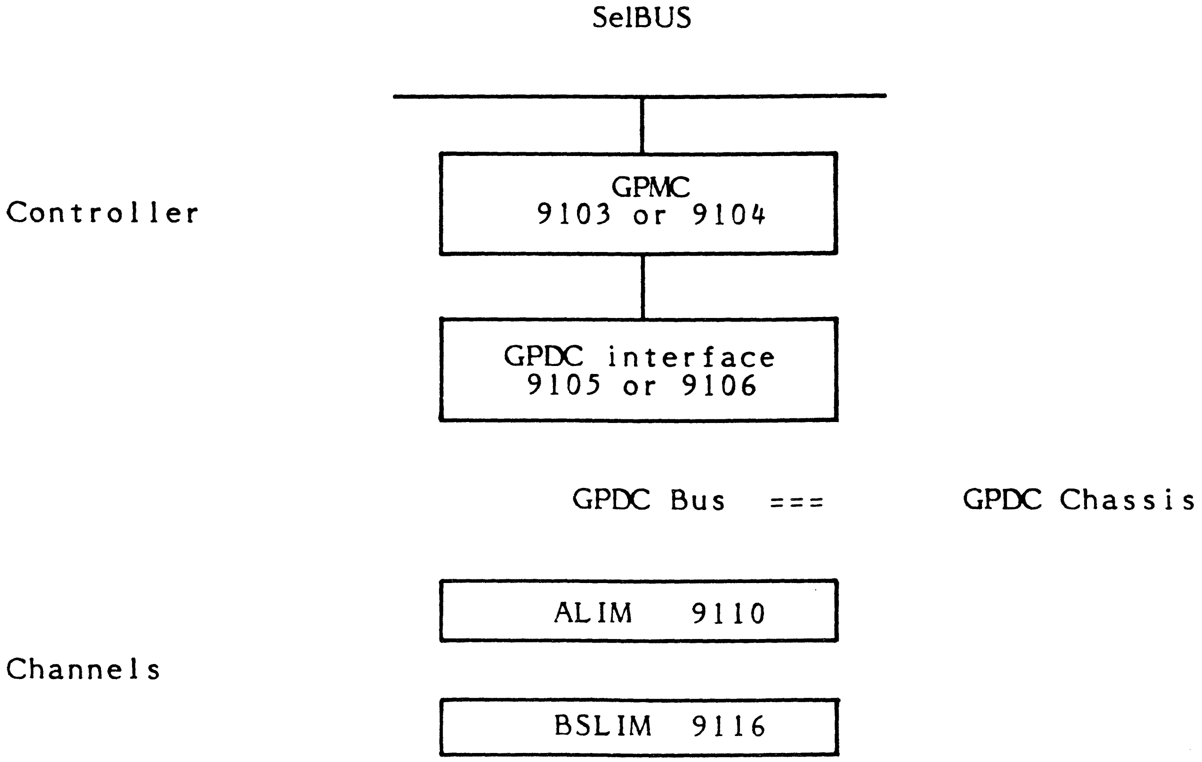
The General Purpose Multiplexer Controller (GPMC) is structured as follows:

- All General Purpose Device Controller (GPDC) handlers are system reentrant, therefore only one copy of a handler is needed per system.
- All handlers and the interrupt executive use common logic where reasonable, reducing duplication of logic. In addition, as much GPMC specific logic as possible is placed within this common logic, removing it from IOCS. If a GPMC is configured, this group of routines is loaded only once.
- SYSGEN creates only one Controller Definition Table (CDT) entry per GPMC (regardless of the number of device addresses) and no longer creates a handler entry point jump table.
- All I/O requests are queued from the Unit Definition Table (UDT) entry.

Current devices supported are:

9103	Extended GPMC, Class D (16MB)
9104	GPMC, Class E (1/2 MB) (32/7X only)
9109	Synchronous Line Interface Module (SLIM)
9110	Asynchronous Line Interface Model (ALIM)
9116	Binary Synchronous Line Interface Module (BLIM)
	Card Reader and/or Punch
	Paper Tape

13.2 Hardware Structure



Refer to the GPMC Technical Reference Manual, 325-329104/9103, for details of the GPMC operation. Refer to the appropriate technical manual part number 325-32XXXX where XXXX is the model number, for details of a particular GPDC channel's operation.

13.3 Software Structure

13.3.1 Input/Output Control System (IOCS)

The GPMC enqueues IOQ entries onto the UDTs. Code previously contained within IOCS that is solely required for GPMC operation is in a separate subroutine module which is loaded only if a GPMC is configured.

13.3.2 GPMC Interrupt Executive (H.MUX0)

The GPMC Interrupt Executive, H.MUX0, is entered every time an interrupt occurs at the level for which the GPMC is configured. H.MUX0 queries the GPMC to find which channel caused the interrupt, and then vectors to the handler's Service or Spurious Interrupt entries via the contents of IB.EP in the device Interrupt Block, which is built by SYSGEN and filled by the handler. Since H.MUX0 is entered only on an interrupt, it contains one entry point which comprises both. IOCS goes directly to the device handler by using the contents of UDT.SIHA.

The handler entry point is found by: 1) locating the device UDT through the list CDT.UTn, 2) locating the device context or "interrupt" block whose address is saved in UDT.CBLK, and 3) branching through IB.EP in the interrupt block.

13.3.3 GPDC Device Handlers (H.??MP)

These handlers contain all eight standard handler entry points, and perform the actual IOCS handler functions. However, their order in the handler address table has been rearranged. They are not entered directly when an interrupt occurs.

The SYSGEN initialization entry point determines how many pre-built device interrupt blocks are needed (maximum is 64) and instructs SYSGEN to overlay the remainder of the blocks with the next handler.

These handlers are system re-entrant and only one copy is needed per MPX-32 system when controller(s) are present. In addition, different handlers use common logic within the module "GPMC.SUB" which is loaded only when GPMC(s) are configured. This reduces their aggregate size.

13.3.4 Normal I/O Logic

When a normal IOCS request for input or output (such as read or write) occurs, the opcode entry processes it, returning appropriately if it is either illegal or a logical "NOP". If an actual I/O operation is needed, the entry point builds an IOQ.

The IOQ is returned to memory pool, either due to I/O completion or a task delete.

In the case of a Model 9104 GPMC which can only reference physical E memory, an OS buffer may also be obtained, and pointed to by IOQ.FBUF. This is performed automatically by IOCS, and it returns the buffer when the IOQ is released.

13.3.5 Execute Channel Program

The user has the ability to perform direct channel I/O using physical addresses from a privileged task. The I/O transfer address points to an I/O Command List (IOCL).

13.4 Data Structures

13.4.1 System Blocks

The following communication region variables are special to GPMC support.

13.4.1.1 Controller Definition Table (CDT)

One CDT is generated for each GPMC, and the CDT fields added for the IOP support are utilized.

CDT Variables Used:

CDT.CLAS — Hexadecimal 0E for Model 9104, hexadecimal 0D for Model 9103.
CDT.FLGS — Bit 2 (CDT.GPMC) is set to indicate a GPMC controller.
CDT.IOST — Bit 0 (CDT.NIOQ) is set for the new structure.
Bit 1 (CDT.MUX) is set to indicate a multiplexing controller.
CDT.SIHA — Points to the H.MUX0 HAT.
CDT.UTn — Where n is a hex digit from 0 to F a table of sixteen entries, each of which corresponds to a channel. If a device is configured, the corresponding entry contains the address of the UDT, or zero if no UDT corresponds.

13.4.1.2 Unit Definition Table (UDT)

One UDT is built by SYSGEN for each configured address and is flagged as having IOQ entries queued from it.

UDT variables used:

UDT.CBLK — points to the interrupt block which corresponds to this device.
UDT.SIHA — points to the appropriate device handler HAT.
UDT.STA2 — bit 0 is set to indicate IOQs are strung from UDT.
UDT.TIAD — is filled with the TI location address (for debug purposes only).
UDT head cell — IOQ entries are queued here.

13.4.2 Architecture

13.4.2.1 Overview

Assume that you have two GPMCs. Each has 3 ALIMs; in addition, one has two BSLIMs and the other has a card punch and a paper tape. You would have the following in memory:

```
-------------------------
!    Subroutines        !
!    common logic       !   Only resident if a GPMC present
-------------------------

GPMC 1:                            GPMC 2:

-------------------------   --------------------------
!    H.MUX0             !   !         H.MUX0         !
-------------------------   --------------------------

-------------------------   --------------------------
!    CDT                !   !          CDT           !
-------------------------   --------------------------
     One per GPMC.

-------------------------
!    UDT                !   28, one per device.
-------------------------

-------------------------   --------------------------
!    H.ASMP             !   !         H.BSMP         !
-------------------------   --------------------------

-------------------------   --------------------------
!    H.CPMP             !   !         H.PTMP         !
-------------------------   --------------------------

-------------------------
!    Interrupt Block    !   28, one per device.
-------------------------
```

13.4.2.2 Interrupt Block

```
                                 Decimal
                                 Offset
+----------------------------+      0 W    current IOQ entry address
!    IB.QEADR                !
+----------------------------+      1 W    address of IOQ entry just
!    IB.CMPQE           *    !             completed
+----------------------------+      2 W    prototype for device CDs
!    IB.CDTD            *    !             & TDs
+----------------------------+      3 W    GPMC status/residual byte
!   H.AUT    H.AUTCNT   *    !             count
+----------------------------+      4 W    address of UDT location
!    IB.DQEAD                !             which points to DQE of
!                            !             task allocating
+----------------------------+      5 W    address of CDT
!    IB.CDTA            *    !
+----------------------------+      6 W    address of UDT
!    IB.UDTA            *    !
+----------------------------+      7 W
!    IB.   IB.   IB.   IB.*  !
!    LICNT SPCNT OPKODE DHBF !
!                            !
!    (See Note 1)            !
!                            !             Bit 6
!                            !
!                            !             Bit 7
!                            !
+----------------------------+      8 W    handler entry
!    IB.EP              *    !             address to
                                           use (queue drive or
!                            !             spurious interrupt)
+----------------------------+      9 W    return address from
!    IB.EXIT                 !             handler entry
+----------------------------+     10 W    time out value
!    IB.LITIM           *    !             (in timer units)
+----------------------------+     11 W    I/O transfer count
!    H.CNT                   !             in bytes
+----------------------------+     12 W    I/O buffer address
!    H.BUF                   !
+----------------------------+     13 W    negative of transfer
!    H.NCT                   !             count remaining
+----------------------------+     14 W    1st doubleword of GPMC
!    IB.IOCL                 !             IOCL
+----------------------------+     16 W
!    OPTIONAL                !
!    DEVICE DEPENDENT        !             handler dependent
!    INFORMATION             !
+----------------------------+
```

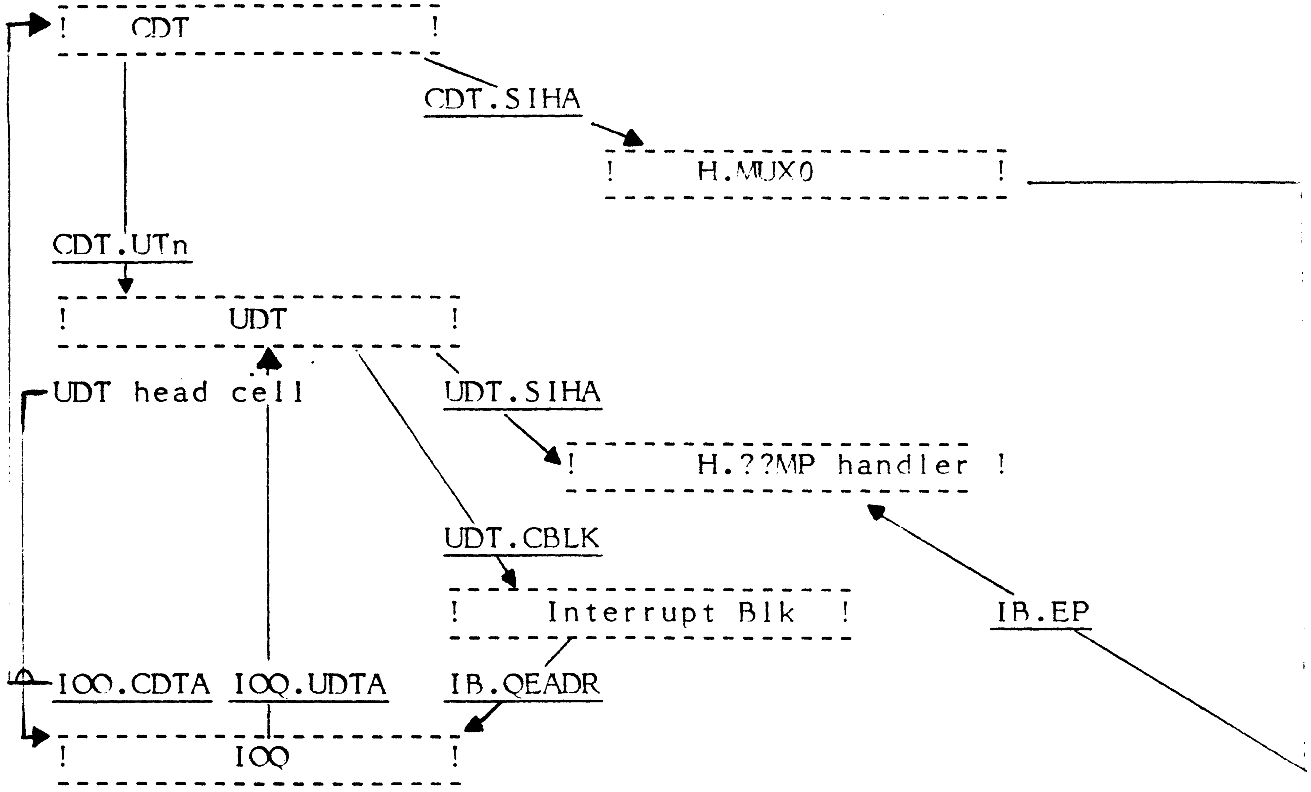
* referenced by GPMC/SUB or H.MUXO

Note 1

LICNT	Lost Interrupt Count
SPCNT	Spurious Interrupt Count
OPKODE	IOCS Byte Operation Code
DHBF	Device Handler Bit Flags as follows:

Bit 6	Postprocessing needed (tells H.MUX0 to report I/O completion)
Bit 7	CD terminate issued by lost interrupt entry

13.4.2.3 Data Block Linkage



Linkage Values

CDT.SIHA points to HAT of H.MUX0.

CDT.UTn where "n" is the device number (0 to F). Points to UDT of corresponding device.

UDT.SIHA points to HAT of appropriate LIM handler (H.??MP).

UDT.CBLK points to Interrupt Block which corresponds to this device.

IB.EP points to either Entry Point 1 or Entry Point 3 in corresponding device handler, depending on whether there is I/O in progress.

IB.QEADR points to the IOQ Entry which is currently being performed.

IOO.CDTA CDT address.

IOQ.UDTA UDT address.

DQE pointers (not shown)

UDT.DQEA points to DQE of task if device not shared.

IB.DQEA points to UDT.DQEA.

13.5 Handler Entry Points

13.5.1 H.MUX0

E.P.

1 Entered on interrupt. Registers saved in register file in GPMC Context Block, and restored on exit to S.EXEC5.

2 SYSGEN initialization. Entered with CDT address in register 7.

13.5.2 H.??MP

Entry points have designated names.

E.P.

SI. Entered by H.MUX0, which finds this address in the interrupt block variable IB.EP. Register 3 points to the UDT. The Interrupt Block is found thru UDT.CBLK.

IQ. Entered by H.IOCS,29, who finds it via UDT.SIHA. Register 3 points to the UDT. The Interrupt Block is found thru UDT.CBLK. Returns through E.P. 1 logic.

SP. Post Transfer processing - returns thru R0.

LI. Entered by S.IOCS5 or H.IOCS,38. Register 2 is the HAT Address, Register 3 is the UDT address.

OP. Entered by H.MUXO if GPMC has an interrupt for this type of device. H.MUXO finds this address thru the interrupt block variable IB.EP. Register 3 points to the UDT. The Interrupt block is found thru UDT.CBLK. Returns to H.MUXO through H.EXIT return address in Interrupt Block.

PX. Entered by H.IOCS,29. Register 1 is the FCB address, register 2 is the HAT address (unused), and register 3 is the UDT address. The Interrupt block is found thru UDT.CBLK.

OI. Error Processing - returns thru R0.

SG. SYSGEN initialization logic. Register 7 points to the CDT.

13.6 Common Logic

The following subroutines are contained in the module GPMC.SUB, which is loaded by SYSGEN if any GPMCs are configured. They permit the GPDC device handlers to be smaller.

13.6.1 Subroutine S.GPMC0 - Report GPMC Status

Entry

R0	Return address
R2	IOQ address
R3	Interrupt Block Address

Exit

R0-R4	Unchanged
R5,R6	Destroyed
R7	Device Status in Right Halfword

13.6.2 Subroutine S.GPMC1 - I/O Initiation Logic

Entry

R2	IOQ Address
R3	Interrupt Block Address
R6	IOCL address

13.6.3 Subroutine S.GPMC2 - Lost Interrupt Logic

Entry

R0	IOCS return address
R2	Handler Address (unused)
R3	UDT address

Exit - Direct to IOCS

R0-R3	Unchanged
R4-R7	Destroyed

13.6.4 Subroutine S.GPMC3 - Operation Initiation and IOQ Entry Acquisition

If the opcode vector table entry bit 0 is set, an IOQ is built for the user. If bit 1 in the word is set, the IOQ is extended by enough space to hold the absolutized IOCL necessary to perform the requested IO.

Entry

R0	H.??MP return address (not used)
R1	FCB address
R2	Opcode vector table address
R3	Interrupt Block address

Exit - via vector table

R0,R1,R3	unchanged
Other Registers	Indeterminate

13.6.5 Subroutine S.GPMC4 - Execute Channel Program Inspection and Absolutizing

New IOCL is produced in a memory pool buffer whose address is placed in IOQ.FBUF, and whose size is placed in IOQ.WOSB. The User's IOCL address is placed in IOQ.TBUF (not yet implemented).

Entry

R0	E.P. 5 return address
R1	FCB address

13.7 GPMC Support Macros

13.7.1 IB.DAT1,IB.DAT2

Functional Description

This macro defines the standard information for a Device Interrupt Block. Special handler information should be inserted between IB.DAT1 and IB.DAT2. The latter macro closes the Interrupt Block Definition and computes its size. This macro must start on a doubleword boundary.

For use by unique handler logic, the SET label H.IOCL always points to the physical address of IB.IOCL in the current Interrupt Block.

Calling Sequence:

```
 IB.DAT1
       :  (handler specific information)
       :
 IB.DAT2
```

Use the REPT directive to get multiple copies of the interrupt block.

13.7.2 M.IB

Functional Description

This macro establishes the Interrupt Block offset labels. It maps to the contents of IB.DAT1. Special handler information may be equated starting at IB.DFSIZ, which is doubleword bounded.

Calling Sequences:

```
 M.IB
```

13.7.3 GPDC.IT

Functional Description

This macro generates the SYSGEN Initialization logic for a GPMC handler.

Calling Sequence:

```
GPDC.IT          lab[,timout]
```

where:

lab	starting label, SG.lab
timout	is a positive number indicating the number of seconds for device timeout. If not provided, a word variable "PA6" should contain the negative timeout count.

13.7.4 M.DIB

Functional Description

This macro is called by GPDC.IT to initialize the Device Interrupt Block. This macro contains special case code for the ALIM (Model 9110) handler.

Calling Sequence:

```
M.DIB   type
```

where:

type	is the information passed to GPDC.IT as "lab"

13.8 User Handlers

Customers writing their own GPMC/GPDC handlers need not use the common logic of GPMC.SUB, although in most cases it can simplify design and development. Customer written handlers do not need to be reentrant, however, if they are not, interrupt reentrancy must be specified in the SYSGEN directives.

Customer written handlers must use the Interrupt Block arrangement because H.MUX0 and GPMC.SUB reference certain locations.

14. DISC PROCESSOR HANDLER

14.1 Overview

The Extended I/O Disc Handler is a software component of MPX-32 intended to provide support for Extended I/O Disc Processors connected to an MPX-based SERIES 32 computer.

The product is designed to support any number and mix of extended I/O disc drives listed below. These include fixed-head discs (FHD), moving-head discs (MHD), and cartridge module drives (CMD).

The design supports IOCS callable I/O service requests as described in the MPX-32 Reference Manual, Volume 1.

An execute channel program capability has been incorporated to allow the priviledged user to execute his own IOCD list. Care should be exercised when using this feature as only minimal support has been included. No validation of commands or addresses is performed. Tasks should be cataloged as resident (see MPX-32 Reference Manual, Volume 1, Chapter 2) and all IOCD addresses made absolute. Basic error conditions are detected and noted in the FCB, however, sense information generation, error correction, and error retry are the responsibility of the user. It should be noted that sense information relates to the last I/O request to the disc which may or may not be that of the requesting user. Reserve and Release IOCDs must never be included within an execute channel program IOCD list (see Sections 14.2.1.3.16 and 14.2.1.3.17).

14.1.1 Discs Supported

The following discs are supported by the extended I/O disc handler.

Compatible Disc Drives

						SYSGEN
					Byte	Device
Manufacturer	ID #	Type	Heads	Cylinders	Capacity	Code
CDC	9320	MHD	5	823	82,958,400	MH080
CDC	9323	MHD	19	823	315,241,920	MH300
CDC	9733-5	FHD	4	64	5,160,960	FH005
CDC	9448-32	CMD	1+1	823	33,183,360	CD032

CDC - Control Data Corporation
MHD - Removable media, moving-head disc
FHD - Captive media, moving-head disc
CMD - Cartridge module drive, removable and captive media

Moving head discs (MHD) are available in both single and dual-port versions.

The 5 megabyte Fixed Head Disc (FHD) is by definition a fixed head device with 256 fixed heads. However, software must treat this unit as a moving head disc with 64 cylinders and 4 heads. The fixed head disc may be single or dual ported.

A Cartridge Module Drive (CMD) is two devices in one package. The first is the removable media (MHD) and the second is the captive media (FHD). MPX-32 software dedicates the even subchannel to the removable media and the odd subchannel to the captive media. Cartridge module drives will only operate in the single-port mode.

14.1.2 Track Format

The disc processor and disc handler support two track formats. One format, designated F16, provides 16 data sectors where each data sector contains storage for 1024 data bytes. The second format, designated F20, provides 20 data sectors where each data sector contains 768 data bytes. While the disc processor and disc handler are capable of supporting both track formats, only the F20 format is supported within the MPX-32 operating system.

14.1.3 Dual Subchannel I/O

The disc processor firmware allows two communication paths to each device. These paths are called subchannels and occur in sequential even and odd pairs. This is to say that a device with a unit address plug of 1 (see Section 14.6) has software subaddress assignments of 02 and 03.

Under MPX-32, dual subchannel I/O is applicable only to cartridge module drives where the even subchannel is dedicated to the removable media and the odd subchannel is dedicated to the captive media. For SYSGEN purposes, this device must be assigned an even subaddress on the DEVICE directive.

For devices other than cartridge module drives, the odd subchannel address is unusable and should never be assigned on the SYSGEN DEVICE directive.

14.1.4 Dual Port Support

14.1.4.1 Normal Support

Dual porting allows two CPUs to share a single disc drive. In order to maintain disc and system integrity, mechanisms must exist to prevent both CPUs from accessing the device at the same time. This is accomplished through device reservation which makes the device unaccessable to the nonreserving CPU. Device reservation may be implicit or explicit.

Implicit device reservation is a disc processor function and is transparent to the operating system. If a drive is defined as dual ported, the disc processor automatically issues a RESERVE command to the device before initiating an I/O request. Once the I/O is complete, the disc processor issues a RELEASE command. An I/O request from the opposing CPU will be postponed from the time the RESERVE is issued until the RELEASE is performed.

Explicit device reservation makes a device unaccessable to the opposing CPU for a user requested period of time. The explicit device reservation is user invoked through the M.RESP service request. The device remains unavailable to the opposing CPU until the user releases the device through the M.RELP service request. When performing explicit device reservation, the release timer switch located on the disc drive must be set to the off position to disable the drive from performing its own release. (This is a drive performed release which is different from the implicit disc processor release mentioned above. Also, the channel 1 and channel 2 inhibit switches located on the disc drive must be in the off position. Should more than one user on the same CPU have a device explicitly reserved at the same time, the drive will not be released until the last such user explicitly releases it.

14.1.4.2 System Failure in Dual-Port Environment

In a dual-port environment, there exists the possibility that one of the systems may fail while having the shared disc reserved. Should this happen, the shared disc would be unaccessable to the opposing CPU because the failing system (CPU or disc processor) would have no way of releasing the drive. A mechanism is available to preclude this from happening.

When an explicit reserve (M.RESP) is issued to a device that is already reserved by the opposing CPU, a SYSGEN defineable time out value (DPTOV) begins to count down. Should this timer expire before the device becomes available, a priority override (POR) command is issued to gain absolute control over the device. A drive that is selected using the POR command remains reserved to the channel gaining control over the drive until such time that the channel releases the drive.

14.1.5 Maximum Byte Transfer and IOCD Generation

The MPX-32 services available for user Read and Write requests allow for a maximum transfer of 1,000,000 hexadecimal bytes (1,048,576 decimal bytes) per request. Requests larger than this are truncated to this amount.

H.IOCS,40 processes Read and Write requests by building data chained IOCDs as necessary to span map blocks. The number of IOCDs generated for any transfer request depends on how large the transfer is and where, within the MAP block, the buffer begins.

14.1.6 Hardware/Software Relationship

The dual-ported XIO disc hardware/software relationship is presented in Figure 14-1. The system consists of disc drives (1.0-n.7) which can be either single or dual port, disc processors (1-n), and the disc handler.

The disc handler consists of three parts. H.EXIO is the interrupt fielder and corresponds one for one with the number of disc processors configured. H.DP02 is a system reentrant handler that processes I/O requests. Context Blocks are areas of storage and record keeping and correspond one for one with the number of active subchannels configured. They are physically located at the end of H.DP02.

Up to eight (8) disc drives can be connected to any disc processor. The number of disc processors configured per system is limited by the number of channels and cabinet space available.

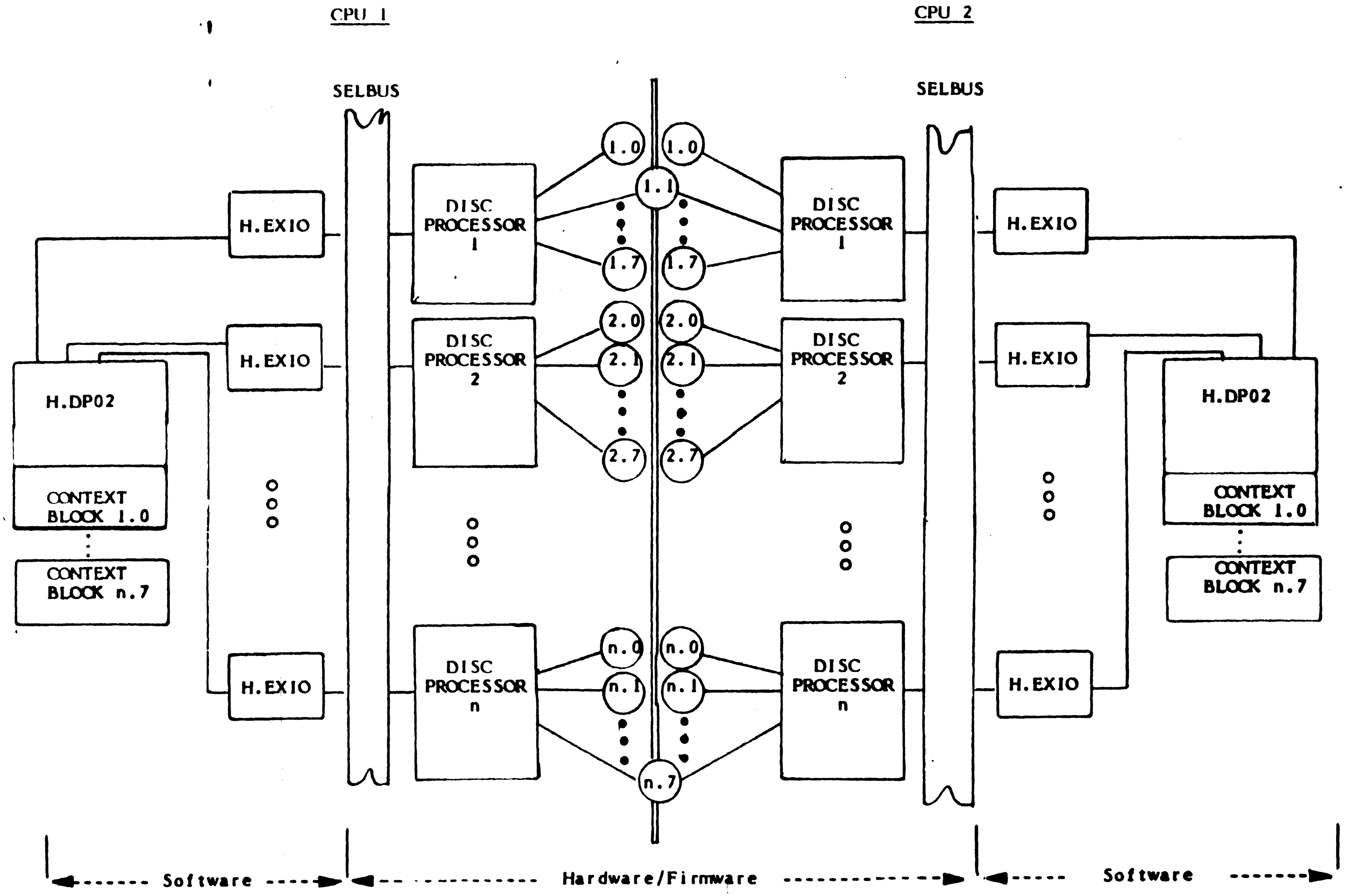


Figure 14-1
Disc Processor Hardware/Software Relationship

14.2 Extended I/O Commands and CPU Instructions

The Extended I/O (XIO) philosophy provides channel commands and CPU instructions for accomplishing I/O requests. The following is a summary of these features and their use by software.

14.2.1 XIO Channel Commands

14.2.1.1 Command Summary

The following is a summary of the XIO channel commands.

		MPX	Used by
Channel Command	Command Code	Service Call	MPX Software
Initiate Channel (INCH)	X'00'	None	Yes
Sense (SENSE)	X'04'	None	Yes
Transfer in Channel (TIC)	X'08'	None	Yes
Write Data (WD)	X'01'	M.WRIT	Yes
Write Sector Label (WSL)	X'31'	None	No
Write Track Label (WTL)	X'51'	None	No
Read Data (RD)	X'02'	M.READ	Yes
Read Sector Label (RSL)	X'32'	None	No
Read Track Label (RTL)	X'52'	None	No
Read Angular Position (RAP)	X'A2'	None	No
No Operation (NOP)	X'03'	None	Yes
Seek Cylinder (SKC)	X'07'	None	Yes
Format for No Skip (FNSK)	X'0B'	None	No
Lock Protected Labels (LPL)	X'13'	None	No
Load Mode Register (LMR)	X'1F'	None	Yes
Reserve (RES)	X'23'	M.RESP	Yes
Release (REL)	X'33'	M.RELP	Yes
Rezero (XEZ)	X'37'	None	Yes
Test Star (TESS)	X'AB'	None	No
Increment Head Address (IHA)	X'47'	None	No
Priority Override (POR)	X'43'	None	Yes
Set Reserve Track Mode (SRM)	X'4F'	None	No
Reset Reserve Track Mode (XRM)	X'5F'	None	No
Read ECC (REC)	X'B2'	None	Yes

14.2.1.2 IOCD Format

All channel commands have the following IOCD format:

	00 07	08 31
Word 1	Command Code	Absolute Date Addresss

	00 15	16 31
Word 2	Flags	Byte Count

Flag bits 00 through 04 have the following significance:

Bit	Mnemonic	Name
00	DC	Data Chain
01	CC	Command Chain
02	SLI	Suppress Incorrect Length Indication
03	SKIP	Skip Read Data
04	PCI	Program Controlled Interrupt

The command code field defines the operation to be performed during command execution.

The data address must be a 24-bit absolute addresss.

TIC branch address must be 24-bit addresses that address word boundaries.

The requirements for IOCD validity apply to all IOCDs regardless of the contents of the command code field.

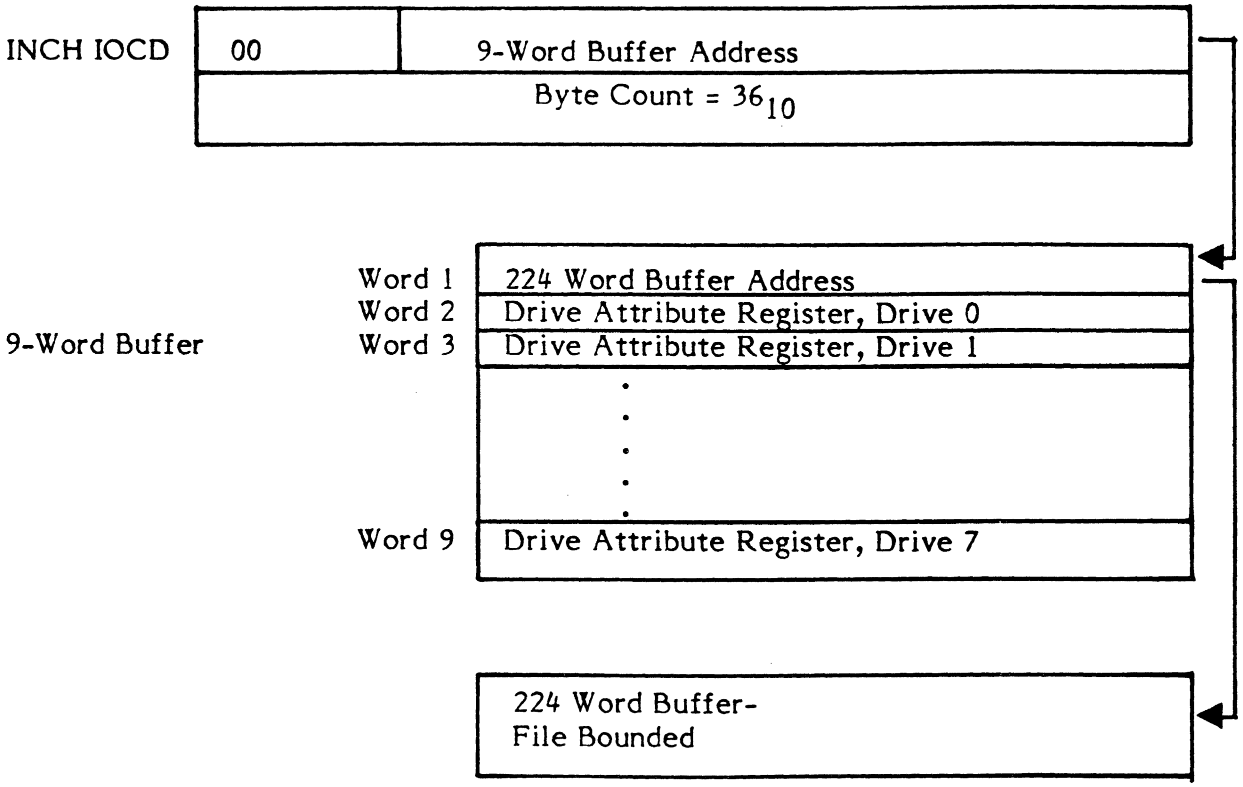
The requirements are:

- o IOCD word 2 bits 05 through 15 must be zero
- o The PCI bit must be zero
- o The byte count must be nonzero

14.2.1.3 Commands

14.2.1.3.1 Initialize Channel (INCH)

This command is the means by which disc drive information is relayed to the disc processor and the means by which a buffer area is declared and made available for use by the Disc Processor. An INCH command must be the first I/O command to any channel that has a disc processor configured and is performed automatically by the disc handler as a result of the first I/O request to the channel. The data address specified in the INCH IOCD points to a nine word buffer that must fall on a word boundary. The first word of this nine-word buffer must contain a 24-bit address that points to a file bounded 224 word buffer that is used by the disc processor for record keeping. The remaining 8 words contain disc drive information pertinent to each of the 8 drives that may be configured. The following diagram shows the overall picture of this relationship.



Drive Attribute Register - layout is as follows:

0 7	8 15	16 23	24 31
Flags	Sector Count	MHD Count	FHD Count

Bits	Meaning
0 1	10 = FHD 01 = MHD 11 = MHD with FHD option 00 = Reserved
2	1 = Cartridge Module Drive
3	Reserved
4	1 = Drive Not Present
5	1 = Drive is Dual Ported
6-7	Reserved for future expansion; must be zero

Sector Count is the number of sectors per track (14_{16}).

MHD Count is the number of heads on the MHD or the number of heads for the removeable media portion of the cartridge module drive.

FHD Count is the number of heads for the captive media portion of the cartridge module drive.

The drive attribute register for a 40 MB single port moving head disc would be generated as follows:

```
DATAW   X'40140500'
          | | | |_______________Zero  FHD count
          | | |_________________5 MHD count
          | |___________________20 Sectors per Track
          |_____________________Moving Head Disc
```

The drive attribute register for an 80 MB dual ported moving head disc would be generated as follows:

```
DATAW   X'44140500'
          ||| | |_______________Zero FHD count
          ||| |_________________5 MHD count
          |||___________________20 Sectors per Track
          ||____________________Dual port indication
          |_____________________Moving Head Disc
```

The drive attribute register for a 5 MB single port fixed head disc would be generated as follows:

```
DATAW   X'40140400'
          | | | |_______________Zero FHD count
          | | |_________________4 MHD count
          | |___________________20 Sectors per Track
          |_____________________Moving Head Disc
```

The drive attribute register for a 32 MB cartridge module drive would be generated as follows:

```
DATAW   X'20140101'
          | | | |_______________1 FHD count
          | | |_________________1 MHD count
          | |___________________20 Sectors per Track
          |_____________________Cartridge Module Drive
```

The drive attribute register for a drive that is not present would be generated as follows:

```
DATAW   X'0800000'
          |_____________________Drive Not Present
```

14.2.1.3.2 Sense (SENSE)

This command is a means by which information can be retrieved concerning the results of the last SIO processed by a subchannel.

The disc handler issues the SENSE command to generate information regarding I/O error completions and uses the information in determining certain retry requirements. The Sense information is stored in the absolute data address specified in the Sense IOCD. When the disc handler issues a SENSE command, it specifies a data address in the context block associated with the device (see Section 14.3.6). Some of the sense information generated is passed to the user (see Section 14.3.12).

Note: The drive is reselected if a byte count greater than 12 is specified.

The following describes the information returned from the SENSE command:

SENSE Information

Word	0 7	8 15	16 23	24 31
1	Cylinder		Track	Sector
2	Mode Byte	Contents of SENSE Buffer Register		
3	Drive Attribute Register			
4	Drive Status		Not Used	

Word 1: current cylinder, track and sector.

Word 2: Mode Byte - bit assignments are as follows:

Bit	Function
0	A 1 implies that the drive carriage will be offset.
1	This bit is effective only when bit 0 is in the 1 state; a zero implies a positive track offset and a 1 implies a negative track offset; a positive offset is an offset toward the next higher cylinder number.
2	A 1 implies a read timing offset.
3	This bit is effective only when bit 2 is in the one state; a 0 implies that a positive read strobe timing adjustment will be used; a 1 implies that a negative read strobe timing adjustment will be used.

Bit	Function
4	A 1 implies diagnostic mode for Error Corrrection Code (ECC) generation and checking.
5	A 1 implies that reserved tracks can be accessed without causing an error; a zero implies that reserved track data cannot be written.
6	A 1 implies that the associated Subchannel (SSC) will access the captive media portion of a cartridge module drive (CMD).
7	A 1 implies that the channel functions will use the RAM buffer for data operations, i.e., buffer mode is invoked.

When all Mode bits are set to the zero state, data operations occur between main memory and a Moving Head Disc (MHD); this setting might be considered to be the normal mode. A Halt Channel Directive (HCHNL) places all channels in this mode. A Halt I/O (HIO) does not change the selected subchannel's mode.

Word 2: Contents of SENSE Buffer Register - bit assignments are as follows:

Bit	Meaning
8	Command Rejected
9	Intervention Requested
10	Spare
11	Equipment Check
12	Data Check
13	Data Over or Under Run
14	Disc Format Error
15	Defective Track Encountered
16	Last Track Flag Encountered
17	At Alternate Track
18	Write Protection Error
19	Write Lock Error
20	Mode Check
21	Invalid Memory Address
22	Release Fault
23	Chaining Error
24	Lost Revolution
25	Disc Addressing or Seek Error
26	Buffer Check
27	ECC Error in Sector Label
28	ECC Error in Data
29	ECC Error in Track Label
30	Reserve Track Access Error
31	Uncorrectable ECC

Word 3: Drive Attribute Register - For layout, see Section 14.2.1.3.1.

Word 4: Drive Status - bit assignments are as follows:

Bit	Meaning
0	Seek End
1	Unit Selected
2	Sector Pulse Counter Bit 0
3	Sector Pulse Counter Bit 1
4	Sector Pulse Counter Bit 2
5	Sector Pulse Counter Bit 3
6	Sector Pulse Counter Bit 4
7	Sector Pulse Counter Bit 5
8	Disc Drive Detected a Fault
9	Seek Error
10	On Cylinder
11	Unit Ready
12	Write Protected
13	Drive is Busy
14	Spare
15	Spare

14.2.1.3.3 Transfer In Channel (TIC)

This command causes Input/Output Command Doubleword (IOCD) execution to continue at the address specified in the TIC command. Effectively, a TIC serves as a branch for IOCD execution. A TIC command cannot point to another TIC command nor can it be the first command in an IOCD list.

The TIC command is used by the handler to link IOCDs located in the Context Block (see Section 14.3.6) to IOCDs located in the I/O Queue (IOQ).

14.2.1.3.4 Write Data (WD)

This command transfers data to the disc from the address specified in the IOCD. Entry point 5 of H.DP02 processes user Write requests by calling H.IOCS,40 to build Write Data IOCDs within the IOQ.

14.2.1.3.5 Write Sector Label (WSL)

This command is the means by which sector labels are written to the disc.

It is not currently used by the disc handler.

14.2.1.3.6 Write Track Label (WTL)

This command is the means by which track labels are written to the disc.

It is not currently used by the disc handler.

14.2.1.3.7 Read Data (RD)

This command transfers data from the disc to the address specified in the IOCD. Entry point 5 of H.DP02 processes user Read requests by calling H.IOCS,40 to build Read Data IOCDs within the IOQ.

14.2.1.3.8 Read Sector Label (RSL)

This command reads sector labels from the disc.

It is not currently used by the disc handler.

14.2.1.3.9 Read Track Label (RTL)

This command reads track labels from the disc.

It is not currently used by the disc handler.

14.2.1.3.10 Read Angular Position (RAP)

This command reads the sector pulse counter from the disc.

It is not currently used by the disc handler.

14.2.1.3.11 No Operation (NOP)

This command executes without selecting the associated disc drive.

Completed Read Data IOCDs within the I/O Queue (IOQ) IOCD list are changed to NOP commands at entry point 1 of H.DP02 when performing error correction code (ECC) logic.

14.2.1.3.12 Seek Cylinder (SKC)

This command causes a disc head seek/select to the specified cylinder, track, and sector. The address specified in the Seek Cylinder command points to a memory word which contains the following:

0 15	16 23	24 31
0 Cylinder	Track	Sector

Entry point 5 of H.DP02 computes the cylinder, track, and sector address for user requested reads and writes, and stores this information into the I/O Queue cell named IOQ.FCT1. S.IOCS12 is then called to build the seek IOCD and store it into the IOQ.

14.2.1.3.13 Format for No Skip (FNSK)

This command is used by the diagnostic to format a disc.

It is not currently used by the disc handler.

14.2.1.3.14 Lock Protect Label (LPL)

This command involves write lock.

It is not currently used by the disc handler.

14.2.1.3.15 Load Mode Register (LMR)

This command is used to identify a byte of information which specifies the manner in which I/O is to take place with the disc. The address specified in the Input/Output Control Doubleword (IOCD) points to this byte of information which is physically located in the I/O Queue. See Section 14.2.1.3.2 (Word 2, Byte 0 of the Sense Information) for interpretation of the mode bits.

The disc handler automatically generates this command as the first IOCD presented for disc access user requests. The command physically resides in the Context Block.

14.2.1.3.16 Reserve (RES)

This command causes a device to be reserved to the requesting CPU until such time as a Release (REL) or Priority Override (POR) is issued. The command is user callable through the M.RESP service routine and is associated with dual-port operations. Execute channel programs must never include a Reserve command and should use the M.RESP service routine when device reservation is desired.

14.2.1.3.17 Release (REL)

This command causes a Reserved device to be released by the reserving CPU. The Release will not be issued if more than one task has the device Reserved. The command is user callable through the M.RELP service routine and is associated with dual-port operations. Execute channel programs must never include a Release command and should use the M.RELP service routine when device release is desired.

14.2.1.3.18 Rezero (XEZ)

This command is effectively a recalibration request to the disc which resets the drive's seek logic and causes the drive to locate cylinder and track zero. Entry point 1 of H.DP02 uses it to recover from seek and drive fault errors. The command physically resides in the Context Block.

14.2.1.3.19 Test Star (TESS)

This command causes the currently addressed cylinder, track, and sector to be compared to that specified by the Test Star IOCD. It can be used to skip the next sequential IOCD.

It is not currently used by the disc handler.

14.2.1.3.20 Increment Head Address (IHA)

This command is used to select sector zero of the next sequential track in the associated disc drive.

It is not currently used by the disc handler.

14.2.1.3.21 Priority Override (POR)

This command provides a mechanism for overriding and disabling dual-ported disc drive Reserve functions. The drive specified in the POR command is absolutely reserved to the requesting channel until the channel Releases the drive.

This command is used by the handler to gain control of a drive that is reserved by the opposing channel in a dual ported configuration if that channel does not Release the drive within a specified period of time. This time value is an optional SYSGEN parameter (DPTOV) which defaults to two timer units.

14.2.1.3.22 Set Reserve Track Mode (SRM)

This command allows all data areas designated as reserve tracks to be read or written. It should be noted that in order for this to work, the reserve track mode jumper must be set on the Device Interface Adapter (DIA) board.

It is not currently used by the disc handler.

14.2.1.3.23 Reset Reserve Track Mode (XRM)

This command makes all data areas designated as reserve tracks unavailable for write operations.

It is not currently used by the disc handler.

14.2.1.3.24 Read ECC (REC)

This command causes the channel to compute and present error correction information needed to recover from a disc read error. The information returned to the address specificied in the Read ECC IOCD contains the following:

0 15	16 31
Displacement	Correction Mask

where:

Displacement — is the number of bits from the end of the last sector transferred to the last bit in the field found to contain the error.

Correction Mask — is a 9-bit correction mask that can be used to correct erroneous data stored in memory.

The Read ECC command is used at entry point 1 of H.DP02 to recover from data errors.

14.2.2 CPU Instructions

14.2.2.1 Instruction Summary

The following is a summary of the Extended I/O CPU instructions.

Mnemonic	Instruction Code	Function
SIO	X'2'	Start I/O
TIO	X'3'	Test I/O
HIO	X'6'	Halt I/O
HCHNL	X'5'	Halt Channel
RSCHNL	X'5'	Reset Channel
STPIO	X'4'	Stop I/O
RSCTL	X'8'	Reset Controller
ECI	X'C'	Enable Channel Interrupt
DCI	X'D'	Disable Channel Interrupt
ACI	X'E'	Activate Channel Interrupt
DACI	X'F'	Deactivate Channel Interrupt

14.2.2.2 CPU Instruction Format

All Extended I/O CPU instructions have the following format:

0 5	6 8	9 12	13 15	16 31
Op Code	R	Instruction Code	Aug Code	Constant

Bits 00-05: specifies the operation code, hex 'FC'.

Bits 06-08: specifies the general register, when nonzero, whose contents will be added to constant to form the logical channel and subaddress.

Bits 09-12: specifies the instruction code.

Bits 13-15: specifies the augment code, hex '7'.

Bits 16-31: specifies a constant that will be added to the contents of R to form the logical channel and subaddress. If R is zero, only constant will be used to specify the logical channel and subaddress.

14.2.2.3 Condition Codes

Condition codes are generated for all extended I/O instructions and indicate the successful or unsuccessful initiation of an I/O instruction. For extended I/O purposes, the four normal condition code bits are interpreted as a four bit hexadecimal number ranging from 0 - F. This means that there are 16 possible condition code responses to an extended I/O instruction. The following is a summary of extended I/O condition code assignments.

Extended I/O
Condition Code Assignments

Condition Code				Hex Value	Meaning
CC1	CC2	CC3	CC4		
0	0	0	0	X'0'	Accepted Will Echo
0	0	0	1	X'1'	Channel Busy
0	0	1	0	X'2'	Channel Inoperable or Undefined
0	0	1	1	X'3'	Subchannel Busy
0	1	0	0	X'4'	Status Stored
0	1	0	1	X'5'	Unsupported Transaction
0	1	1	0	X'6'	Unassigned
0	1	1	1	X'7'	Unassigned
1	0	0	0	X'8'	Request Accepted
1	0	0	1	X'9'	Unassigned
1	0	1	0	X'A'	Unassigned
1	0	1	1	X'B'	Unassigned
1	1	0	0	X'C'	Unassigned
1	1	0	1	X'D'	Unassigned
1	1	1	0	X'E'	Unassigned
1	1	1	1	X'F'	Unassigned

14.2.2.4 Condition Code Checking

Condition code checking within the disc handler varies depending on the instruction issued. The following summarizes condition code checking and disc handler action.

14.2.2.4.1 For SIO Instruction

Hex Condition Codes	Meaning	Action
X'8'	Request Accepted	Continue normal processing
X'1'	Channel busy	Retry 1 time, if unsuccessful process next I/O request
X'3'	Subchannel Busy	Retry 1 time, if unsuccessful process next I/O request
X'2'	Channel Inoperative or Undefined	Set operator intervention bit, show error condition for FCB, abort I/O request
X'4'	Status Stored	Branch to H.EXIO and process as though an interrupt had occurred
X'5'	Unsupported Transaction	Show error condition for FCB, abort the I/O request
X'6',X'7' X'9',X'F'	Unassigned	Show error condition for FCB, abort the I/O request
X'0'	Not supported by the disc processor	

14.2.2.4.2 For HIO Instructions

Hex Condition Codes	Meaning	Action
X'4'	Status Stored	Branch to H.EXIO and process as though an interrupt had occurred

NO OTHER CONDITION CODES ARE CHECKED

14.2.2.4.3 For Remaining Instructions

For ACI, DACI, RSCTL, RSCHNL, DCI, and ECI instructions:

NO CONDITION CODES ARE CHECKED

14.2.2.5 Instructions

14.2.2.5.1 Start I/O (SIO)

The SIO instruction is used to begin I/O execution if the subchannel number is valid and the channel has no pending final status. If the channel has pending final status, the SIO instruction is rejected with a status stored condition code response. The status stored response is equivalent to an interrupt status presentation and must be treated as such as no further indication of the I/O completion is given.

The SIO instruction is used by entry point 1 of H.DP02 as indicated above.

14.2.2.5.2 Test I/O (TIO)

The TIO instruction is used to test controller status and to return appropriate condition codes and status reflecting the state of the channel and addressed subchannel.

The TIO instruction is not used by the disc handler.

14.2.2.5.3 Halt I/O (HIO)

The HIO instruction is directed to a particular subchannel; the channel and subchannel respond to the HIO by terminating all activities in the subchannel at the end of the current sector. The instruction will not halt the I/O to a malfunctioning device. The HIO instruction does not affect subchannels other than the subchannel addressed: however, it will generate a status stored response if status is pending in any of the channel's subchannels and reject the HIO instruction. The status stored response is equivalent to an interrupt status presentation and must be treated as such as no further indication of the I/O completion is given.

The HIO instruction is used by entry point 4 of H.DP02 to recover from I/O requests that time-out.

14.2.2.5.4 Halt Channel (HCHNL) and Reset Channel (RSCHNL)

HCHNL and RSCHNL are the same instruction.

These instructions terminate all activity in the channel. Issuance of these instructions requires that an INCH command (see Section 14.2.1.3.1) be performed before any subsequent I/O to the affected channel.

The RSCHNL instruction is used at entry point 5 of H.DP02 prior to issuing the INCH command.

14.2.2.5.5 Stop I/O (STPIO)

The STPIO instruction is used to perform an orderly termination of an IOCD list by stopping IOCD execution at the completion of the current IOCD. The STPIO instruction applies only to the addressed subchannel; however, if there is pending status for any subchannel associated with the addressed channel, the instruction will not be executed and a status stored condition code response will be returned. The status stored response is equivalent to an interrupt status presentation and must be treated as such as no further indication of the I/O completion is given.

The STPIO instruction is not used by the disc handler.

14.2.2.5.6 Reset Controller (RSCTL)

The RSCTL instruction causes the addressed subchannel to terminate its I/O operation immediately. If the subchannel is in a hung condition the device will be reset so that normal I/O operations may resume. The RSCTL instruction will always be accepted, will never generate a status stored response, and will never generate an interrupt.

The RSCTL instruction is used to recover from the HIO instruction issued at entry point 4 of H.DP02 should it also time out. This condition would indicate that the device is actually broken. It should be noted that when the RSCTL instruction is issued at EP4, a HIO instruction is also issued to gain access to EP1 processing.

14.2.2.5.7 Enable Channel Interrupt (ECI)

The ECI instruction causes the addressed channel to be enabled to request interrupts from the CPU.

The ECI instruction is used at entry point 5 of H.DP02 prior to issuing the INCH command.

14.2.2.5.8 Disable Channel Interrupt (DCI)

The DCI instruction causes the addressed channel to be disabled from requesting interrupts from the CPU.

The DCI instruction is used at entry point 5 of H.DP02 prior to issuing an Enable Channel Interrupt (ECI) instruction.

14.2.2.5.9 Activate Channel Interrupt (ACI)

The ACI instruction causes the addressed channel to begin actively contending with other interrupt levels preventing its level and all lower priority levels from requesting an interrupt.

The ACI instruction is used by H.DP02 for protection of certain sensitive code paths.

14.2.2.5.10 Deactivate Channel Interrupt (DACI)

The DACI instruction causes the addressed channel to remove its interrupt level from contention.

The DACI instruction is used by H.EXIO just prior to entry point 1 and 3 exits. It is also used to clear explicit ACI instructions within H.DP02.

14.3 Related Data Structures

This section outlines the data structures used by the disc handler.

14.3.1 I/O Queue (IOQ)

See the MPX-32 Technical Manual, Chapter 2.

14.3.2 Unit Definition Table (UDT)

See the MPX-32 Technical Manual, Chapter 2.

14.3.3 Controller Definition Table (CDT)

See the MPX-32 Technical Manual, Chapter 2.

14.3.4 File Control Block (FCB)

See the MPX-32 Technical Manual, Chapter 2.

14.3.5 File Assignment Table (FAT)

See the MPX-32 Technical Manual, Chapter 2.

14.3.6 Context Block

A Context Block exists for each active subchannel and serves as a storage area for information regarding the subchannel and its operation. The Context Blocks are physically located at the end of the disc handler (H.DP02) and contain the following:

Context Block Format

Word	
0	# of H.DP02 Entry Points
1	H.DP02 EP1 Address
2	H.DP02 EP2 Address
3	H.DP02 EP3 Address
4	H.DP02 EP4 Address
5	H.DP02 EP5 Address
6	H.DP02 EP6 Address
7	H.DP02 EP7 Address
8	Status Word 1 [1]
9	Status Word 2 [1]
10	Active I/O Queue Address [2]
11	Flagword [3]
12	CDT Address [4]
13	Sense Buffer Word 1 [5]
14	Sense Buffer Word 2 [5]
15	Sense Buffer Word 3 [5]
16	Sense Buffer Word 4 [5]
17	Channel Address and Subaddress
18	Address of IOCD Pointer Within H.EXIO [6]
19	# Error Retries Attempted
20	# Sectors/Cylinder
21	ECC Data
22	Rezero IOCD Word 1
23	Rezero IOCD Word 2
24	Load Mode IOCD Word 1
25	Load Mode IOCD Word 2

Word		
26	TIC IOCD Word 1	
27	TIC IOCD Word 2	
28	Sense IOCD Word 1	
29	Sense IOCD Word 2	
30	Read ECC IOCD Word 1	
31	Read ECC IOCD Word 2	
32	POR IOCD Word 1	
33	POR IOCD Word 2	
34	Working Status Word 1 [7]	
35	Working Status Word 2 [7]	
36	H.EXIO OLDPSD Address [8]	
37	Successful SIO Counter	
38	Error Retry Counter	
39	Lost Interrupt Counter	
40	# of Reset Controller Commands Issued	
41	# of ECC Corrections Performed	
42	ECC Bit Displacement	ECC Byte Displacement
43	Not Used	
44	Last Buffer Address for Previous IOCD (ECC Logic)	
45	Start Buffer Address for Current IOCD (ECC Logic)	
46	Address of End of Erring Sector (ECC Logic)	
47	Address of Erring Halfword (ECC Logic)	
48	# of Bytes Transferred for Current IOCD (ECC Logic)	
49	End of Current IOCD Buffer (ECC Logic)	
50	Dual Port Time Out Value[9]	
51	# of Reserves Active for this Subchannel	

Notes:

1. This is the status doubleword returned when an interrupt fires or when a status stored response is generated as a result of the SIO and HIO instructions (see Section 14.3.7).

2. This is the address of the IOQ for which an interrupt is pending.

3. Flagword bits have the following meaning:

Bit	Meaning
1	if set, lost interrupt
2	if set, I/O completed for a device that had timed out
0, 3-31	reserved

4. CDT address associated with this channel.

5. See Section 14.2.1.3.2 for format of these four words.

6. Contains the address of the fourth word of the interrupt context block located in H.EXIO.

7. Used for error retry logic.

8. Contains the address of the interrupt context block located in H.EXIO.

9. This is the SYSGEN specifiable time-out value for dual-ported (DPTOV) discs.

14.3.7 Status Doubleword

A Status Doubleword is presented each time an interrupt is generated or as a result of the "status stored" response to a SIO or HIO instruction. It has the following format:

Word	0 8	9 15	16 31
1	Subchannel	IOCD Address	
2	Status		Residual Byte Count

Word 1:

Subchannel - subchannel address of interrupting device
IOCD Address - this address points 8 bytes past the last IOCD processed

Word 2:

Status - Status bits are defined as follows:

Bit	Mnemonic	Name
0	ECHO	ECHO
1	PCI	Program Controlled Interrupt
2	IL	Incorrect Length
3	PCK	Program Check
4	CDC	Channel Data Check
5	CCC	Channel Control Check
6	ICC	Interface Control Check
7	CC	Chaining Check
8	DB	Device Busy
9	SM	Status Modified
10	CNE	Controller End
11	ATT	Attention
12	CE	Channel End
13	DE	Device End
14	UC	Unit Check
15	UE	Unit Exception

Residual Byte Count - number of bytes not transferred for the last IOCD processed

14.3.8 Input/Output Control Doubleword (IOCD)

See Section 14.2.1.2.

14.3.9 Interrupt Context Block

See the 32/70 Series Reference Manual.

14.3.10 Sense Buffer

See Section 14.2.1.3.2.

14.3.11 INCH Buffer

See Section 14.2.1.3.1.

14.3.12 Status Returned to User's FCB

The handler places the following information into the IOQ which is relayed back to the user's File Control Block (FCB) by IOCS. Not all of the sense information obtained by the handler is returned to the user.

	IOQ Word		From	Becomes FCB Word		
IOQ.IOST	Special Bits Set by handler	Status Bits	STATUS + 2H	Special Bits Set by handler	Status Bits	FCB.SFLG
IOQ.IST1	Mode Byte	Contents of Sense Buffer Register	SENSE BUFFER WORD 2	Mode Byte	Contents of Sense Buffer Register	FCB.IST1
IOQ.IST2	Status Bits	Residual Byte Count	STATUS WORD 2	Status Bits	Residual Byte Count	FCB.IST2
IOQ.UTRN	Number of bytes transferred		COMPUTED BY HANDLER	Number of bytes transferred		FCB.RECL

Note: For Execute Channel Program requests, IOQ.IST1 contains STATUS WORD 1 and IOQ.UTRN is invalid.

14.4 XIO Disc Interrupt Fielder (H.EXIO)

One copy of H.EXIO exists for each channel that is connected to a disc processor. It serves as the interrupt/event fielder for that channel and as the link to entry points 1 and 2 of H.DP02.

14.4.1 Entry Point Summary

H.EXIO has four entry points as follows:

Entry Point	Handler Entry Point Name	Function
EP1	EP1AND3	Queue Drive Interrupt Service Routine
EP2	EP2	Queue Start Service Routine
EP3	EP1AND3	Spurious Interrupt Service Routine
EP8	EXIO.8	SYSGEN Initialization

14.4.2 Entry Point 1 and 3 - Queue Drive Interrupt Service Routine Spurious Interrupt Service Routine

Because it is not known whether a valid or spurious interrupt/event has occurred until some processing has taken place within H.EXIO, entry points 1 and 3 are physically the same. Entry point 1 and 3 is entered any time one of the following occurs:

- o an interrupt fires
- o status is stored as a result of the SIO instruction in entry point 1 of H.DP02
- o status is stored as a result of the HIO instruction in entry point 4 of H.DP02

14.4.2.1 Processing - When an Interrupt Fires

Calling Sequence:

Entered through Service Interrupt (SI) vector location, where the new Program Status Doubleword (PSD) causes the system to go unmapped with interrupts unblocked and the SI level active. This is always the initial mechanism for reporting I/O complete for a given channel.

Registers Passed in Calling Sequence: None.

Control Transferred to:

- o Entry point 1 of H.DP02 if valid interrupt
- o Exit routine of H.EXIO if spurious interrupt

14.4.2.2 Processing - As a Result of Status Stored for Start I/O (SIO)

Calling Sequence:

Entered from entry point 1 of H.DP02 by:

- o setting bit within H.EXIO indicating entry is from H.DP02
- o BU *XIO.EP13,X3 X3 = H.EXIO OLDPSD address

Registers Passed in Calling Sequence:

R0 = context block address that was in effect when the Status Stored Condition Code was generated in response to the SIO instruction.

Control Transferred To:

- o Entry point 1 of H.DP02 if status was stored for a valid I/O request
- o Entry point 2 of H.DP02 if status was stored for a spurious event

14.4.2.3 Processing - As a Result of Status Stored for Halt I/O (HIO)

Calling Sequence:

Entered from entry point 4 of H.DP02 by:

- o setting bit within H.EXIO indicating entry is from H.DP02
- o setting bit within H.EXIO indicating entry is from EP4 of H.DP02
- o BU *XIO.EP13, X2 X2 = H.EXIO OLDPSD address

Registers Passed in Calling Sequence:

R0 = context block address that was in effect when the Status Stored Condition Code was generated in response to the HIO instruction.

Control Transferred To:

- o Entry point 1 of H.DP02 if status was stored for a valid I/O request
- o Entry point 2 of H.DP02 if status was stored for a spurious event

14.4.2.4 Common Processing

External Routines Called:

- o Entry point 1 of H.DP02
- o Entry point 2 of H.DP02

Functions Performed:

- o increments global interrupt count (C.GINT) - only when an interrupt fires
- o saves registers - only when an interrupt fires
- o determines the validity of the interrupt or status stored event
- o fetches the context block address of the offending device
- o transfers control to appropriate processing as defined above

Exit Sequence:

o	LPSD	return to the mapped mode
o	BEI	block external interrupts
o	DACI	deactivate interrupt level
o	BL	S.EXEC5 (no return)

Registers Passed in Exit Sequence:

R6 and R7 - PSD in effect when the interrupt occurred
R2 - address of the register save area

14.4.3 Entry Point 2 - Queue Start Service Routine

Entry point 2 of H.EXIO is entered from IOCS each time it queues an I/O request and the channel is not busy. It is entered with interrupts blocked and the appropriate context block address.

Calling Sequence:

Entered from H.IOCS,29 by:

```
        BEI                         Block External Interrupts
        BL        *XIO.EP2,X2       X2 = H.EXIO  OLDPSD address
```

Registers Passed in Calling Sequence:

R3 = Context Block address

External Routines Called:

- o Entry point 2 of H.DP02

Functions Performed:

- o sets bit indicating a handler EP2 entry
- o transfers control to EP2 of H.DP02

Exit Sequence:

```
TRSW    R0
```

Registers Passed in Exit Sequence:

All registers returned unchanged.

14.4.4 Entry Point 8 - SYSGEN Initialization

Entry point 8 of H.EXIO is called by SYSGEN when configuring a new system.

Calling Sequence:

```
BL    *1W,X2        X2 = H.EXIO HAT address
```

Registers Passed in Calling Sequence:

R7 = CDT address

External Routines Called: None.

Functions Performed:

- provides a buffer area used by the disc processor for storage and record keeping
- builds the deactivate channel interrupt (DACI) instruction for EP1 and EP3 exit routines
- computes the SI vector location address and initializes it with the H.EXIO execution address
- stores the SI scratchpad address into CDT.SIAD
- builds the drive attribute information table needed for channel initialization (INCH)

Exit Sequence:

```
M.XIR   HAT   (Special SYSGEN Initialization Termination Macro)
```

Note: 'HAT' is a label equated to 0 relative within the H.EXIO handler.

Registers Passed in Exit Sequence: None.

14.5 XIO Disc Handler (H.DP02)

One copy of H.DP02 exists for each system that supports the disc processor. Its function is to process I/O requests on behalf of the user.

14.5.1 Entry Point Summary

H.DP02 has 8 entry points as follows:

Entry Point	Handler Entry Point Name	Function
EP1	DP02.1	Queue drive interrupt service routine
EP2	DP02.2	Queue start service routine
EP3	DP02.3	Spurious interrupt service routine
EP4	DP02.4	Lost (timed out) interrupt processor
EP5	DP02.5	Op code processing routine
EP6	DP02.6	Post transfer processing routine
EP7	DP02.7	Error processing routine
EP8	DP02.8	SYSGEN initialization

14.5.2 Entry Point 1 - Queue Drive Interrupt Service Routine

Entry point 1 of H.DP02 performs I/O post-access processing for completed I/O requests and pre-access processing associated with all queued I/O requests for non busy devices on the interrupting channel.

Calling Sequence:

This entry point is called from H.EXIO entry point 1 by a:

```
        BU              *JUMPLOC
```

where JUMPLOC contains the address of EP1 of H.DP02.

Registers Passed In Calling Sequence:

R3 = Context Block address of completing subchannel

External Routines Called:

S.IOCS3	unlink IOQ from Controller Definition Table (CDT) chain
S.IOCS15	delete IOQ for task delete requests
S.IOCS29	report I/O complete

Functions Performed:

Entry point 1 performs the following functions

- stores status into IOQ
- computes and stores transfer count into IOQ
- issues SENSE commands when needed
- stores SENSE information into IOQ
- determines if error conditions exist
- processes error conditions and does retries as required
- initiates I/O for queued I/O requests
- takes various action in response to condition codes returned from SIO commands

- processes device time-outs
- unlinks IOQ from CDT chain
- processes requests for task deletes
- reports I/O complete
- issues priority override for dual ported discs that time out

Exit Sequence:

Entry point 1 always exits with an unconditional branch to the H.EXIO exit processing routine.

```
        BU        *XIO.XIT,X3          X3 = H.EXIO  OLDPSD address
```

Registers Passed in Exit Sequence: None.

14.5.3 Entry Point 2 - Queue Start Service Routine

Entry point 2 of H.DP02 serves as the link between EP2 of H.EXIO and the pre-access processing code of H.DP02 EP1.

Calling Sequence:

This entry point is called from H.EXIO entry point 1 by a

```
        BU    *JUMPLOC
```

where JUMPLOC contains the address of EP2 of H.DP02

Registers Passed In Calling Sequence:

R3 = context block address associated with the I/O request

External Routines Called: None.

Functions Performed:

Branches to pre-access processing code of H.DP02 EP1.

Exit Sequence:

Entry point 2 merges with entry point 1 and uses its exit sequence to return to H.EXIO.

Registers Passed in Exit Sequence: None.

14.5.4 Entry Point 3 - Spurious Interrupt Service Routine

Entry point 3 of H.DP02 is a dummy entry point which is never called because spurious interrupts are discovered and processed at EP3 of H.EXIO.

14.5.5 Entry Point 4 - Lost Interrupt Service Routine

Entry point 4 of H.DP02 is entered from S.IOCS5 to take corrective measures appropriate to the device when an expected Service Interrupt (SI) fails to occur. It is also entered from H.IOCS,38 when a 'KILL' command is issued for I/O that has started but has not completed. Because a Halt I/O (HIO) instruction is issued, EP4 contains logic to handle a status stored response.

Calling Sequence:

This entry point is called from either S.IOCS5 or H.IOCS,38 with:

```
        BEI         Block External Interrupts
        BL          *4W,X2  X2 = H.DP02 Halfword Address Table (HAT) address
```

Register Passed in Calling Sequence:

R1 = UDT address

External Routines Called:

H.EXIO	if HIO produces status stored response

Functions Performed:

- terminates device activity and sets lost interrupt flag

Exit Sequence:

Entry point 4 returns to the calling sequence with a:

```
        TRSW        R0
```

Registers Passed in Exit Sequence: None.

14.5.6 Entry Point 5 - Opcode Processing Service Routine

14.5.6.1 Common Processing

Entry point 5 of H.DP02 is called by IOCS to process various user I/O requests.

Calling Sequence:

```
        BL    *5W,R3    R3 = H.DP02 Halfword Address Table (HAT) address
```

Registers Passed in Calling Sequence:

R1 = FCB address
R2 = FAT address

External Routines Called:

S.IOCS13	build an IOQ
H.IOCS,40	set up IOCDs in IOQ
S.IOCS33	update File Assignments Table (FAT) and File Control Block (FCB); return files starting block number
S.IOCS10	delete IOQ
S.IOCS12	set up IOCDs in IOQ

Exit Sequence:

All entry point 5 returns are to IOCS through one of the following calls:

o BL IOLINK	used when normal device access is required to service the request
o BL SERVCOMP	used when request has been serviced with no device access required
o BL ILOPCODE	used when op code passed to EP5 is in error

Registers Passed in Exit Sequence: None.

14.5.6.2 Opcode Dependent Processing

14.5.6.2.1 Opcode Summary Table

The following table summarizes FCB op code assignments.

Op Code	Timeout Value	Function	EP5 return to IOCS
X'00'	0	Open File	Service Complete [1]
X'01'	0	Rewind File	Service Complete
X'02'	4	Read	Queue Request
X'03'	4	Write	Queue Request
X'04'	0	Write End-of-File	Service Complete
X'05'	0	Execute Channel Program	Queue Request
X'06'	0	Advance Record	Service Complete
X'07'	0	Advance File	Service Complete
X'08'	0	Backspace Record	Service Complete
X'09'	0	Backspace File	Service Complete
X'0A'	0	Upspace	Service Complete
X'0B'	0	Erase	Service Complete
X'0C'	0	Eject	Service Complete
X'0D'	0	Close File	Service Complete
X'0E'	2	Reserve	Queue Request
X'0F'	0	Release	Queue Request

[1] Open commands normally take the service complete return except for the first I/O request to the channel which forces channel initialization (INCH).

where:

Op code is a hexadecimal number from 00-0F used by IOCS and EP5 only
Timeout Value is in seconds
Service Complete returns to IOCS SERVCOMP routine
Queue Request returns to IOCS IOLINK routine

14.5.6.2.2 Open

Open commands are normally processed in EP5 by taking the service complete return with no other action. The exception to this is when OPEN is called the first time for a given channel. The first OPEN call will effect a channel initialization (INCH) by:

- calling S.IOCS13 to build an IOQ
- issuing a Reset Controller (RSCTL) instruction
- issuing a Reset Channel (RSCHNL) instruction
- delaying for 32000 timer units
- issuing a Disable Channel Interrupt (DCI) instruction
- issuing an Enable Channel Interrupt (ECI) instruction
- setting up the IOQ with an execute channel program request which points to an INCH command IOCD
- setting the bit in IOQ.FLGS specifying execute channel program
- storing a positive timeout value in seconds into IOQ.FCT4
- taking the queue request return to IOCS

14.5.6.2.3 Read

Read commands are processed in EP5 by:

- calling S.IOCS13 to build an IOQ
- calling S.IOCS33 to update the FAT and FCB and return the absolute starting block number for the requested file
- computing a seek address
- calling S.IOCS12 to set up a Seek command IOCD within the IOQ
- calling H.IOCS,40 to set up a Read IOCD or IOCDs within the IOQ
- building mode byte within the IOQ
- storing a positive timeout value in seconds into IOQ.FCT4
- taking the queue request return to IOCS

14.5.6.2.4 Write

Write commands are processed in EP5 by:

- calling S.IOCS13 to build an IOQ
- calling S.IOCS33 to update the FAT and FCB and return the absolute starting block number for the requested transfer file
- computing a seek address
- calling S.IOCS12 to set up a Seek command IOCD within the IOQ
- calling H.IOCS,40 to set up a Write IOCD or IOCDs within the IOQ
- building mode byte within the IOQ
- storing a positive timeout value in seconds into IOQ.FCT4
- taking the queue request return to IOCS

14.5.6.2.5 Rewind

Rewind commands are processed in EP5 by:

- zeroing the current disc address in the FAT
- setting the Beginning of Media (BOM) flag in the FCB
- taking the service complete return to IOCS

14.5.6.2.6 Write End-of-File

Write End-of-File commands are processed in EP5 by taking the service complete return to IOCS with no other action.

14.5.6.2.7 Execute Channel Program

Execute Channel Program commands are processed in EP5 by:

- calling S.IOCS13 to build an IOQ
- setting the bit in IOQ.FLGS specifying execute channel program
- storing a positive timeout value in seconds into IOQ.FCT4
- taking the queue request return to IOCS

14.5.6.2.8 Advance Record

Advance Record commands are processed in EP5 by:

- calling S.IOCS33 with a count of +1 to update the FAT and FCB
- taking the service complete return to IOCS

14.5.6.2.9 Advance File

Advance File commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.10 Backspace Record

Backspace Record commands are processed in EP5 by:

- calling S.IOCS33 with a count of -1 to update the FAT and FCB
- taking the service complete return to IOCS

14.5.6.2.11 Backspace File

Backspace File commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.12 Upspace

Upspace commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.13 Erase

Erase commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.14 Eject

Eject commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.15 Close File

Close File commands are processed in EP5 by taking the service complete return to IOCS.

14.5.6.2.16 Reserve

Reserve commands are processed in EP5 by:

- calling S.IOCS13 to build an IOQ
- calling S.IOCS12 to set up a Reserve IOCD within the IOQ
- setting bit in IOQ specifying Reserve command
- building mode byte within the IOQ
- storing timeout value in seconds into IOQ.FCT4
- taking queue request return to IOCS

14.5.6.2.17 Release

Release commands are processed in EP5 by:

- calling S.IOCS13 to build an IOQ
- calling S.IOCS12 to set up a Release IOCD within the IOQ
- setting bit in IOQ specifying Release command
- building mode byte within the IOQ
- storing timeout value in seconds into IOQ.FCT4
- taking queue request return

14.5.7 Entry Point 6 - Post Transfer Processing Service

Entry point 6 of H.DP02 is a dummy entry point and should never be called.

14.5.8 Entry Point 7 - Error Processing for Operator Intervention

Entry point 7 of H.DP02 does an immediate TRSW R0 return thus disallowing any operator intervention error retry.

14.5.9 Entry Point 8 - SYSGEN Initialization

Entry point 8 of H.DP02 is entered by SYSGEN when SYSGEN encounters an extended I/O disc specified in the Controller Definition Table (CDT). It is called only once regardless of the number of channels configured with extended I/O discs.

Calling Sequence:

```
BL    *8W,R2    R2 = H.DP02 Handler Address Table (HAT) address
```

Registers Passed In Calling Sequence:

R7 = CDT address

External Routines Called: None.

Function Performed:

Entry point 8 performs the following functions for all extended I/O disc CDTs, UDTs and context blocks:

- provides memory space allocation for context blocks
- stores handler Halfword Address Table (HAT) address into CDT.SIHA
- stores context block address into UDT.CBLK
- stores CDT address into context block
- stores max transfer byte count into UDT.MBX
- stores channel address and device subaddress into context block
- stores SYSGEN specifiable timeout value (DPTOV) into context block for dual ported devices
- computes and stores number of sectors per cylinder into context block
- stores service interrupt vector address into context block
- stores channel IOCD vector address location into context block
- converts EP5 time out values to seconds and stores into EP5 time out table

Exit Sequence:

```
M.XIR    HAT    (Special SYSGEN Initialization Termination Macro)
```

Note: 'HAT' is a label equated to 0 relative within the H.DP02 handler.

Registers Passed in Exit Sequence: None.

14.5.10 Error Processing for Conventional I/O Requests

When an I/O operation completes, the 16 status bits presented in the Status Doubleword are checked for error conditions. If only Channel End (CE) and Device End (DE) are presented, the I/O operation is considered complete with no errors and normal post-access processing is continued. If other bits are found to be set, the handler issues a SENSE command to gain additional information regarding the error. The Sense information is stored in the context block for the device (see Section 14.2.1.3.2). The status bits, sense bits, and drive status bits are then interrogated to determine appropriate action as described next.

14.5.10.1 Abort the I/O Request

The following Status bits, Sense bits, and Drive Status bits will abort the I/O request:

Status Bit	Meaning
2	Incorrect Length
3	Program Check
4	Channel Data Check
5	Channel Control Check

Sense Bit	Meaning
8	Command Reject
9	Operator Intervention Required*
10	Spare
12	Data Check
14	Disc Format Error
18	Write Protect Error**
19	Write Lock Error
20	Mode Check
21	Invalid Memory Address
23	Chaining Error
30	Reserve Track Access Error

Drive Status Bit	Meaning
12	Write Protected**

All of the above errors will cause the Error Condition Found bit (1) in Word 3 of the FCB to be set which indicates to the user that the I/O operation completed abnormally.

14.5.10.2 Retry the I/O Request

The following Sense bits will cause five (5) retries of the whole IOCD list before aborting the I/O request:

Sense Bit	Meaning
11	Equipment Check
13	Data Over or Under Run
27	ECC Error in Sector Label
29	ECC Error in Track Label

* This condition will also cause the Device Inoperable bit (4) in Word 3 of the FCB to be set.

** If I/O request was a write, this condition will also cause the Write Protection Violation bit (3) in Word 3 of the FCB to be set.

I/O requests which timeout will be retried five (5) times.

If error retry is unsuccessful, the Error Condition Found bit (1) in Word 3 of the FCB will be set which indicates to the user that the I/O operation completed abnormally.

There exists the capability within the disc processor to retry erring read commands using various combinations of carriage and timing offsets as defined by the "Mode" byte specified by the Load Mode Register command (see Section 14.2.1.3.2). This capability is extremely useful for recovering data from transportable disc packs and shall be incorporated in a future version of the XIO disc handler.

14.5.10.3 Perform Read ECC Correction Logic

The following Sense bits will cause the ECC logic to be performed:

Sense Bit	Meaning
28	ECC Error in Data

When an ECC Error in Data is detected, the Read ECC command is issued to obtain correction information. This information is invalid if the Status returned from the Read ECC command contains other than Channel End (CE) and Device End (DE) or if the bit displacement exceeds the sector size. In either case the error correction logic is bypassed and error retry is initiated as per Section 14.5.10.2.

When the correction information is determined to be valid, the bit displacement is used to locate the address of the erring bits. If the address is outside the users buffer, no error correcting takes place and the I/O request is considered complete without error. If the address is within the users buffer, the data is corrected. The IOCD list is then modified to begin data transfer from the point of interruption and the I/O transfer is continued.

14.5.10.4 Rezero and Retry

The following Sense bits and Drive Status bits will cause drive recalibration and retry of the whole IOCD list:

Sense bit	Meaning
25	Disc Addressing or Seek Error

Drive Status Bit	Meaning
8	Disc Drive Detected a Fault
9	Seek Error

The Sense bits and Drive Status bits not specifically mentioned above are considered 'don't care' items and normal post-access processing is continued.

14.5.11 Error Processing for Execute Channel Program Requests

Error processing for execute channel programs is impossible to perform at the handler level; therefore, only a minimal amount of support can be given. Information returned will consist of Status Words 1 and 2 passed to FCB Words 11 and 12. If bits other than Channel End (CE) and Device End (DE) are present, bit 1 (error condition found) of Word 3 in the FCB will be set. Bits 16-31 of Word 3 are valid. Sense information generation, error correction, and error retry are the responsibility of the user.

14.6 SYSGEN Directives

SYSGEN 'CONTROLLER' and 'DEVICE' directives are used to define XIO discs configured in an MPX-32 system. See Volume III of the MPX-32 Reference Manual for details.

Each disc drive attached to a disc processor is assigned a unique unit address in the range 0 to 7. This unit address is determined by a unit address plug installed in the drive or by switches contained in the drive depending upon drive type. The device subaddress specified on the SYSGEN 'DEVICE' directive is the unit address from above multiplied by 2 and converted to its hexadecimal equivalent, i.e., unit address 7 is specified as 0E. Sample SYSGEN directives follow:

```
CONTROLLER=DM08, PRIORITY=16, CLASS=F, MUX=XIO, HANDLER=(H.EXIO,I) 1

DEVICE=00, DTC=DM, HANDLER=(H.DP02,S), DISC=MH0402

DEVICE=02, DTC=DM, DISC=MH080 3

DEVICE=04, DTC=DM, DISC=(MH040,D), DPTOV=34

DEVICE=06, DTC=DM, DISC=CD0325
```

1. The CONTROLLER directive specifies an 'F' class, extended I/O moving head disc on channel 8 at priority level 16. The handler name (interrupt fielder) is H.EXIO and is channel reentrant (one copy per channel).

2. This DEVICE directive specifies a 40 MB moving head disc assigned to subaddress 00. The handler is H.DP02 and is system reentrant (one copy per system).

3. This DEVICE directive specifies an 80 MB moving head disc assigned to subaddress 02.

4. This DEVICE directive specifies a 40 MB dual-ported moving head disc assigned to subaddress 04 with a dual-port timeout value of three (3) timer units.

5. This DEVICE directive specifies a 32 MB cartridge module drive. The removable media portion (MHD) of the cartridge module drive is assigned subaddress 06 and the captive media portion (FHD) is assigned subaddress 07 (by SYSGEN).

15.0 INPUT/OUTPUT PROCESSOR (IOP) HANDLER

15.1 Introduction

This chapter describes in functional terms the software requirements for integrating Input/Output Processor (IOP) hardware into MPX-32. It also serves as a guide to programmers whose task is to design device handler programs for IOP controllers.

This chapter is not intended to supply detailed IOP hardware information. Such information can be found in the IOP Technical Manual and in the individual IOP Controller Technical Manuals.

15.2 Hardware Overview

15.2.1 Hardware Block Diagram Description

A block diagram of the IOP and associated hardware elements is contained in Figure 15-1 and is explained in the paragraphs which follow.

15.2.1.1 The Input/Output Processor (IOP)

The IOP is a multifunction processor which plugs into the SelBUS and requires two SelBUS addresses. The IOP is an I/O multiplexing channel capable of interfacing up to 16 IOP controllers to the SelBUS. Each of the 16 controllers can communicate with multiple devices.

The IOP also provides two ports for the connection of two TTY compatible devices, each of which can function as both System Control Panel and operator console.

Optionally, the IOP can also provide the same real-time interrupt input capability that one RTOM board provides with the exception that only 4 external interrupts are available to the user.

15.2.1.2 IOP Control Panel/Operator Console

The IOP eliminates the need for a System Control Panel by providing the most commonly used System Control Panel functions as interactive commands on a CRT or TTY which is connected to one of the two integral IOP ports. The same CRT or TTY also functions as the operator console.

Each port can operate in the full duplex mode, and will support a TTY compatible device or a modem for remote operation. One port is designated as the master port and the second port is designated as the slave port.

Upon system power-up, the devices connected to both ports are automatically activated in the control panel mode. While in control panel mode, the CRT or TTY is driven strictly by IOP firmware. Automatic switching from control panel mode to operator console mode is effected when the CPU enters the RUN mode. In the operator console mode, the CRT or TTY is driven by software. Upon transition of the CPU from RUN to

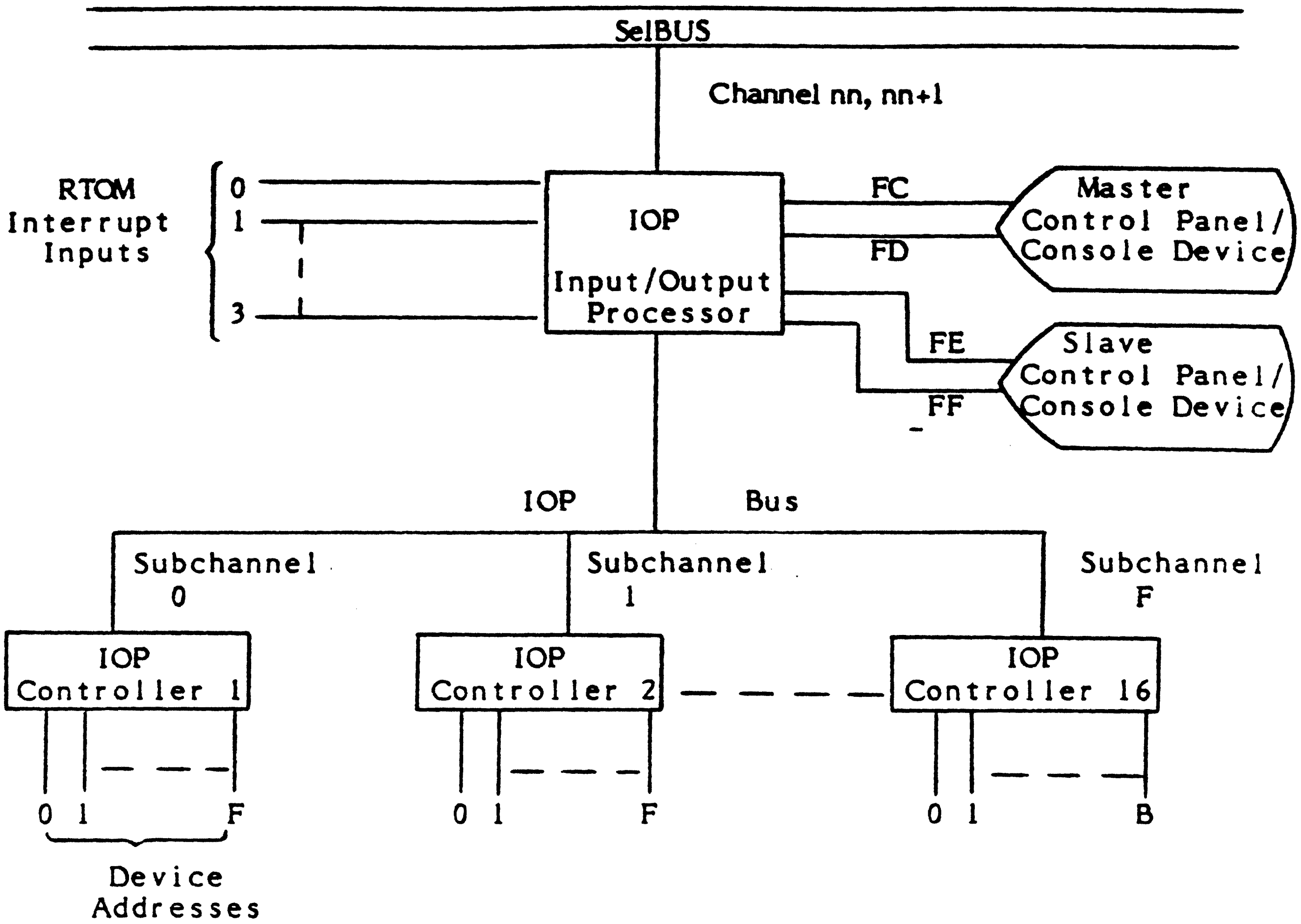


Figure 15-1. IOP Hardware Block Diagram

HALT, the CRT or TTY device is automatically switched to the control panel mode. The transition from operator console mode to control panel mode can also be effected via software (i.e., operator command).

For a list of control panel mode displays and interactive commands, consult the IOP Technical Manual. A description of the operation of the slave port can also be found in the IOP Technical Manual.

15.2.1.3 IOP Controllers

The IOP supports up to 16 independent device controllers. Each controller connects to the multipurpose bus. All controllers connected to a particular IOP share both the channel address and service interrupt (SI) priority of their parent IOP.

Each IOP controller has a subchannel address which is its multipurpose bus address. Each IOP controller supports up to 16 devices except the controller on subchannel F, since four subchannel F addresses are committed to the control panel/console devices.

Even though the IOP is theoretically capable of addressing 254 devices, the amount of RAM configured on the IOP limits IOP addressability to 124 devices.

15.2.1.4 IOP Support for Real-Time Interrupts

As an option, the IOP can, in addition to its other previously described functions, assume the role of an RTOM. That is, it can process up to 16 real-time priority interrupt levels.

If the RTOM function is required for a CONCEPT/32 CPU, the appropriate real-time interrupt levels should be configured in CPU scratchpad using the odd SelBUS address of the IOP's even/odd address pair. If the RTOM function is required for a 32/7x CPU, the IOP's even/odd address pair must be 78/79 with the appropriate interrupt levels configured in CPU scratchpad using address 79. If the RTOM function is not required on a 32/7x CPU, the IOP SelBUS address pair should start at less than 78.

15.2.2 IOP Device Addressing

IOP device addresses have the following format:

CCSD, where
CC = Channel/IOP Address
S = Subchannel/Controller Address
D = Device Address

15.2.3 IOP Protocol

IOP devices use standard F-class I/O (XIO) instructions with the following exceptions: Enable WCS (EWCS), Write Channel WCS (WCWCS), and Grab Controller (GRIO) except in the case of the console which uses a special form of GRIO to facilitate usage of the MPX-32 System Debugger.

Operation of the IOP/XIO instructions is explained in the IOP Technical Manual.

15.2.4 Detailed IOP Hardware Information

For detailed IOP hardware information, see the IOP Technical Manual.

15.3 Software Overview

15.3.1 Software Block Diagram Description

A block diagram of MPX-32 software related to the IOP and the control of IOP devices is contained in Figure 15-2. The various elements of Figure 15-2 are explained in the paragraphs which follow.

15.3.1.1 IOP Device Handler Programs

IOP device handler programs are designed to provide generalized MPX-32 I/O support for specific IOP device controllers. Each handler consists of six entry points. Each entry point represents a functional block of code as follows:

Entry Point OP. - Operation Code Processor

This entry point performs device specific processing of user I/O requests including the formatting of I/O queue entries.

Entry Point IQ. - I/O Queue Processor

This entry point processes I/O queue entries in order to initiate actual device I/O.

Entry Point SI. - Service Interrupt Processor

This entry point performs all service interrupt processing, status reporting, and automatic error recovery if applicable.

Entry Point LI. - Lost Interrupt Processor

This entry point performs handler recovery from a lost service interrupt due to device timeout.

Entry Point PX. - Special Post Transfer Processor

This entry point performs optional handler related processing after completion of the I/O request but before return to the requesting task. This entry point operates at task priority level.

Entry Point SG. - SYSGEN Initialization Processor

Executed by SYSGEN during MPX-32 image construction and subsequently overlayed. Builds and initializes DCAs, and initializes other MPX-32 I/O data structures.

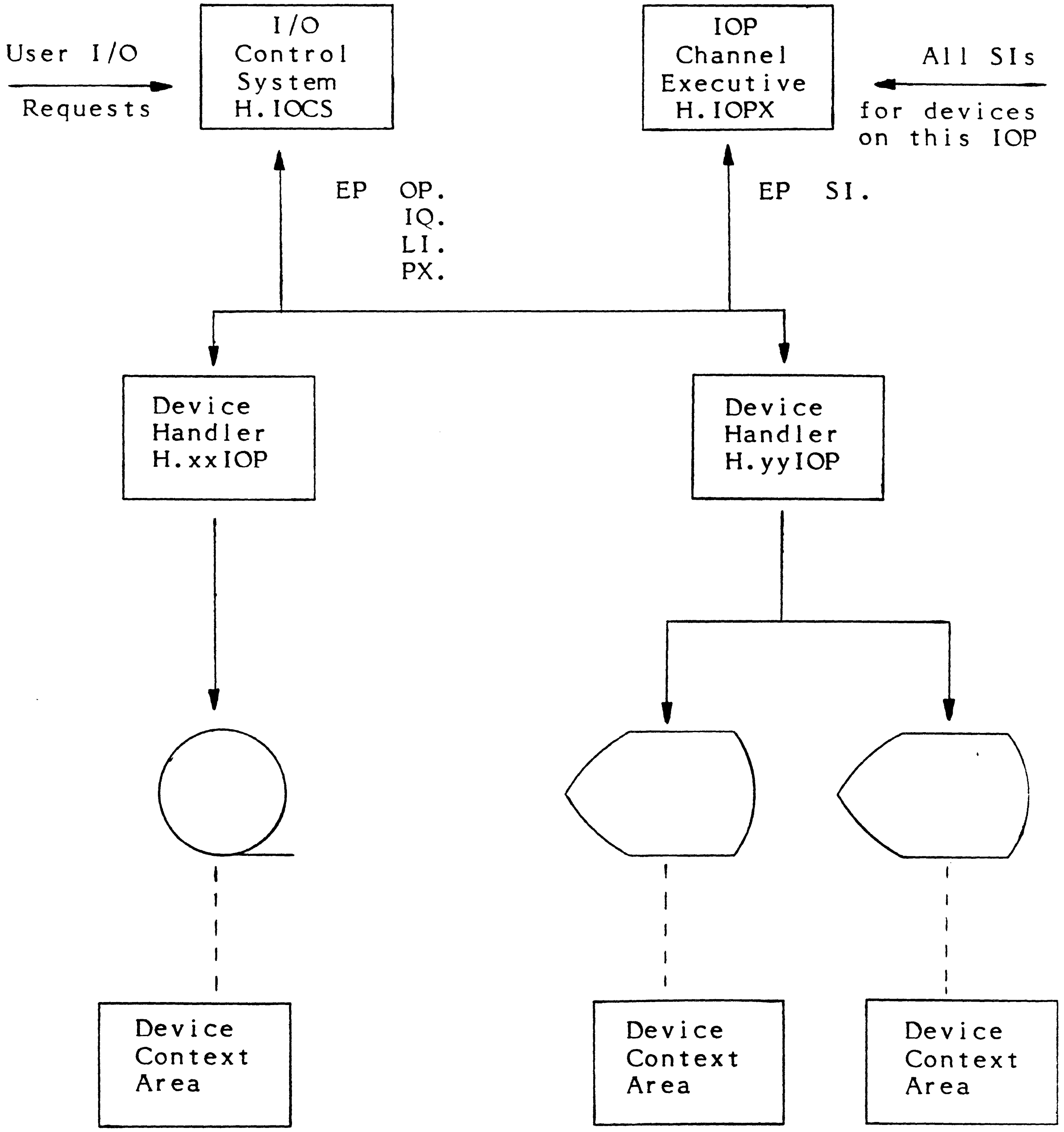


Figure 15-2. Software Block Diagram

15.3.1.2 IOP Device Context Areas

Associated with each device connected to an IOP controller is an IOP Device Context Area (DCA). The IOP DCA allows IOP device handlers to be system level reentrant by containing pointers to system tables associated with the device, and by providing temporary storage locations for each device.

The configuration of the DCA is discussed in detail in subsequent paragraphs.

15.3.1.3 IOP Channel Executive

The IOP Channel Executive program fields interrupts on behalf of all controllers connected to a particular IOP. Subsequent to fielding an interrupt the IOP Channel Executive passes control to the interrupt servicing entry point of the appropriate IOP device handler. The IOP Channel Executive locates the correct IOP handler by locating the appropriate UDT.

Returns from handler interrupt servicing entry points are made via the IOP Channel Executive.

15.3.1.4 H.IOCS/IOP Handler Interaction

H.IOCS makes calls to entry points OP., IQ., LI., and PX. of IOP device handlers to perform the device specific processing required when users execute I/O requests.

15.4 Required Data Structure Additions and Modifications for IOP Implementation

15.4.1 Data Structure Additions

15.4.1.1 Channel Definition Table (CHT)

The CHT as shown in Section 2.4.11 of the MPX-32 Technical Manual will be constructed to support the IOP Channel Executive program and IOP Handlers. The first 14 words will include a register save area, CHT.REGS, and a service interrupt control area comprised of CHT.OPSD, CHT.NPSD, CHT.IOCL, and CHT.STAT. Locating these items in the CHT serves to centralize them for easy scrutiny. CHT.REGS must begin on a register file boundary.

Fields CHT.IPL, CHT.CHAN, and CHT.FLGS will provide IOP attribute descriptors.

Fields CHT.CDT0 through CHT.CDTF will contain the addresses of the CDT entries for the controllers connected to the corresponding IOP. These addresses will provide for the fastest possible service interrupt response times by allowing the IOP Channel Executive the most convenient access method to the IOP handler interrupt servicing entry point. These addresses will be used together with the analogous UDT addresses (CDT.UT0 through CDT.UTF) described in Section 2.4.5 of the MPX-32 Technical Manual. Entries for unimplemented controllers will be set to zero.

Field CHT.STDW is a double word location into which the IOP will store its channel status.

Field CHT.EXIT contains the address of the exit entry point in the corresponding IOP Channel Executive program. IOP handler interrupt servicing entry points will branch to the exit procedure via the address contained in CHT.EXIT.

Field CDT.INCH contains the address of the channel initialization entry point in the corresponding IOP Channel Executive program.

Only IOPs will have a CHT entry, and these entries will not affect the operation of any currently implemented device handlers.

15.4.2 Data Structure Modifications

15.4.2.1 Controller Definition Table (CDT)

The CDT is modified as shown in Section 2.4.5 of the MPX-32 Technical Manual. First, to allow for the possibility of an I/O queue per UDT (i.e., per device), bit 0 of CDT.IOST when set indicates that no I/O queue entries are linked to the corresponding CDT.

Second, bit 6 of CDT.FLGS, when set, indicates that the corresponding CDT represents an IOP controller.

Third, in order to achieve the fastest possible IOP device interrupt response times, each CDT contains the addresses of all associated UDTs in ascending order of device address. Entries for unimplemented device addresses contain zero.

These modifications will not affect operation of existing device handlers.

15.4.2.2 Unit Definition Table (UDT)

Several significant modifications have been made to the UDT for IOP implementation as shown in Section 2.4.6 of the MPX-32 Technical Manual and are described in the following paragraphs.

To allow for the queueing of I/O requests on a per device basis, the UDT contains the standard MPX-32 linked list head cell format as represented by fields UDT.FIOQ, UDT.BIOQ, UDT.LPRI, and UDT.IOCT. In addition, bit 0 of UDT.STA2, a new status byte, indicates that an I/O queue is linked from the corresponding UDT when set.

Bit 1 of UDT.STA2 indicates, when set, that the corresponding UDT represents an IOP device.

These modifications will not affect operation of existing device handlers.

15.4.2.3 Communication Region Variables

The following communication region variables have been added to support the CHT:

C.CHTA	Address of Channel Definition Table (CHT)
C.CHTN	Number of Channel Definition Table (CHT) entries

15.5 Required Modifications to Existing Software For IOP Implementation

15.5.1 IOCS Modifications

IOCS is modified to take into account the differences in IOCS to IOP handler communication, I/O queueing on a per device basis, and other features new to IOP handlers.

15.5.1.1 IOCS to IOP Handler Communications

15.5.1.1.1 IOCS Calls to IOP Handler Entry Points OP. and IQ.

H.IOCS,29 is modified to determine whether calls to the opcode processor entry point (OP.) or the I/O queue processor entry point (IQ.) are for IOP devices, and, if so , to locate the handler via the UDT or initialized I/O queue entry (IOQ.DCAA) rather than via the CDT.

15.5.1.1.2 IOCS Calls to IOP Handler Entry Point PX.

S.IOCS1 is modified to determine whether calls to the special device post processor entry point (PX.) are for IOP devices, and, if so, to locate the handler via the I/O entry queue (IOQ.DCAA) rather than via the CDT.

15.5.1.1.3 IOCS Calls to IOP Handler Entry Point LI.

S.IOCS5 is modified to determine whether calls to the lost interrupt processor entry point (LI.) are for IOP devices, and, if so, to locate the handler via the UDT rather than via the CDT.

15.5.1.2 I/O Queue Entry Linking and Unlinking

15.5.1.2.1 Linking an Entry to an I/O Queue

H.IOCS,29 is modified to determine whether I/O queue entries are to be linked to the CDT or the UDT, and present S.EXEC8 (common link subroutine) with the appropriate head cell address.

15.5.1.2.2 Unlinking an Entry from an I/O Queue

S.IOCS3 is modified to determine whether I/O queue entries are to be unlinked from the CDT or the UDT, and to perform the unlinking accordingly.

15.5.1.3 Kill I/O Queue Search

H.IOCS,38 is modified to search for I/O queue entries linked from UDTs as well as from CDTs.

15.5.2 SYSGEN Modifications

15.5.2.1 Data Structure Requirements

SYSGEN has been modified to support the data structure additions and modifications as described in Section 15.4 of this chapter.

15.5.2.2 Device Requirements

Samples of SYSGEN directives required for IOP support are shown in Figure 15-3.

15.5.2.2.1 IOP Controller Directive

All IOP controllers connected to the same IOP have the same channel address. One copy of the IOP Channel Executive program (H.IOPX) is loaded per IOP. SYSGEN keys off the MUX=IOP directive for this purpose unless the optional "HANDLER=XXXX" keyword is present on the directive. In addition, one CHT entry is created per IOP.

The "SUBCH=n" keyword indicates which IOP subchannel a controller is connected to, and is used to verify proper device address specifications on subsequent device directives (i.e., the subchannel should match the first device address digit).

Note that the device mnemonic (and therefore device type) may be different on the controller and device directives.

15.5.2.2.2 IOP Device Directive

Each IOP device directive contains the name of the handler which services it. However, IOP handlers are assumed to be system level reentrant, and, therefore, SYSGEN configures only one copy of each required handler.

```
CONTROLLER = LF12, PRIORITY=1A, CLASS=F, MUX=IOP, SUBCH=1
DEVICE=(10,2), DTC=FL, HANDLER=H.FL0IOP
DEVICE=18, DTC=LP, HANDLER=H.LP0IOP

                    FL1210, FL1211         Floppy Disks
                    LP1218                 Line Printer

CONTROLLER=TY12, PRIORITY=1A, CLASS=F, MUX=IOP, SUBCH=2
DEVICE=20, DTC=TY, HANDLER=H.AL0IOP
DEVICE=21, DTC=TY, HANDLER=H.AL0IOP

                    TY1220, TY1221         8-Line Asynchs

CONTROLLER=CT12, PRIORITY=1A, CLASS=F, MUX=IOP, SUBCH=0
DEVICE=00+01, DTC=CT,    HANDLER=H.CT0IOP

                    CT00                   Console Terminal

PRIORITY=13, RTOM=(13,0C), PROGRAM=H.IP13
```

Software interrupt level 13 corresponding to IOP channel 13 (the odd address of even/odd address pair 12 and 13) and subaddress C (which is the 1's complement of the relative physical priority, 3).

Figure 15-3. Sample IOP SYSGEN Directives

15.5.2.2.3 IOP RTOM Function Directive

SYSGEN does not require modification to process priority interrupt directives for the IOP and construct the required scratch pad cell images. Note that the interrupt physical address must be an odd number address of an even/odd address pair, and that the even number address must appear in the controller directive set. Since the interrupt physical subaddress is the one's complement of the relative physical priority (0-F), the unused and superfluous first hexdigit should be zero.

15.5.2.3 IOP SYSGEN Initialization

SYSGEN calls the initialization entry point of the IOP Channel Executive and of each IOP handler with the address of the CHT in register 1 (R3).

15.6 New Software Required For IOP Implementation

15.6.1 IOP Channel Executive - H.IOPX

15.6.1.1 Interrupt Fielding Entry Point

When an interrupt occurs on an IOP channel the interrupt fielding entry point in the corresponding copy of H.IOPX will automatically be entered. The interrupt fielding entry point will, after updating the global interrupt count and saving all registers, locate the handler servicing the interrupting device via the appropriate CHT, CDT, and UDT entries. Control will then be passed in the unmapped mode to the service interrupt processor (entry point SI.) of the device handler.

15.6.1.2 Interrupt Exit Entry Point

In order to exit an IOP interrupt level, the service interrupt processor of the IOP handler will call the interrupt exit entry point of the appropriate copy of the IOP Channel Executive. The address of the interrupt exit entry point will be located via the CDT. The interrupt exit entry point will restore the mapped mode, deactivate the interrupt level, and branch to S.EXEC5.

15.6.1.3 Initialize Channel Entry Point

The initialize channel entry point is called only once to perform a channel initialization (INCH) to the IOP. This consists of a single IOCD which indicates to the IOP where interrupt status from the IOP is to be posted. This entry point is called by the device handler of the first device on an IOP to be opened.

15.6.1.4 Initialization Entry Point

The initialization entry point is an overlayable procedure called by SYSGEN to initialize certain local and global variables as follow:

Variable Name	Description
N/A	SI vector location
CHT.NPSD	New PSD address in Service Interrupt Control Area (SICA)
CHT.EXIT	Address of H.IOPX interrupt exit entry point
CHT.INCH	Address of H.IOPX initialize channel entry point
IOPADDR	IOP channel address
INCHIOCD	IOCD used to perform initialize channel (INCH)

15.6.2 IOP Device Handlers

15.6.2.1 IOP Device Context Area - H.DCAxxx

15.6.2.1.1 Device Context Area Configuration

The configuration of a prototype DCA is shown in Figure 15-4. The purpose of each element of the DCA is also explained in the figure.

DCA.SIZE	REZ	1W	Size of this DCA
DCA.UADD	REZ	1W	Address 1 This Unit
DCA.CHTA	REZ	1W	CHT Entry Address This Unit
DCA.CDTA	REZ	1W	CDT Entry Address This Unit
DCA.UDTA	REZ	1W	UDT Entry Address This Unit
DCA.IOQA	REZ	1W	Current IOQ Address This Unit
DCA.LINC	REZ	1W	Lost Interrupt This Unit
DCA.SINC	REZ	1W	Spurious Interrupt Count This Unit
DCA.RETR	REZ	1W	Retry Count This Unit
DCA.FLAG	REZ	1W	Bit Flags This Unit ← [illegible]
DCA.TIMO	REZ	16W	Timeout Value Table - 1 entry, 1 opcode
DCA.SENI	REZ	1W	IOCD for Sense Status
DCA.SENS	REZ	NB	Sense Status This Unit

Figure 15-4. IOP Device Context Area

15.6.2.1.2 Device Context Area Construction and Initialization

Entry point SG. of each IOP handler will construct and initialize the appropriate number of DCAs. Space for the DCAs will be allocated at assembly time via assembly in entry point SG. of a system macro named DCA.DATA. A repeat count will be set representing the maximum number of that type of device which is possible to configure. During SYSGEN initialization, entry point SG. will scan the UDT for its handler name, and will initialize only the required number via system macros DCA.INI1 and DCA.INI2. Entry point SG. will then cause the remaining DCAs to be overlayed by SYSGEN. Provisions will be made in the design of the aforementioned macros to allow for additional space in the DCA and for insertion of user code to perform any required unique initialization.

15.6.2.1.3 Device Context Area Element Access

Elements of a DCA will be referenced via symbolic offsets which are defined in the system macro DCA.EQUS. The names of the symbolic offsets defined in DCA.EQUS will be the labels shown in Figure 15-4.

15.6.2.2 IOP Device Handler Programs

Section 15.8 contains an IOP device handler prototype or skeleton which can be "filled in" with device specific code to produce a unique IOP device handler. This handler prototype should be used as a time saving device, a learning tool, and a guide to coding standards to be followed in all IOP device handlers.

Section 15.9 contains an actual sample IOP device handler.

15.6.3 Miscellaneous New Software

15.6.3.1 Report I/O Complete - S.IOCS29

This subroutine is called from entry point 1 of IOP handlers to report I/O completion to the MPX-32 Executive program. S.IOCS29 determines the type of I/O starting and call the appropriate Executive subroutine.

15.6.3.2 Mark Units Offline - S.IOCS31

This subroutine is called from entry point 4 of IOP handlers when it is determined that a device is broken. S.IOCS31 marks all UDT entries of a particular controller off-line and malfunctioning, and marks the CDT malfunctioning.

15.7 IOP Floppy Disc Handler Program (H.FLIOP)

The IOP floppy disc handler program (H.FLIOP) is system level reentrant, i.e., only one copy of H.FLIOP is required regardless of the number of floppy discs or IOPs configured in a system. Reentrancy is accomplished via device context areas (DCAs). One DCA is created and initialized in the SYSGEN initialization entry point for each floppy disc configured.

As with all IOP device handlers, interrupts are fielded by the IOP channel executive program (H.IOPX), and then turned over to the service interupt entry point of H.FLIOP for processing.

15.7.1 Functional Characteristics By Entry Point

15.7.1.1 Opcode Processor (OP.)

The following operations are supported by H.FLIOP:

Operation	User Call	Remarks
Open	M.FILE	1. Performs IOP channel initialization if first IOP operation. 2. Initializes FAT. Rewind is implicit
Rewind	M.RWND	Resets current disc address in FAT to zero.
Read	M.READ	1. Resets EOF detection flag if user inhibits data formatting, if read is blocked, or if read is random access. 2. Saves original FAT parameters in I/O queue for EOF processing in service interrupt entry point. 3. Updates FAT for read operation. 4. Builds IOCL consisting of SEEK + READ commands with data chaining as necessary.
Write	M.WRIT	Same processing as for read with exceptions: (1) No EOF processing considerations (2) IOCL consists of SEEK + WRITE commands.
Write End-of-File	M.WEOF	Same processing as for write except that X'0F000000' is written to the current block.
Execute Channel Program	N/A	Not supported in MPX-32 Release 1.5.
Advance Record	M.FWRD	Adds one (block) to the current disc address in FAT.
Advance File	M.FWRD	Same processing as for read with exception: (1) I/O queue indicators set such that read is repeated until EOF detected. (2) Read buffer is local to handler and read count is 1 word.
Backspace Record	M.BKSP	Subtracts one (block) from the current disc address in FAT.

Operation	User Call	Remarks
Backspace File	M.BKSP	Same processing as for Advance File except that read is in the backward direction.
Upspace	M.UPSP	Formats diskette to MPX-32 format which is 256 bytes per sector, double density.

15.7.1.2 I/O Queue Processor (IQ.)

The I/O queue processor performs standard I/O queue scheduling as outlined in the prototype IOP device handler.

15.7.1.3 Service Interrupt Processor (SI.)

The service interrupt processor performs standard I/O interrupt processing for the floppy disc as outlined in the prototype IOP device handler with the following additional considerations.

15.7.1.3.1 Read End-of-File Processing

Unless explicitly inhibited by the user (via bit 2, FCB.CBRA) or implicitly inhibited (via blocked or random access operation) SI. will look for the pseudo EOF character (X'0F000000') when processing an interrupt for a read. The first word of each 192 word block read is examined for the pseudo EOF character. If found, the EOF indicator in the I/O queue entry is set. This will eventually cause the EOF indicator in the users FCB to be set.

Since the current disc address in the FAT is updated for the read prior to the operation, and since the EOF is likely to be detected on the first block (or some block other than the last block read), the current disc address in the FAT will have to be adjusted after the EOF is detected. In order to accomplish the adjustment, the original FAT parameters (prior to being updated for the read) are saved in the I/O queue entry. These values are then updated via subroutine S.FLIO1 upon EOF detection.

The FAT is not available to the service interrupt entry point (because it runs unmapped) so actual updating of the FAT is performed in the special post transfer processor (PX.) which runs at task priority level.

15.7.1.3.2 Backspace File and Skip File Processing

For both these operations, a one block read is set up in the opcode processor (OP.). During service interrupt processing, the first word of the block is tested for EOF. If EOF was not detected, the original IOCL is updated to point to the next or previous block, and the original FAT parameters now in the I/O queue entry are updated (via subroutine S.FLIO1). The IOCL is then reexecuted. When the EOF is detected, processing is as described in Section 15.7.1.3.1.

15.7.1.3.3 Error Recovery Processing

If interrupt status indicates an error condition (i.e., unit check reported) a sense status command is executed to gather sense data. Refer to the Line Printer/Floppy Controller Technical Manual for a listing of sense data generated by the floppy controller. Error conditions are categorized as unrecoverable error, seek error and data error. If unrecoverable errors are reported, no retry is attempted. If a seek error is reported, the floppy is rezeroed followed by up to 5 reexecutions of the user's original request. If data errors are reported, the user's original request is reexecuted up to 5 times. If still unsuccessful, the seek error recovery procedure is followed.

15.7.1.4 Lost Interrupt Processor (LI.)

The Lost Interrupt Processor performs standard lost interrupt and kill I/O processing as in the prototype IOP device handler.

15.7.1.5 Special Post Transfer Processor (PX.)

The special post transfer processor updates the FAT from I/O queue entry parameters in order to complete processing for a read in which EOF was detected, a backspace file, or an advance file operation. See section 15.7.1.3.1.

15.7.1.6 SYSGEN Initialization Processor (SG.)

The SYSGEN Initialization Processor creates (at assembly time) and initializes (at SYSGEN time) floppy disc device context areas (DCAs). Currently, up to two DCAs will be created and initialized. If more than 2 floppy discs are to be configured in a system, the count on the repeat (REPT) assembler directive immediately following label 'DCASTART' must be increased, and H.FLIOP reassembled.

```
*XXXXXXXXXXXXXXXXXX HANDLER          RELEASE 1.5 MPX        HEADER  H.XXIOP
        SPACE     10
************************************************************************
*                                                                      *
*           >>>>> RESTRICTED RIGHTS LEGEND <<<<<                       *
*                                                                      *
*       USE, DUPLICATION, OR DISCLOSURE IS SUBJECT TO THE              *
*       RESTRICTIONS STATED IN SYSTEMS' LICENSE AGREEMENT              *
*       (FORM NO. 1218) OR, FOR GOVERNMENT CUSTOMERS,                  *
*       DAR 7-104.9A.                                                  *
*                                                                      *
************************************************************************
        TITLE     H.XXIOP          XXXXXXXXXXXXXXXXXXX DEVICE HANDLER
        PROGRAM   H.XXIOP
************************************************************************
*                                                                      *
*           XXXXXXXXXXXXXXXXXXXX HANDLER PROGRAM                       *
*                                                                      *
*    THIS HANDLER IS SYSTEM REENTRANT WHICH MEANS THAT ONLY ONE COPY   *
*    SHOULD BE CONFIGURED INTO AN MPX-32 SYSTEM REGARDLESS OF THE      *
*    NUMBER OF DEVICES OF THIS TYPE AND THE NUMBER OF IOP'S CONFIGURED *
*    INTO THE SYSTEM.                                                  *
*                                                                      *
************************************************************************
        SPACE     3
        M.EQUS
        M.TBLS
        DCA.EQUS
        EXT       S.EXEC5
        EXT       S.IOCS3
        EXT       S.IOCS10
        EXT       S.IOCS13
        EXT       S.IOCS15
        EXT       S.IOCS29
        EXT       S.IOCS31
        EXT       ILOPCODE
        EXT       SERVCOMP
        EXT       IOLINK
        EXT       POSTPROS
        SPACE     3
HAT     DATAW     6                HANDLER ADDRESS TABLE
        ACW       OP.              OPCODE PROCESSOR ADDRESS
        ACW       IQ.              I/O QUEUE PROCESSOR ADDRESS
        ACW       SI.              SERVICE INT PROCESSOR ADDRESS
        ACW       LI.              LOST INTERRUPT PROCESSOR ADDRESS
        ACW       PX.              POST XFER PROCESSOR ADDRESS
*                                  *ADDITIONAL ENTRY POINTS ,IF REQ'D,
*                                  *MUST BE ADDED HERE.
SG.ADDR ACW       SG.              SYSGEN INITIALIZATION PROCESSOR ADDR
*                                  *THIS MUST THE LAST ENTRY POINT
        TITLE     H.XXIOP          OPCODE PROCESSOR
************************************************************************
*                                                                      *
*                 OP. - OPCODE PROCESSOR                               *
*                                                                      *
************************************************************************
*                                                                      *
*    THIS ENTRY POINT IS ACTUALLY A SUBROUTINE EXTENSION OF H.IOCS,29,  *
*    A PORTION OF IOCS LOGIC COMMON TO ALL I/O SERVICES WHICH ARE       *
*    CAPABLE INITIATING A PHYSICAL DEVICE ACCESS. IT IS CALLED TO       *
*    PROCESS THE OPCODE PLUGGED INTO THE FCB BY THE I/O SERVICE         *
*    ORIGINALLY CALLED BY THE USER.                                     *
*                                                                       *
*                                                                       *
*    THE PURPOSE OF OP. IS TO EXAMINE THE OPCODE AND OTHER PERTINENT    *
*    FCB CONTROL SPECIFICATIONS, AND TO INDICATE TO H.IOCS,29 WHAT      *
*    ACTION IS TO BE TAKEN. IN ORDER TO INDICATE WHAT ACTION IS TO BE   *
*    TAKEN, OP. TAKES 1 OF 4 POSSIBLE RETURNS TO H.IOCS,29 AS FOLLOWS:  *
*                                                                       *
*         BU         ILOPCODE      1. OPCODE ILLEGAL FOR THIS DEVICE    *
*         BU         SERVCOMP      2. SERVICE COMPLETE, NO DEV ACCESS REQD*
*         BU         IOLINK        3. LINK REQUEST TO I/O QUEUE         *
*         BU         POSTPROS      4. LINK REQUEST + POST PROCESSING REQD *
*                                                                       *
*                                                                       *
*    IF RETURN 3 (IOLINK) IS TAKEN OP. MUST FIRST:                      *
*                                                                       *
*         1. CALL IOCS SUBROUTINE S.IOCS13 TO ALLOCATE AND INITIALIZE   *
*            AN I/O QUEUE ENTRY. FOUR IOQ ENTRY WORDS, CALLED HANDLER   *
*            FUNCTION WORDS ARE OF SPECIAL INTEREST TO THE PROGRAMMER.  *
*            UPON ENTRY INTO THE OPCODE PROCESSING ROUTINE THEY ARE     *
*            SET UP AS FOLLOWS:                                         *
*                   IOQ.FCT1 - BITS 0-7 = WORD ADJ OPCODE,BITS 8-31 FREE*
*                   IOQ.FCT2 - USER'S LOGICAL DATA ADDRESS              *
*                   IOQ.FCT3 - USER'S BYTE TRANSFER COUNT               *
*                   IOQ.FCT4 - AVAILABLE                                *
*                                                                       *
*         2. BUILD INTO THE I/O QUEUE ENTRY IN THE SPACE RESERVED FOR   *
*            IT, AN I/O COMMAND LIST (IOCL) WITH THE PROPER COMMAND     *
*            CODES AND FLAGS USING IOCS SUBROUTINE S.IOCS12 AND IOCS    *
*            ENTRY POINT H.IOCS,40.                                     *
*                                                                       *
*         3. OR, IF THE USER REQUESTS EXECUTION OF HIS CHANNEL PROGRAM, *
*            VALIDATE THE DATA AREAS, AND ABSOLUTIZE THE DATA ADDRESSES *
*            USING IOCS SUBROUTINE S.IOCS33                             *
*                                                                       *
*                                                                       *
*    TAKING RETURN 4 (POSTPROS) INDICATES THAT AN I/O REQUEST IS TO BE  *
*    LINKED TO THE I/O QUEUE FOR THIS DEVICE AS IN RETURN 3 (IOLINK),   *
*    AND ALSO THAT AFTER THE REQUEST HAS BEEN COMPLETED BUT BEFORE      *
*    RETURN TO OR NOTIFICATION OF THE USER, THE SPECIAL DEVICE POST     *
*    PROCESSING ENTRY POINT (PX.) WILL BE CALLED.                       *
*                                                                       *
*************************************************************************
*                                                                       *
*         CALLER:                      H.IOCS,29                        *
*                                                                       *
*         CALLERS PRIORITY:            TASK LEVEL                       *
*                                                                       *
*         INTERRUPTS:                  UNBLOCKED                        *
*                                                                       *
*         CALLING SEQUENCE:            BL     *1W,X2                    *
*                                                                       *
*         REGISTERS IN:                R1 = FCB ADDRESS
*                                        R2 = MAT ADDRESS                          *
*                                        R3 = UDT ADDRESS                          *
*                                                                                  *
*         RETURN SEQUENCE:               BU    IOLINK   OR,                        *
*                                        BU    SERVCOMP OR,                        *
*                                        BU    ILOPCODE OR,                        *
*                                        BU    POSTPROS  ,'                        *
*                                                                                  *
*         REGISTERS OUT:                 R1 = FCB ADDRESS                          *
*                                                                                  *
************************************************************************************
          SPACE     3
*
** THE OPCODE VECTOR TABLE WHICH FOLLOWS CONTAINS THE ADDRESSES OF
** THE OPCODE PROCESSING PROCEDURES ORDERED BY OPCODE NUMBER. EACH
** ENTRY HAS BIT 0 SET IF THE ASSOCIATED PROCEDURE REQUIRES AN I/O
** QUEUE ENTRY. SYMBOLIC OPCODES ARE ALSO LISTED.
*
OPTAB     EQU       $
          GEN       1/X,31/W(OPEN)  OPEN
OPEN.OP   EQU       0
          GEN       1/X,31/W(RWND)  REWIND
RWND.OP   EQU       1
          GEN       1/X,31/W(READ)  READ RECORD
READ.OP   EQU       2
          GEN       1/X,31/W(WRIT)  WRITE RECORD
WRIT.OP   EQU       3
          GEN       1/X,31/W(WEOF)  WRITE END-OF-FILE RECORD
WEOF.OP   EQU       4
          GEN       1/X,31/W(EXCP)  EXECUTE CHANNEL PROGRAM
EXCP.OP   EQU       5
          GEN       1/X,31/W(ADVR)  ADVANCE RECORD
ADVR.OP   EQU       6
          GEN       1/X,31/W(ADVF)  ADVANCE FILE
ADVF.OP   EQU       7
          GEN       1/X,31/W(BKSR)  BACKSPACE RECORD
BKSR.OP   EQU       8
          GEN       1/X,31/W(BKSF)  BACKSPACE FILE
BKSF.OP   EQU       9
          GEN       1/X,31/W(UPSP)  UPSPACE
UPSP.OP   EQU       10
          GEN       1/X,31/W(ERPT)  ERASE OR PUNCH TRAILER
ERPT.OP   EQU       11
          GEN       1/X,31/W(EJCT)  EJECT
EJCT.OP   EQU       12
          GEN       1/X,31/W(CLSE)  CLOSE
CLSE.OP   EQU       13
          GEN       1/X,31/W(UNUS)  UNUSED
          GEN       1/X,31/W(UNUS)  UNUSED
          SPACE     3
          BOUND     1W
P.        EQU       $

* PROCEDURE TO ALLOCATE AN I/O QUEUE ENTRY,IF NECESSARY, AND
* VECTOR TO THE APPROPIATE OPCODE PROCESSING PROCEDURE

          LB        R2,FCB.OPCD,X1  GET OPCODE FROM FCB
```

```
         SLL       R2,2                WORD ADJUST OPCODE
         TRR       R2,R6               SAVE OPCODE IN R6
         LW        R4,OPTAB,X2         GET OPCODE PROCEDURE ADDRESS
         BNN       OP.0.00             BR IF I/O QUEUE ENTRY NOT REQ'D
         BL        S.IOCS13            GET I/O QUEUE ENTRY
         LW        R2,FCB.IOQA,X1      GET IOQ ENTRY ADDRESS
         STB       R6,IOQ.FCT1,X2      SAVE OPCODE IN IOQ ENTRY
OP.0.00  EQU       $
         LW        R3,UDT.DCAA,X3      GET ADDRESS OF DEVICE CONTEXT AREA
         TRR       R6,R2               RESTORE OPCODE TO R2
         BU        *OPTAB,R2           VECTOR TO OPCODE PROCEDURE
*
** PROCEDURE TO PERFORM OPEN
*
OPEN     EQU       $
*** INITIALIZE IOP CHANNEL IF NECESSARY
OP.1.00  EQU       $
         LW        R2,DCA.CHTA,X3      GET CHT ADDRESS
         TBM       0,CHT.FLGS,X2       HAS INCH BEEN PERFORMED?
         BS        OP.1.05             BR IF IOP ALREADY INCHED
         BL        *CHT.INCH,X2        GO TO H.IOPX TO PERFORM INCH
OP.1.05  EQU       $
         XXX                           DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM REWIND
*
RWND     EQU       $
OP.2.00  EQU       $
         XXX                           DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM READ
*
READ     EQU       $
OP.3.00  EQU       $
         XXX                           DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM WRITE
*
WRIT     EQU       $
OP.4.00  EQU       $
         XXX                           DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM WRITE END-OF-FILE
*
WEOF     EQU       $
OP.5.00  EQU       $
         XXX                           DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM EXECUTE CHANNEL PROGRAM
*
EXCP     EQU       $
OP.6.00  EQU       $
         BU        ILOPCODE            EXCP NOT SUPPORTED IN MPX 1.4
*
** PROCEDURE TO PERFORM ADVANCE RECORD
*
ADVR     EQU       $
```

```
OP.7.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM ADVANCE FILE
*
ADVF     EQU       $
OP.8.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM BACKSPACE RECORD
*
BKSR     EQU       $
OP.9.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM BACKSPACE FILE
*
BKSF     EQU       $
OP.A.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM UPSPACE
*
UPSP     EQU       $
OP.B.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM ERASE OR PUNCH TRAILER
*
ERPT     EQU       $
OP.C.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM EJECT
*
EJCT     EQU       $
OP.D.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
*
** PROCEDURE TO PERFORM CLSE
*
CLSE     EQU       $
OP.E.00  EQU       $
         XXX                        DEVICE DEPENDENT CODE GOES HERE
** ILLEGAL OPCODE FOR THIS DEVICE
*
UNUS     EQU       $
OP.F.00  EQU       $
         BU        ILOPCODE
         TITLE     H.XXIOP          I/O QUEUE PROCESSOR
*********************************************************************
*                                                                   *
*                  IQ. - I/O QUEUE PROCESSOR                        *
*                                                                   *
*********************************************************************
*                                                                   *
*   THIS ENTRY POINT IS CALLED BY H.IOCS,29 AFTER LINKING THE I/O   *
*   QUEUE ENTRY CREATED BY OP. TO THE I/O QUEUE FOR THIS DEVICE,       *
*   IF THE I/O QUEUE WAS EMPTY WHEN THE ENTRY WAS LINKED.              *
*                                                                       *
*   THIS ENTRY POINT IS ALSO CALLED BY OP. TO AUTOMATICALLY PROCESS    *
*   THE I/O QUEUE AFTER SERVICE INTERRUPT PROCESSING IS COMPLETE.      *
*                                                                       *
*   THIS ENTRY POINT IS ALSO CALLED BY LI. IF DEVICE TERMINATION       *
*   FAILS, REQUIRING THE I/O QUEUE TO BE RESTARTED.                    *
*                                                                       *
*************************************************************************
*                                                                       *
*       CALLER:                      H.IOCS,29     OR,                  *
*                                    SI.           OR,                  *
*                                    LI.                                *
*                                                                       *
*       CALLERS PRIORITY:            TASK LEVEL IF H.IOCS,29            *
*                                    IOP INTERRUPT LEVEL IF SI.         *
*                                    REAL-TIME CLOCK INT LVL IF LI.     *
*                                                                       *
*       INTERRUPTS:                  BLOCKED IF H.IOCS,29               *
*                                    IOP LEVEL ACTIVE IF SI.            *
*                                    RT CLOCK LEVEL ACTIVE IF LI.       *
*                                                                       *
*       CALLING SEQUENCE:            BL    *1W,X2    IF H.IOCS,29       *
*                                    BL    IQ.       IF SI. OR LI.      *
*                                                                       *
*       REGISTERS IN:                R2 = HAT ADDRESS    IF H.IOCS,29   *
*                                    R3 = UDT ADDRESS                   *
*                                    R7 = IOQ ADDRESS    IF H.IOCS,29   *
*                                                                       *
*       RETURN SEQUENCE:             TRSW  R0                           *
*                                                                       *
*       REGISTERS OUT:               R7    MUST NOT BE DISTURBED        *
*                                                                       *
*                                                                       *
*************************************************************************
        SPACE     3
        BOUND     1W
IQ.     EQU       $
        LW        R2,UDT.DCAA,X3  GET DCA ADDRESS
        ZBM       0,DCA.FLAG,X2   CLEAR UNEXPECTED INTERRUPT INDICATOR
*
** PROCEDURE TO PROCESS NEXT I/O QUEUE ENTRY. THIS PROCEDURE ASSUMES
** THE I/O QUEUE FOR THIS DEVICE IS LINKED TO ITS UDT, AND MUST BE
** MODIFIED IF THE I/O QUEUE IS LINKED TO THE CDT FOR THIS DEVICE.
*
IQ.1.00 EQU       $
        LB        R6,UDT.IOCT,X3  ANY IOQ ENTRIES?
        BZ        IQ.1.05         BR IF NO IOQ ENTRIES
*** MORE IOQ ENTRIES TO PROCESS
        LW        R6,UDT.FIOQ,X3  GET NEXT IOQ ENTRY
        STW       R6,DCA.IOQA,X2  SAVE IN DCA
        TRR       R3,R5           UDT ADDRESS TO R5
        TRR       R6,R3           IOQ ENTRY ADDRESS TO R3
        SBM       0,IOQ.STAT,X3   INDICATE IOQ ENTRY ACTIVE
        LW        R1,DCA.CHTA,X2  GET CHT ADDRESS
        LA        R6,IOQ.IOCD,X3  GET ADDRESS OF IOCL
        STW      R6,CHT.IOCL,X1  STORE INTO CHT IOCL ADDRESS
        LA       R1,DCA.TIMO,X2  GET ADDRESS OF DCA TIMEOUT TABLE
        ADMB     R1,IOQ.FCT1,X3  OFFSET INTO DCA.TIMO BASED ON OPCODE
        LW       R6,0,X1         GET TIMEOUT VALUE THIS OPERATION
        TRR      R5,R1           UDT ADDRESS TO R1
        STW      R6,UDT.PTOV,X1  STORE TIMEOUT VALUE IN UDT
        LH       R6,DCA.UADD,X2  GET UNIT ADDRESS'
        SBM      1,UDT.FLGS,X1   INDICATE I/O OUTSTANDING THIS DEVICE
        SIO      R6,0            ISSUE START I/O
        BU       IQ.1.10
*** NO MORE IOQ ENTRIES TO PROCESS - IOQ IS EMPTY
IQ.1.05 EQU      $
        ZMW      UDT.PTOV,X3     ZERO CURRENT TIMEOUT VALUE
        ZMW      DCA.IOQA,X2     ZERO CURRENT IOQ ENTRY ADDRESS
        ZBM      1,UDT.FLGS,X3   INDICATE NO I/O OUTSTANDING THIS DEV
        SBM      0,DCA.FLAG,X2   INDICATE INTERRUPTS NOT EXPECTED
*** RETURN TO CALLER
IQ.1.10 EQU      $
        TRSW     R0
        TITLE    H.XXIOP         SERVICE INTERRUPT PROCESSOR
*****************************************************************
*                                                               *
*                 SI. - SERVICE INTERRUPT PROCESSOR             *
*                                                               *
*****************************************************************
*                                                               *
*   THIS ENTRY POINT IS ENTERED WHENEVER AN INTERRUPT FROM THIS DEVICE *
*   IS FIELDED BY H.IOPX. SERVICE INTERRUPT PROCESSING INCLUDES THE    *
*   FOLLOWING:                                                  *
*                                                               *
*       1. DETERMINE THE REASON FOR INTERRUPTION.               *
*       2. POST STATUS FOR USER.                                *
*       3. FETCH ADDITIONAL SENSE DATA, IF NECESSARY.           *
*       4. ATTEMPT ERROR RECOVERY, IF NECESSARY.                *
*       5. UNLINK IOQ ENTRY FROM IOQ.                           *
*       6. REPORT I/O COMPLETE TO EXEC.                         *
*       7. AUTOMATICALLY PROCESS NEXT IOQ ENTRY                 *
*                                                               *
*****************************************************************
*                                                               *
*       CALLER:                  H.IOPX                         *
*                                                               *
*       CALLERS PRIORITY:        IOP INTERRUPT LEVEL            *
*                                                               *
*       INTERRUPTS:              IOP LEVEL ACTIVE               *
*                                                               *
*       CALLING SEQUENCE:        BU    *3W,X2                   *
*                                                               *
*       REGISTERS IN:            R2 = HAT ADDRESS               *
*                                R3 = UDT ADDRESS               *
*                                                               *
*       RETURN SEQUENCE:         BU    *CHT.EXIT,XN             *
*                                                               *
*       REGISTERS OUT:           RN = CHT ADDRESS               *
*                                                               *
*****************************************************************
        SPACE    3
```

```
         BOUND     1W
SI.      EQU       $
*
** PROCEDURE TO DETERMINE THE REASON FOR INTERRUPTION
*
         LW        R3,UDT.DCAA,X3  GET DCA ADDRESS
         LW        R1,DCA.CHTA,X3  GET CHT ADDRESS
         LW        R2,DCA.IOQA,X3  GET CURRENT IOQ ENTRY ADDRESS
         TBM       0,DCA.FLAG,X3   UNEXPECTED INTERRUPT?
         BS        SI.2.00         BR IF UNEXPECTED INTERRUPT
         TBM       22,IOQ.FLGS,X2  WAS IT HIO FOR KILL?
         BS        SI.3.00         BR IF KILL
         ZBM       1,DCA.FLAG,X3   WAS IT HIO FOR DEVICE TIMEOUT?
         BS        SI.4.00         BR IF TIMEOUT
         ZBM       1,IOQ.STAT,X2   WAS IT A SENSE STATUS SIO?
         BS        SI.5.00         BR IF SENSE STATUS
         TBM       1,CHT.CHST,X1   WAS IT A POST PGM CONTROLLED INT?
         BS        SI.7.00         BR IF PPCI
         TBM       23,IOQ.FLGS,X2  WAS IT AN EXECUTE CHANNEL PROGRAM?
         BS        SI.2.00         BR IF EXEC CHAN PGM
*
** PROCEDURE TO PROCESS INTERRUPT FROM NORMAL SIO
*
         TBM       13,CHT.CHST,X1  DEVICE END SET?
         BNS       *CHT.EXIT,X1    EXIT INTERRUPT LEVEL IF NO DEVICE END
         LH        R6,CHT.CHST,X1  GET CHAN/DEV STATUS
         STH       R6,IOQ.IOST+1H,X2  STORE STATUS IN USER STATUS WORD
         LNH       R6,CHT.RBC,X1   GET NEGATIVE RESIDUAL BYTE COUNT
         ADMW      R6,IOQ.FCT3,X2  ADD REQUESTED BYTE XFER COUNT
         STW       R6,IOQ.UTRN,X2  UPDATE ACTUAL BYTE XFER COUNT
         TBM       15,CHT.CHST,X1  UNIT EXCEPTION SET?
         BNS       SI.1.00         BR IF NO UNIT EXCEPTION
         XXX                       DEV DEPENDENT UNIT EX CODE GOES HERE
SI.1.00  EQU       $
         TBM       14,CHT.CHST,X1  UNIT CHECK SET?
         BNS       SI.1.05         BR IF NO UNIT CHECK
*** FETCH SENSE STATUS IF UNIT CHECK
         TBM       3,IOQ.CONT,X2   STATUS CHECKING INHIBITED?
         BS        SI.1.10         BR IF INHIBITED
         SBM       1,IOQ.STAT,X2   INDICATE SENSE STATUS SIO ISSUED
         LA        R6,DCA.SENI,X3  GET ADDRESS OF SENSE STATUS IOCD
         STW       R6,CHT.IOCL,X1  STORE IN CHT (IOCLA OF SIOA)
         TRR       R1,R5           SAVE CHT ADDRESS IN R5
         LA        R1,DCA.TIMO,X3  GET ADDRESS OF DCA TIMEOUT TABLE
         ADI       R1,READ.OP*4    USE TIMEOUT FOR READ
         LW        R6,0,X1         GET TIMEOUT VALUE THIS OPERATION
         LW        R1,DCA.UDTA,X3  GET UDT ADDRESS
         STW       R6,UDT.PTOV,X1  STORE TIMEOUT VALUE IN UDT
         TRR       R5,R1           RESTORE CHT ADDRESS TO R1
         LH        R6,DCA.UADD,X3  GET ADDRESS THIS UNIT
         SIO       R6,0            ISSUE SENSE STATUS SIO
         BU        *CHT.EXIT,X1    EXIT INTERRUPT LEVEL
*** CHECK FOR CHANNEL ERRORS
CHERMASK DATAW     X'1F00'         CHANNEL ERROR MASK
SI.1.05  EQU       $
         LW        R4,CHERMASK     GET CHANNEL ERROR MASK
         LMH       R6,CHT.CHST,X1  APPLY TO CHANNEL STATUS
         BZ        SI.1.10         BR IF NO CHANNEL ERRORS
         SBM       1,IOQ.IOST,X2   INDICATE ERROR IN USER STATUS
*** UNLINK IOQ ENTRY, REPORT I/O COMPLETE, AND PROCESS NEXT IOQ ENTRY
SI.1.10  EQU       $
         LW        R3,DCA.CDTA,X3  GET CDT ADDRESS
         BL        S.IOCS3         UNLINK IOQ ENTRY FROM IOQ
         LB        R1,IOQ.PRGN,X2  GET OQE NUMBER
         BL        S.IOCS29        REPORT I/O COMPLETE TO EXEC
         LW        R2,IOQ.DCAA,X2  GET DCA ADDRESS
         LW        R3,DCA.UDTA,X2  GET UDT ADDRESS
         BL        IQ.             PROCESS NEXT IOQ ENTRY
         LW        R2,DCA.CHTA,X2  GET CHT ADDRESS
         BU        *CHT.EXIT,X2    EXIT INTERRUPT LEVEL VIA H.IOPX
*
** PROCEDURE TO PROCESS AN UNEXPECTED INTERRUPT. THIS PROCEDURE TREATS
** UNEXPECTED INTERRUPTS AS SPURIOUS. IF THE DEVICE IS CAPABLE OF
** GENERATING ASYNCHRONOUS INTERRUPTS, THIS PROCEDURE MUST BE MODIFIED.
*
SI.2.00  EQU       $
         ABM       31,DCA.SINC,X3  INCREMENT LOCAL SPURIOUS INT COUNT
         ABM       15,CHT.SPUR,X1  INCREMENT GLOBAL SPURIOUS INT COUNT
         BU        *CHT.EXIT,X1    EXIT INTERRUPT LEVEL
*
** PROCEDURE TO PROCESS HIO INTERRUPT FOR A KILL
*
SI.3.00  EQU       $
         LW        R3,DCA.CDTA,X3  GET CDT ADDRESS
         BL        S.IOCS3         UNLINK IOQ ENTRY FROM IOQ
         TRR       R2,R3           IOQ ENTRY ADDRESS TO R3
         BL        S.IOCS15        DELETE IOQ ENTRY
         LW        R2,IOQ.DCAA,X1  GET DCA ADDRESS
         LW        R3,DCA.UDTA,X2  GET UDT ADDRESS
         BL        IQ.             PROCESS NEXT IOQ ENTRY
         LW        R2,DCA.CHTA,X2  GET CHT ADDRESS
         BU        *CHT.EXIT,X2    EXIT INTERRUPT LEVEL VIA H.IOPX
*
** PROCEDURE TO PROCESS HIO INTERRUPT FOR A DEVICE TIMEOUT
*
SI.4.00  EQU       $
         SBM       10,IOQ.IOST,X2  INDICATE TIMEOUT IN USER STATUS
         SBM       6,IOQ.FLGS,X2   INDICATE UNRECOVERABLE I/O ERR
         SBM       1,IOQ.IOST,X2   INDICATE ERROR IN USER STATUS
         BU        SI.1.10         COMPLETE PROCESSING
*
** PROCEDURE TO PROCESS SIO SENSE STATUS INTERRUPT
*
UNREMASK DATAW     XXX             DEV DEPENDENT UNRECOVERABLE ERR MASK
RTRYMASK DATAW     XXX             DEV DEPENDENT RECOVERABLE ERROR MASK
*
SI.5.00  EQU       $
         LW        R6,DCA.SENS,X3  GET ERROR STATUS INFO
         STW       R6,IOQ.IST1,X2  STORE IN IOQ ENTRY
         LW        R4,UNREMASK     GET UNRECOVERABLE ERROR MASK
         LMW       R6,DCA.SENS,X3  MASK WITH ERROR STATUS
         BZ        SI.5.05         BR IF NO UNRECOVERABLE I/O ERRORS
         SBM       6,IOQ.FLGS,X2   SET IOQ ENTRY ERROR INDICATOR
         SBM       1,IOQ.IOST,X2   INDICATE ERROR IN USER STATUS
         XXX                            SET OTHER DEV DEP STATUS INDICATORS
         BU        SI.1.10              COMPLETE PROCESSING
SI.5.05  EQU       $
         LW        R4,RTRYMASK          GET RECOVERABLE ERROR MASK
         LMW       R6,DCA.SENS,X3       MASK WITH ERROR STATUS
         BZ        SI.1.10              BR IF NO ERRORS
*
** PROCEDURE TO PERFORM ERROR RECOVERY PROCESSING
*
RETRYLIM EQU       XXX                  DEV DEPENDENT MAX # OF RETRIES
*
SI.6.00  EQU       $
         TBM       1,IOQ.CONT,X2        USER INHIBIT ERROR PROCESSING?
         BNS       SI.6.05              BR IF INHIBITED
         SBM       1,IOQ.IOST,X2        INDICATE ERROR IN USER STATUS
         BU        SI.1.10              COMPLETE PROCESSING
SI.6.05  EQU       $
         ABM       31,DCA.RETC,X3       UPDATE RETRY COUNTER
         LI        R6,RETRYLIM          GET MAX # RETRIES
         CAMW      R6,DCA.RETC,X3       RETRY LIMIT EXCEEDED?
         BGT       SI.6.05              BR IF NOT EXCEEDED
         ZMW       DCA.RETC,X3          CLEAR RETRY COUNTER
         SBM       6,IOQ.FLGS,X2        SET IOQ ENTRY ERROR INDICATOR
         SBM       1,IOQ.IOST,X2        INDICATE ERROR IN USER STATUS
         XXX                            SET OTHER DEV DEP STATUS INDICATORS
         BU        SI.1.10              COMPLETE PROCESSING
SI.6.10  EQU       $
         XXX                            DEV DEPENDENT RETRY LOGIC GOES HERE
*
** PROCEDURE TO PROCESS INTERRUPT FROM A CHANNEL PROGRAM
*
SI.8.00  EQU       $
*   EXECUTE CHANNEL PROGRAM NOT IMPLEMENTED IN MPX 1.4
         HALT
         TITLE     H.XXIOP              LOST INTERRUPT PROCESSOR
*************************************************************************
*                                                                       *
*                  LI. - LOST INTERRUPT PROCESSOR                       *
*                                                                       *
*************************************************************************
*                                                                       *
*   THIS ENTRY POINT IS CALLED BY S.IOCS5 (ON BEHALF OF THE REAL-TIME   *
*   CLOCK INTERRUPT HANDLER) WHEN A DEVICE TIMEOUT OCCURS, AND BY        *
*   H.IOCS,38 IN ORDER TO KILL AN OUTSTANDING I/O REQUEST.              *
*                                                                       *
*   IN BOTH CASES THE CURRENT I/O REQUEST IS TERMINATED WITH A HALT     *
*   I/O (HIO) INSTRUCTION. IF THE CONTROLLER RESPONDS TO THE HIO,       *
*   OP. PERFORMS THE REQUIRED INTERRUPT HANDLING. IF THE HIO TIMES OUT  *
*   THE CONTROLLER AND DEVICE ARE PRESUMED MALFUNCTIONING.              *
*                                                                       *
*   S.IOCS5 AND H.IOCS,38 ENTER LI. WITH INTERRUPTS BLOCKED. IN         *
*   ADDITION H.IOCS,38 ENTERS LI. WITH CONTEXT SWITCHING INHIBITED.     *
*                                                                       *
*************************************************************************
*
*       CALLER:                         S.IOCS5
*
```

```
*         CALLERS PRIORITY:          REAL-TIME CLOCK INTERRUPT LEVEL      *
*                                                                         *
*         INTERRUPTS:                RT CLOCK INTERRUPT LEVEL ACTIVE      *
*                                    INTERRUPTS BLOCKED                   *
*                                    CONTEXT SWITCHING INHIBITED          *
*                                                                         *
*         CALLING SEQUENCE:          BL    *4W,X2                         *
*                                                                         *
*         REGISTERS IN:              R2 = HAT ADDRESS                     *
*                                    R3 = UDT ADDRESS                     *
*                                                                         *
*         RETURN SEQUENCE:           TRSW  R0                             *
*                                                                         *
*         REGISTERS OUT:             NONE REQ'D                           *
*                                                                         *
***************************************************************************
          SPACE      3
          BOUND      1W
LI.       EQU        $
*
** PROCEDURE TO TERMINATE DEVICE AND FORCE AN INTERRUPT VIA HALT I/O
*
          LW         R2,UDT.DCAA,X3  GET DCA ADDRESS
          ABM        31,DCA.LINC,X2  UPDATE LOST INTERRUPT COUNT
          SBM        1,DCA.FLAG,X2   INDICATE HALT I/O ISSUED
          BS         LI.1.00         BR IF HALT I/O TIMED OUT
          LW         R6,LI.TIMO      GET TIMEOUT VALUE FOR HALT I/O
          STW        R6,UDT.PTOV,X3  STORE IN UDT
          LH         R6,UDT.CHAN,X3  GET DEVICE ADDRESS
          HIO        R6,0            ISSUE HALT I/O
          TRSW       R0              RETURN TO CALLER
*
** PROCEDURE TO CLEAN UP WHEN HALT I/O TIMES OUT
*
LI.1.00   EQU        $
          ZBM        1,DCA.FLAG,X2   CLEAR HIO ISSUED FLAG
          STW        R0,LI.RETN      SAVE RETURN ADDRESS
          LW         R3,UDT.DCAA,X3  GET DCA ADDRESS
          LW         R1,DCA.CDTA,X3  GET CDT ADDRESS FOR THIS UNIT
          TRR        R3,R4           DCA ADDRESS TO R4
          BL         S.IOCS31        MARK UNITS OFFLINE AND MALFUNCTIONING
          TRR        R4,R3           RESTORE DCA ADDRESS TO R3
          LW         R2,DCA.IOQA,X3  GET CURRENT I/O QUEUE ENTRY ADDRESS
          LW         R3,DCA.CDTA,X3  GET CDT ADDRESS
          BL         S.IOCS3         UNLINK IOQ ENTRY FROM IOQ
          TBM        22,IOQ.FLGS,X2  THIS A KILL REQUEST?
          BNS        LI.1.05         BR IF DEVICE TIME OUT
*** FINAL CLEANUP FOR KILL REQUEST
          TRR        R2,R3           IOQ ENTRY ADDRESS TO R3
          BL         S.IOCS10        DEALLOCATE IOQ ENTRY
          BU         LI.1.10
*** FINAL CLEANUP FOR DEVICE TIME OUT
LI.1.05   EQU        $
          LW         R6,IOQ.UDTA,X2  GET UDT ADDRESS
          STW        R6,LI.UDTA      SAVE FOR LATER
          SBM        6,IOQ.FLGS,X2   INDICATE UNRECOVERABLE I/O ERR
          SBM        4,IOQ.IOST,X2   INDICATE DEVICE INOPERABLE
```

```
         SBM       10,IOQ.IOST,X2  INDICATE DEVICE TIME OUT
         LB        R1,IOQ.PRGN,X2  GET DQE NUMBER
         BL        S.IOCS29        REPORT I/O COMPLETE
         LW        R3,LI.UDTA      RESTORE UDT ADDRESS
         BL        IQ.             RESTART IOQ TO FLUSH EXISTING ENTRIES
LI.1.10  EQU       $
         LW        R0,LI.RETN      GET RETURN ADDRESS
         TRSW      R0              RETURN TO CALLER
LI.RETN  RES       1W              LI. RETURN ADDRESS SAVE AREA
LI.UDTA  RES       1W              LI. UDT ADDRESS SAVE AREA
LI.TIMO  DATAW     -2              TIMEOUT VALUE FOR HIO
         TITLE     H.XXIOP         SPECIAL POST XFER PROCESSOR
************************************************************************
*                                                                      *
*                  PX. - SPECIAL POST XFER PROCESSOR                   *
*                                                                      *
************************************************************************
*                                                                      *
*   THIS ENTRY POINT IS CALLED BY S.IOCS1 (GENERALIZED I/O POST        *
*   PROCESSING) WHENEVER RETURN FROM OP. IS VIA "POSTPROS". THIS       *
*   ENTRY POINT EXECUTES AT TASK PRIORITY, AND THEREFORE CAN BE        *
*   USED TO PROVIDE ANY SPECIAL PROCESSING ASSOCIATED WITH A DATA      *
*   TRANSFER (EG., DATA TRANSLATION) AT A LOW LEVEL OF SYSTEM          *
*   OVERHEAD.                                                          *
*                                                                      *
************************************************************************
*                                                                      *
*        CALLER:                   S.IOCS1                             *
*                                                                      *
*        CALLERS PRIORITY:         TASK LEVEL                          *
*                                                                      *
*        INTERRUPTS:               UNBLOCKED                           *
*                                                                      *
*        CALLING SEQUENCE:         BL   *5W,X2                         *
*                                                                      *
*        REGISTERS IN:             R1 = FCB ADDRESS                    *
*                                  R2 = HAT ADDRESS                    *
*                                  R3 = UDT ADDRESS                    *
*                                                                      *
*        RETURN SEQUENCE:          TRSW  R0                            *
*                                                                      *
*        REGISTERS OUT:            R1 = FCB ADDRESS                    *
*                                                                      *
************************************************************************
         SPACE     3
         BOUND     1W
PX.      EQU       $
*   DEVICE DEPENDENT CODE GOES HERE IF PX. IS REQUIRED
         TRSW      R0
         TITLE     H.XXIOP         SYSGEN INITIALIZATION PROCESSOR
************************************************************************
*                                                                      *
*                  SG. - SYSGEN INITIALIZATION PROCESSOR               *
*                                                                      *
************************************************************************
*                                                                      *
*   THIS ENTRY POINT IS CALLED BY SYSGEN FOR THE PURPOSE OF INITIALIZ- *
*   ING CERTAIN HANDLER PARAMETERS, INITIALIZING DEVICE CONTEXT AREAS   *
*   (DCA'S) AND INITIALIZING CERTAIN DATA STRUCTURE ELEMENTS DURING     *
*   THE CONSTRUCTION OF AN MPX-32 IMAGE.                                *
*                                                                       *
*   DCA'S FOR THIS HANDLER ARE CREATED VIA THE REPEATED ASSEMBLY OF     *
*   THE MACRO, "DCA.DATA". A MAXIMUM NUMBER OF DCA'S IS                 *
*   CREATED DURING HANDLER ASSEMBLY. DURING THE EXECUTION OF THIS       *
*   ENTRY POINT, ONE DCA IS INITIALIZED FOR EACH UDT ENTRY CONTAINING   *
*   THE NAME OF THIS HANDLER. ANY REMAINING DCA'S AND THE REMAINDER OF  *
*   THE CODE IN THIS ENTRY WILL BE OVERLAID BY SYSGEN. THEREFORE THIS   *
*   MUST PHYSICALLY BE THE LAST ENTRY POINT OF THE HANDLER.             *
*                                                                       *
*************************************************************************
*                                                                       *
*        CALLER:                      SYSGEN DURING IMAGE CONSTRUCTION  *
*                                                                       *
*        CALLERS PRIORITY:            N/A                               *
*                                                                       *
*        INTERRUPTS:                  N/A                               *
*                                                                       *
*        CALLING SEQUENCE:            BL    LAST PHYSICAL ENTRY POINT   *
*                                                                       *
*        REGISTERS IN:                R3 = CHT ADDRESS                  *
*                                                                       *
*        RETURN SEQUENCE:             TRSW  R0 (VIA M.XIR)              *
*                                                                       *
*        REGISTERS OUT:               NONE REQ'D                        *
*                                                                       *
*************************************************************************
         SPACE     3
         LPOOL                        LITERALS HERE TO AVOID OVERLAYING
DCASTART EQU       $                  START OF DCA'S FOR THIS DEVICE
         REPT      XXX                REPEAT COUNT FOR DCA CREATION
         DCA.DATA  X,X,X,X,X,X,X,;    LIBRARY MACRO - CREATE 1 DCA
                   X,X,X,X,X,X,X,X,X,X,X
         ENDR
         BOUND     1W
SG.      EQU       $ -
         M.EIR                        LIBRARY MACRO - ENTER INIT ROUTINE
         DCA.INI1  H.XXIOP            LIBRARY MACRO - DCA INITIALIZATION
         XXX                          USER CODE HERE IF REQUIRED
         XXX                          NOTE: ALL REGS AVAILABLE, R2=DCA ADDR
         DCA.INI2                     LIBRARY MACRO - DCA INITIALIZATION
         M.XIR     HAT                LIBRARY MACRO - EXIT INIT ROUTINE
         END
```

15.9 Sample IOP Device Handler

```
*  IOP LINE PRINTER HANDLER        RELEASE 1.4 MPX      HEADER  H.LPIOP
         SPACE     10
************************************************************************
*                                                                      *
*            >>>>> RESTRICTED RIGHTS LEGEND <<<<<                      *
*                                                                      *
*        USE, DUPLICATION, OR DISCLOSURE IS SUBJECT TO THE             *
*        RESTRICTIONS STATED IN SYSTEMS' LICENSE AGREEMENT             *
*        (FORM NO. 1218) OR, FOR GOVERNMENT CUSTOMERS,                 *
*        DAR 7-104.9A.                                                 *
*                                                                      *
************************************************************************
         TITLE     H.LPIOP            IOP LINE PRINTER DEVICE HANDLER
         PROGRAM   H.LPIOP
************************************************************************
*                                                                      *
*              IOP LINE PRINTER HANDLER PROGRAM                        *
*                                                                      *
*   THIS HANDLER IS SYSTEM REENTRANT WHICH MEANS THAT ONLY ONE COPY    *
*   SHOULD BE CONFIGURED INTO AN MPX-32 SYSTEM REGARDLESS OF THE       *
*   NUMBER OF DEVICES OF THIS TYPE AND THE NUMBER OF IOP'S CONFIGURED  *
*   INTO THE SYSTEM.                                                   *
*                                                                      *
************************************************************************
         SPACE     3
         M.EQUS
         M.TBLS
         DCA.EQUS
         EXT       S.EXEC5
         EXT       S.IOCS3
         EXT       S.IOCS10
         EXT       S.IOCS12
         EXT       S.IOCS13
         EXT       S.IOCS15
         EXT       S.IOCS29
         EXT       S.IOCS31
         EXT       ILOPCODE
         EXT       SERVCOMP
         EXT       IOLINK
         EXT       POSTPROS
         SPACE     3
HAT      DATAW     6               HANDLER ADDRESS TABLE
         ACW       OP.             OPCODE PROCESSOR ADDRESS
         ACW       IQ.             I/O QUEUE PROCESSOR ADDRESS
         ACW       SI.             SERVICE INT PROCESSOR ADDRESS
         ACW       LI.             LOST INTERRUPT PROCESSOR ADDRESS
         ACW       PX.             POST XFER PROCESSOR ADDRESS
*                                  *ADDITIONAL ENTRY POINTS ,IF REQ'D,
*                                  *MUST BE ADDED HERE.
SG.ADDR  ACW       SG.             SYSGEN INITIALIZATION PROCESSOR ADDR
*                                  *THIS MUST THE LAST ENTRY POINT
         SPACE     5
*
**  IOP LINE PRINTER COMMAND TABLE
*
CMD.TBL  EQU       $
*
```

```
****  COMMANDS TO PRINT BUFFER THEN DO FORMS CONTROL
*
LP.CMD1  DATAW    X'01000000'     PRINT ONLY - NO FORMS CONTROL
LP.CMD2  DATAW    X'05000000'     PRINT BUFFER, <CR>
LP.CMD3  DATAW    X'15000000'     PRINT BUFFER, <LF>
LP.CMD4  DATAW    X'25000000'     PRINT BUFFER, <LF> <LF>
LP.CMD5  DATAW    X'35000000'     PRINT BUFFER, <LF> <LF> <LF>
LP.CMD6  DATAW    X'45000000'     PRINT BUFFER, <FF>
LP.CMD7  DATAW    X'65000000'     PRINT BUFFER, <CR>, THEN CLEAR BUFFER
*
****  COMMANDS TO DO FORMS CONTROL AND THEN PRINT BUFFER.
****  NOTE: THESE COMMANDS ARE ARRANGED SO THAT BY USING THE INDEX
****        OF THE FORMS CONTROL TABLE AND A OFFSET INTO THIS TABLE
****        YOU CAN GET THE APPROPRIATE COMMAND FOR THE FC CHAR.
*
LP.CMD8  DATAW    X'00000000'     <CR>, PRINT BUFFER, <CR>
LP.CMD9  DATAW    X'40000000'     <FF>, PRINT BUFFER, <CR>
         DATAW    X'40000000'     <FF>, PRINT BUFFER, <CR>
LP.CMD10 DATAW    X'20000000'     <LF> <LF>, PRINT BUFFER <CR>
LP.CMD11 DATAW    X'10000000'     <LF>, PRINT BUFFER, <CR>
LP.CMD12 DATAW    X'30000000'     <LF> <LF> <LF>, PRINT, <CR>  (SPARE)
*
****  COMMANDS THAT DO ONLY FORMS CONTROL (NO PRINTING)
*
LP.CMD13 DATAW    X'03000000'     <CR>
LP.CMD14 DATAW    X'47000000'     <FF>
         DATAW    X'47000000'     <FF>
LP.CMD15 DATAW    X'27000000'     <LF> <LF>
LP.CMD16 DATAW    X'17000000'     <LF>
LP.CMD17 DATAW    X'37000000'     <LF> <LF> <LF>                (SPARE)
         SPACE    3
*
** LINE PRINTER FORMS CONTROL TABLE
*
LPFCNUM  DATAW    5               NUMBER OF ENTRIES IN THE FC TABLE
LPFCTBL  EQU      $
         DATAB    C'+'            FORMS CONTROL FOR CR THEN PRINT
         DATAB    C'1'            FORMS CONTROL FOR FF THEN PRINT
         DATAB    C'-'            FORMS CONTROL FOR FF THEN PRINT
         DATAB    C'0'            FORMS CONTROL FOR 2 LF'S THEN PRINT
         DATAB    C' '            FORMS CONTROL FOR LF THEN PRINT
         SPACE    2
*
** COMMAND TABLE OFFSETS EQUATES
*
PR.OFSET EQU      7W              OFFSET TO CMDS THAT DO FC AND PRINT
CM.OFSET EQU      12W             OFFSET TO CMDS THAT ONLY DO FC
         TITLE    H.LPIOP         OPCODE PROCESSOR
**********************************************************************
*                                                                    *
*                 OP. - OPCODE PROCESSOR                             *
*                                                                    *
**********************************************************************
*                                                                    *
*   THIS ENTRY POINT IS ACTUALLY A SUBROUTINE EXTENSION OF H.IOCS,29, *
*   A PORTION OF IOCS LOGIC COMMON TO ALL I/O SERVICES WHICH ARE     *
*   CAPABLE INITIATING A PHYSICAL DEVICE ACCESS. IT IS CALLED TO     *
*   PROCESS THE OPCODE PLUGGED INTO THE FCB BY THE I/O SERVICE              *
*   ORIGINALLY CALLED BY THE USER.                                          *
*                                                                           *
*                                                                           *
*   THE PURPOSE OF OP. IS TO EXAMINE THE OPCODE AND OTHER PERTINENT         *
*   FCB CONTROL SPECIFICATIONS, AND TO INDICATE TO H.IOCS,29 WHAT           *
*   ACTION IS TO BE TAKEN. IN ORDER TO INDICATE WHAT ACTION IS TO BE        *
*   TAKEN, OP. TAKES 1 OF 4 POSSIBLE RETURNS TO H.IOCS,29 AS FOLLOWS:       *
*                                                                           *
*        BU         ILOPCODE     1. OPCODE ILLEGAL FOR THIS DEVICE          *
*        BU         SERVCOMP     2. SERVICE COMPLETE, NO DEV ACCESS REQD*
*        BU         IOLINK       3. LINK REQUEST TO I/O QUEUE               *
*        BU         POSTPROS     4. LINK REQUEST + POST PROCESSING REQD *
*                                                                           *
*                                                                           *
*   IF RETURN 3 (IOLINK) IS TAKEN OP. MUST FIRST:                           *
*                                                                           *
*        1. CALL IOCS SUBROUTINE S.IOCS13 TO ALLOCATE AND INITIALIZE        *
*           AN I/O QUEUE ENTRY. FOUR IOQ ENTRY WORDS, CALLED HANDLER        *
*           FUNCTION WORDS ARE OF SPECIAL INTEREST TO THE PROGRAMMER.       *
*           UPON ENTRY INTO THE OPCODE PROCESSING ROUTINE THEY ARE          *
*           SET UP AS FOLLOWS:                                              *
*                  IOQ.FCT1 - BITS 0-7 = WORD ADJ OPCODE,BITS 8-31 FREE*
*                  IOQ.FCT2 - USER'S LOGICAL DATA ADDRESS                   *
*                  IOQ.FCT3 - USER'S BYTE TRANSFER COUNT                    *
*                  IOQ.FCT4 - AVAILABLE                                     *
*                                                                           *
*        2. BUILD INTO THE I/O QUEUE ENTRY IN THE SPACE RESERVED FOR        *
*           IT, AN I/O COMMAND LIST (IOCL) WITH THE PROPER COMMAND          *
*           CODES AND FLAGS USING IOCS SUBROUTINE S.IOCS12 AND IOCS         *
*           ENTRY POINT H.IOCS,40.                                          *
*                                                                           *
*        3. OR, IF THE USER REQUESTS EXECUTION OF HIS CHANNEL PROGRAM,      *
*           VALIDATE THE DATA AREAS, AND ABSOLUTIZE THE DATA ADDRESSES      *
*           USING IOCS SUBROUTINE S.IOCS33                                  *
*                                                                           *
*                                                                           *
*   TAKING RETURN 4 (POSTPROS) INDICATES THAT AN I/O REQUEST IS TO BE       *
*   LINKED TO THE I/O QUEUE FOR THIS DEVICE AS IN RETURN 3 (IOLINK),        *
*   AND ALSO THAT AFTER THE REQUEST HAS BEEN COMPLETED BUT BEFORE           *
*   RETURN TO OR NOTIFICATION OF THE USER, THE SPECIAL DEVICE POST          *
*   PROCESSING ENTRY POINT (PX.) WILL BE CALLED.                            *
*                                                                           *
*****************************************************************************
*                                                                           *
*        CALLER:                      H.IOCS,29                             *
*                                                                           *
*        CALLERS PRIORITY:            TASK LEVEL                            *
*                                                                           *
*        INTERRUPTS:                  UNBLOCKED                             *
*                                                                           *
*        CALLING SEQUENCE:            BL    *1W,X2                          *
*                                                                           *
*        REGISTERS IN:                R1 = FCB ADDRESS                      *
*                                     R2 = HAT ADDRESS                      *
*                                     R3 = UDT ADDRESS                      *
*                                                                           *
*        RETURN SEQUENCE:          BU   IOLINK   OR,                    *
*                                  BU   SERVCOMP OR,                    *
*                                  BU   ILOPCODE OR,                    *
*                                  BU   POSTPROS                        *
*                                                                       *
*        REGISTERS OUT:            R1 = FCB ADDRESS                     *
*                                                                       *
************************************************************************
         SPACE     3
*
** THE OPCODE VECTOR TABLE WHICH FOLLOWS CONTAINS THE ADDRESSES OF
** THE OPCODE PROCESSING PROCEDURES ORDERED BY OPCODE NUMBER. EACH
** ENTRY HAS BIT 0 SET IF THE ASSOCIATED PROCEDURE REQUIRES AN I/O
** QUEUE ENTRY. SYMBOLIC OPCODES ARE ALSO LISTED.
*
         BOUND     1W
OPTAB    EQU       $
         GEN       1/0,31/W(OPEN)  OPEN
OPEN.OP  EQU       0
         GEN       1/1,31/W(RWND)  REWIND
RWND.OP  EQU       1
         GEN       1/0,31/W(READ)  READ RECORD
READ.OP  EQU       2
         GEN       1/1,31/W(WRIT)  WRITE RECORD
WRIT.OP  EQU       3
         GEN       1/0,31/W(WEOF)  WRITE END-OF-FILE RECORD
WEOF.OP  EQU       4
         GEN       1/0,31/W(EXCP)  EXECUTE CHANNEL PROGRAM
EXCP.OP  EQU       5
         GEN       1/1,31/W(ADVR)  ADVANCE RECORD
ADVR.OP  EQU       6
         GEN       1/0,31/W(ADVF)  ADVANCE FILE
ADVF.OP  EQU       7
         GEN       1/0,31/W(BKSR)  BACKSPACE RECORD
BKSR.OP  EQU       8
         GEN       1/0,31/W(BKSF)  BACKSPACE FILE
BKSF.OP  EQU       9
         GEN       1/1,31/W(UPSP)  UPSPACE
UPSP.OP  EQU       10
         GEN       1/0,31/W(ERPT)  ERASE OR PUNCH TRAILER
ERPT.OP  EQU       11
         GEN       1/1,31/W(EJCT)  EJECT
EJCT.OP  EQU       12
         GEN       1/0,31/W(CLSE)  CLOSE
CLSE.OP  EQU       13
         GEN       1/0,31/W(UNUS)  UNUSED
         GEN       1/0,31/W(UNUS)  UNUSED
         SPACE     3
         BOUND     1W
OP.      EQU       $
*
** PROCEDURE TO ALLOCATE AN I/O QUEUE ENTRY,IF NECESSARY, AND
** VECTOR TO THE APPROPIATE OPCODE PROCESSING PROCEDURE
*
         LB        R2,FCB.OPCO,X1  GET OPCODE FROM FCB
         SLL       R2,2            WORD ADJUST OPCODE
         TRR       R2,R6           SAVE OPCODE IN R6
         LW        R4,OPTAB,X2      GET OPCODE PROCEDURE ADDRESS
         BNN       OP.0.00          BR IF I/O QUEUE ENTRY NOT REQ'D
         BL        S.IOCS13         GET I/O QUEUE ENTRY
         LW        R2,FCB.IOQA,X1   GET IOQ ENTRY ADDRESS
         STB       R6,IOQ.FCT1,X2   SAVE OPCODE IN IOQ ENTRY
OP.0.00  EQU       $
         LW        R3,UDT.DCAA,X3   GET ADDRESS OF DEVICE CONTEXT AREA
         XCR       R6,R2            PLACE OPCODE IN R2 AND IOQA IN R6
         BU        *OPTAB,R2        VECTOR TO OPCODE PROCEDURE
         SPACE     2
*
*          OPCODE PROCESSING SECTION
*
*        ENTRY : X1 = FCB ADDR
*                X2 = OPCODE
*                X3 = DCA ADDR
*                R6 = IOQ ADDR
         SPACE      2
*
** PROCEDURE TO PERFORM OPEN
*
OPEN     EQU       $
*** INITIALIZE IOP CHANNEL IF NECESSARY
OP.1.00  EQU       $
         LW        R2,DCA.CHTA,X3   GET CHT ADDRESS
         TBM       0,CHT.FLGS,X2    HAS INCH BEEN PERFORMED?
         BS        OP.1.05          BR IF IOP ALREADY INCHED
         BL        *CHT.INCH,X2     GO TO H.IOPX TO PERFORM INCH
OP.1.05  EQU       $
         BU        SERVCOMP         GOTO H.IOCS,29 WITH SERVICE COMPLETE
         SPACE     1
*
** PROCEDURE TO PERFORM REWIND
*
RWND     EQU       $
         TRR       R6,R2            GET IOQ ADDR INTO R2
         SBM       31,IOQ.IOST,X2   SET BIT IN IOST TO REPRESENT EOF
*
** PROCEDURE TO PERFORM EJECT
*
EJCT     EQU       $
         TRR       R6,R2            GET IOQ ADDR INTO R2
         LW        R6,LP.CMD14      GET I/O CMD FOR FF (NO PRINT)
         ZR        R7               ZERO REG TO BE USED FOR I/O FLAG WRD
         BU        OP.5.00          BR TO SETUP IOCD IN IOQ
         SPACE     1
*
** PROCEDURE TO PERFORM UPSPACE AND ADVANCE RECORD
*
UPSP     EQU       $
ADVR     EQU       $
         TRR       R6,R2            GET IOQ ADDR INTO R2
         LW        R6,LP.CMD16      GET I/O CMD FOR LF (NO PRINT)
         ZR        R7               ZERO REG TO BE USED FOR I/O FLAG WOR
         SPACE     2
*
** PROCEDURE TO SETUP IOCD FOR COMMAND WITH NO DATA TRANSFER
**        ENTRY : R1 = FCB ADDR
**                R2 = IOQ ADDR
**                R3 = DCA ADDR
**                R6 = IOCD MSW (DATA ADDR = 0)
**                R7 = IOCD LSW (COUNT = 0)
*
OP.5.00   EQU        $
          LW         R5,IOQ.IST1,X2  LOAD IOQ.IOCD ADDR'INTO R5
          LW         R3,DCA.CHTA,X3  GET CHT ADDR FROM DCA
          STW        R5,CHT.IOCL,X3  SAVE IOQ.IOCD ADDR IN CHT
          TRR        R2,R3           GET IOQ ADDR INTO R3
          BL         S.IOCS12        SETUP IOCD IN IOQ FOR I/O CONTROL CMD
          BU         IOLINK          RETURN TO IOCS TO LINK THE IOQ
          SPACE      2
*
** PROCEDURE TO PROCESS WRIT OPCODE
*
WRIT      EQU        $
OP.6.00   EQU        $
          TRR        R6,R2           GET IOQ ADDR INTO REG 2
          SBM        11,IOQ.FLGS,X2  SET WRITE FLAG IN IOQ
          TBM        2,FCB.GCFG,X1   IS DATA FORMATTING INHIBITED ??
          BNS        OP.6.10         NO, GO DO FORMATTING
          ZR         R7              ZERO REG TO BE USED FOR IOCD LSW
          TBM        8,FCB.GCFG,X2   IS THERE SUPPOSE TO BE NO FORM CNTRL
          BS         OP.6.05         YES, NO FORMS CONTROL IS NEEDED
          LW         R6,LP.CMD11     DEFAULT FORMS CNTRL IS SINGLE LF
          BU         OP.7.00         GO SETUP IOCD FOR OUTPUT
OP.6.05   EQU        $
          LW         R6,LP.CMD1      NO FORMS CNTRL ONLY PRINT BUFFER
          BU         OP.7.00         GO SETUP IOCD FOR OUTPUT
*** FORMAT OUTPUT ACCORDING TO FIRST CHAR IN DATA BUFFER
OP.6.10   EQU        $
          TRR        X1,R5           SAVE FCB ADDR IN REG 5
          LW         R1,IOQ.FCT2,X2  GET DAT BUFFER ADDR FROM IOQ
          LB         R4,0,R1         GET FIRST BYTE IN DATA BUFFER
          ABM        31,IOQ.FCT2,X2  INCR DATA BUFFER ADDR 1 BYTE
          LW         R6,IOQ.FCT3,X2  GET XFER COUNT FROM IOQ
          SUI        R6,1            DECR BY 1 BYTE
          STW        R6,IOQ.FCT3,X2  STORE NEW COUNT IN IOQ
***** CHECK FC CHAR AGAINST FC TABLE
          LNW        R1,LPFCNUM      GET NEG NUMBER OF CHARS IN FC TABLE
OP.6.15   EQU        $
          CAMB       R4,LPFCTBL+5B,R1   COMPARE FC CHAR TO FC TABLE
          BEQ        OP.6.20         BR IF MATCH
          BIB        R1,OP.6.15      INCR NEG CNT AND BR IF NOT ZERO
          SUI        R1,1            IF TABLE EMPTIES SET R1 = -1; FC =!
***** CONVERT FC TO I/O COMMAND TO BE USED IN IOCD
OP.6.20   EQU        $
          ADMW       R1,LPFCNUM      MAKE NEG CNT A POSITIVE TABLE INDEX
          SLL        R1,2            MAKE INDEX A WORD INDEX
          ZR         R7              CLEAR REG TO BE USED FOR IOCD LSW
          LW         R6,IOQ.FCT3,X2  TEST XFER COUNT TO SEE IF ZERO
          BNZ        OP.6.25         IF NOT ZERO, BR TO DO FC & DATA XFER
***** GET I/O COMMAND FROM CMD TABLE FOR CONTROL ONLY (NO DATA XFER)
          ADI        R1,CM.OFSET     ADD OFFSET INTO CMD TABLE TO FC INDE
          LW         R6,CMD.TBL,R1   GET I/O COMMAND FOR FORMS CONTROL
```

```
         TRR       R5,R1           RESTORE FCB ADDR
         BU        OP.5.00         BR TO SETUP IOCD IN IOQ FOR CONTROL
***** GET I/O COMMAND FROM CMD TABLE FOR FC AND PRINT BUFFER
OP.6.25  EQU       $
         ADI       R1,PR.OFSET     ADD OFFSET INTO CMD TABLE TO FC INDEX
         LW        R6,CMD.TBL,R1   GET I/O COMMAND FOR FORMATTED PRINT
         TRR       R5,R1           RESTORE FCB ADDR
         SPACE     1
*
** PROCEDURE TO SETUP IOCD FOR A DATA XFER
**       ENTRY : R1 = FCB ADDR
**               R2 = IOQ ADDR
**               R3 = DCA ADDR
**               R6 = IOCD MSW WITHOUT DATA BUFFER ADDR
**               R7 = IOCD LSW WITHOUT XFER COUNT
*
OP.7.00  EQU       $
         LW        R5,IOQ.IST1,X2  GET IOQ.IOCD ADDR FROM IOQ
         LW        R3,DCA.CHTA,X3  GET CHT ADDR FROM DCA
         STW       R5,CHT.IOCL,X3  STORE IOCD ADDR INTO CHT
         TRR       R2,R3           GET IOA ADDR INTO R3
         M.CALL    H.IOCS,40       CALL IOCS 40 TO SETUP IOCD IN IOQ
         BU        IOLINK          RETURN TO IOCS TO LINK IOQ
         SPACE     1
*
** PROCEDURE TO PERFORM EXECUTE CHANNEL PROGRAM
** (NOT IMPLEMENTED UNDER MPX-1.4)
*
EXCP     EQU       $               EXECUTE CHANNEL PROGRAM
         BU        ILOPCODE        ILLEGAL OPCODE FOR MPX-1.4
         SPACE     1
*
** OPCODES THAT ARE RETURNED SERVICE COMPLETE WITHOUT ANY PROCESSING
*
CLSE     EQU       $               CLOSE DEVICE
WEOF     EQU       $               WRITE END-OF-FILE
ERPT     EQU       $               ERASE OR PUNCH TRAILER
         BU        SERVCOMP        GOTO H.IOCS,29 WITH SERVICE COMPLETE
         SPACE     1
*
** ILLEGAL OPCODE FOR THIS DEVICE
*
UNUS     EQU       $
READ     EQU       $               ILLEGAL TO READ FROM LINEPRINTER
ADVF     EQU       $               LINEPRINTER IS NOT A FILE
BKSR     EQU       $               CANNOT BACKSPACE LINEPRINTER
BKSF     EQU       $
         BU        ILOPCODE
         TITLE     H.LPIOP         I/O QUEUE PROCESSOR
************************************************************************
*                                                                      *
*                  IQ. - I/O QUEUE PROCESSOR                           *
*                                                                      *
************************************************************************
*                                                                      *
*   THIS ENTRY POINT IS CALLED BY H.IOCS,29 AFTER LINKING THE I/O      *
*   QUEUE ENTRY CREATED BY OP. TO THE I/O QUEUE FOR THIS DEVICE,       *
*   IF THE I/O QUEUE WAS EMPTY WHEN THE ENTRY WAS LINKED.                 *
*                                                                         *
*   THIS ENTRY POINT IS ALSO CALLED BY OP. TO AUTOMATICALLY PROCESS       *
*   THE I/O QUEUE AFTER SERVICE INTERRUPT PROCESSING IS COMPLETE.         *
*                                                                         *
*   THIS ENTRY POINT IS ALSO CALLED BY LI. IF DEVICE TERMINATION          *
*   FAILS, REQUIRING THE I/O QUEUE TO BE RESTARTED.                       *
*                                                                         *
***************************************************************************
*                                                                         *
*       CALLER:                       H.IOCS,29     OR,                   *
*                                     SI.           OR,                   *
*                                     LI.                                 *
*                                                                         *
*       CALLERS PRIORITY:             TASK LEVEL IF H.IOCS,29             *
*                                     IOP INTERRUPT LEVEL IF SI.          *
*                                     REAL-TIME CLOCK INT LVL IF LI.      *
*                                                                         *
*       INTERRUPTS:                   BLOCKED IF H.IOCS,29                *
*                                     IOP LEVEL ACTIVE IF SI.             *
*                                     RT CLOCK LEVEL ACTIVE IF LI.        *
*                                                                         *
*       CALLING SEQUENCE:             BL    *2W,X2    IF H.IOCS,29        *
*                                     BL    IQ.       IF SI. OR LI.       *
*                                                                         *
*       REGISTERS IN:                 R2 = HAT ADDRESS    IF H.IOCS,29    *
*                                     R3 = UDT ADDRESS                    *
*                                     R7 = IOQ ADDRESS    IF H.IOCS,29    *
*                                                                         *
*       RETURN SEQUENCE:              TRSW  R0                            *
*                                                                         *
*       REGISTERS OUT:                R2 = DCA ADDRESS                    *
*                                     R7 = UNDISTURBED                    *
*                                                                         *
*                                                                         *
***************************************************************************
        SPACE      3
        BOUND      1W
IQ.     EQU        $
        LW         R2,UDT.DCAA,X3  GET DCA ADDRESS
        ZBM        0,DCA.FLAG,X2   CLEAR UNEXPECTED INTERRUPT INDICATOR
*
** PROCEDURE TO PROCESS NEXT I/O QUEUE ENTRY. THIS PROCEDURE ASSUMES
** THE I/O QUEUE FOR THIS DEVICE IS LINKED TO ITS UDT, AND MUST BE
** MODIFIED IF THE I/O QUEUE IS LINKED TO THE CDT FOR THIS DEVICE.
*
IQ.1.00 EQU        $
        LB         R6,UDT.IOCT,X3  ANY IOQ ENTRIES?
        BZ         IQ.1.05         BR IF NO IOQ ENTRIES
*** MORE IOQ ENTRIES TO PROCESS
        LW         R6,UDT.FIOQ,X3  GET NEXT IOQ ENTRY
        STW        R6,DCA.IOQA,X2  SAVE IN DCA
        TRR        R3,R5           UDT ADDRESS TO R5
        TRR        R6,R3           IOQ ENTRY ADDRESS TO R3
        LW         R1,DCA.CHTA,X2  GET CHT ADDRESS
        LA         R6,IOQ.IOCD,X3  GET ADDRESS OF IOCL
        STW        R6,CHT.IOCL,X1  STORE INTO CHT IOCL ADDRESS
```

```
         LW        R1,DCA.TIMO,X2  GET ADDRESS OF DCA TIMEOUT TABLE
         ADMB      R1,IOQ.FCT1,X3  OFFSET INTO DCA.TIMO BASED ON OPCODE
         LW        R6,0,X1         GET TIMEOUT VALUE THIS OPERATION
         TRR       R5,R1           UDT ADDRESS TO R1
         STW       R6,UDT.PTOV,X1  STORE TIMEOUT VALUE IN UDT
         LH        R6,DCA.UADD,X2  GET UNIT ADDRESS
         SIO       R6,0            ISSUE START I/O
         SBM       1,UDT.FLGS,X1   SET I/O OUTSTANDING BIT IN UDT
         SBM       0,IOQ.STAT,X3   SET I/O OUTSTANDING BIT IN IOQ
         BU        IQ.1.10
*** NO MORE IOQ ENTRIES TO PROCESS - IOQ IS EMPTY
IQ.1.05  EQU       $
         ZMW       UDT.PTOV,X3     ZERO CURRENT TIMEOUT VALUE
         ZMW       DCA.IOQA,X2     ZERO CURRENT IOQ ENTRY ADDRESS
         ZBM       1,UDT.FLGS,X3   CLEAR I/O OUTSTANDING BIT IN UDT
         SBM       0,DCA.FLAG,X2   INDICATE INTERRUPTS NOT EXPECTED
*** RETURN TO CALLER
IQ.1.10  EQU       $
         TRSW      R0
         TITLE     H.LPIOP         SERVICE INTERRUPT PROCESSOR
*************************************************************************
*                                                                       *
*                   SI. - SERVICE INTERRUPT PROCESSOR                   *
*                                                                       *
*************************************************************************
*                                                                       *
*    THIS ENTRY POINT IS ENTERED WHENEVER AN INTERRUPT FROM THIS DEVICE *
*    IS FIELDED BY H.IOPX. SERVICE INTERRUPT PROCESSING INCLUDES THE    *
*    FOLLOWING:                                                         *
*                                                                       *
*         1. DETERMINE THE REASON FOR INTERRUPTION.                     *
*         2. POST STATUS FOR USER.                                      *
*         3. FETCH ADDITIONAL SENSE DATA, IF NECESSARY.                 *
*         4. ATTEMPT ERROR RECOVERY, IF NECESSARY.                      *
*         5. UNLINK IOQ ENTRY FROM IOQ.                                 *
*         6. REPORT I/O COMPLETE TO EXEC.                               *
*         7. AUTOMATICALLY PROCESS NEXT IOQ ENTRY                       *
*                                                                       *
*************************************************************************
*                                                                       *
*         CALLER:                      H.IOPX                           *
*                                                                       *
*         CALLERS PRIORITY:            IOP INTERRUPT LEVEL              *
*                                                                       *
*         INTERRUPTS:                  IOP LEVEL ACTIVE                 *
*                                                                       *
*         CALLING SEQUENCE:            BU    *3W,X2                     *
*                                                                       *
*         REGISTERS IN:                R2 = HAT ADDRESS                 *
*                                      R3 = UDT ADDRESS                 *
*                                                                       *
*         RETURN SEQUENCE:             BU    *CHT.EXIT,XN               *
*                                                                       *
*         REGISTERS OUT:               R3 = CHT ADDRESS                 *
*                                                                       *
*************************************************************************
         SPACE     3
         BOUND     1W
SI.      EQU       $
*
** PROCEDURE TO DETERMINE THE REASON FOR INTERRUPTION
*
         LW        R1,UDT.DCAA,R3  GET DCA ADDRESS FROM UDT
         LW        R3,DCA.CHTA,X1  GET CHT ADDRESS
         LW        R2,DCA.IOQA,X1  GET CURRENT IOQ ENTRY ADDRESS
         TBM       0,DCA.FLAG,X1   UNEXPECTED INTERRUPT?
         BS        SI.2.00         BR IF UNEXPECTED INTERRUPT
         ZBM       0,IOQ.STAT,X2   CLEAR IOQ BUSY BIT ON INTERRUPT
         TBM       22,IOQ.FLGS,X2  WAS IT HIO FOR KILL?
         BS        SI.3.00         BR IF KILL
         ZBM       1,DCA.FLAG,X1   WAS IT HIO FOR DEVICE TIMEOUT?
         BS        SI.4.00         BR IF TIMEOUT
         ZBM       1,IOQ.STAT,X2   WAS IT A SENSE STATUS SIO?
         BS        SI.5.00         BR IF SENSE STATUS
         TBM       1,CHT.CHST,X3   WAS IT A POST PGM CONTROLLED INT?
         BS        SI.2.00         BR IF PPCI
         TBM       23,IOQ.FLGS,X2  WAS IT AN EXECUTE CHANNEL PROGRAM?
         BS        SI.8.00         BR IF EXEC CHAN PGM
         SPACE     1
*
** PROCEDURE TO PROCESS INTERRUPT FROM NORMAL SIO
*
         TBM       13,CHT.CHST,X3  DEVICE END SET?
         BNS       *CHT.EXIT,X3    EXIT INTERRUPT LEVEL IF NO DEVICE END
         LH        R6,CHT.CHST,X3  GET CHAN/DEV STATUS
         STH       R6,IOQ.IOST+1H,X2  STORE STATUS IN USER STATUS WORD
         LNH       R6,CHT.RBC,X3   GET NEGATIVE RESIDUAL BYTE COUNT
         ADMW      R6,IOQ.FCT3,X2  ADD REQUESTED BYTE XFER COUNT
         STW       R6,IOQ.UTRN,X2  UPDATE ACTUAL BYTE XFER COUNT
SI.1.00  EQU       $
         TBM       14,CHT.CHST,X3  UNIT CHECK SET?
         BNS       SI.1.05         BR IF NO UNIT CHECK
*** FETCH SENSE STATUS IF UNIT CHECK
         TBM       3,IOQ.CONT,X2   STATUS CHECKING INHIBITED?
         BS        SI.1.10         BR IF INHIBITED
         SBM       1,IOQ.STAT,X2   INDICATE SENSE STATUS SIO ISSUED
         LA        R6,DCA.SENI,X1  GET ADDRESS OF SENSE STATUS IOCD
         STW       R6,CHT.IOCL,X3  STORE IN CHT (IOCLA OF SICA)
         TRR       R3,R5           SAVE CHT ADDRESS IN R5
         LA        R3,DCA.TIMO,X1  GET ADDRESS OF DCA TIMEOUT TABLE
         ADI       R3,READ.OP*4    USE TIMEOUT FOR READ
         LW        R6,0,X3         GET TIMEOUT VALUE THIS OPERATION
         LW        R3,DCA.UDTA,X1  GET UDT ADDRESS
         STW       R6,UDT.PTOV,X3  STORE TIMEOUT VALUE IN UDT
         LH        R6,DCA.UADD,X1  GET ADDRESS THIS UNIT
         TRR       R5,R3           RESTORE CHT ADDRESS TO R3
         SIO       R6,0            ISSUE SENSE STATUS SIO
         SBM       0,IOQ.STAT,X2   SET I/O BUSY BIT IN IOQ FOR SENSE
         BU        *CHT.EXIT,X3    EXIT INTERRUPT LEVEL
*** CHECK FOR CHANNEL ERRORS
CHERMASK DATAW     X'1F00'         CHANNEL ERROR MASK
SI.1.05  EQU       $
         LW        R4,CHERMASK     GET CHANNEL ERROR MASK
         LMH       R6,CHT.CHST,X3  APPLY TO CHANNEL STATUS
         BZ       SI.1.10         BR IF NO CHANNEL ERRORS
         SBM      1,IOQ.IOST,X2   INDICATE ERROR IN USER STATUS
*** UNLINK IOQ ENTRY, REPORT I/O COMPLETE, AND PROCESS NEXT IOQ ENTRY
***      ENTRY : R1 = DCA ADDR
***              R2 = IOQ ADDR
SI.1.10  EQU      $
         LW       R3,DCA.CDTA,X1  GET CDT ADDRESS
         BL       S.IOCS3         UNLINK IOQ ENTRY FROM IOQ
         LB       R1,IOQ.PRGN,X2  GET DQE NUMBER
         BL       S.IOCS29        REPORT I/O COMPLETE TO EXEC
         LW       R2,IOQ.DCAA,X2  GET DCA ADDRESS
         LW       R3,DCA.UDTA,X2  GET UDT ADDRESS
         BL       IQ.             PROCESS NEXT IOQ ENTRY
         LW       R3,DCA.CHTA,R2  GET CHT ADDRESS FROM DCA
         BU       *CHT.EXIT,X3    EXIT INTERRUPT LEVEL VIA H.IOPX
         SPACE    1
*
** PROCEDURE TO PROCESS AN UNEXPECTED INTERRUPT. THIS PROCEDURE TREATS
** UNEXPECTED INTERRUPTS AS SPURIOUS. IF THE DEVICE IS CAPABLE OF
** GENERATING ASYNCHRONOUS INTERRUPTS, THIS PROCEDURE MUST BE MODIFIED.
**       ENTRY : R1 = DCA ADDR
**               R2 = IOQ ADDR
**               R3 = CHT ADDR
*
SI.2.00  EQU      $
         ABM      31,DCA.SINC,X1  INCREMENT LOCAL SPURIOUS INT COUNT
         ABM      15,CHT.SPUR,X3  INCREMENT GLOBAL SPURIOUS INT COUNT
         BU       *CHT.EXIT,X3    EXIT INTERRUPT LEVEL
         SPACE    1
*
** PROCEDURE TO PROCESS HIO INTERRUPT FOR A KILL
**       ENTRY : R1 = DCA ADDR
**               R2 = IOQ ADDR
**               R3 = CHT ADDR
*
SI.3.00  EQU      $
         LW       R3,DCA.CDTA,X1  GET CDT ADDRESS
         BL       S.IOCS3         UNLINK IOQ ENTRY FROM IOQ
         TRR      R2,R3           IOQ ENTRY ADDRESS TO R3
         BL       S.IOCS15        DELETE IOQ ENTRY
         LW       R2,IOQ.DCAA,X1  GET DCA ADDRESS
         LW       R3,DCA.UDTA,X2  GET UDT ADDRESS
         BL       IQ.             PROCESS NEXT IOQ ENTRY
         LW       R3,DCA.CHTA,R2  GET CHT ADDRESS FROM DCA
         BU       *CHT.EXIT,X3    EXIT INTERRUPT LEVEL VIA H.IOPX
         SPACE    1
*
** PROCEDURE TO PROCESS HIO INTERRUPT FOR A DEVICE TIMEOUT
**       ENTRY : R1 = DCA ADDR
**               R2 = IOQ ADDR
**               R3 = CHT ADDR
*
SI.4.00  EQU      $
         SBM      1,IOQ.CONT,X2   INDICATE ERROR PROC INHIBITED
         SBM      6,IOQ.FLGS,X2   INDICATE UNRECOVERABLE I/O ERR
         SBM      1,IOQ.IOST,X2   INDICATE ERROR IN USER STATUS
         SBM      10,IOQ.IOST,X2  INDICATE TIMEOUT IN USER STATUS
         BU       SI.1.10          COMPLETE PROCESSING
         SPACE 1
*
** PROCEDURE TO PROCESS SIO SENSE STATUS INTERRUPT
**       ENTRY : R1 = DCA ADDR
**               R2 = IOQ ADDR
**               R3 = CHT ADDR
*
UNREMASK DATAW    X'FFF00000'      DEV DEPENDENT UNRECOVERABLE ERR MASK
*
SI.5.00  EQU      $
         LW       R6,DCA.SENS,X1   GET ERROR STATUS INFO
         STW      R6,IOQ.IST1,X2   STORE IN IOQ ENTRY
         LW       R4,UNREMASK      GET UNRECOVERABLE ERROR MASK
         LMW      R6,DCA.SENS,X1   MASK WITH ERROR STATUS
         BZ       SI.1.10          BR IF NO UNRECOVERABLE I/O ERRORS
         SBM      6,IOQ.FLGS,X2    SET IOQ ENTRY ERROR INDICATOR
         SBM      1,IOQ.IOST,X2    INDICATE ERROR IN USER STATUS
**** COMPLETE ERROR PROCESSING FOR UNRECOVERABLE ERROR
         TBM      0,IOQ.CONT,X2    TEST FOR NO-WAIT I/O
         BS       SI.5.02          BR IF NO-WAIT I/O
         TBM      1,IOQ.CONT,X2    TEST FOR ERROR PROCESSING INHIBITED
         BNS      SI.5.05          BR IF ERROR PROC NOT INHIBITED
SI.5.02  EQU      $
         LW       R3,IOQ.CDTA,X2   GET CDT ADDRESS FOR UNLINKING IOQ
         BL       S.IOCS3          UNLINK IOQ FOR NO-WAIT I/O W/ ERRORS
SI.5.05  EQU      $
         LB       R1,IOQ.PRGN,X2   GET DQE NUMBER FROM IOQ
         BL       S.IOCS29         REPORT I/O COMPLETE
         LW       R2,IOQ.DCAA,X2   GET DCA ADDRESS
         LW       R3,DCA.UDTA,X2   GET UDT ADDRESS
         ZMW      UDT.PTOV,X3      ZERO TIMEOUT VALUE
         ZBM      1,UDT.FLGS,X3    CLEAR I/O OUTSTANDING FLAG IN UDT
         SBM      0,DCA.FLAG,X2    INDICATE INTERRUPTS NOT EXPECTED
         LW       R3,DCA.CHTA,X2   GET CHT ADDRESS
         BU       *CHT.EXIT,X3     EXIT INTERRUPT LEVEL VIA M.IOPX
         SPACE    1
*
** PROCEDURE TO PROCESS_INTERRUPT FROM A CHANNEL PROGRAM
*
SI.8.00  EQU      $
*   EXECUTE CHANNEL PROGRAM NOT IMPLEMENTED IN MPX 1.4
         HALT
         TITLE    H.LPIOP          LOST INTERRUPT PROCESSOR
************************************************************************
*                                                                      *
*                  LI. - LOST INTERRUPT PROCESSOR                      *
*                                                                      *
************************************************************************
*                                                                      *
*   THIS ENTRY POINT IS CALLED BY S.IOCS5 (ON BEHALF OF THE REAL-TIME  *
*   CLOCK INTERRUPT HANDLER) WHEN A DEVICE TIMEOUT OCCURS, AND BY      *
*   H.IOCS,38 IN ORDER TO KILL AN OUTSTANDING I/O REQUEST.             *
*                                                                      *
*   IN BOTH CASES THE CURRENT I/O REQUEST IS TERMINATED WITH A HALT    *
*   I/O (HIO) INSTRUCTION. IF THE CONTROLLER RESPONDS TO THE HIO,      *
*   OP. PERFORMS THE REQUIRED INTERRUPT HANDLING. IF THE HIO TIMES OUT *
*   THE CONTROLLER AND DEVICE ARE PRESUMED MALFUNCTIONING.                 *
*                                                                          *
*   S.IOCS5 AND H.IOCS,38 ENTER LI. WITH INTERRUPTS BLOCKED. IN            *
*   ADDITION H.IOCS,38 ENTERS LI. WITH CONTEXT SWITCHING INHIBITED.        *
*                                                                          *
****************************************************************************
*                                                                          *
*        CALLER:                      S.IOCS5 ; H.IOCS,38                  *
*                                                                          *
*        CALLERS PRIORITY:            REAL-TIME CLOCK INTERRUPT LEVEL      *
*                                                                          *
*        INTERRUPTS:                  RT CLOCK INTERRUPT LEVEL ACTIVE      *
*                                     INTERRUPTS BLOCKED                   *
*                                     CONTEXT SWITCHING INHIBITED          *
*                                                                          *
*        CALLING SEQUENCE:            BL    *4W,X2                         *
*                                                                          *
*        REGISTERS IN:                R2 = HAT ADDRESS                     *
*                                     R3 = UDT ADDRESS                     *
*                                                                          *
*        RETURN SEQUENCE:             TRSW  R0                             *
*                                                                          *
*        REGISTERS OUT:               NONE REQ'D                           *
*                                                                          *
****************************************************************************
         SPACE     3
         BOUND     1W
LI.      EQU       $
*
** PROCEDURE TO TERMINATE DEVICE AND FORCE AN INTERRUPT VIA HALT I/O
**       ENTRY : R2 = DCA ADDR
**               R3 = UDT ADDR
*
         LW        R2,UDT.DCAA,X3  GET DCA ADDRESS
         ABM       31,DCA.LINC,X2  UPDATE LOST INTERRUPT COUNT
         SBM       1,DCA.FLAG,X2   INDICATE HALT I/O ISSUED
         BS        LI.1.00         BR IF HALT I/O TIMED OUT
         LW        R6,LI.TIMO      GET TIMEOUT VALUE FOR HALT I/O
         STW       R6,UDT.PTOV,X3  STORE IN UDT
         LH        R6,UDT.CHAN,X3  GET DEVICE ADDRESS
         HIO       R6,0            ISSUE HALT I/O
         TRSW      R0              RETURN TO CALLER
         SPACE     2
*
** PROCEDURE TO CLEAN UP WHEN HALT I/O TIMES OUT
*
LI.1.00  EQU       $
         ZBM       1,DCA.FLAG,X2   CLEAR HIO ISSUED FLAG
         STW       R0,LI.RETN      SAVE RETURN ADDRESS
         LW        R1,DCA.COTA,X2  GET COT ADDRESS FOR THIS UNIT
         BL        S.IOCS31        MARK UNITS OFFLINE AND MALFUNCTIONIN
         LW        R3,DCA.COTA,X2  GET COT ADDRESS
         LW        R2,DCA.IOQA,X2  GET CURRENT I/O QUEUE ENTRY ADDRESS
         BL        S.IOCS3         UNLINK IOQ ENTRY
         TBM       22,IOQ.FLGS,X2  THIS A KILL REQUEST?
         BNS       LI.1.05         BR IF DEVICE TIME OUT
*** FINAL CLEANUP FOR KILL REQUEST
```

```
         TRR       R2,R3              IOQ ENTRY ADDRESS TO R3
         BL        S.IOCS10           DEALLOCATE IOQ ENTRY
         BU        LI.1.10
*** FINAL CLEANUP FOR DEVICE TIME OUT
LI.1.05  EQU       $
         LW        R6,IOQ.UDTA,X2     GET UDT ADDRESS
         STW       R6,LI.UDTA         SAVE FOR LATER
         SBM       6,IOQ.FLGS,X2      INDICATE UNRECOVERABLE I/O ERR
         SBM       4,IOQ.IOST,X2      INDICATE DEVICE INOPERABLE
         SBM       10,IOQ.IOST,X2     INDICATE DEVICE TIME OUT
         LB        R1,IOQ.PRGN,X2     GET DQE NUMBER
         BL        S.IOCS29           REPORT I/O COMPLETE
         LW        R3,LI.UDTA         RESTORE UDT ADDRESS
         BL        IQ.                RESTART IOQ TO FLUSH EXISTING ENTRIES
LI.1.10  EQU       $
         LW        R0,LI.RETN         GET RETURN ADDRESS
         TRSW      R0                 RETURN TO CALLER
LI.RETN  RES       1W                 LI. RETURN ADDRESS SAVE AREA
LI.UDTA  RES       1W                 LI. UDT ADDRESS SAVE AREA
LI.TIMO  DATAW     -2                 TIMEOUT VALUE FOR HIO
         TITLE     H.LPIOP            SPECIAL POST XFER PROCESSOR
************************************************************************
*                                                                      *
*                  PX. - SPECIAL POST XFER PROCESSOR                   *
*                                                                      *
************************************************************************
*                                                                      *
*   THIS ENTRY POINT IS CALLED BY S.IOCS1 (GENERALIZED I/O POST        *
*   PROCESSING) WHENEVER RETURN FROM OP. IS VIA "POSTPROS". THIS       *
*   ENTRY POINT EXECUTES AT TASK PRIORITY, AND THEREFORE CAN BE        *
*   USED TO PROVIDE ANY SPECIAL PROCESSING ASSOCIATED WITH A DATA      *
*   TRANSFER (EG., DATA TRANSLATION) AT A LOW LEVEL OF SYSTEM          *
*   OVERHEAD.                                                          *
*                                                                      *
************************************************************************
*                                                                      *
*        CALLER:                      S.IOCS1                          *
*                                                                      *
*        CALLERS PRIORITY:            TASK LEVEL                       *
*                                                                      *
*        INTERRUPTS:                  UNBLOCKED                        *
*                                                                      *
*        CALLING SEQUENCE:            BL    *5W,X2                     *
*                                                                      *
*        REGISTERS IN:                R1 = FCB ADDRESS                 *
*                                     R2 = HAT ADDRESS                 *
*                                     R3 = UDT ADDRESS                 *
*                                                                      *
*        RETURN SEQUENCE:             TRSW  R0                         *
*                                                                      *
*        REGISTERS OUT:               R1 = FCB ADDRESS                 *
*                                                                      *
************************************************************************
         SPACE     3
         BOUND     1W
PX.      EQU       $
*   DEVICE DEPENDENT CODE GOES HERE IF PX. IS REQUIRED
```

```
          TRSW      R0
          TITLE     H.LPIOP             SYSGEN INITIALIZATION PROCESSOR
**********************************************************************
*                                                                    *
*                  SG. - SYSGEN INITIALIZATION PROCESSOR             *
*                                                                    *
**********************************************************************
*                                                                    *
*   THIS ENTRY POINT IS CALLED BY SYSGEN FOR THE PURPOSE OF INITIALIZ-*
*   ING CERTAIN HANDLER PARAMETERS, INITIALIZING DEVICE CONTEXT AREAS *
*   (DCA'S) AND INITIALIZING CERTAIN DATA STRUCTURE ELEMENTS DURING  *
*   THE CONSTRUCTION OF AN MPX-32 IMAGE.                             *
*                                                                    *
*   DCA'S FOR THIS HANDLER ARE CREATED VIA THE REPEATED ASSEMBLY OF  *
*   THE MACRO, "DCA.DATA". A MAXIMUM NUMBER OF DCA'S IS              *
*   CREATED DURING HANDLER ASSEMBLY. DURING THE EXECUTION OF THIS    *
*   ENTRY POINT, ONE DCA IS INITIALIZED FOR EACH UDT ENTRY CONTAINING *
*   THE NAME OF THIS HANDLER. ANY REMAINING DCA'S AND THE REMAINDER OF *
*   THE CODE IN THIS ENTRY WILL BE OVERLAID BY SYSGEN. THEREFORE THIS *
*   MUST PHYSICALLY BE THE LAST ENTRY POINT OF THE HANDLER.          *
*                                                                    *
**********************************************************************
*                                                                    *
*         CALLER:                       SYSGEN DURING IMAGE CONSTRUCTION *
*                                                                    *
*         CALLERS PRIORITY:             N/A                          *
*                                                                    *
*         INTERRUPTS:                   N/A                          *
*                                                                    *
*         CALLING SEQUENCE:             BL   LAST PHYSICAL ENTRY POINT *
*                                                                    *
*         REGISTERS IN:                 R3 = CHT ADDRESS             *
*                                                                    *
*         RETURN SEQUENCE:              TRSW  R0 (VIA M.XIR)         *
*                                                                    *
*         REGISTERS OUT:                NONE REQ'D                   *
*                                                                    *
**********************************************************************
          SPACE     3
          LPOOL                         LITERALS HERE TO AVOID OVERLAYING
DCASTART  EQU       $                   START OF DCA'S FOR THIS DEVICE
          REPT      2                   REPEAT COUNT FOR DCA CREATION
          DCA.DATA  2,0,4,4,4,4,4,;     LIBRARY MACRO - CREATE 1 DCA
                    4,4,4,4,4,4,4,4,4,4,4
          ENDR
          BOUND     1W
SG.       EQU       $
          M.EIR                         LIBRARY MACRO - ENTER INIT ROUTINE
          DCA.INI1  H.LPIOP,1           LIBRARY MACRO - DCA INITIALIZATION
*         XXX                           USER CODE HERE IF EXTRA DCA TO INIT
*         XXX                           NOTE: ALL REGS AVAILABLE, R2=DCA ADD
          DCA.INI2                      LIBRARY MACRO - DCA INITIALIZATION
          M.XIR     HAT                 LIBRARY MACRO - EXIT INIT ROUTINE
          END
```

16. IOP EIGHT-LINE FULL DUPLEX HANDLER (H.F8IOP)

The IOP Eight-Line Full Duplex Handler (H.F8IOP) is system level reentrant, i.e., only one copy is required regardless of the number of eight-line communication multiplexers configured in a system. Reentrancy is accomplished via Device Context Areas (DCA's). The appropriate number of DCA's are initialized in the SYSGEN initialization entry point. As with all XIO device handlers, interrupts are fielded by the XIO channel executive program (H.IFXIO) and then turned over to the service interrupt entry point of H.F8IOP for processing.

16.1 Entry Points

16.1.1 Opcode Processor (OP.)

This entry point provides the interface for processing an opcode stored in the user's File Control Block (FCB). Depending on the particular opcode, this processing may or may not involve device access.

The 8-line handler supports fourteen IOCS opcodes enabling support of all command codes recognized by the eight line multiplexer plus necessary standard functions, such as open. The opcodes and their functions are as follows:

OP	CALL	HANDLER FUNCTION	8 LINE COMMAND
00	M.FILE	Open - Perform inch if necessary	--
01	M.RWND	Sense Status	04
02	M.READ	Read - See Section 16.2	02 or 06
03	M.WRIT	Write - See Note 1	01,05 or FF
04	M.WEOF	NOP	03
05	SVC 1,X'25'	EXCPGM - Execute Channel Program	--
06	M.FWRD,R	Set Data Terminal Ready	17
07	M.FWRD	Reset Data Terminal Ready	13
08	M.BACK,R	Define ACE Parameters (INIT)	FF
09	M.BACK	Reset Request to Send	1B
0A	M.UPSP	Set Request to Send	1F
0B	SVC 1,X'3E'	Set or Reset Break - See Note 2	37 or 33
0C	SVC 1,X'0D'	Define Special Character	0B
0D	M.CLSE	Close	--
0E	---	Invalid Opcode - Abort Caller (IO07)	--
0F	--	Invalid Opcode - Abort Caller (IO07)	--

For an explanation of the individual command codes, see 8-Line Asynchronous Communications Multiplexer Technical Manual, Publication Number 303-328510.

Note 1 - If bit 1 is set in FCB.SCFG, the write is interpreted as an init and a "define ACE parameters" command (FF) is issued. If bit 1 in FCB.SCFG is reset and bit 3 in FCB.SCFG is set, a "write with input subchannel monitoring" command (05) is issued. If both bits 1 and 3 in FCB.SCFG are reset, a "write" command (01) is issued.

Note 2 - If bit 0 in FCB.SCFG is set, a "set break" command (37) is issued. If bit 0 in FCB.SCFG is reset, a "reset break" command (33) is issued. Note it is not necessary to issue a "reset break" after a break interrupt is received.

16.1.2 I/O Queue Processor (IQ.)

This entry point performs standard I/O queue scheduling.

16.1.3 Service Interrupt Processor (SI.)

This entry point consists of the six standard subroutines described in Section 16.3.

16.1.4 Lost Interrupt Processor (LI.)

This entry point performs standard lost interrupt and kill I/O processing.

16.1.5 Post Transfer Processing (PX.)

This entry point performs conversion of lower case ASCII to upper case on formatted read operations. This processing can be inhibited by setting bit 10 in FCB.GCFG.

In addition, reads performed for TSM tasks must have the data copied from the operating system buffer to the user's buffer.

16.1.6 Pre-SIO Processor (PRE.SIO)

This entry point is called by XIO.SUB prior to issuing an SIO. This entry point checks for asynchronous attention interrupts on the read subaddress while a write is in progress for half duplex devices. If this is true, a break is issued to the task and the interrupt routine is exited.

16.1.7 SYSGEN Initialization Processor (SG.)

This entry point creates DCA's at assembly time and initializes them at SYSGEN time. In order to implement the full duplex capabilities of H.F8IOP, it is necessary to have two Unit Definition Table (UDT) entries per port (i.e., one UDT for read, one UDT for write). To accomplish this, SYSGEN directives must specify both the read subaddresses configured (0 through 7) and the write subaddresses configured (8 through F). If half duplex operation is desired, only the read subaddresses need to be specified in the SYSGEN directives. TSM terminals must have only the read subaddresses specified.

Note that devices SYSGENed as half duplex can still be initialized as full duplex in order to make use of full duplex commands such as read echoplex.

16.2 Options

The following options can be requested for reads.

16.2.1 Read Echoplex

This option causes a "read" command (02) to be issued if bit 1 is set in FCB.SCFG. If the bit is not set, a "read echoplex" command (06) is issued.

16.2.2 ASCII Control Character Detect

This option allows input to terminate whenever a control character (X'00' through X'1F') or a delete (X'7F') is detected. This option can be selected by setting bit 0 in FCB.SCFG before issuing the read. This option is implied in the read echoplex command.

16.2.3 Special Character Detect

This option allows input to terminate whenever a predefined 8 bit character is detected. This option can be selected by setting bit 3 in FCB.SCFG before issuing the read.

16.2.4 Purge Input Buffer

This option specifies any input data held in the "type ahead" buffer is to be purged before any new incoming data. This option can be selected by setting bit 4 in FCB.SCFG before issuing the read. This option is forced following a ring or break interrupt.

16.3 Subroutines

16.3.1 NORMAL

This subroutine is called by XIO.SUB when an I/O operation completes normally or when error processing is inhibited. Special processing is not performed unless the operation is a formatted read.

For formatted reads that are not on a TSM device, a check is made for an ETX character. If one is found, the end of file indicator is set.

For formatted reads that are on a TSM device, in addition to ETX, a check is made for a carriage return, backspace, tab, or delete. Appropriate actions are taken if any of these are found.

16.3.2 UNEXPT

This subroutine is called by XIO.SUB when an unexpected interrupt occurs. A sense status command is issued in order to determine the reason for the interrupt and the interrupt routine is exited.

16.3.3 SNSNOIOQ

This subroutine is called by XIO.SUB in response to an interrupt generated by a sense command issued by the UNEXPT subroutine. The sense data is examined to determine whether the unexpected interrupt was due either to a ring, break, or DSR or RLSD failure. If none of these is true, the routine is exited. If the interrupt was due to a ring, bit UDT.LOGO is set in the UDT. If the interrupt was due to DSR or RLSD failure, bit

UDT.DEAD is set in the UDT. A break is then issued to the task which has the device allocated. If another ring is expected, the task must reset UDT.LOGO.

16.3.4 SENSE

This subroutine is called by XIO.SUB when a I/O operation completes in error. A sense status command is executed prior to calling this subroutine. This subroutine examines the status to determine if the error was due to DSR or RLSD failure. If so, bit UDT.DEAD is set in the UDT and an error flag is set in the IOQ.

Next, the status is tested for attention, unit check or unit exception. If any of these occurred, a break is issued to the task and the error bit is set. Status is then tested for channel errors. If errors are found, the error bit is set in the IOQ.

16.3.5 CENODE

This subroutine is called by XIO.SUB when channel end but no device end is found in the status. This condition does not occur with the eight-line asynch.

16.3.6 TIMEO

This subroutine is called by XIO.SUB when an I/O operation does not complete and times out. An HIO (halt I/O) is issued prior to calling this subroutine. TIMEO processing consists of setting the error processing inhibit bit in the IOQ and returning to XIO.SUB.

APPENDIX A

MPX-32 MACRO CROSS REFERENCE

MACRO NAME	USAGE	SECTION
M.ACTV	Activate Task	3.2.15, 3.2.53
M.ADRS	Memory Address Inquiry	3.2.3
M.ALOC	Allocate File or Peripheral Device	3.2.21, 3.2.55
M.ANYW	Wait for any No-Wait Operation Complete; Message Interrupt or Break	3.2.37
M.ASYNCH	Set Asynchronous Task Interrupt	3.2.68
M.BACK	Backspace File or Record	1.12.1, 3.4.9, 3.4.19
M.BORT	Abort Specified Task	3.2.19
	Abort Self	3.2.20
	Abort with Extended Message	3.2.28
M.BRK	Break/Task Interrupt Link	3.2.46
M.BRKXIT	Exit from Task Interrupt Level	3.2.48
M.CALL	Call to System Module	1.12.2
M.CDJS	Submit Job from Disc File	3.2.27
M.CLSE	Close File	1.12.3, 3.4.23
M.CONADB	Convert ASCII Decimal to Binary	3.6.7
M.CONAHB	Convert ASCII Hexadecimal to Binary	3.6.8
M.CONBAD	Convert Binary to ASCII Decimal	3.6.9
M.CONBAH	Convert Binary to ASCII Hexadecimal	3.6.10
M.CONN	Connect Task to Interrupt	3.2.10
M.CREATE	Create Permanent File	3.7.12
M.CWAT	System Console Wait	3.4.26

MACRO NAME	USAGE	SECTION
M.DALC	Deallocate File or Peripheral Device	3.2.22
M.DATE	Date and Time Inquiry	3.2.70
M.DEBUG	Load and Execute Interactive Debugger	3.2.29
M.DELETE	Delete Permanent File or Non-SYSGEN Memory Partition	3.7.14
M.DELTSK	Delete Task	3.2.31
M.DEVID	Get Device Mnemonic or Type Code	3.2.71
M.DFCB	Create a File Control Block (FCB)	1.12.4
M.DFCBE	Create an Expanded File Control Block (FCB)	1.12.5
M.DISCON	Disconnect Task from Interrupt	3.2.38
M.DLTT	Delete Timer Entry	3.2.6
M.DSMI	Disable Message Task Interrupt	3.2.57
M.DSUB	Disable User Break Interrupt	3.2.73
M.DUMP	Memory Dump Request	3.2.12
M.EAWAIT	End Action Wait	3.1.40
M.EIR	Resident System Module Initialization Entry Macro	1.12.6
M.ENMI	Enable Message Task Interrupt	3.2.58
M.ENUB	Enable User Break Interrupt	3.2.72
M.EXCL	Free Shared Memory (EXCLUDE)	3.5.14
M.EXIT	Terminate Task Execution	3.2.18, 3.2.52
M.FADD	Permanent File Address Inquiry	3.2.2
M.FD	Free Dynamic Extended Indexed Data Space	3.5.9
M.FE	Free Dynamic Task Execution Space	3.5.11
M.FILE	Open File	3.4.1
M.FSLR	Release Synchronization File Lock	3.7.25

MACRO NAME	USAGE	SECTION
M.FSLS	Set Synchronization File Lock	3.7.24
M.FWRD	Advance File or Record	1.12.7, 3.4.7, 3.4.8
M.FXLR	Release Exclusive File Lock	3.7.23
M.FXLS	Set Exclusive File Lock	3.7.22
M.GADRL	Get Address Limits	3.2.41
M.GD	Get Dynamic Extended Indexed Data Space	3.5.8
M.GE	Get Dynamic Task Execution Space	3.5.10
M.GMSGP	Get Message Parameters	3.2.35
M.GRUNP	Get Run Parameters	3.2.36
M.HOLD	Program Hold Request	3.2.25
M.ID	Get Task Number	3.2.32
M.INCL	Get Shared Memory (INCLUDE)	3.5.13
M.INIT	Initialize Device Handler Parameters	1.12.8
M.INITX	Initialize Device Handler Parameters	1.12.9
M.INT	Activate Task Interrupt	3.2.47
M.IOFF	Inhibit Interrupt Signals	1.12.10
M.IONN	Allow Interrupt Signals	1.12.11
M.IVC	Connect Entry Point to Interrupt Vector Location	1.12.12
M.KILL	Halt Computer	1.12.13
M.LOG	Permanent File Log	3.2.33, 3.2.61
M.MODT	Build Module Address Table Entry	1.12.14
M.OLAY	Load Overlay Segment	3.2.13
	Load and Execute Overlay Segment	3.2.14
M.OPEN	Allow Context Switching	1.12.15
M.PDEV	Physical Device Inquiry	3.2.1, 3.2.56

MACRO NAME	USAGE	SECTION
M.PERM	Change Temporary File to Permanent	3.7.13
M.PGOW	Task Option Word Inquiry	3.2.24
M.PRIL	Change Priority Level	3.2.9
M.PTSK	Parameter Task Activation	3.2.40
M.RCVR	Receive Message Link Address	3.2.43
M.READ	Read Record	3.4.3
M.RELP	Release Dual Ported Disc/Release FHD Port	3.4.27
M.RESP	Reserve Dual Ported Disc/Reserve FHD Port	3.4.24
M.RRES	Release Channel Reservation	3.4.13
M.RSML	Resourcemark Lock	3.2.62
M.RSMU	Resourcemark Unlock	3.2.63
M.RSRV	Reserve Channel	3.4.12
M.RTNA	Return to Specified Address	1.12.16
M.RTRN	Return In-Line	1.12.17
M.RWND	Rewind File	3.4.2
M.SETS	Set User Status Word	3.2.7
M.SETT	Create Timer Entry	3.2.4
M.SHARE	Share Memory with Another Task	3.5.12
M.SHUT	Inhibit Context Switching	1.12.18
M.SMSGR	Send Message to Specified Task	3.2.44
M.SMULK	Unlock and Dequeue Shared Memory	3.5.19
M.SPAD	Scratchpad Reference	1.12.19
M.SRUNR	Send Run Request to Specified Task	3.2.45
M.SUAR	Set User Abort Receiver Address	3.2.26
M.SUME	Resume Task Execution	3.2.16

MACRO NAME	USAGE	SECTION
M.SUSP	Suspend Task Execution	3.2.17, 3.2.54
M.SVCT	Build SVC Table Entry	1.12.20
M.SYNCH	Set Synchronous Task Interrupt	3.2.67
M.TBRKON	Break Processing Entry	3.6.6
M.TDAY	Time-of-Day Inquiry	3.2.11
M.TRAC	System Trace	1.12.21
M.TSCAN	Syntax Scanner	3.6.2
M.TSTE	Arithmetic Exception Inquiry	3.2.23
M.TSTS	Test User Status Word	3.2.8
M.TSTT	Test Timer Entry	3.2.5
M.TURNON	Activate Program at Given Time-of-Day	3.2.66
M.TYPE	OPCOM Console Type	1.12.22, 3.4.14
M.UPSP	Upspace	3.4.20
M.USER	Username Specification	3.2.34
M.USHUT	Inhibit User Task Context Switching	1.12.23
M.WAIT	Wait I/O	3.4.25
M.WEOF	Write End-of-File (EOF)	3.4.5
M.WRIT	Write Record	3.4.4
M.XBRKR	Exit from Task Interrupt Level	3.2.48
M.XIEA	No-Wait I/O End Action Return	3.4.34
M.XIR	Resident System Module Initialization Exit Macro	1.12.24
M.XMEA	Exit from Message End Action Routine	3.2.50
M.XMSGR	Exit from Message Receiver	3.2.39
M.XREA	Exit from Run Request End Action Routine	3.2.51
M.XRUNR	Exit Run Receiver	3.2.49

MACRO NAME	USAGE	SECTION
M.XTIME	Task CPU Execution Time	3.2.65
HMP.INIT	MIOP Initialization	1.12.25
IB.INIT	MIOP Initialization	1.12.26

APPENDIX B

COMPRESSED SOURCE FORMAT

Compressed source files are blocked files that consist of 120 byte records. The last record may be less than 120 bytes and has a Data Type Code of 9F. The structure of a compressed record is described below.

Each record contains 6 control bytes:

1 byte	Data Type Code, BF (9F indicates last record)
1 byte	Byte Count, number of data bytes in record
2 bytes	Checksum, halfword sum of data bytes
2 bytes	Sequence Number, record sequence number starting at zero

Data is recorded as follows:

1 byte	Blank Count, number of blanks before data
1 byte	Data Count, number of data bytes
n-bytes	Actual ASCII data
.	
.	(this sequence is repeated until the end of a line is reached)
.	
1 byte	EOL character, FF