Gould MPX-32™

Release 3.3

Reference Manual

Volume I

Overview and System Services

December 1986

Publication Order Number: 323-001551-300

GOULD
Electronics

Gould Inc., Computer Systems Division
6901 West Sunrise Boulevard
Fort Lauderdale, FL 33313

HISTORY

The MPX-32 Release 3.0 Reference Manual, Publication Order Number **323-001550-000,** was printed June, 1982.

Publication Order Number **323-001551-100,** (Revision 1, Release 3.2) was printed June, 1983.

Publication Order Number **323-001551-200,** (Revision 2, Release 3.2B) was printed March, 1985.

Publication Order Number **323-001551-201,** (Change 1 to Revision 2, Release 3.2C) was printed December, 1985.

Publication Order Number **323-001551-300,** (Revision 3, Release 3.3) was printed December, 1986.

Note: For Release 3.3 of MPX-32, the reference material formerly included at the back of each MPX-32 Reference Manual volume was placed in a separate document and assigned Publication Order Number: **323-001550-300.**

The manual contains the following pages:

CONTENTS

CHAPTER 2 TASK STRUCTURE AND OPERATION OVERVIEW

CHAPTER 3 RESOURCE MANAGEMENT OVERVIEW

CHAPTER 4 VOLUME RESOURCE MANAGEMENT

CHAPTER 5 RESOURCE ASSIGNMENT/ALLOCATION AND I/O

CHAPTER 6 NONBASE MODE SYSTEM SERVICES

CHAPTER 7 BASE MODE SYSTEM SERVICES

FIGURES

TABLES

Documentation Conventions

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

lowercase letters

In directive 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. For example,

```
!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 directive syntax, uppercase letters specify a keyword must be entered as shown for input, and is printed as shown in output. For example,

```
SAVE filename
```

means enter SAVE followed by a file name. For example,

```
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,

```
{counter  }
{startbyte}
```

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 can 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 can be repeated. For example,

name [,name]...

means one or more names separated by commas can be entered.

Vertical Ellipsis

.
.
.

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

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

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

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. For example, (234).

Underscore

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

ACTIVATE taskname

means spell out the directive 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.

CHAPTER 1

INTRODUCTION

1.1 System Description

The Mapped Programming Executive (MPX-32) is a disc-oriented, multiprogramming operating system that supports concurrent execution of multiple tasks in interactive, batch, and real-time environments. MPX-32 provides memory management, terminal support, multiple batch streams, and intertask communication.

MPX-32 employs the SelMAP to fully support the 16MB physical address space of the CONCEPT/32 computers. Each task executes in a unique address space which may be expanded under task control up to 2MB of memory on the 32/27 and 32/87, or 16 MB on the 32/67 and 32/97. An integrated CPU scheduler and a swap scheduler provide efficient use of main memory by balancing the in-core task set based on time-distribution factors, software priorities, and task state queues. The SelMAP is used to perform dynamic relocation of tasks during inswap.

Tasks operating under MPX-32 can be activated and/or resumed by hardware interrupts, system service requests, interactive commands, job control directives, or by the expiration of timers. Multiple copies of a task can be executed concurrently in interactive, batch, or real-time environments. Through its various scheduling capabilities, MPX-32 provides the flexibility needed to adapt system operation to changing real-time conditions.

The MPX-32 software package is composed of various software modules including the resident operating system (IOCS, CPU and swap schedulers, Resource Allocator, Volume Management Module, re-entrant system services, and device and interrupt handlers), a Terminal Service Manager (TSM), a system generator (SYSGEN), and utilities such as a Volume Formatter and Volume Manager. Figure 1-1 describes the system nucleus, processors, and utilities.

The Internal Processing Unit (IPU) is a second central processor designed to work with a CPU to increase system throughput. The IPU is attached to the SelBUS like the CPU and shares all memory (including the resident operating system area) with the CPU. The IPU's function is to execute user task level code in parallel with CPU operation. (The IPU is optional hardware and must be specified during SYSGEN for use on a system.)

To avoid contention between the IPU and CPU, there are limitations on what the IPU can do:

- It cannot communicate with peripherals (perform I/O).
- It cannot process all supervisor call (SVC) system services.
- It cannot execute interrupt control instructions.

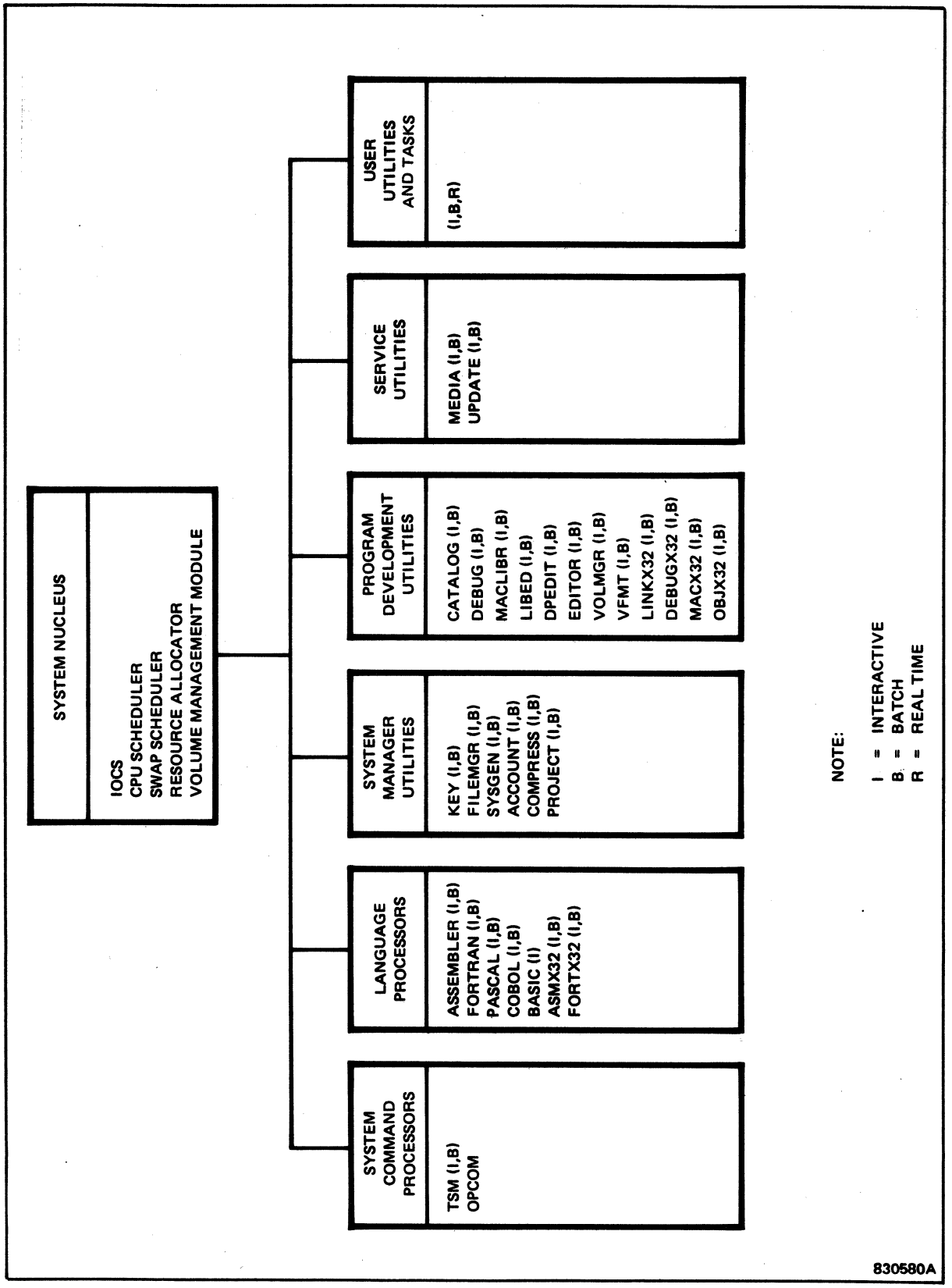


Figure 1-1. MPX-32 Processors and Utilities

Therefore, the IPU and CPU manage task execution transparently around the IPU limitations. For example, if the IPU is executing a task and encounters a service it cannot perform, a trap is sent to the CPU, the CPU takes over execution of the task at that point, and the task remains in the CPU until completion or reselection for IPU execution.

MPX-32 standard features include:

- Support for the full 16MB physical addressability of the CONCEPT/32 computers
- Up to 255 tasks executing concurrently
- 64 software priority levels, 10 of which are time distributed
- Servicing of all standard XIO peripheral devices
- Standard handlers for interrupts and traps
- Intertask communications, including send/receive
- Intertask shared memory partitions, such as Global Common and Datapool
- Dynamic allocation and deallocation of memory and peripherals
- Multiple batch streams, including multiple spooled input and output queues
- Wait and no-wait I/O capabilities, including automatic blocking, buffering, and queueing
- Terminal support for up to 64 devices, including device independent operation and an extensive repertoire of on-line commands
- Automatic task reentrancy through separation of pure code and data areas
- Reentrant system services available to all tasks
- Several levels of system security, including access restrictions based on task ownership
- File management, assignment, and security
- Up to 245 logical files (files or devices) opened concurrently per task if both static and dynamic assignments are used
- Project accounting capability
- Transparent support of the IPU
- Automatic mounting of public volumes at IPL

MPX-32 uses hardware and software priorities for scheduling and executing tasks. Figure 1-2 shows the various MPX-32 software elements and the hardware and software priority levels that are assigned to each.

1.1.1 Hardware Interrupts/Traps

The CONCEPT/32 computers support up to 96 hardware interrupts and traps. See Tables 1-1 and 1-2. The exact number in a particular system is dependent on the user's requirements and the number of peripheral devices in the configuration.

The highest hardware priority levels in the system are reserved for the basic system integrity interrupts and traps. These include the power fail/power up traps and system override interrupts and traps. Lower levels are used for the I/O transfer interrupts, memory parity trap, console interrupt, and I/O service interrupts.

The next lower group of interrupts and traps are used for exceptional conditions, supervisor call requests, and real-time clock. The exceptional conditions include nonpresent memory trap, undefined instruction trap, privilege violation trap, and arithmetic exception interrupt.

All lower hardware priority levels are used for external interrupts. User tasks can be connected directly or indirectly to the external interrupts.

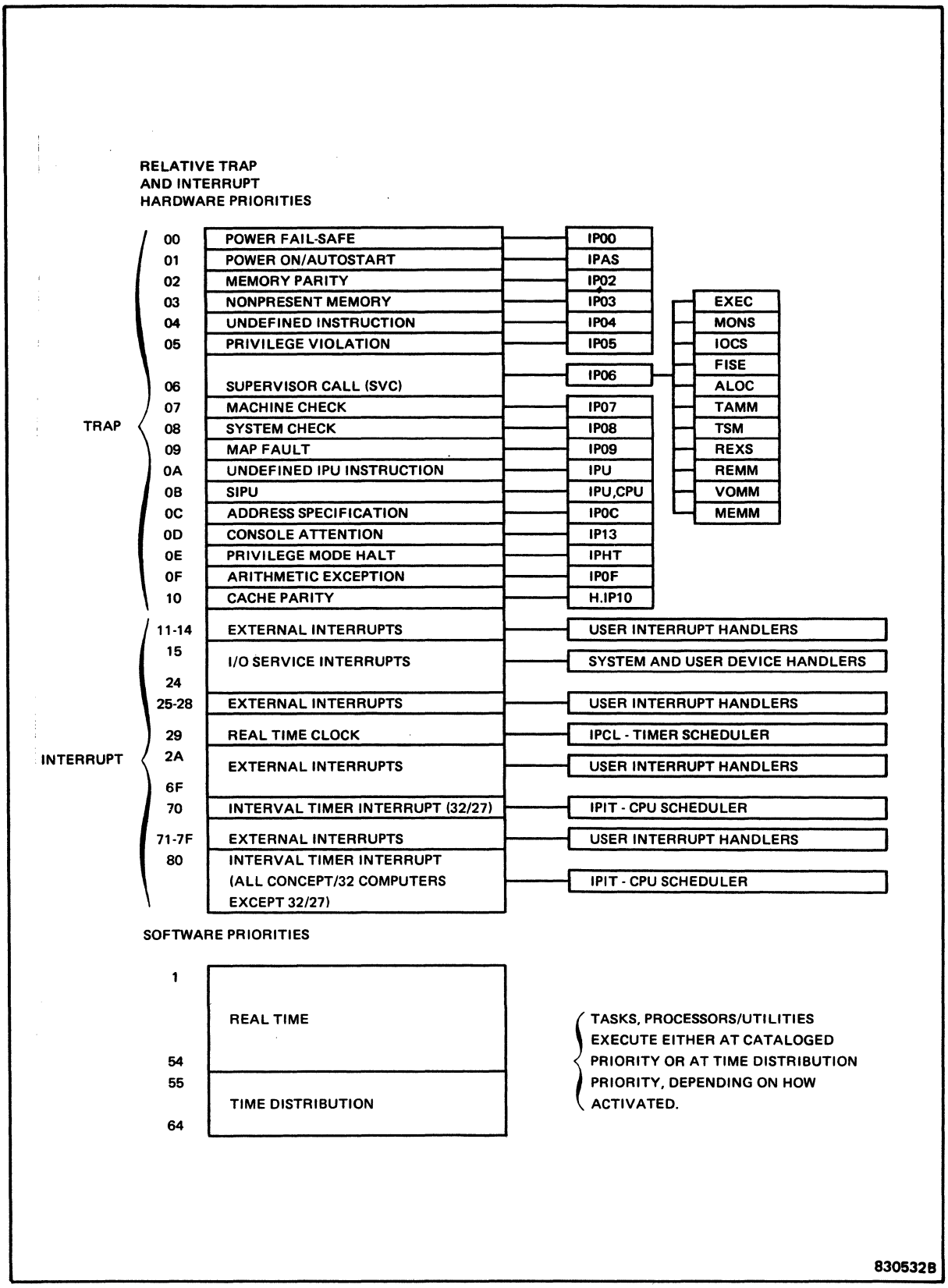


Figure 1-2. Hardware/Software Priorities

Table 1-1
CONCEPT/32 Trap Vectors

Relative priority	Default Trap Vector Location (TVL)		Trap condition
	CPU	IPU	
00	80	20	Power fail trap (power down)
01	84	24	Autostart trap (power up)
02	88	28	Memory parity trap
03	8C	2C	Nonpresent memory trap
04	90	30	Undefined instruction trap
05	94	34	Privilege violation trap
06	98	38	Supervisor call trap
07	9C	3C	Machine check trap
08	A0	40	System check trap
09	A4	44	MAP fault trap
0A	A8	48	Not used
0B	AC	4C	Undefined IPU instruction trap
0C	B0	50	Address specification trap
0D	B4	54	Console attention trap
0E	B8	58	Privilege mode halt trap
0F	BC	5C	Arithmetic exception trap
10	C0	60	Cache memory parity trap (all CONCEPT/32 computers except the 32/27)

Table 1-2
CONCEPT/32 Interrupt Vectors

Relative Priority	Default Interrupt Vector Location (IVL)	Interrupt condition
00	100	External/software interrupt 0
01	104	External/software interrupt 1
02	108	External/software interrupt 2
03	10C	External/software interrupt 3
04	110	I/O channel 0 interrupt
05	114	I/O channel 1 interrupt
06	118	I/O channel 2 interrupt
07	11C	I/O channel 3 interrupt
08	120	I/O channel 4 interrupt
09	124	I/O channel 5 interrupt
0A	128	I/O channel 6 interrupt
0B	12C	I/O channel 7 interrupt
0C	130	I/O channel 8 interrupt
0D	134	I/O channel 9 interrupt
0E	138	I/O channel A interrupt
0F	13C	I/O channel B interrupt
10	140	I/O channel C interrupt
11	144	I/O channel D interrupt
12	148	I/O channel E interrupt
13	14C	I/O channel F interrupt
14	150	External/software interrupts
↕	↕	↕
17	15C	External/software interrupts
18	160	Real-time clock interrupt
19	164	External/software interrupts
↕	↕	↕
5E	278	External/software interrupts
5F	27C	Interval timer interrupt (32/27)
60	280	External/software interrupts (all CONCEPT/32 computers except the 32/27)
↕	↕	↕
6F	2BC	Interval timer interrupt (all CONCEPT/32 computers except the 32/27)

1.1.2 Software Interrupt System

MPX-32 provides 64 software priority levels for controlling the user's application. All system scheduling is performed by priority. Multiple tasks can be assigned to any priority level, thereby achieving a high level of multiprogramming versatility. The software priority levels are used by the Resource Allocator for peripheral and memory allocation, by the I/O supervisor for the queueing of I/O requests, and by MPX-32 whenever CPU control is allocated.

1.1.3 Task Priority Levels

Priorities 55 to 64 are time-sliced to provide for round-robin time distribution among tasks of the same priority. Priorities 1 to 54 are not time distributed. A task's cataloged priority will be altered based on its eligibility to run. For example, a task's priority is boosted when an I/O operation is completed and restored after a minimal time quantum. Priority migration ensures maximum response to real-time events.

1.1.4 Supervision and Allocation

CPU scheduling is maintained through a set of state queues including the priority state chains and such execution states as suspended, queued for memory, queued for peripheral, I/O wait, etc. Each CPU dispatch queue entry defines all scheduling attributes of a single task. The entry typically migrates among the state queues as the task's execution eligibility changes. These state chains are also used by the swap scheduler to select candidates for swapping.

The CPU scheduler is invoked whenever a scheduling event occurs. Scheduling events include:

- External interrupts
- I/O completion
- Timer expiration
- Resource deallocation
- System service completion

IPU scheduling is maintained through state queues consisting of biased tasks (C.RIPU) scheduled in addition to MPX-32 normal state queues for nonbiased tasks (SQRT through SQ64). Any biased tasks are prioritized among themselves, and are scheduled for execution based on priority. Any nonbiased tasks are also prioritized among themselves, and are scheduled for execution according to priority. If a nonbiased task waiting for execution has a higher priority level than a biased task also waiting for execution, the nonbiased task will be selected for execution.

An optional scheduling algorithm is available to allow the user to boost the priority of IPU tasks and allow them to run in the CPU.

1.1.5 Memory Allocation

The unit of memory allocation is a map block, which is 2KW on the CONCEPT/32 computer. Memory is allocated to tasks as needed. All tasks are loaded discontiguously into a whole number of physical map blocks, utilizing the SelMAP to create their contiguous logical address space. No partial map blocks are allocated.

The MPX-32 memory allocation scheme allows tasks to dynamically expand and contract their address space by system service calls.

The unit of memory protection is called a protection granule and is 512W. Thus, it is possible to have protected data areas within a map block.

1.1.5.1 Dynamic Allocation

Dynamic allocation and deallocation are performed by the allocate and deallocate system services. These services can be used to dynamically allocate and deallocate any peripheral device, permanent and temporary disc files, or the System Listed Output (SLO) and System Binary Output files (SBO). By allocating peripheral devices dynamically, each task will have exclusive use of a peripheral only during the time required to perform the task's I/O. Therefore, when peripherals are unallocated, other tasks can use them on an as-needed basis.

Because the allocation of system-wide peripheral devices that are requested dynamically cannot be guaranteed, a task must be prepared to accept a denial return.

A task requesting additional memory is automatically queued until the memory can be allocated. For peripherals and file space, the caller can optionally queue for allocation or take alternative action.

1.1.6 File Management

In the MPX-32 operating environment, files are used in several ways. Permanent files are created for user programs, user data, and system programs. Temporary files provide system scratch storage, user scratch storage, and system output data storage for the system printer and card punch. Separation is maintained among files belonging to different users.

The file management system for MPX-32 consists of the resident Volume Management Module and the nonresident Volume Manager. Together, they supervise all file space on the discs.

1.1.6.1 Permanent Files

Residing in disc storage, permanent files are defined by entries in a directory which specify each file's name, binary creation date and time, absolute block number of resource descriptor, resource ID flag and type and other directory entry control information. Permanent files remain defined to the operating system until they are explicitly deleted.

All permanent files are referenced by pathname, and any number of tasks may access any permanent file for both input and output. To locate the directory entry for each file, MPX-32 employs a hashing technique which translates the characters in the file name to a specific location in the directory. For a complete description of pathnames, see Chapter 4.

Permanent files are classified as either fast or slow depending on the speed at which their directory entries can be located. A fast permanent file is one whose entry can be located with one disc access. Slow permanent files are not necessarily characterized by a unique mapping of names in the directory, and, therefore, two or more disc accesses may be required to find each file's entry.

1.1.6.2 Temporary Files

Temporary files are files whose definitions are eliminated from the system upon completion of the task requiring the space. Temporary file space is allocated and deallocated by the Volume Management Module which is responsible for maintaining space allocation maps for all available discs. Temporary files are typically used for system scratch storage and user scratch storage.

1.1.6.3 Random Access Files

Any disc file may be accessed randomly by record number through standard IOCS calls. The user sets a bit and specifies the relative disc block number in a File Control Block (FCB) to utilize this feature.

1.1.6.4 Disc File Protection

File protection mechanisms are available to prevent unauthorized access to and deletion of permanent files. Protection of individual files can be specified when the files are created. User files can also be protected on a per user basis. If a key is associated with an owner name in the M.KEY file, it must be entered at logon. Specific access rights are defined for each file by owner, project group, and other.

1.1.6.5 Dedicated System Files

To increase system throughput and minimize I/O delay time, IOCS supports disc buffered I/O by using special system files. Four dedicated file codes exist in the system. One file code is for buffered system input (SYC), two file codes are for buffered system output (SLO, SBO), and one file code is for a System Object file (SGO). A system file can be assigned to a file code in the same manner a device is assigned to a file code.

1.1.6.6 Multiprocessor Files

MPX-32 allows tasks executing in separate system environments to obtain concurrent access to selected files. The operating system maintains resource integrity on these files within the scope of volume management described in Chapter 4 of this manual. These files must reside on a volume accessible by a multiported device.

1.1.7 System Services

MPX-32 offers an extensive array of resident system service routines that can perform frequently required operations with maximum efficiency. Using the Supervisor Call instruction, tasks running in batch, interactive, or real-time environments can call these routines.

All system service routines are re-entrant. Thus, each service routine is always available to the currently active task.

The system service routines are standard modular components of MPX-32. The open-ended design of the system, however, gives each user freedom to add any service routines required to tailor MPX-32 to a specific application.

1.1.8 Input/Output Operations

The Input/Output Control System (IOCS) provides I/O services that relieve the programmer of detailed chores. While keeping software overhead to an absolute minimum, IOCS receives and processes all I/O requests for both user and tasks. It performs all logical error checking and parameter validation. IOCS also logically processes all I/O operations and assigns I/O control to the appropriate device handler. The device handler, in turn, executes the I/O data exchange, processes service interrupts, and performs device testing.

Input/output operations under MPX-32 include the following general capabilities: direct I/O, queued I/O requests, device independent I/O, device interchangeability, device reassignment, and disc-buffered (blocked) I/O.

1.1.8.1 Direct I/O

I/O can be issued directly to acquire data at rates which prohibit the overhead of IOCS. Mechanisms are provided in IOCS to ensure that no conflict occurs with IOCS file operations. The interface facilities provided in IOCS for direct I/O enable a task to gain exclusive use of an I/O channel.

1.1.8.2 Device-independent I/O

Normal I/O operations in the system occur to and from user-specified logical file codes. These file codes are assigned and reassigned to the physical device where the I/O commands are ultimately routed.

1.1.8.3 Logical File Codes

The user logical file code consists of one to three ASCII characters. For each file code defined and referenced by a user task, there is an entry in a File Assignment Table (FAT). The FAT entry describes the device controller channel and the device the file is assigned to. In the case of a disc that is a shared device, additional addressing information is provided for complete identification of the file. Each user task is allowed a maximum of 245 static and dynamic logical file assignments.

1.1.8.4 File Access

Both random and sequential file access is supported by IOCS. Random or sequential access is specified by the user. All files assigned to devices other than disc are considered sequential. A file assigned to disc may be referenced by both random and sequential transfers. Attempts to perform a write operation on a file specified as read-only, or attempts to circumvent disc file protection and security, are aborted.

1.1.9 Communications Facilities

MPX-32 offers complete facilities for providing communications between individual users, internal system elements, user tasks, and the operator and the system. Users communicate with one another through sharing permanent files, shared images, the communication region, Global Common and Datapool partitions, and job status flags which can be set and interrogated by system service routines. Tasks communicate with one another by messages or run requests.

1.1.9.1 Intertask Messages

Tasks can establish message receivers for intertask communication. Messages are buffered by MPX-32 in memory pool until the receiving task is eligible to receive. The receiving task is interrupted asynchronously and optionally responds to the sender. The sender optionally waits for a reply or elects to be interrupted asynchronously by a response. Messages can be queued to an arbitrary depth.

1.1.9.2 Run Requests

A task can send a run request to any other task. A run request is similar to a message, except that with a run request the receiver may not yet be in execution. In such cases, the receiving task is activated before the message is queued. The receiving task can process run requests at any time.

1.1.9.3 Global Common

Global Common is an area of memory that many programs can access by using symbolic names to identify specific storage cells. In this respect, Global Common is comparable to local common. Unlike local common, however, access to Global Common is not restricted to programs within a single task. Rather, programs belonging to many independent tasks can freely access the same data and exchange control information within the Global Common area.

1.1.9.4 Shared Images

A shared image allows base mode tasks to share both code and data, for example, shared subroutines and common data partitions. A shared image is built on disc by the base mode linker (LINKX32) and is loaded into memory upon inclusion by a task. In addition, nonbase mode tasks may include the shared image as an initialized dynamic data partition.

Shared images are distinct from static and dynamic common in that the memory is initialized with data from disc.

1.1.9.5 Datapool

Like Global Common, Datapool is an area of memory that many tasks can access using symbolic references. In addition to providing all the advantages of Global Common, Datapool provides a much wider range of structuring flexibility. For example, where Global Common symbolic references must follow the same order as the locations of the data in memory, symbolic references to Datapool may be entirely independent of the actual positioning of data within the memory area.

1.1.9.6 Internal Communications

Internal system elements communicate through temporary files, system queues, and the system communications region. The system communications region occupies approximately 2KW of lower memory. It contains information common to all system modules and processors.

1.1.10 Trap Processors

Trap processors are entered when any exceptional condition trap occurs. Certain traps indicate task errors, such as a reference to nonpresent memory, a privilege violation, or execution of an unimplemented instruction. These traps cause the violating task to be aborted. When the arithmetic exception trap occurs, the overflow condition is noted for use by the task in execution.

1.1.11 Timer Scheduler

The timer scheduler schedules events such as task activation, task resumption, flag setting and resetting, and interrupt activation on a timed basis.

1.1.12 Time Management

Time is kept in two different formats. The system maintains the time as a count of clock interrupts, and the date as an ASCII constant. In order to allow for easy time stamping of resources with the file system capabilities, time is also kept as a binary count of 100 microsecond units since midnight, and the date as the binary number of days since January 1, 1960. (See Appendix H for more details.)

When entering the date and time, the user is allowed to specify that daylight savings time is in effect and a correction factor for time zone. These features are provided for the user who wishes to have the system use a standard time base, such as GMT, for system operations but have the displayed values of date and time be equal to local time. For example, if a user states that local time is 10:00:00, daylight savings time is in effect, and there is a two-hour correction for time zone, the time kept by MPX-32 will indicate 07:00:00. The correction factor is kept so any user access of time indicates the local value.

Another feature allows the use of the international date format for entering the date. Instead of entering 10/17/80, 17OCT80 can be entered. The date is always displayed in the same format as it is entered at IPL time.

1.1.13 System Nonresident Media Mounting Task

The system nonresident media mounting task, J.MOUNT, mounts both formatted and unformatted media. J.MOUNT is normally in the Waiting for Run Request (RUNW) queue. When a program requires a volume to be mounted, a run request is sent to J.MOUNT which issues the mount message to the system console and then processes the user's reply.

1.2 System Command Processors

The Terminal Services Manager (TSM) and the interactive Operator Communications (OPCOM) command processor provide access to MPX-32 interactive, batch. and real-time processing environments.

1.2.1 Terminal Services Manager (TSM)

The Terminal Services Manager (TSM) provides interactive, time-shared access to the MPX-32 system for terminals connected either through ALIM or ACM controllers. It allows the terminal user to:

- Logon to MPX-32
- Access any MPX-32 processor
- Communicate with on-line users or the operator
- Account for use of computer resources
- Specify and pass parameters to interactive and batch tasks
- Automate a series of tasks into a job, or submit a stream of jobs
- Request assignment of any MPX-32 resource
- Specify alternative actions conditionally
- Logoff from MPX-32

1.2.2 Operator Communications (OPCOM)

Operator Communications (OPCOM) provides commands that can be used to interrogate the system and tune it for optimum response to changing conditions. The commands allow the user to:

- List the status of all queues, tasks, I/O controllers, and mounted volumes
- Control spooled print and punch output

- Hold and continue execution of tasks
- Activate and abort tasks
- Connect tasks to interrupts
- Establish resident and nonresident tasks
- Display time-of-day clock
- Create and delete timer scheduler queue entries
- Delete allocation queue entries
- Enable, disable, and trigger hardware interrupts
- Reserve devices, release them, and place them off-line or on-line
- Change the assignment of the system input device, the SGO file, and the destination of the SLO and SBO spooled output files
- Initiate the reading of the batch stream
- Issue system debugging commands

1.2.3 Batch Processing

Batch processing consists of spooling batch jobs to disc, interpreting job control statements, and directing listed and binary spooled output to destination files and devices. Multiple jobs are processed concurrently within limits established by SYSGEN and the availability of computer resources. Tasks comprising batch processing compete with each other and with nonbatch tasks for computer resources under standard MPX-32 allocation algorithms.

Each job is spooled to a separate System Control (SYC) disc file prior to processing. Jobs may be spooled to SYC files from card, magnetic tape, and paper tape peripheral devices, and from blocked, temporary, and permanent disc files. The OPCOM BATCH directive may be used to initiate spooling from peripheral devices and permanent files. The batch job from entry system service (M.BATCH) is used by TSM, and the Text Editor and can be invoked by a task to initiate spooling from permanent and temporary disc files. The TSM $BATCH or $SUBMIT directives can also be used to submit batch jobs.

Job sequence numbers reflect the order in which jobs are entered and uniquely identify each job and its tasks.

Upon job completion, a job's spooled listed and binary output is automatically routed to usable peripheral devices if no particular device(s) or permanent file(s) are specified for the job. Usable devices for automatic selection are specified by SYSGEN and OPCOM directives. Spooled output destination devices include line printers, card punches, magnetic tapes, paper tapes, and disc files. Spooled output is selected for processing based on the software priority of jobs and, within a given priority, on the order in which jobs complete processing.

1.3 Program Development Utilities

MPX-32 supports both nonbase and base mode programs. Refer to Section 2.2 for the nonbase and base task differences. Nonbase and base modes cannot be mixed; therefore, separate program development utilities exist for each mode.

Of the following utilities, only VOLMGR and J.VFMT are included on the MPX-32 Master SDT.

1.3.1 Task Cataloging (CATALOG)

By exercising the facilities of the cataloger, users create permanent nonbase mode load modules that execute as tasks on the MPX-32 system. During cataloging, relocatable object modules produced by the assembler or compilers are loaded and linked internally and externally to library subroutines. The linked body of code thus produced is then sent to a selected permanent file in relocatable or absolute format. In addition, the cataloger places a preamble on this file. This preamble contains a summary of the resources required by the task, such as memory, permanent files, and peripheral devices, and defines special task characteristics (shared, resident, etc.). Once created, a task is known to the system by the name of the permanent file where it resides. The task can then be activated, saved, restored, or otherwise operated on by specifying its name in the appropriate job control statement, system service call, or terminal directive.

1.3.1.1 Privilege

Whether a task is privileged or unprivileged can be defined by cataloger directives. The ability to specify a privileged operation for a task can be restricted by owner name.

By specifying whether tasks are privileged or unprivileged, users can control system security. Tasks designated to run privileged are free to execute any instruction in the instruction repertoire. They also have read/write access to all memory locations.

1.3.1.2 Overlays

For efficient use of memory, the cataloger provides the user with facilities for dividing large nonbase mode programs into overlays. Both the main program segment, the root, and the overlay segments can be cataloged in relocateable format. Individual overlays can be cataloged separately, permitting the user to modify or replace any overlay without disturbing any of the others. Flexible symbol linkage is provided between the root and its associated overlays and between individual overlays of various levels.

1.3.2 Task Debugger (MPXDB)

The task debugger is a directive-oriented processor that debugs a single, cataloged, nonbase mode user task. It can be accessed with a DEBUG directive in TSM, with a $DEBUG statement in batch, by coding an M.DEBUG service call within the cataloged task, or by using the break key after a task has been activated with TSM, in which case TSM provides the option of calling M.DEBUG.

If the task the debugger is connected to has a shared CSECT, the debugger must be attached at task activation (by the DEBUG directive in TSM or $DEBUG statement in batch). The shared CSECT task is then loaded as multicopied and breakpoints set in the CSECT does not impact other users of the shared CSECT.

MPXDB directives allow the user to:

- Trace task execution
- Set debugging traps within the task
- Display and/or alter contents of the task's logical address space, general purpose registers, etc.
- Watch for privileged task entry into the operating system or other areas of memory not usually accessed even by a privileged task
- Perform other operations that facilitate task debugging

1.3.3 Macro Assembler (ASSEMBLE)

The Macro Assembler translates nonbase assembler directives and source code into binary instructions for the CONCEPT/32 CPU.

1.3.4 Macro Library Editor (MACLIBR)

With the Macro Library Editor, nonbase mode macros that are used frequently can be placed in a macro library where they are available for use by the Macro Assembler. During execution, the Macro Library Editor transfers the macros from the Source Input File to the Macro Library File. The macros entered into the library are listed on an output file.

1.3.5 Subroutine Library Editor (LIBED)

The Subroutine Library Editor provides facilities for creating and modifying the nonbase mode System Subroutine Library and any number of user subroutine libraries. The user is provided with a listing of directives, module names, external definitions, the quantity of library and directory space remaining on the disc, and the modules that were specified for deletion but were not located in the library.

1.3.6 Datapool Editor (DPEDIT)

The Datapool Editor provides the ability to create and maintain dictionaries for access to static or dynamic Datapool common memory partitions.

1.3.7 Text Editor (EDIT)

The Text Editor provides directives for building and editing text files, merging files or parts of files into one file space, copying existing text from one location to another, and, in general, for performing editing functions familiar to users of interactive systems.

EDIT is typically used to create source files and to build job control files and general text files. A job file built in the editor can be copied directly into the batch stream using the editor BATCH directive.

1.3.8 Volume Manager (VOLMGR)

The Volume Manager creates or deletes permanent disc file space, special Global partitions, and/or a Datapool partition (one that can be dynamically allocated in memory when required by tasks). A primary use is to provide system and user permanent file backup.

1.3.9 Volume Formatter (J.VFMT)

The Volume Formatter formats volumes (discs). It can operate on a fully functional MPX-32 system or a starter system by the SDT.

1.3.10 Assembler/X32 (ASMX32)

The Assembler/X32 translates base mode assembler directives and source code into binary base mode instructions for the CONCEPT/32 CPU.

1.3.11 Macro Librarian/X32 (MACX32)

The Macro Librarian/X32 builds and maintains base mode macro libraries that are accessed by the Macro Assembler/X32. Frequently used base mode macros can be placed in the macro libraries for easy access.

1.3.12 Object Librarian/X32 (OBJX32)

The Object Librarian/X32 provides facilities for creating and modifying user object libraries. The object libraries contain object files to be used in base mode programs. The object librarian provides a log of the number of object files entered, the names of the object files, when each file was entered, and the amount of available library space.

1.3.13 Linker/X32 (LINKX32)

The Linker/X32 creates permanent base mode load modules that can execute as tasks on MPX-32. During linking, object modules produced by the Macro Assembler/X32 or compilers are loaded and linked internally and externally to library subroutines. The linked body of code becomes an executable image.

1.3.14 Symbolic Debugger/X32 (DEBUGX32)

The Symbolic Debugger/X32 is a directive-oriented processor used to debug base mode executable images created by the Linker/X32.

The debugger allows the user to:

- Debug interactively, with debugger directives controlling the execution of the program
- Access program locations (memory addresses) by using hexadecimal addresses, bases, or the global symbols defined in the source program. Addresses are displayed in hexadecimal format or relative to a base or global symbol.
- Display data in ASCII, hexadecimal, or instruction format
- Execute program instructions one at a time, showing the result after each instruction is executed, or set traps to allow execution to proceed through many instructions to a designated program checkpoint
- Define bases

- Debug privileged programs
- Print a record of the debugging session

1.4 Service Utilities

1.4.1 Source Update (UPDATE)

The Source Update utility provides facilities for revising source files. It permits the user to enter new files as well as to update existing files by adding, replacing, and deleting source statements. Input can be in either standard or compressed format, and either format may be selected for the output file. Source Update can also produce a listing of the control stream as it generates the output file.

1.4.2 Media Conversion (MEDIA)

The Media Conversion utility performs functions ranging from card duplication to merging multiple media inputs into single or multiple media outputs. It provides media editing, media-to-media conversion, code conversion, media copying, and media verification. Rather than restricting the user to a fixed set of functions, the Media Conversion utility is controlled by a language of directives.

1.5 System Manager Utilities

1.5.1 M.KEY Editor (KEY)

KEY is a utility used to build an M.KEY file for the MPX-32 system. The M.KEY file specifies valid owner names on the system and optionally sets, for each owner name:

- A key to restrict access to the owner name during logon and to the user name when accessing files
- OPCOM indicators restricting the owner's use of OPCOM directives
- An indicator that prevents the owner from cataloging privileged tasks (tasks that use privileged system services or privileged variations of unprivileged system services)
- An indicator that prevents the owner from activating tasks cataloged as privileged
- Default tab settings
- Default working volume and directory specification
- Default alphanumeric project names/numbers for accounting purposes

After KEY has been run, only those owners established in the M.KEY file are allowed to logon to the system and access files.

1.5.2 MPX-32 System Start-up, Generation, and Installation (SYSGEN)

Under MPX-32, the user can install a starter system by booting from the master System Distribution Tape (SDT). Using the starter system, which is fully operational, a user-configuration of the system can be generated with the SYSGEN utility (running either interactively or in batch). An on-line RESTART directive is available to test user-configured systems. When a system has been tested, the user can create his own SDT using the VOLMGR SDT directive.

When SYSGEN runs, system tables are constructed and linked to the resident system modules, handlers, and user-supplied resident modules and handlers as specified by SYSGEN directives. A resident system image is formed and subsequently written to a dynamically acquired permanent disc file. Concurrent with this process, a listing of directives is built and a load map of the system is generated. The load map can be saved on a system symbol table file specified by the user with the SYMTAB directive and used subsequently in patching the system.

A system debugger can also be configured in the resident system image to assist in patching or debugging resident system code, including user interrupt and I/O handlers.

1.6 Libraries

Subroutine Libraries

Subroutine libraries can be utilized to simplify the development of applications. Subroutines can be added, modified, or deleted. This permits one routine to be changed without having to reassemble or recompile all of the subroutines needed for a task. Only the task must be recataloged.

Subroutines on a subroutine library can be used by programs written in various languages, including Assembly. They are accessed as object modules when a task is cataloged. The subroutine library and directory for MPX-32 are called MPXLIB and MPXDIR. User subroutine libraries can be built and modified by the LIBED utility.

System Macro Libraries

Two macro libraries are supplied as part of the MPX-32 system. They are used only with programs written in Assembly language. The first, M.MPXMAC, should be accessed when code that uses MPX-32 system services is assembled. The second, M.MACLIB, is used when code contains RTM monitor service calls. These macro libraries provide macros containing equates for MPX-32 communication region variables.

The user can expand, contract, or modify a macro library by using the MACLIBR utility.

Other

The Scientific Subroutine Library is optionally available. It contains math and statistical routines for scientific and engineering applications. A user group library is also available. It is provided by and for users.

1.7 Minimum Hardware Configuration for MPX-32

Minimum hardware requirements for MPX-32 operation on a CONCEPT/32 computer are as follows:

- 128KW Memory
- Magnetic tape (class F) or IOP floppy disc
- IOP console
- Extended I/O disc

The minimum configuration must also include the prerequisites required to support the items listed, for example, controllers, formatters, etc.

Devices supported by MPX-32 are listed in Table 1-3. Where appropriate, the code used to access a device is shown in parentheses. The code indicating the appropriate device, such as TY for a terminal on an ALIM, is used when accessing devices connected with a communications link.

Table 1-3
MPX-32 Device Support

Model Number	Description
1603	Vector Processor 3300
1604	Vector Processor 6410
2345	Real-Time Option Module
3050	Multiprocessor Shared Memory System (MS)2
7410	Analog Digital Interface (ADI)
8000	I/O processor*
8001	I/O processor
8031	Line printer/Floppy disc controller (LP)
8033	IOP disc controller
8050	High Speed Tape Processor (HSTP) (XIO)
8055	Disc processor II
8110	32MB cartridge module disc*
8114	IOP disc controller with 32MB cartridge module disc*
8120	80MB Sealed media disc and IOP disc controller
8121	80MB Sealed media disc processor subsystem
8122	80MB Additional sealed media disc
8130	80MB Disc processor subsystem
8134	80MB Removable IOP disc subsystem
8140	300MB Disc processor subsystem
8144	300MB Removable IOP disc subsystem
8148	160MB Additional disc
8150	675MB Fixed module disc processor subsystem
8155	675MB additional fixed module disc
8160	Cache disc accelerator
8170	Floppy disc with controller (FL)*
8174	Floppy disc with controller (FL)
8175	Dual floppy disc with controller (FL)
8176	1.6MB additional floppy disc drive
8177	1.6MB floppy disc drive with rack mount enclosure
8190	Dual-port option kit
8191	Disc processor eight-drive option
8210	High speed tape processor subsystem 75 ips 9-Track 1600/6250 bpi (M9)*
8211	High speed tape processor subsystem 125 ips 9-Track 1600/6250 bpi (M9)*
8212	High speed tape processor subsystem 125 ips 9-Track 800/1600/6250 bpi (M9)
8255	Buffered tape processor
8310	Band printer (300 lpm) (64 character) (LP)
8311	Band printer (600 lpm) (64 character) (LP)
8312	Band printer with form length select switch
8313	Band printer with VFU (300 lpm) (64 character) (LP)
8314	Band printer with VFU (600 lpm) (64 character) (LP) lpm
8315	Band printer with VFU (1000 lpm) (64 character) (LP)
8317	96-character option set
8410	Quarter-inch tape drive
8510	Eight-line asynchronous communication controller*

* This product is no longer available but remains supported by MPX-32 in existing installations.

Table 1-3
MPX-32 Device Support (Cont.)

Model Number	Description
8511	Asynchronous Communication Multiplexer (ACM)
8512	Asynchronous Communication Multiplexer (ACM)
8520	Synchronous Communications Multiplexer (SCM)
8610	Alphanumeric CRT (CT or TY)
8846	160MB Disc processor subsystem
8856	340MB Disc processor subsystem
8858	340MB Additional disc
9020	Low Speed Tape Processor (LSTP) (XIO)
9024	Disc processor I (XIO)*
9103	Extended (Class D) General Purpose Multiplexer Controller (GPMC)
9109	Synchronous Line Interface Module (SLIM)
9110	Asynchronous Line Interface Module (ALIM)
9116	Binary Synchronous Line Interface Module (BLIM)
9131	High Speed Data Interface II (HSD II)
9132	High Speed Data Interface I (HSD I)*
9201	KSR Teletypewriter (10 cps) (CT or TY)*
9202	Teletypewriter (30 cps) (CT or TY)
9203	Alphanumeric CRT (95 character) (CT or TY)*
9223	Matrix printer (340 cps) (LP)
9225	Line printer (300 lpm) (64 character) (LP)*
9226	Line printer (600 lpm) (64 character) (LP)*
9237	Line printer (900 lpm) (64 character) (' P)*
9245	Line printer (260 lpm) (96 character) (LP)*
9246	Line printer (436 lpm) (96 character) (LP)*
9247	Line printer (600 lpm) (96 character) (LP)*
9260	Paper tape reader (PT)*
9264	Paper tape punch/spooler (120 cps) (PT)
9342	80mb removable expansion disc drive
9346	300mb removable expansion disc drive
9363	45 ips slave magnetic tape unit
9377	75 ips slave magnetic tape unit
9448-32	32 MB Cartridge module disc (XIO) (DM)*
9460	Paper tape reader with controller (300 cps) (PT)
9462	Paper tape reader/spooler (300 cps) (PT)
9508	10 MB Master cartridge disc drive (DM)*
9520	80 MB Master moving head disc (DM)*
9521	40 MB Master moving head disc (DM)*
9523	300 MB Master moving head disc (DM)
9561	45 ips Master magnetic tape unit 9-track (M9)*
9563	45 ips Master magnetic tape unit 9-track (M9)*
9567	Low speed tape processor subsystem 45 ips 9-track 800 bpi (M9)*
9568	Low speed tape processor subsystem 45 ips 9-track 800/1600 bpi (M9)
9571	Low speed tape processor subsystem 75 ips 9-track 800/1600 bpi (M9)
9577	75 ips Master magnetic tape unit 9-track (M9)*
9733-5	5 MB Fixed head disc (XIO) (DF)*

* This product is no longer available but remains supported by MPX-32 in existing installations.

CHAPTER 2

TASK STRUCTURE AND OPERATION OVERVIEW

2.1 Task Identification

Under MPX-32, the user can communicate with and control tasks either by task name or task number (unless a task is multicopied, in which case the task number is required). The task name is the name of the load module or executable image file containing the task. The task number is assigned when the task is activated and is a sequential 24-bit number concatenated with an eight-bit DQE index. Task numbers are unique for each task in the system.

Each task is also associated with an owner. Valid owner names are specified in the M.KEY file, if it exists; otherwise, all owner names are valid. An owner can have any number of tasks with the same or different task names active on the system at any point in time.

In addition to the task numbers, each batch job is assigned a unique sequence number when the job is entered in the batch stream.

Active tasks can be listed by:

- Task number
- Owner name
- Task name
- Batch sequence number (if batch)
- Pseudonym used by MPX-32 to further identify the task, e.g., by the terminal it is running on
- Any combination of the above

The system provides the OPCOM LIST directives and the system service M.ID for listing any active task by specifying a unique combination of these attributes.

2.2 Task Structures

A task is structured to meet a user's particular requirements by defining the contents of a unique address space. A unique address space is a mapped logical address space whose maximum size varies, according to computer type. The unique address maximum executable code region size depends on whether the nonbase or base instruction set is being used. See Table 2-1.

Table 2-1.
Nonbase Mode vs. Base Mode

	Nonbase	Base
Supported On	All CONCEPT/32 computers	All CONCEPT/32 computers except 32/27
Maximum Task Size	2 MB (32/27, 32/87) 16 MB (All others)	2 MB (32/87) 16 MB (All others)
Code/Data Size	0.5 MB	2 MB (32/87) 16 MB (All others)
Data-Only Size	1.5 MB (32/27, 32/87) 15.5 MB (All Others)	N/A
Shareable Area Called	CSECT	Read-only
Nonshareable Area Called	DSECT	Read/write
Created By	CATALOG	LINKER/X32
Exists On Disc As	Load module	Executable image

All tasks activated on a 32/27 or 32/87 have a 2MB logical address space.

Base mode tasks activated on all other CONCEPT/32 computers have a logical address space of 2MB or 32KW plus the task size, whichever is larger. The automatic logical address space sizing can be overridden by the SET LAS LINKX32 directive or the TSM $SPACE directive.

Nonbase mode tasks activated on computers other than the 32/67 and 32/97 have a logical address space of 2MB unless overridden by the $SPACE TSM directive.

A unique address space contains a copy of MPX-32 and a task that can:

- be nonshared
- share reentrant code and data with another task
- share memory (common storage or user defined use) with another task

The memory size minus the operating system size equals the maximum task size. The operating system size includes any static memory partitions and 4KW for use by the Volume Management Module.

Shared memory considerations are described in Chapter 3.

2.2.1 Nonbase, Nonshared Tasks

This type of address space contains a single task including its Task Service Area (TSA), its code section (CSECT - write protected memory containing code and pure data), and its data section (DSECT - read/write memory containing impure data). See Figure 2-1. Tasks which are not sectioned have only a DSECT, which contains the code and all data.

2.2.2 Base, Nonshared Tasks

This type of address space contains a single task including a Task Service Area (TSA), program stack, read/write image section, and read-only image section.

2.2.3 Multicopied Tasks

An owner or several owners can have tasks with the same name and the same load module active concurrently. This is accomplished by cataloging the task as multicopy. The task name is not sufficient to communicate with multicopy tasks; the task number must be used.

2.2.4 Shared Tasks

When a task is created, the user can specify that a program section is to be shared. A program section, CSECT or read only, consists of code and pure data. This section is write protected and mapped into the logical address space of each copy of the task. A separate data section, DSECT or read/write, is mapped into each logical address space, as illustrated in Figures 2-2 and 2-3. Shared tasks are implicitly multicopied tasks.

2.2.5 Unique Tasks

Although only one copy of a task that is unique can be active on the system at a given time, the MPX-32 run request mechanism can be used to queue run requests to the task, so that as soon as one user stops executing, another can begin. See Intertask Communication, Section 2.7.

2.3 Task Execution

Nonbase mode tasks are introduced to the system by a request to activate the cataloged load module by name. Activation can be requested in several different ways.

- Batch and interactive tasks are activated by the job control or TSM $RUN and $EXECUTE directives.

- Real-time tasks can be activated by the M.ACTV or M.PTSK system services. The requestor uses the M.PTSK service to rename the task or to specify additional or alternate resource requirements for the task. Real-time tasks can also be activated by the job control statement $ACTIVATE or the OPCOM directive, ACTIVATE.

Real-time tasks can also be activated by timers or interrupts.

Base mode tasks are entered by the operating system in one of two ways: the initial dispatch of the task, or as a result of an asynchronous event which includes messages, breaks, or end-action notification. Regardless of which way a task is entered, entry is through a call instruction.

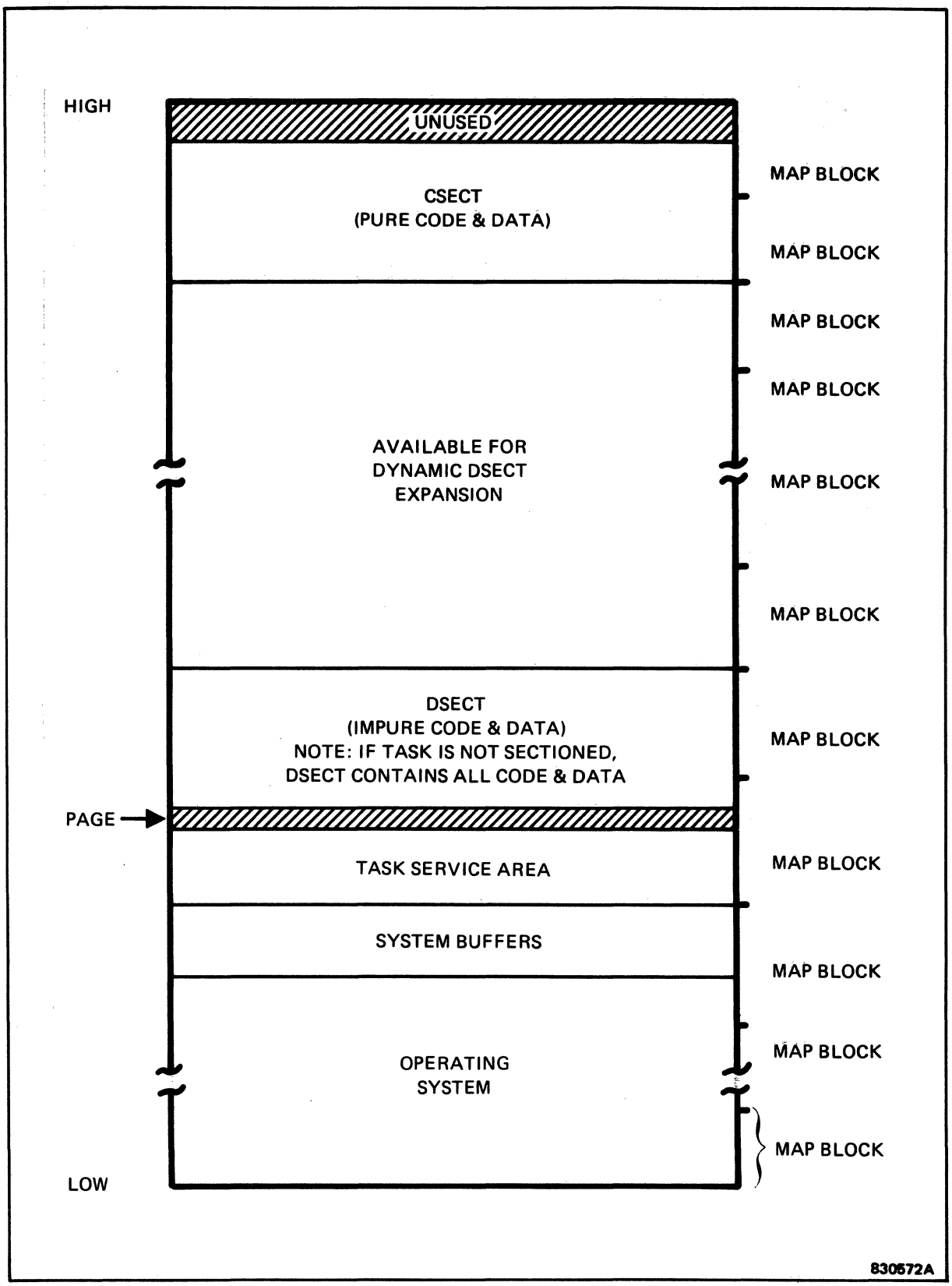


Figure 2-1. Nonbase Mode Nonshared Task Address Space

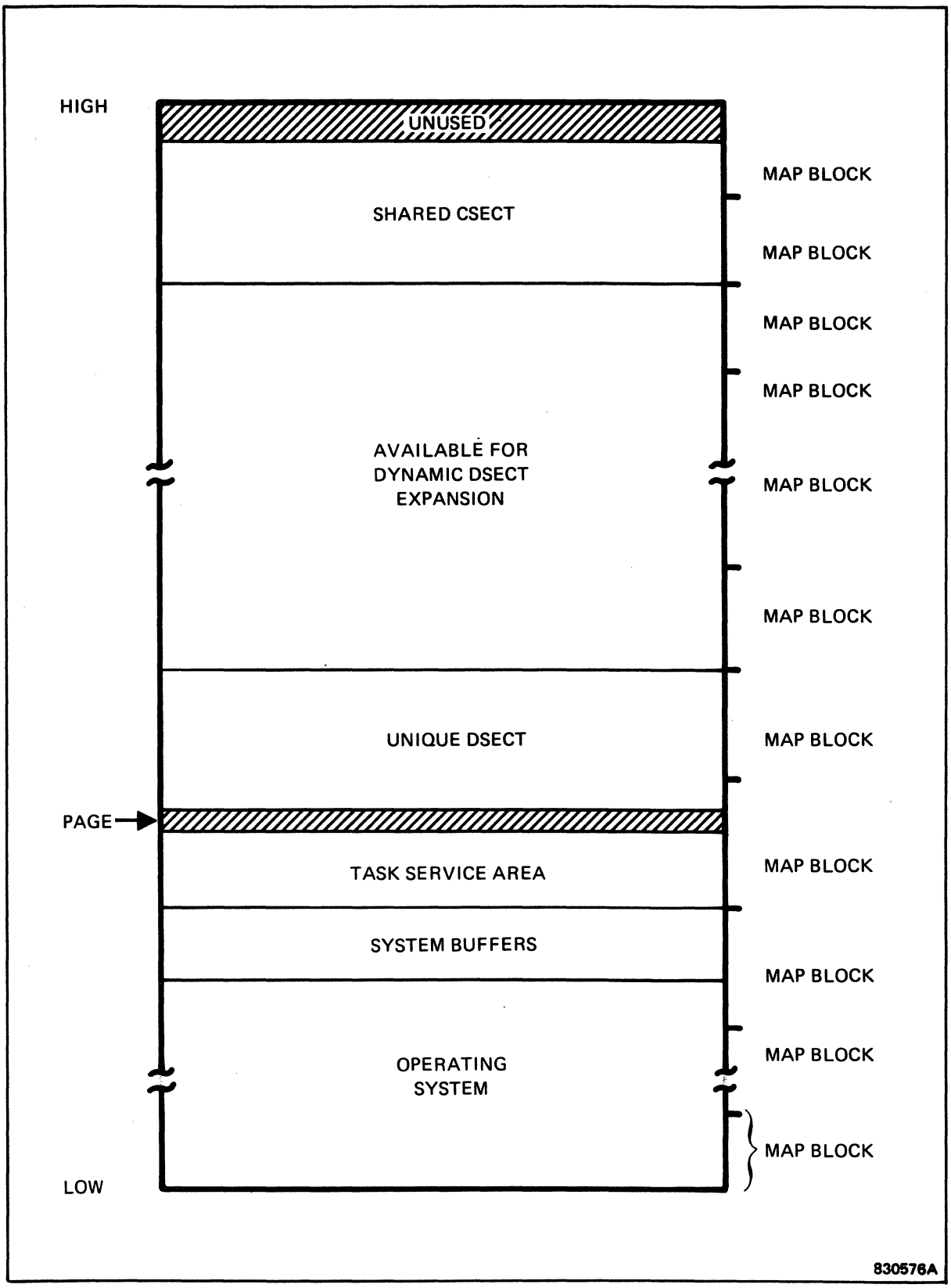


Figure 2-2. Nonbase Mode Shared Task Address Space

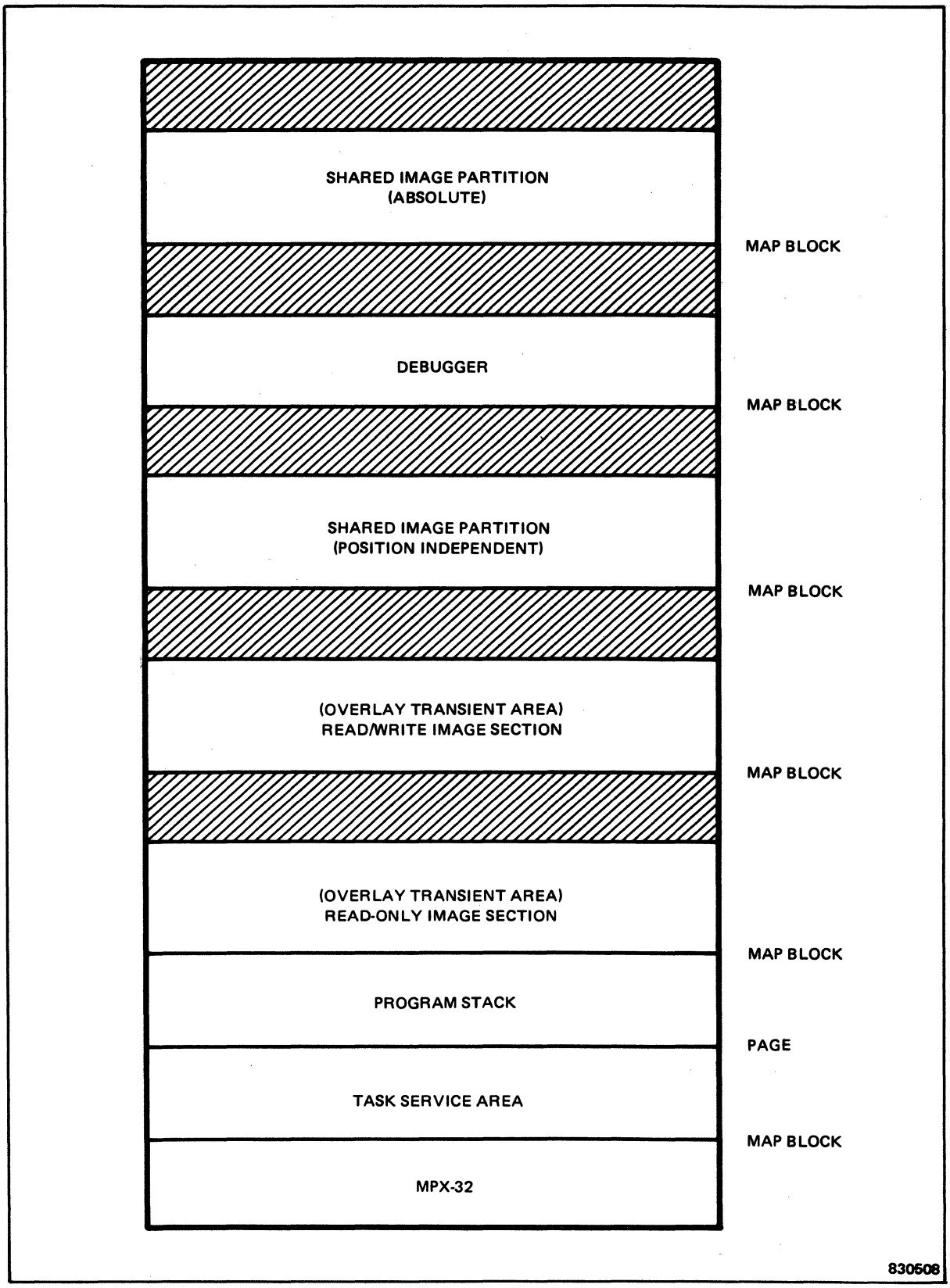


Figure 2-3. Base Mode Shared Task Address Space

2.3.1 Task Activation Sequencing (M.ACTV, M.PTSK)

The MPX-32 task management module performs task activation in the two following phases:

Phase 1 -- Activation

When a task is activated on the MPX-32 system, either by the M.ACTV service or the M.PTSK service, the MPX-32 resource manager runs for the task that issues the service call (the activating task). In many cases, the activating task is TSM or OPCOM. Running at the priority of the activating task, the resource manager constructs a rudimentary Task Service Area (TSA) for the new task in the task's address space and a rudimentary Dispatch Queue Entry (DQE) in the communications region. Data in the prototypes include: a task number, parameters passed with the task (M.PTSK), the Load Module Information Table (LMIT), and other basic data that define the task.

Initially, the DQE for the task is unlinked from the list of free DQE's maintained by the CPU scheduler and linked to the preactivation state queue (PREA). See Section 2.4.4. After the prototype TSA and DQE are constructed, the DQE is unlinked from the PREA state queue and linked to the appropriate ready-to-run queue. A context is set up in the prototype TSA so that the resource manager can gain control for the second phase of activation as soon as the new task becomes the highest priority ready-to-run task on the system. There are several cases where task activation does not continue at the end of phase 1:

- Activation with a run request for a single-copied task that is already active
- Timer activation requests (M.SETT)
- RTM-compatible activation on an interrupt (CALM X'66' or M.CONN)

In the first case, the CPU scheduler can link the run request to an existing DQE. See Section 2.7.3. In the last two cases, the task remains in the preactivation state queue until the timer expires or the interrupt fires. At that point, such tasks are linked to the appropriate ready-to-run queue as described previously.

Phase 2 -- Activation

In this phase, the resource manager operates for the new task, and runs at the new task's specified priority. It reads in the Resource Requirements Summary (RRS) from the load module file, merges them with static assignments, and validates the results. Resources are allocated. The task's DQE can be linked and unlinked to various state queues as it moves through stages of device and memory allocation.

If any parameters, assignments, or other task resource requirements specified in the load module or by Job Control or TSM assignments are invalid, the resource manager aborts the task during this phase and the task exits as described in Section 2.11.

When the new task has allocated all resources required for execution, it is loaded into memory, relocated, and the resource manager transfers control to the task at its specified transfer address.

There are two exceptions to the control transfer at the end of phase 2. The first is a task that has been initiated by the OPCOM ESTABLISH directive. This task is linked into the suspended state queue (SUSP) instead of going into execution. The purpose is to provide a capability within the MPX-32 structure equivalent to the capability in RTM (Real-Time Monitor) for activating a task that resides permanently in memory (a resident task). In MPX-32, resident means locked in memory. When an activating request occurs for a task that has been established (a timer expires, an interrupt fires, or the task is resumed), the task is fully ready to execute and is brought into execution with just a context switch. If the task has been cataloged as resident, no inswap is required.

The second exception is a task that has been activated with the MPX-32 Debugger attached (TSM or job control DEBUG task name directive). Instead of transferring control to the task, the resource manager first loads and then transfers control to the debugger.

2.3.2 Task Service Area (TSA)

The Task Service Area (TSA) is a section of memory associated with each active task. 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 task's logical address space size and the amount of space reserved for I/O activity.

As depicted in Figure 2-4, the number of blocking buffers, File Assignment Table (FAT) entries, and the File Pointer Table (FPT) entries is variable among tasks. For all tasks, a fixed number of buffers, FAT, and FPT entries are reserved for MPX-32 use; for example, they are present in every TSA.

The pushdown stack area in the TSA provides reentrancy in calls to system modules. At each call to a system module entry point, the stack pointer (T.REGP) is incremented to the next 32-word pushdown level where the contents of the general purpose registers and Program Status Doubleword (PSD) are saved. Within this 32-word level, 22 words are available for scratchpad storage by the module entry point being called. T.REGP is decremented to the previous pushdown level upon return to the entry point caller. 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.

2.4 Central Processing Unit (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.

2.4.1 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 to 54. Time-distribution tasks operate in the priority range 55 to 64.

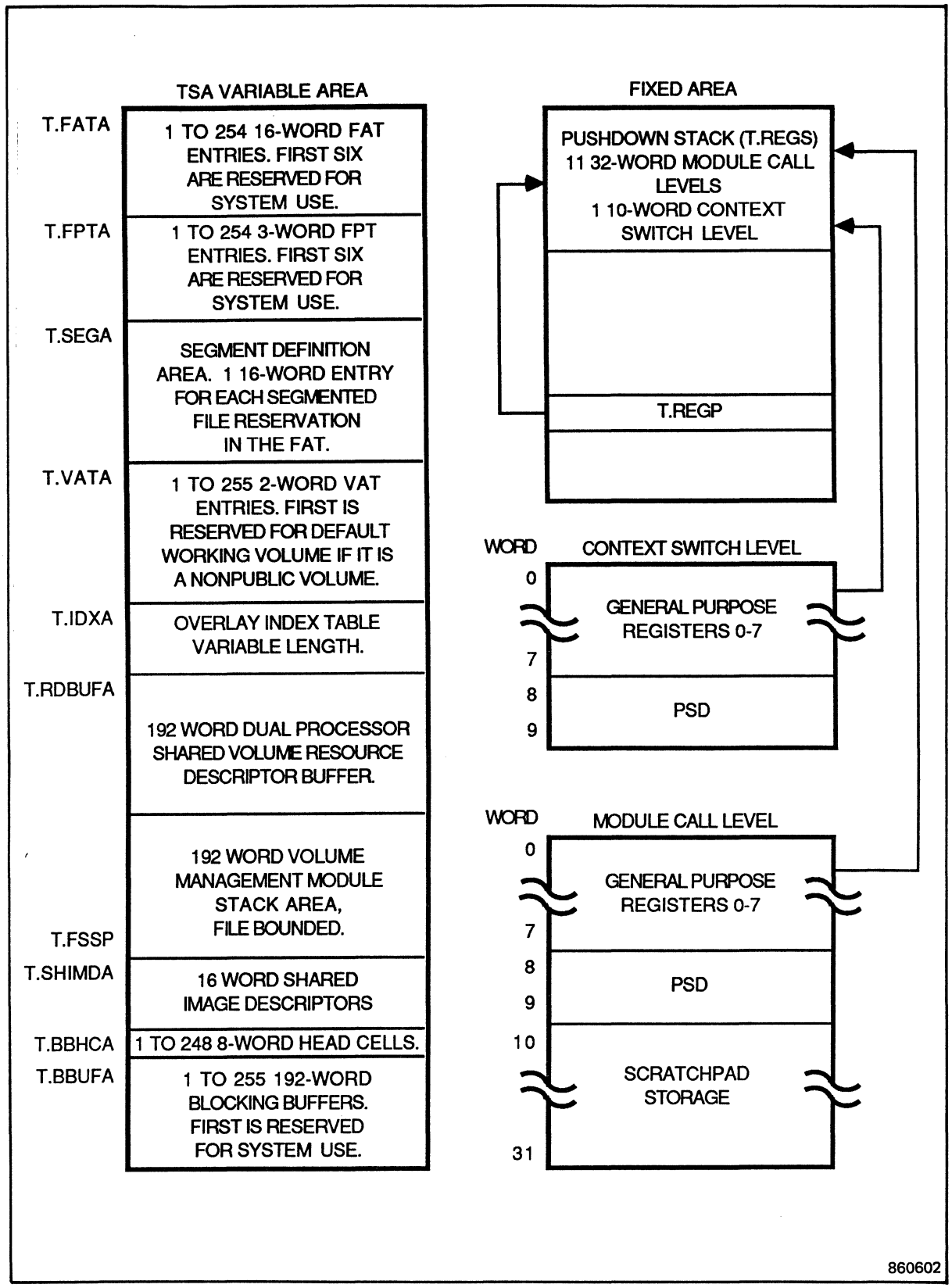


Figure 2-4. Task Service Area (TSA) Structure

2.4.2 Real-Time Priority Levels (1 to 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.

2.4.3 Time-Distribution Priority Levels (55 to 64)

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

2.4.3.1 Priority Migration

The specified execution priority of a time-distribution task is used as the task's base execution priority. Each time-distribution task's current execution priority is determined by the base priority level as adjusted by any situational priority increment. The current execution priority is further adjusted by increasing the priority (by one level) whenever execution is 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.

2.4.3.2 Situational Priority Increments

Time-distribution tasks are given situational priority increments in order to increase 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
Preempted by real-time task	Level 55

2.4.3.3 Time-Quantum Controls

The MPX-32 system allows for the specification of two time-quantum values at SYSGEN. If these values are not specified, system default values are used. The two quantum values are provided for scheduling control of time-distribution tasks. The first quantum value (stage 1) indicates the minimum amount of CPU execution time guaranteed to a task before 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 its 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, for example, suspends, performs wait I/O, etc.

2.4.4 State Chain Management

The current state of a task ready-to-run, waiting for I/O, etc., is reflected by the linkage of the dispatch queue entry (DQE) associated with the task into the appropriate state chain. Linkage is established via string forward and string backward addresses and a state queue index in each DQE. The string forward address for a given DQE points to the closest lower priority DQE and the string backward address points to the closest higher priority DQE in a given state. The index points to a state chain head cell, which contains the link forward/backward addresses from the DQE at the top (highest priority task) of the state chain. At a given time, from any one DQE or from a head cell, an entire state chain queue can be examined by moving either backward or forward through the DQE linkages.

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.

Table 2-2
MPX-32 State Queues

State index	Label	Meaning	
0	FREE	DQE is available (in free list)	
1	PREA	Task activation in progress	
2	CURR	Task is executing	
3	SQRT	Task is ready-to-run (priority level 1-54)	
4	SQ55	Task is ready-to-run (priority level 55)	
5	SQ56	Task is ready-to-run (priority level 56)	
6	SQ57	Task is ready-to-run (priority level 57)	
7	SQ58	Task is ready-to-run (priority level 58)	
8	SQ59	Task is ready-to-run (priority level 59)	Ready-to-run queues
9	SQ60	Task is ready-to-run (priority level 60)	
10	SQ61	Task is ready-to-run (priority level 61)	
11	SQ62	Task is ready-to-run (priority level 62)	
12	SQ63	Task is ready-to-run (priority level 63)	
13	SQ64	Task is ready-to-run (priority level 64)	
14	SWTI	Task is waiting for terminal input	
15	SWIO	Task is waiting for I/O	Operation wait queues
16	SWSM	Task is waiting for message complete	
17	SWSR	Task is waiting for run request complete	
18	SWLO	Task is waiting for low speed output	
19	SUSP	Task is waiting for one of the following: . Timer expiration . Resume request . Message request interrupt	
20	RUNW	Task is waiting for one of the following: . Timer expiration . Run request	Execution wait queues
21	HOLD	Task is waiting for a continue request	

Table 2-2
MPX-32 State Queues (Cont.)

State index	Label		Meaning
22	ANYW	Task is waiting for one of the following: . Timer expiration . No-wait I/O complete . No-wait message complete . No-wait run request complete . Message request interrupt . Break interrupt	Execution wait queues (continued)
23	SWDC	Task is waiting for disc space allocation	Resource wait queues
24	SWDV	Task is waiting for device allocation	
25	N/A	Reserved	
26	MRQ	Task is waiting for memory allocation	
27	SWMP	Task is waiting for memory pool allocation	
28	SWGQ	Task is waiting in general wait queue	
29	CIPU	Current IPU task	IPU state queues*
30	RIPU	Requesting IPU execution	

* See the MPX-32 Technical Manual, Volume I for further details.

2.5 Internal Processing Unit (IPU)

The IPU is a user-transparent device managed by the MPX-32 operating system. The IPU is scheduled as an additional resource to offload the CPU and improve system throughput in a multitasking environment. The IPU can be used to execute task level code and a limited set of system services. The CPU is responsible for laoding the task from disc, allocating memory for the task, building system resident structures identifying the task, rolling the task out to disc (if needed), and executing all I/O instructions.

Scheduling of tasks for IPU execution is controlled by the CPU executive (H.EXEC) working with the IPU executive (H.CPU) for the standard scheduler. An optional scheduler uses a different CPU executive, H.EXEC2, and a different IPU executive, H.CPU2. The optional CPU/IPU scheduling logic is enabled by the SYSGEN DELTA directive. The standard scheduler is more processor oriented whereas the optional scheduler is more priority oriented. The following sections apply to both schedulers; any differences are noted.

2.5.1 Options

Options for the IPU can be specified at catalog or execution time by TSM. The IPU options are:

- IPUBIAS -- When set, tasks that are IPU eligible are run by the IPU. At any point during execution where eligibility ceases, the CPU is trapped and the task is scheduled to execute at its Cataloged priority in the CPU.

- CPUONLY -- When set, the IPU is ignored and the task is executed by the CPU.

If not specified, the default is tasks are executed by the first eligible processor.

Tasks that are compute bound may be biased to the IPU; tasks that are I/O bound may be designated to run only in the CPU.

The CPU/IPU scheduling logic automatically adapts to tasks that alternate between bursts of computing and bursts of I/O for NONBIASED tasks.

2.5.2 Biased Task Prioritization

Standard CPU/IPU Scheduler

If the IPU scheduler finds more than one IPU-biased task waiting for processing, they are placed in a ready-to-run queue (C.RIPU), in priority order among themselves, and are eligible for swapping while waiting.

Optional CPU/IPU Scheduler

Tasks that are IPU biased are not enqueued on the IPU ready to run queue (C.RIPU). These tasks are linked to the ready to run lists SQRT thru SQ64 with other task types. Since the IPU-biased tasks do not enter a wait state, they are less likely candidates for swapping than tasks that are in the wait state.

IPU-biased tasks may have their priority boosted using the SYSGEN DELTA directive. If the DELTA directive is set to zero, scheduling occurs on a priority basis only. If the

DELTA value is greater than zero and less than or equal to 54, the value will be subtracted from the cataloged priority (boosting its priority) at scheduling time. For example, when the DELTA is set to 5, a priority 20 IPU biased task will compete for the IPU at priority 15. Similarly, when an IPU bias task needs the CPU for a system service, the boosted priority (15) will be used to compete for the CPU. The DELTA does not apply when an IPU-biased task executes task level code in the CPU.

2.5.3 Nonbiased Task Prioritization

If the IPU scheduler finds more than one nonbiased task waiting for processing (any task in ready state queues SQRT thru SQ64), they are placed in priority order among themselves and scheduled for processing after execution. The highest priority IPU-eligible task is scheduled in the IPU regardless of its bias or unbiased attribute.

2.5.4 IPU Task Selection and Execution

When the IPU task scheduler has found a task, it checks for IPU eligibility. For a task to be eligible for IPU execution, the following conditions must be present:

- No pending task interrupts
- No system action requests, for example, aborts
- Not CPU biased
- Current execution address outside of resident O.S.

If a task fails any one of these tests, it is ineligible for IPU execution (i.e., ignored) and the task scheduler proceeds to select the next task, if any.

If a task has been selected and is determined eligible for IPU processing, it is linked to the current IPU task queue (C.CIPU), a Start IPU (SIPU) is executed from the CPU, the IPU executive (H.IPU) fields the trap, loads the task's map registers (LPSDCM), and executes the task.

Tasks running with batch priorities (55-64) are not subject to time distribution while being executed in the IPU.

Note: Tasks running with batch priorities (55-64) can not have their priorities boosted via the DELTA value.

2.5.5 CPU Execution of IPU Tasks

Standard CPU/IPU Scheduler

Unbiased tasks require CPU execution for code sequences requiring OS execution. Unbiased tasks are also free to execute task level code in the CPU.

IPU biased tasks will be executed by the CPU for only those code sequences requiring OS execution. When the PSD points back into the task, its CPU execution is terminated immediately and the task is linked to the IPU request queue (C.RIPU). If the IPU is running and this new task has a higher priority (lower number) than the task the IPU is

executing, the executing task is preempted by the new task and replaced by the higher priority task. If the IPU is running and the new task has a lower priority (higher number) than the task currently under execution, the new task is placed in the IPU ready-to-run queue (C.RIPU).

Optional CPU/IPU Scheduler

When the highest and second highest priority tasks are IPU biased, the CPU will execute task level code of the second highest priority task. However, the task's priority will not be boosted by the DELTA value in this case.

2.5.6 Priority versus Biasing

When there is a task in the IPU and it encounters a code sequence requiring CPU execution, the task is linked to a ready to run state chain at its base priority.

Note: For the optional CPU/IPU scheduler, an IPU bias task will be linked to the ready to run state at base priority minus the DELTA value for code sequences requiring CPU execution.

An IPU task that requires some CPU execution **cannot** execute in the CPU if a CPU-only task of the same priority is in the CPU.

An IPU task that requires some CPU execution **can** execute in the CPU if:

- a non-CPU-only task of the same priority is in the CPU.
- the task in the CPU has a lower priority.
- there is no task in the CPU.

2.5.7 IPU Accounting

When the IPU and its interval timer handler are specified during SYSGEN, and the IPU is used for task execution, the following message will be displayed at EOJ and when logging off a terminal:

IPU EXECUTION TIME = xx HOURS- xx MINUTES- xx.xx SECONDS

where xx is a decimal number.

2.5.8 IPU Executable System Services

When the execution address of the task is within the resident OS, the task cannot be scheduled to be executed by the IPU. However, when the execution address of the task is within the task, the task can be executed by the IPU. Once the task has gained entry into the IPU, there is a limited set of system services that the IPU can execute. These are memory reference only system services, since the IPU cannot execute any I/O instructions. The system services that are executable in the IPU are listed in the Nonbase Mode and Base Mode System Services chapters.

2.5.9 IPU Scheduling

Although the IPU is scheduled transparently by the operating system, users can restrict a task to execute only in the CPU or bias a task to execute in the IPU. Tasks designated as

CPU only cannot execute in the IPU. IPU biased tasks and unbiased tasks must meet certain requirements to execute in the IPU:

- the task cannot have any pending or active task interrupts. For example, I/O end action, breaks, etc. A pending interrupt is an interrupt that has been recognized by the operating system but has not yet been dispatched to the task.
- the task cannot have any system action requests, for example, abort, hold, etc.
- the task cannot have context switching inhibited
- the task cannot be temporarily inhibited from IPU execution because it contains instructions not executable by the IPU. For example, CD, BEI, etc.
- the execution address of the task must be outside the resident operating system
- the starting logical address of the task's Task Service Area (TSA) must begin at the end of the resident operating system (i.e., at C.TSAD). All user tasks meet this requirement. SYSGEN and system resident modules and tasks do not meet this requirement.

If a task does not meet the above requirements, it cannot be executed in the IPU and will be scheduled for execution in the CPU.

Scheduling unbiased tasks includes checks for the existence of the following conditions:

- is a task currently executing in the CPU
- is a task currently executing in the CPU that is not eligible to execute in the IPU
- is a task currently executing in the CPU that is eligible to execute in the IPU

If a task is not executing in the CPU, S.EXEC20, the main scheduling routine, attempts to schedule a task for the IPU. If IPU eligible tasks are found, the task with the highest priority is scheduled for IPU execution. S.EXEC20 then schedules the highest priority ready-to-run task for CPU execution. IPU eligible tasks are automatically considered to be CPU eligible. If an eligible task for either processor cannot be found, that processor remains idle.

Note: For the optional CPU/IPU scheduler, tasks that are IPU biased may run on the CPU.

If a task is currently executing in the CPU and it is not eligible for IPU execution, it will continue to be executed by the CPU. S.EXEC20 attempts to schedule a task for the IPU. If an IPU eligible task cannot be found, the IPU remains idle.

If a task is currently executing in the CPU and is IPU eligible, the following factors are also taken into consideration in the order described by S.EXEC20.

- if the current task is IPU biased and the IPU is idle, the task is scheduled for IPU execution.
- if the current task is IPU biased and the IPU is executing a task with a higher priority than the current task, the current task is placed in the IPU request state (RIPU). If the current task has a higher priority than the task executing in the IPU, the task executing in the IPU is removed from execution and the current IPU biased task is scheduled for execution.

Note: For the optional CPU/IPU scheduler, if the current task is IPU biased, but the IPU is executing a higher priority task, the CPU will run the current task.

- if the IPU is idle, S.EXEC20 performs a check to see if another task is requesting CPU execution. If no other task is found, the current task remains in execution in the CPU. If another task is found, the current CPU task is moved to the IPU for execution. The highest priority task of the other tasks found is scheduled for CPU execution.

  Note: For the optional CPU/IPU scheduler, if the IPU is idle and the current task is IPU eligible, the task is scheduled for the IPU.

- if the IPU is busy, S.EXEC20 performs a check to see if another task is requesting CPU execution. If no other task is found, the current task remains in execution in the CPU. If a nonreal time task is found, the current task remains in the CPU. If a real-time task is found, the priority of the current task executing in the CPU is compared with the priority of the current task executing in the IPU. If the CPU task has a higher priority, the task in the IPU is replaced by the task in the CPU. Otherwise, the current task remains in execution in the CPU.

These additional factors which are considered in IPU scheduling allow for a more predictable operation and eliminate unnecessary scheduling overhead. Unless the user can be assured of benefits through the use of IPU biasing or CPU only restrictions, it is recommended that tasks be run unbiased, thereby allowing the MPX-32 executive to make the decision on IPU usage.

2.6 MPX-32 Task Interrupt Scheduling

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

2.6.1 Task Interrupt Levels

Individual tasks operating in the MPX-32 environment may be organized to take advantage of task unique software interrupt levels. Each task in the MPX-32 system can have six levels of software interrupt, sometimes referred to as pseudo-interrupts:

Level priority	Description
0	Reserved for operating system use
1	Debug
2	Break
3	End action
4	Message
5	Normal execution - run request

2.6.1.1 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 debugger interrupt level is used by the system to process tasks running in debug mode (either MPXDB or SYMDB). The end action interrupt level is used for system postprocessing of no-wait I/O, message, or run requests. It is also used for executing end action routines specified by the user task. The normal execution level is used for run-request processing and general base level task execution.

2.6.1.2 Scheduling

Task interrupt processing is gated by the MPX-32 CPU scheduler during system service processing. If a task interrupt request occurs while the task is executing in a system service, the scheduler defers the interrupt until the service returns to the user task execution area. If service calls are nested, the scheduler defers the task interrupt until the last service executes and returns to the user task execution area. The user can defer task interrupts through calls to synchronize task interrupts (M.SYNCH) or disable message interrupts (M.DSMI).

2.6.1.3 System Service Calls from Task Interrupt Levels

A task can utilize the complete set of system services from any task interrupt level. It is prohibited, however, from making a wait-for-any call (M.ANYW, M.EAWAIT) from task interrupt levels.

2.6.1.4 Task Interrupt Context Storage

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

2.6.1.5 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 system service call. When the level active status is reset, any queued request is processed.

2.6.2 User Break Interrupt Receivers (M.BRK, M.BRKXIT)

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

2.7 Intertask Communication

MPX-32 provides both message request and run request send/receive processing. Run request services allow a task to queue an execution request with optional parameter passing 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. Message and run-request services use the software interrupt scheduling structure described in Section 2.6.

2.7.1 User End-action Receivers (M.XMEA, M.XREA, M.XIEA)

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

End-action Type	End-action Exit Service
I/O	SVC 1,X'2C'
Send Message	M.XMEA
Send Run Request	M.XREA

2.7.2 User Message Receivers (M.RCVR, M.GMSGP, M.XMSGR)

A task can enable the message interrupt level by calling the M.RCVR system service to establish a message interrupt receiver address. The level becomes active as the result of a message send request specifying this task as the destination task.

When the message level is active, normal execution processing is inhibited. The task's receiver may optionally call a service M.GMSGP to store the message in a user receiver buffer. After appropriate processing, the message interrupt level may be reset by calling the M.XMSGR system service to exit from the message interrupt receiver.

2.7.3 User Run Receivers (M.GRUNP, M.XRUNR)

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 by the M.SRUNR service, the task load module name may be used to identify the task to be executed. If a task of that load module name is currently active and single-copied, the run request is queued from its existing DQE. If the specified task is not active, or if the task is not a single-copied task, it is activated and the run request is then linked to the new DQE. A new copy will be activated for each run request sent to a multicopied task by load module name or pathname vector. If the multicopied task is waiting for a run request, for example, in the RUNW state chain, the task number must be specified.

The task receiving the run request may optionally call a service M.GRUNP to store the run parameters in a user receiver buffer. After appropriate processing, the run-receiver task can exit by calling the M.XRUNR system service. Any queued run requests are then processed.

When a task which is in run receiver mode enters its abort receiver, the run request has already been terminated and the task issuing the run request has already received status or call back depending on the options used. A new copy of the task is activated to satisfy any queued run requests.

2.7.4 Receiving Task Services

2.7.4.1 Establishing Message Receivers (M.RCVR)

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 system service M.RCVR, and providing the receiver routine address as an argument with the call.

2.7.4.2 Establishing Run Receivers

Any valid task can be a run receiver. Although a set of special run-receiver services are provided, in the most simple case, they 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 is queued until the task exits. If a task of that load module name is currently active, but is not a single-copied task, the load module is activated (multicopied) to process this request. When a single-copied task exits, any queued run requests are executed. If a run request is issued for a task that is not currently active, the task is activated automatically.

2.7.4.3 Execution of Message Receiver Programs

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

2.7.4.4 Execution of Run Receiver Programs

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

2.7.4.5 Obtaining Message Parameters (M.GMSGP)

When the message receiver is entered, register one 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.

2.7.4.6 Obtaining the Run Request Parameters (M.GRUNP)

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.

2.7.4.7 Exiting the Message Receiver (M.XMSGR)

When processing of the message is complete, the message interrupt level must be exited by calling the M.XMSGR service. When M.XMSGR is called, the address of a two-word Receiver Exit Block (RXB) must be provided. The RXB contains the address of the return parameter buffer, and the number of bytes (if any) to be returned to the sending task. The RXB 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 is examined for any additional messages to process. If none exists, a return to the base level interrupted context is performed.

2.7.4.8 Exiting the Run Receiver Task (M.EXIT, M.XRUNR)

When run-request processing is complete, the task can use either the standard exit call (M.EXIT), or the special run-receiver exit service (M.XRUNR).

If the standard exit service (M.EXIT) exits the run-receiver task, no user status or parameters are returned. Only completion status is posted in the scheduler status word of the Parameter Send Block (PSB) in the sending task. After completion processing for the run request is accomplished, the run receiver queue for this task is examined, and any queued run request causes the task to be re-executed. If the run receiver queue for this task is empty, a standard exit is performed.

If the special exit (M.XRUNR) exits the run-receiver task, the address of a two-word Receiver Exit Block (RXB) must be provided as an argument with the call. The RXB contains the address of the return parameter buffer, and the number of bytes (if any) to be returned to the sending task. The RXB also contains a return status byte to be stored in the 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 is put into a wait-state, waiting for the receipt of new run requests. Execution of the task does not resume until such a request is received. If the terminate exit option is used, any queued run requests are processed. If the run receiver is empty, however, a standard exit is performed.

2.7.4.9 Waiting for the Next Request (M.SUSP, M.ANYW, M.EAWAIT)

In addition to the wait options described under the heading "Exiting the Run Receiver Task", a task can use M.SUSP, M.ANYW, or M.EAWAIT services. When operating at the base execution level, a task that has established a message receiver can use the M.SUSP service call to enter a wait-state until the next message is received.

A task may also make use of the special 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 is ended only upon receipt of a message interrupt, timer expiration, or resume, the M.ANYW wait-state is ended upon receipt of any message, end action, or break software interrupt.

M.EAWAIT is similar to M.ANYW except that if no requests are outstanding, an immediate return is made to the caller.

2.7.5 Sending Task Services

2.7.5.1 Message Send Service (M.SMSGR)

A task can send a message to another active task, providing the destination task has established a message receiver. The sending task must identify the destination task by task number. When the send message service (M.SMSGR) is called, the doubleword bounded address of a Parameter Send Block (PSB) must be provided as an argument. The PSB specifies the message to be sent, whether or not any parameters are to be returned, and the address of a user end-action routine. User status can be returned by the destination task. The operating system also returns completion status in the PSB. No-wait and no-call-back control options are also provided. An unprivileged user is limited to five no-wait messages.

2.7.5.2 Send Run-request Service (M.SRUNR)

A task can send a run request to any active or inactive task, identifying the task by load module name. When the run request service (M.SRUNR) is called, the doubleword bounded address of a Parameter Send Block (PSB) must be provided as an argument. The PSB format allows for the specification of the run request parameters to be sent, any parameters to be returned, scheduler and user status, as well as the address of a user end action routine. No-wait and no-call-back control options are also provided. An unprivileged user is limited to five no-wait run requests.

2.7.5.3 Waiting for Message Completion

A message can be sent in either the wait or no-wait mode. If the wait mode is used, execution of the sending task is deferred until processing of the message by the destination task is complete. If the no-wait mode is used, execution of the sending task continues 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 can issue a series of no-wait mode messages followed by a call to the M.ANYW or M.EAWAIT system wait service. This allows a task to wait for the completion of any no-wait messages previously sent. The completion of such a message causes resumption at the point after the M.ANYW or M.EAWAIT call.

2.7.5.4 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.

2.7.5.5 Message End-action Processing (M.XMEA)

User specified end-action routines associated with no-wait 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.

2.7.5.6 Run-Request End-action Processing (M.XREA)

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.

2.7.6 Parameter Blocks

Parameters for run requests and messages are passed by parameter blocks established within the user task. The parameter blocks are described in this section.

2.7.6.1 Parameter Send Block (PSB)

The Parameter Send Block (PSB) describes a send request issued from one task to another. The same PSB format is used for both message and run requests. The address of the PSB (doubleword bounded) must be presented as an argument when either the M.SMSGR or M.SRUNR services are invoked.

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

Please note that a task number, not a load module name, must be used if sending a message request or if sending a run request to a multicopied task which is waiting for a run request.

Word

	0 7	8 15	16 23	24 31
0	Load module name (or task number if message or run request to multicopied task). See Note 1.			
1	Load module name or pathname vector or RID vector if activation (or zero if message or run request to multicopied task). See Note 2.			
2	Priority (PSB.PRI). See Note 3.	Reserved.	Number of bytes to be sent (PSB.SQUA). See Note 4.	
3	Reserved	Send buffer address (PSB.SBA). See Note 5.		
4	Return parameter buffer length (bytes) (PSB.RPBL). See Note 6.		Number of bytes actually returned (PSB.ACRP). See Note 7.	
5	Reserved	Return parameter buffer address (PSB.RBA). See Note 8.		
6	Reserved	No-wait request end-action address (PSB.EAA). See Note 9.		
7	Completion status (PSB.CST). See Note 10.	Processing start status (PSB.IST). See Note 11.	User Status (PSB.UST). See Note 12.	Options (PSB.OPT). See Note 13.

Notes:

1. Word 0, bits 0-31: For send message: Task number of the task to receive the message.

   For run request: Zero if using pathname vector or RID vector in word 1, else task number (word 1 must be zero), else characters one to four of the name of the load module to receive the run request.

2. Word 1, bits 0-31: For send message: Zero.
For run request: Zero if using task number in word 0, else pathname vector or rid vector (word 0 must be zero), else characters 5-8 of the load module to receive the run request.

3. Word 2, bits 0-7: Priority (PSB.PRI) - contains the priority of the send request (1-64). If the value of this field is zero, the priority used defaults to the execution priority of the sending task. This field is examined if the sending task is privileged.

4. Word 2, bits 16-31: Number of bytes to be sent (PSB.SQUA) - specifies the number of bytes to be passed (0 to 768) with the message or run request.

5. Word 3, bits 8-31: Send buffer address (PSB.SBA) - contains the word address of the buffer containing the parameters to be sent.

6. Word 4, bits 0-15: Return parameter buffer length (PSB.RPBL) contains the maximum number of bytes (0-768) that may be accepted as returned parameters.

7. Word 4, bits 16-31: Number of bytes actually returned (PSB.ACRP) is set by the send message or run request service upon completion of the request.

8. Word 5, bits 8-31: Return parameter buffer address PSB.RBA) contains the word address of the buffer into which any returned parameters are stored.

9. Word 6, bits 8-31: No-wait request end-action address (PSB.EAA) contains the address of a user routine to be executed at an interrupt level upon completion of the request.

10. Word 7, bits 0-7: Completion status (PSB.CST) is a bit encoded field that contains completion status information posted by the operating system as follows:

Bit	Meaning When Set
0	Operation in progress (busy)
1	Destination task was aborted before completion of processing for this request
2	Destination task was deleted before completion of processing for this request
3	Return parameters truncated (attempted return exceeds return parameter buffer length)
4	Send parameters truncated (attempted send exceeds destination task receiver buffer length)
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)
6-7	Reserved

11. Word 7, bits 8-15: Processing start (initial) status (PSB.IST) is a value encoded field that contains initial status information posted by the operating system as follows:

Code	Definition
0	Normal initial status
1	Message request task number invalid
2	Run request load module name not found in directory
3	Reserved
4	File associated with run request load module name does not have a valid load module format
5	Dispatch Queue Entry (DQE) space is unavailable for activation of the load module specified by a run request
6	An I/O error was encountered while reading the directory to obtain the file definition of the load module specified in a run request
7	An I/O error was encountered while reading the file containing the load module specified in a run request
8	Memory unavailable
9	Invalid task number for run request to multicopied load module in RUNW state
10	Invalid priority specification. Note: An unprivileged task may not specify a priority which is higher than its own execution priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end action routine address
14	Memory pool unavailable
15	Destination task receiver queue is full

12. Word 7, bits 16-23: User Status (PSB.UST) - As defined by sending and receiving tasks.

13. Word 7, bits 24-31: Options (PSB.OPT) 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.
25	Do not postcompletion 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.

2.7.6.2 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.

Word

	0 7	8 15	16 23	24 31
0	Status (PRB.ST). See Note 1.	Parameter receiver buffer address (PRB.RBA). See Note 2.		
1	Receiver buffer length (Bytes) (PRB.RBL). See Note 3.		Number of bytes actually received (PRB.ARQ). See Note 4.	
2	Owner name of sending task (Word 1) (PRB.OWN). See Note 5.			
3	Owner name of sending task (Word 2) (PRB.OWN). See Note 5.			
4	Task number of sending task (PRB.TSKN)			

Notes:

1. Status (PRS.ST) contains the status-value encoded status byte:

Code	Definition
0	Normal status
1	Invalid PRB address (PRB.ER01)
2	Invalid receiver buffer address or size detected during parameter validation (PRB.RBAE)
3	No active send request (PRB.NSRE)
4	Receiver buffer length exceeded (PRB.RBLE)

2. Parameter Receiver Buffer Address (PRB.RBA) contains the word address of the buffer where the sent parameters are stored.

3. Receiver buffer length (PRB.RBL) contains the length of the receiver buffer (0 to 768 bytes).

4. Number of bytes actually received (PRB.ARQ) is set by the operating system and is clamped to a maximum equal to the receiver buffer length.

5. Owner name of sending task (PRB.OWN) is a doubleword that is set by the operating system to contain the owner name of the task that issued the parameter send request.

6. Task number of sending task (PRB.TSKN) is set by the operating syste- to contain the task activation sequence number of the task that issued the parameter send request.

2.7.6.3 Receiver Exit Block (RXB)

The Receiver Exit Block (RXB) controls 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.

Word

	0 7	8 15	16 23	24 31
0	Return status (RXB.ST). See Note 1.	Return parameter buffer address (RXB.RBA). See Note 2.		
1	Options (RXB.OPT). See Note 3.	Reserved	Number of bytes to be returned (RXB.RQ). See Note 4.	

Notes:

1. Return status (RXB.ST) 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.

2. Return parameter buffer address (RXB.RBA) contains the word address of the buffer containing the parameters which are to be returned to the task which issued the send request.

3. Options (RXB.OPT) 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

4. Number of bytes to be returned (RXB.PQ) contains the number of bytes (0 to 768) of information to be returned to the sending task.

2.7.7 User Abort Receivers (M.SUAR)

User abort receivers execute at the normal task execution base level. The user task can establish an abort receiver by calling the M.SUAR service.

If an abort condition is encountered during task operation, control is transferred to the task's abort receiver. Before entry, any active software interrupt level is reset, all outstanding operations or resource waits are completed, and all no-wait requests are processed. End-action routines associated with no-wait requests that complete whi'e the abort is outstandinq are not executed. Status bits reflecting this are posted in the appropriate FCBs. Any files opened or resources allocated at the time the abort condition is encountered remain opened and/or allocated when the abort receiver is executed.

The TSA stack is clean. The context at the time the abort condition is encountered is stored in T.CONTXT. When the abort receiver is entered, register five reflects task interrupt status when the abort condition was encountered.

Bit	Meaning if Set
0-18	N/A
19	User break interrupt active
20	End-action interrupt active
21	Message interrupt active
22-31	N/A

The standard exit service described in Section 2.11 exits from a task's abort receiver. If another abort condition is encountered while a task is executing an abort receiver, the task is deleted.

A privileged task can reestablish its abort receiver through the M.SUAR service. An unprivileged task is not allowed to reestablish its abort receiver after an abort condition has been encountered. An attempt to do so results in a task delete.

2.7.8 Task Interrupt Services Summary

Table 2-3 summarizes the services described in this section including required parameter blocks. For a detailed description of the parameter blocks for run and message requests, see Section 2.7.

2.7.9 Arithmetic Exception Handling

MPX-32 maintains a trap handler with the capability to do special handling of arithmetic exceptions generated by a task. Provided that the arithmetic exception trap is enabled, any task can test for the occurrence of an exception via the T.EXCP flag in the T.BIT1 field of the TSA. For certain instructions, the destination register values are modified as a result of an arithmetic exception. The H.IP0F Register Fixup table (Table 2-4) shows how the different instruction types are modified. This capability exists for both base and nonbase mode tasks.

The arithmetic exception trap is enabled by the setting of the arithmetic exception bit (bit 7) of the task's PSD. By default, this bit is set when a task is activated. Instructions are provided in the base and nonbase mode instruction sets to manipulate this bit (see the EAE and DAE instructions). When the trap is disabled, only the condition code results are available to indicate that the exception has occurred, and the nature of the exception type.

Table 2-3
Task Interrupt Operation/Services Summary

	Task Interrupt Priority	Level 5		Level 4		Level 3		Level 2
	Task Interrupt Functions	Run Requests	Abort Requests	Message Requests	End-action Run	End-action Message	End-action I/O	Break Requests
	Issue Request	M.SRUNR	M.BORT OPCOM Abort	M.SMSGR OPCOM Send	MPX	MPX	MPX	Hardware Break OPCOM Break M.INT
Sending Task Functions	Send Block	PSB	N/A	PSB	N/A	N/A	N/A	N/A
	Wait for Completion (wait)	PSB	N/A	PSB	N/A	N/A	N/A	N/A
	Establish End Action Receiver	PSB	N/A	PSB	N/A	N/A	N/A	N/A
	Wait for Completion (no-wait)	M.ANYW	N/A	M.ANYW	N/A	N/A	N/A	N/A
	Call-Back Information	PSB	N/A	PSB	N/A	N/A	N/A	N/A
Receiving Task Functions	Establish Receiver	N/A	M.SUAR	M.RCVR	PSB	PSB	FCB	M.BRK
	Get Parameters	M.GRUNP	N/A	M.GMSGP	PSB	PSB	R1 Points to FCB	N/A
	Receive Block	PRB	N/A	PRB	PSB	PSB	FCB	TY's UDT or Contents of T.BREAK
	Exit Receiver	M.EXIT M.XRUNR	M.EXIT	M.XMSGR	M.XREA	M.XMEA	SVC 1,X'2C'	M.BRKXIT
	Exit Block	RXB	N/A	RXB	N/A	N/A	N/A	N/A
	Wait for Next Request	RXB (if M.XRUNR)	N/A	M.SUSP M.ANYW	N/A	N/A	N/A	M.ANYW
	Disable Interrupt Level	N/A	N/A	M.DSMI	N/A	N/A	N/A	N/A
	Enable Interrupt Level	N/A	N/A	M.ENMI	N/A	N/A	N/A	N/A

Base mode provides the capability of further arithmetic exception handling within a task. By establishing an exception handler address within the task, the user provides the operating system an entry point to the task upon occurrence of an arithmetic exception.

Table 2-4
H.IP0F Register Fixup

Instruction Type	Exception Type	Destination Register Results
Floating Point Arithmetic (ADRFW, ADRFD, SURFW, SURFD DVRFW, DVRFD MPRFW, MPRFD, ADFW, ADFD SUFW, SUFD DIVFW, DIVFD MPFW, MPFD)	Exponent Under Flow- Positive or Negative Fraction	0
	Exponent Overflow - Positive Fraction	Largest positive number (7F...F)
	Exponent Overflow - Negative Fraction	Largest negative number (80...1)
	Division by Zero	Largest positive number (7F...F)
Fixed Point Arithmetic (ADMB, ADMH, ADMW, ADMD, ADR, ADRM, ARMW, ARMD, ADI, SUMB, SUMH, SUMW, SUMD, SUR, SURM, SUI, DVMB, DVMH, DVMW, DVR, DVI, RND)	Any	No change (See specific CPU Reference Manual)
ABM, ABR FIXW, FIXD LNW, LND SLA, SLAD TRN, TRNM	Any	No change (See specific CPU Reference Manual)

The occurrence of an arithmetic exception with traps enabled causes the following events to occur:

1. The CPU generates a trap and transfers control to the arithmetic exception trap handler (H.IP0F).

2. The trap handler sets the T.EXCP flag in the TSA and determines what type of instruction caused the exception. If the exception was caused by one of the floating point arithmetic instructions, then the trap handler modifies the destination register values as described in Table 2-4.

3. If the exception occurred during a base mode task which has an exception handler established, and was caused by one of the floating point arithmetic instructions that causes register results to be changed, an argument list is constructed, and control is passed to the handler within the task via the CALL* instruction. When the task's exception handling routine is complete, control may be transferred back to the trap handler by using the RETURN* instruction.

4. The exception trap handler restores all original register and condition code values, as well as the value of the task's current PSD, and allows the CPU to transfer control back to the task which caused the exception. Task execution will resume at the instruction following the trapped instruction.

Condition code values generated as a result of an arithmetic exception are defined in each of the CONCEPT/32 reference manuals.

*This refers to Call/Return and argument passing standards established for FORTRAN 77/X32.

2.7.9.1 Exception Handler Establishment

The M_SETEXA (Set Exception Handler) system service establishes an exception handler for base mode tasks. This service accepts as input either the address of the new handler to be established or zero if the handler is to be ignored by the system. It provides as output the previous handler address. This allows a procedure to establish a handler and reset the previous handler when it no longer needs to handle exception conditions.

2.7.9.2 Changing a Return Address from an Exception Handler

The M_SETERA (Set Exception Return Address) system service can be called from an established exception handler to change the return address. This service accepts either the destination address where control is transferred upon exit from the handler or zero if the destination address remains unchanged (for example, execution is continued from the point of the trap).

2.7.9.3 Exception Handler Input Arguments

When an arithmetic exception handler exists within a task, it is entered from the exception trap handler via the CALL* instruction. The use of this instruction implies an argument list address in base mode register 3. This list is a FORTRAN standard list containing three arguments as follows:

- The program counter (PC) of the point of the exception
- An array containing data specific to arithmetic exceptions
- An exception type and status indicator

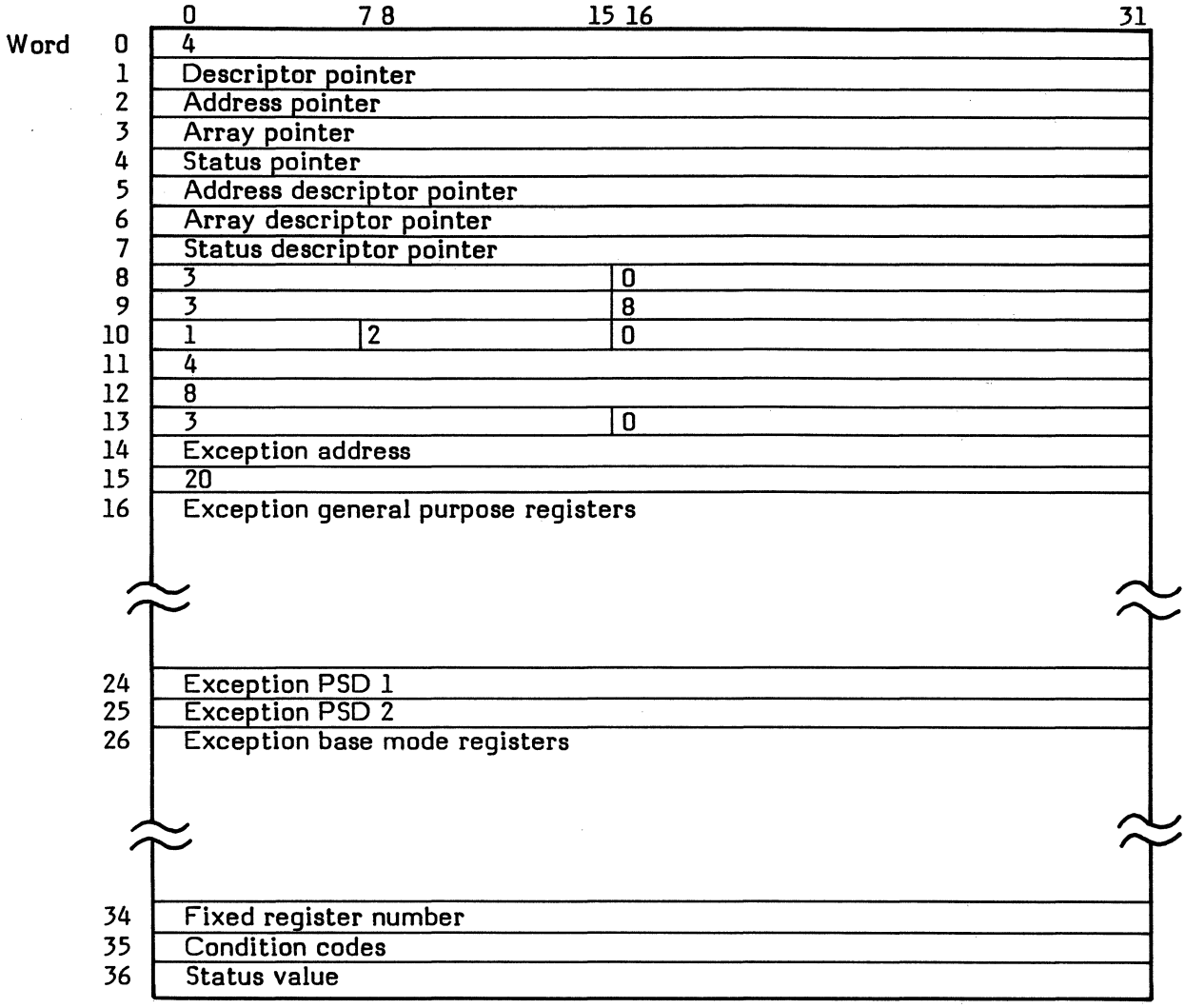
The arguments are passed according to the FORTRAN standard for argument list construction. Assembly language programmers should take care in extracting desired information.

The program counter (PC) points to the instruction causing the exception. This may be either a left or right halfword, or a fullword instruction. The C-bits in the Program Status Doubleword (PSD) must be interpreted to determine the type of instruction.

The array contains information relevant to the arithmetic exception, for example, register contents at the time of the exception, the exception PSD, the register number to which the arithmetic modification was applied by the system arithmetic exception trap handler, and the condition codes at the time of exception.

The status value contains information used by the FORTRAN run-time routines. It includes the severity, system group, functional group, and type of exception.

The data structure of the argument list passed to the arithmetic exception handler of a base mode task is shown below and individual words are described immediately following.



Word	Description
0	This is the number of words of pointer information that follow. In this example, the first word is a pointer to the descriptor list pointers for each argument in the list and the remaining three words are pointers to the arguments themselves.
1	This is the address of the descriptor vector. The vector contains one entry for each argument in the list, in this case there are three. Each entry points to the information which describes the data type and size of each argument.
2	This is the address of the word which contains the address of the instruction causing the arithmetic exception (word 14). Care must be taken if this parameter is used as the instruction may be a halfword or a fullword instruction.

3 This is the address of an array of information collected when the exception occurred (words 15-35). The first word of the array contains the number of words in the array. This is FORTRAN standard. The following 21 words contain the eight general purpose registers, the PSD, the eight base mode registers, the register which was modified by the trap handler, and the condition codes at the time of the trap.

4 This is the address of a status word supplied by the handler (word 36).

5 Contains a pointer to the argument descriptor (word 8) for the first argument value. The argument value itself is contained in word 14.

6 Contains a pointer to the argument descriptor (words 9-12) for the second argument. The argument value itself is contained in words 15 through 35.

7 Contains a pointer to the argument descriptor (word 13) for the third argument. The argument value itself is contained in word 36.

8 Contains the FORTRAN data descriptor for the first argument. The '3' in the left halfword indicates this argument is a word length integer data item. The '0' in the right halfword indicates there is no additional descriptive information about this data item.

9-12 Four words containing descriptive information for the second argument, the exception array. The '3' in the first halfword indicates this argument is a word length integer data item. The '2' in the right halfword indicates 2 more pieces of descriptive information follow. The '1' in the first byte of word 10 indicates that data is the size in bytes of one element of the argument. The '2' in the second byte indicates that data is the size in bytes of the entire argument. The '4' in word 11 indicates each element of the array is 4 bytes in length. The '80' in word 12 is the total byte length of the array (20 elements, 4 bytes each).

13 Contains the FORTRAN data descriptor for the third argument. The '3' in the left halfword indicates this argument is a word length integer data item. The '0' in the right halfword indicates there is no additional descriptive information about this data item.

14 Contains the address of the instruction causing the exception. It may be the address of a fullword instruction, a left halfword instruction, or a right halfword instruction. The C-bits of the exception PSD must be interpreted to determine the type of instruction. The PSD C-bits are interpreted as follows:

Bit 30	Bit 31	Definition
0	0	Fullword instruction
0	1	Right halfword instruction
1	0	Left halfword instruction
1	1	Invalid

15 The first word of the exception array. FORTRAN uses this word to contain the number of entries in the array for subscript validation.

16-23 Eight words containing the contents of the 8 general purpose registers at the time of the exception. The destination register of the instruction causing the exception has had the modified value inserted.

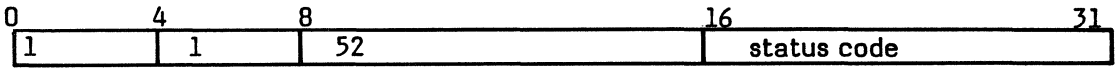
24-25 Two words containing the PSD at the time of the exception. (Points to either 2 or 4 bytes past the instruction which caused the exception, see word 14).

26-33 Eight words containing the contents of the 8 base mode registers at the time of the exception.

34 Contains the register number which was modified as a result of the arithmetic exception processing.

35 Contains the 4 bit condition code value, extracted and right justified, which was contained in the exception PSD.

36 Contains status information generated by the arithmetic exception trap handler in the following format:



bits 0-3 Severity - value 1 = warning
4-7 System group - value 1 = O/S Support Library
8-15 Functional Group - value 52 = Arithmetic exception
16-31 Status code, contains:

1 - Exponent underflow, positive fraction
2 - Exponent overflow, positive fraction
3 - Exponent underflow, negative fraction
4 - Exponent overflow, negative fraction
5 - Divide by zero
6 - Fixed point exception if DQE.AF is set

2.7.9.4 Exception Handler Restrictions

Exception handlers execute with the following restrictions:

- Arithmetic exceptions encountered during execution of the user's arithmetic exception handler are processed by the system arithmetic exception trap handler, but do not cause the user's handler to be reentered. Using the modified registers, execution is continued from the point of the trap or at the address specified to the set return address service.

- A user's arithmetic exception handler can be established only by base mode tasks.

2.7.9.5 Related Arithmetic Exception Information

The M.TSTE (Arithmetic Exception Inquiry) system service accesses the arithmetic exception flag, T.EXCP, in the T.BIT1 field of the task's TSA. The results of this service

indicate whether an exception has occurred, and reset the T.EXCP flag. The output of the service consists of condition code results. Once the T.EXCP flag is set, it is not reset until this service is used or the task terminates. This is a nonbase mode system service.

2.8 CPU Dispatch Queue Area

The CPU Dispatch Queue is a variable length table built at SYSGEN and contains a maximum of 255 Dispatch Queue Entries (DQE's). 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 memory-resident information necessary to describe the task to the system.

For example, the task sequence number, owner name, load module name, TSA address, priority, and current state chain pointers are kept in the DQE, as are abort codes, message and run receiver queue addresses, etc.

Additional (swappable) information is maintained in the Task Service Area (TSA). While a task is active, its DQE is linked to one of the various ready-to-run or wait state chains provided by the CPU scheduler to describe the task's current status. When a task exits, its DQE is again linked to the free list.

2.9 I/O Scheduling

I/O scheduling is designed to provide efficient service to I/O bound tasks while keeping the CPU busy with compute-bound tasks. This allows the fullest possible utilization of both the CPU and I/O devices.

A task that has been waiting for I/O to complete (SWTI or SWIO) is changed to an executable state at a priority slightly higher than a similar compute-bound task when the I/O completes as described in Section 2.4.3.2. At that time, the CPU scheduler interrupts the execution of the compute-bound task so that the I/O-bound task can execute. The I/O-bound user requires minimal CPU time before initiating another I/O request and returning to the SWTI or SWIO state. The compute-bound task then resumes execution. The CPU scheduler automatically adapts to tasks that alternate between bursts of computing and bursts of I/O.

2.10 Swap Scheduling

The swap scheduler task (J.SWAPR) processes entries in the Memory Request Queue (MRQ). It provides memory allocation and swap scheduling as appropriate to service individual requests for memory.

2.10.1 Structure

The swap scheduler is a memory resident, privileged task that is not operating system resident. It executes at the priority of the highest priority task in the Memory Request Queue. The swap scheduler always occupies the first DQE. If a task requires memory, the swap scheduler maps the task's TSA on top of its address space.

The swap scheduler remains suspended until resumed by the executive in response to a swap scheduler event.

2.10.2 Entry Conditions

The swap scheduler task is normally suspended. It is relinked to the ready-to-run queue by the executive in response to a system service calling the executive to report a swap scheduler event. There are four basic types of swap scheduler events.

2.10.2.1 Dynamic Expansion of Address Space (M.GE/M.GD, M.MEMB)

Whenever there is insufficient memory to satisfy a dynamic memory request for a task, the task is linked into the memory request queue and the swap scheduler is resumed.

Memory is allocated in 2KW increments on a CONCEPT/32. These increments are called map blocks.

2.10.2.2 Deallocation of Memory (M.FE/M.FD, M.MEMFRE)

Whenever a task deallocates some or all of its memory, and the memory request queue is not empty, the swap scheduler is resumed. Those tasks in the MRQ are then allocated some or all of the deallocated memory.

2.10.2.3 Request for Inswap

Whenever a currently outswapped task becomes eligible for execution, it is linked into the memory request queue. The swap scheduler is resumed to process the inswap request.

2.10.2.4 Change in Task Status

Whenever a task which had been previously ineligible for swapping becomes eligible, the swap scheduler is resumed. Such status changes include the completion of an unbuffered I/O operation, the release of a lock-in-memory flag, or the expiration of a stage one time quantum.

2.10.3 Exit Conditions

The swap scheduler signals the executive when it cannot process any more outstanding requests, or when the memory request queue is empty. The swap scheduler is unlinked from the ready-to-run queue and placed in a special wait-for-memory-event state.

2.10.4 Selection of Inswap and Outswap Candidates

The swap scheduler attempts to allocate the memory required for the highest priority task in the memory request queue. If there is insufficient free memory, the swap scheduler examines the state queues on a priority basis, searching for the memory class and number of map blocks required.

When the first outswap candidate that satisfies any current memory request is determined, the task is outswapped.

When sufficient memory is available, the inswap process is initiated. The swap scheduler processes entries in the memory request queue until the queue is empty or until an available outswap candidate for a task requesting memory cannot be found.

Both outswap and inswap are serial processes which go to completion before the memory request queue is reexamined. Dynamic memory requests are similar to inswap requests, except that there is no associated disc file to read. Some tasks in the memory request queue can be queued for both inswap and a dynamic request. There must be sufficient memory for the inswap and dynamic requests before the inswap process can proceed.

2.10.4.1 Outswap Process

The outswap process is initiated when inswap or dynamic memory is requested. The outswap priority order can be specified by the system administrator. See the System Administrator Chapter in MPX-32 Reference Manual, Volume III. The default outswap priority is:

Default
Outswap Priority Order

HOLD SUSP RUNW SWDV SWDC SWSR SWSM SWLO SWFI MRQ ANYW SWGQ SWTI SWIO SWMP	Wait queues
SQ64 • • SQ55 RIPU SQRT	Ready-to-run queues

The TSA of the outswap candidate is mapped into the swap scheduler and used to construct a new address space which represents the swappable map blocks in a logically contiguous format. Then, the swap space is allocated and opened by the swap scheduler. For F-class swap devices, a single write request is given to IOCS. Command and data chains are built in the handler to perform the specified transfer.

Once output is complete, the memory is deallocated, and the memory request queue is reexamined to find the highest priority candidate for inswap.

2.10.4.2 Inswap Process

When sufficient memory is available, the swap scheduler allocates the memory required by the highest priority task in the memory request queue. If the request is simply a dynamic one, the swap scheduler adjusts the TSA of the requestor to reflect the newly allocated memory, and informs the CPU scheduler.

If the request requires an inswap, the swap scheduler reads the swapped image into the newly allocated memory. For F-class swap devices, a single read request is given to IOCS. Command and data chains are built in the handler to perform the specified transfer.

Once inswap is complete, the swap scheduler cleans its map and reexamines the memory request queue for the next inswap candidate.

2.11 Task Termination Sequencing

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

2.11.1 Nonbase Mode Exit Task (M.EXIT)

The exit task service is called by a task that wishes to terminate its execution in a normal fashion. The sequence of system processing on task exit is described in Table 2-5.

2.11.2 Abort Task (M.BORT)

The nonbase mode abort task service is called by a task that wants 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 nonpresent 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 owner name or by the OPCOM ABORT directive. The sequence of system processing on task abort is described in Table 2-5.

2.11.3 Delete Task (M.DELTSK)

The delete task service is called by the system for a task that encounters a second abort condition when processing an initial abort request. This service may also be initiated asynchronously by another task of the same owner name or by the OPCOM KILL directive. The sequence of system processing on task delete is reflected in Table 2-5.

2.11.4 Base Mode Exit Task (M_EXIT)

The base mode task entry structure allows base mode tasks to exit in a uniform manner. Exit sequences require a zero or an ASCII status code to be placed in R0. Any entry into a subroutine may be exited by the execution of a RETURN instruction.

A base mode assembler task must exit any end action receiver by a RETURN instruction. If the exit from end action requires a Receiver Exit Block (RXB), the RXB address must be in register one.

Table 2-5
Task Termination Sequencing
(EXIT, ABORT, and DELETE)

Task Has	System Action		
	Task Exit	Task Abort	Task Delete
Outstanding I/O	Defers processing until any outstanding I/O is complete.	Same as exit, except inhibits execution of user no-wait I/O end-action routines. Task abort is reflected in appropriate FCB(s).	Terminates all outstanding I/O.
Outstanding Messages in Receiver Queue	Unlinks all outstanding messages; posts complete with abnormal status.	Same as exit.	Same as exit.
Outstanding No-wait Run Requests with Call Back	Defers processing until the destination task completes. The exiting task is placed in the ANYW state until the destination task has completed.	Defers abort processing until all requests are complete. Task abort status is reflected in run request parameter block.	Call backs are ignored.
Run Requests in Receiver Queue	Terminates the current run request and posts appropriate status in run request parameter block. Then activates a new copy of the task for next run request in queue, if any.	Same as exit.	Same as exit.

Table 2-5
Task Termination Sequencing
(EXIT, ABORT, and DELETE)
(Continued)

Task Has	System Action		
	Task Exit	Task Abort	Task Delete
Task Abort Receiver	Not processed.	Transfers control to task after other steps taken above. Files are not closed; devices and memory are not deallocated. (Remaining abort processing by system is discontinued.)	Not processed.
Files Open	Closes all open files automatically. Preserves integrity of both user and system files.	Same as exit.	Does not close files automatically; preserves integrity of system critical files. User files are left as is.
Devices/ Memory Allocated	Deallocated automatically.	Same as exit.	Same as exit.

2.12 Task-Synchronized Access to Common Resources

MPX-32 provides the structure for tasks to voluntarily synchronize access to a common resource such as a disc file, a shareable device, a common data area, a shared/included procedure area, or any other physical resource.

The capability provided by MPX-32 is a general resourcemark mechanism. Each task using a marked resource must:

- use the M.RSML and M.RSMU (Resourcemark Lock/Unlock) services to synchronize access to a resourcemark with other tasks
- make the association of a particular resourcemark with an actual resource

What MPX-32 provides is a table of resourcemarks that are currently in use, a mechanism for queuing tasks for each mark, and automatic unlock on a resourcemark when a task terminates (aborts, exits, or is deleted), if the task has not unlocked the resourcemark on its own.

A resourcemark is a decimal value from 1 to 64. Values 1-32 are for internal use, values 33-64 are available for customer use. The default size of 64 can be increased by using the SYSGEN RMTSIZE directive. However, MPX-32 does not enforce resource access restrictions on privileged tasks. The system does not associate a particular resource with a particular resourcemark. Thus, if several tasks use synchronization service calls to gain access to a resourcemark and another task does not, the outside task will gain the resource just as if no restrictions were active for it.

Tasks synchronizing use of resources are responsible for using resourcemarks that uniquely identify resources across the system. MPX-32 ensures only that a specified mark is within the legal numeric range.

To use resource marking, each cooperating task:

- uses M.RSML to lock the resourcemark
- performs the access which requires synchronization
- uses M.RSMU to unlock the resourcemark and release the highest priority task queued for the resourcemark

The task has several options available if the resourcemark is locked when it issues the M.RSML call. As specified in the call, it can:

- obtain an immediate denial return and go on
- wait until it can gain ownership of the lock
- wait until it can gain ownership or until a specified number of timer units have expired, whichever occurs first

If a single task uses more than one resourcemark, and it is synchronizing access to more than one resource, the user must exercise care to avoid deadlock situations; for example, Task A is in wait for a lock owned by Task B while Task B is in turn waiting for a lock owned by Task A.

A task using more than one resourcemark can avoid deadlocks by unlocking all locked resourcemarks if it cannot succeed in locking any one of them. The task then waits for the critical unlock to occur before reattempting locks on all the other resourcemarks in the set.

Sample Resourcemark Use by a Task

```
        PROGRAM T4
        M.REQS
        LIST NGLIST
T4      EQU         $
        M.RSML 33,0             LOCK RSM,INDEF.WAIT,NORM SWAP
        M.WRIT ABC              WRITE TO CRITICAL FILE
        M.RSMU 33               UNLOCK RSM
        M.EXIT
ABC     DATAW G'ABC'            LFC SETUP
        GEN 12/B 80,20/B(SBUF)
        REZ 6W
SBUF    RES 80B
        END         T4
```

2.13 MPX-32 Faults/Traps and Miscellaneous Interrupts

MPX-32 provides interrupt and trap processors for all standard interrupts and traps. A list of these interrupts with associated information is shown in Table 2-5.

Processing for trap levels 03, 04, 05, and 09 is dependent on the location of the instruction causing the trap. A system crash (M.KILL; not OPCOM KILL) results if the offending instruction is issued from a location within the MPX-32 system area. If the instruction is issued from a location within a task area, the task is aborted.

When a system crash occurs as a result of a trap handler entry, the CPU halts with the registers containing the following information:

Register	Contents
0	PSD Word 0 (when trap generated)
1	PSD Word 1 (when trap generated)
2	Real address of instruction causing trap
3	Instruction causing trap
4	CPU status word (from trap handler)
5	Crash code: MP01=X'4D503031' (Memory Parity Error - H.IP02) NM01=X'4E4D3031' (Nonpresent Memory - H.IP03) UI01=X'55493031' (Undefined Instruction - H.IP04) PV01=X'50563031' (Privilege Violation - H.IP05) MC01=X'4D433031' (Machine Check - H.IP07) SC01=X'53433031' (System Check - H.IP08) MF01=X'4D463031' (Map Fault - H.IP09) CP01=X'42543031' (Cache Parity - H.IP10) 32/67 and 32/87 AD01=X'41443031' (Address Specification - H.IP0C) HT01=X'48543031' (Privilege Halt Trap - H.IPHT)
6	Real address of register save block
7	C'TRAP'=X'54524150'

Table 2-6
MPX Faults/Traps and Miscellaneous Interrupts

Relative Priority	Logical Priority	Dedicated TVL		Description	System Action: PSD in OS Area/ in Task Area	Abort Code
		CPU	IPU			
00	00	80	20	Power Fail Trap	Halt/Halt	N/A
01	01	84	24	Power On Trap	Halt/Halt	AU01
02/12	12	88	28	Memory Parity Trap	M.KILL/Abort	MP01
03/24	24	8C	2C	Nonpresent Memory Trap	M.KILL/Abort Task	NM01
04/25	25	90	30	Undefined Instruction Trap	M.KILL/Abort Task	UI01
05/26	26	94	34	Privileged Violation Trap	M.KILL/Abort Task	PV01
06		98	38	SVC Trap	Process SVC/ Process SVC	See Note 1
07		9C	3C	Machine Check Trap	M.KILL/M.KILL	MC01
08		A0	40	System Check Trap	M.KILL/M.KILL	SC01
09		A4	44	Map Fault Trap	M.KILL/Abort Task	MF01
0A			48	Undefined IPU Instruction Trap	M.KILL/Abort Task	UI01
0C	0E	B0	50	Address Specification Trap	M.KILL/Abort Task	AD02
0D	0D	B4	54	Console Attention Trap	Process Int./ Process Int.	N/A
0E	27	B8	58	CPU Halt Trap	M.KILL/Abort Task	HT01
0F/29	29	BC	5C	Arithmetic Exception Trap	Not Enabled/ Record in TSA	N/A
10		C0	60	Cache Memory Parity Error Trap	M.KILL/Abort Task	CP01
18	18	160	N/A	Real Time Clock Interrupt	Process Int./ Process Int.	N/A

Note 1: SVC Abort Codes

SV01	Unprivileged task using M.CALL
SV02	Invalid SVC number
SV03	Unprivileged task using privileged service
SV04	Invalid SVC type
SV05	Unprivileged task using M.RTRN
SV07	Invalid SVC for base register operation

CHAPTER 3

RESOURCE MANAGEMENT OVERVIEW

3.1 General Resource Management

A generalized resource management scheme means all resource operations work in a standard and predictable manner on every resource. A resource is any source of aid or support existing which is external to a task's body and is required by the task in order for that task to perform its function.

3.2 Support for Resource Types

The operating system recognizes two major types of resources, physical and logical. A physical resource is any physical hardware supported by the operating system. A logical resource is any entity existing only because of a mechanism provided by software. Most often, the mechanism is merely a named and predictable structure of data imposed on a physical medium.

3.2.1 Physical Resources

The primary physical resources supported by the operating system are: the central processing unit (CPU), computer memory (main storage), and input/output devices. Generally in the support of physical resources, all resource functions are supported in the same manner. Definition (creation) of physical resources is accomplished by the system generation (SYSGEN) process. Deletion of physical resources is also accomplished by the SYSGEN process. Resource attachment, access, and detachment are a subset of the functions allowed for logical resources. Physical resource inquiry is more primitive and resource dependent than logical resource inquiries. The attributes assigned to the management of physical resources are more resource dependent than the attributes associated with logical resources. Attributes of physical resources can only be modified by the SYSGEN process.

3.2.2 Logical Resources

The primary logical resources supported by the operating system are disc volumes, directories, files, and memory partitions. Generally, in the support of logical resources all resource functions are supported in the same manner. Definition (creation) of logical resources is accomplished by utilities or system services. Deletion of logical resources is also accomplished by utilities or system services. Resource attachment, access, detachment, inquiry, and attribute modifications are provided for logical resources and are implemented by system services.

3.3 Support for Resource Functions

In order to support all resources in a similar manner, all functions required to manage resources must be provided and those resources must operate in a similar manner. The following is a list of the functions provided:

- Resource creation - The ability to define a resource to the operating system
- Resource deletion - The ability to remove the definition of a resource from the operating system
- Resource attachment - The ability to connect to a resource for the purpose of using the resource
- Resource access - The process of using the resource -- generally, the process of transferring data to or from a resource
- Resource detachment - The ability to disconnect a resource so it can be used by others
- Resource inquiry - The process of inquiring about a resource in order to determine specific information about it
- Resource attribute modification - The process of modifying the attributes of a resource in order to change its operational characteristics

3.3.1 Resource Creation

Before any resource can be used by a task in the operating system, the required resource must be defined to the operating system. Utilities are provided for defining resources. In most cases, the resources can be defined by directives or services issued to the operating system. When a resource is created, all of its attributes are defined.

3.3.2 Resource Deletion

The resource definition must be deleted so the resource can no longer be used. This is accomplished by the use of a utility program. Usually, resources can be removed by directives or services provided by the operating system.

3.3.3 Resource Attachment

Resource attachment is the process of securing a resource for use by a task. Commands and/or service requests are issued to the operating system in order to attach a resource. When a resource is to be attached, various parameters are specified indicating how the resource is to be used. The directive or service to attach a resource is unique to each type of resource. For example, volumes are mounted, memory partitions are included, and files/devices are assigned.

There are two types of resource attachment provided by the operating system, static and dynamic.

3.3.3.1 Static Allocation

Static allocation is invoked by declaring a task's resource requirements when the task is cataloged or activated. Static allocation serves several purposes. Most importantly, this form of allocation guarantees that a cataloged load module, when activated, will have all the resources required before it begins execution. Secondly, by declaring all of a task's resources when it is cataloged or activated, the operating system can match sets of resource requirements among all tasks in the activation state and more effectively manage resources. Static allocation enables the operating system to allocate sets of resources, thereby avoiding deadlocks that can occur when a task requires multiple resources and must dynamically allocate each one. Statically - allocated resources may be overridden at task activation.

3.3.3.2 Dynamic Allocation

In some applications, a task does not know what resources it requires until it begins execution. For this reason, dynamic resource allocation is provided. Dynamic allocation is invoked by service requests from an executing task.

3.3.4 Resource Access

Resource access is the process of transferring data to or from a resource, positioning a resource, or otherwise manipulating a resource. Various applications require many levels of access to a resource. The operating system provides the following levels of resource access. The levels are described in order from the most device-dependent to the least device-dependent access levels.

3.3.4.1 Device Level

The operating system provides integration of device-dependent I/O drivers. The user is not required to design and code an MPX-32 I/O device handler. However, a user-supplied I/O driver can be integrated into the operating system by the SYSGEN utility.

3.3.4.2 Execute Channel Program Level

In order to perform device-dependent I/O operations where the operating system queues the I/O requests, starts the I/O requests, and processes the operating system termination functions, the user is allowed to build and execute physical or logical channel programs. These channel programs can only be executed on devices that use extended I/O (XIO) protocol.

3.3.4.3 Logical Device Level

With this level of access, the user is able to transact with a device while using specific physical capabilities offered by the device. Logical device I/O supports applications which require the use of a specific device for a capability provided explicitly by the device.

With this level of access, a user performs I/O requests using a File Control Block (FCB). The data format inhibit option must be set in the FCB to gain access to the device at this level.

3.3.4.4 Logical File Level

With this level of access, a user achieves a degree of device independence. When device access is performed at this level, device-dependent characteristics are masked from the user. This access level supports applications which require the illusion of device commonality and independence.

3.3.4.5 Blocked Level

This is the highest level of device access provided by the operating system. Explicitly intended for use by utility programs requiring the highest degree of device commonality or sameness, blocked I/O is designed to work only with magnetic tape and disc media. In order for all operations to perform identically regardless of which type of media is manipulated, this access level emulates the operation of magnetic tape on both types of media. All operations that are valid for magnetic tape media are provided. The user can issue rewind, write end-of-file, advance file, backspace file, advance record, backspace record, read record, and write record operations.

With this method of operation, the user is constrained to access the media in a sequential manner. Records can be transacted with the media in lengths of 254 bytes or less; longer records are truncated. The operating system automatically performs intermediate record blocking and buffering to or from the media.

For the maximum degree of device independence, the user is advised to use the following subset of allowable operations: rewind, read record, and write record.

Note: Append access is available on disc media but is not allowed on magnetic tape media.

3.3.5 Resource Detachment

Resource detachment allows attached resources to be released and made available for use by other tasks. When resources are detached, other tasks that can be enqueued awaiting the availability of the resource are resumed to contend for attachment to the resource.

Resources are detached explicitly by the appropriate dismount, exclude, or deassign functions. Additionally, any resources a task has attached at the normal or abnormal termination of the task are automatically detached by the system.

Detachment of a resource may cause the system to perform clean-up operations, such as purging partially filled blocking buffers and releasing exclusive locks outstanding on the resource.

3.3.6 Resource Inquiry

This capability is provided so a task can determine the attributes of a resource.

3.3.6.1 Inquiry of Unattached Resources

At times it is necessary for a task to inquire about a resource. Given the name or some other legal identifier, directives and services are provided to return information about resources. The operating system provides this information through a resource descriptor (RD). At a place common to all resource descriptors, the inquirer can determine the type of resource. Once this information is obtained, the inquirer can examine resource type-dependent data in the descriptor for more specific information. Usually, such inquiries are made before the resource is attached.

3.3.6.2 Inquiry of Attached Resources

This type of inquiry is most often used when a resource has been statically attached. It determines the logical or physical attributes of the connection, such as the access modes, access level, physical device address, and parameters.

The user must furnish the logical file code, File Control Block (FCB) address, or allocation index to identify the desired resource.

3.3.7 Resource Attribute Modification

At times it is necessary to modify the protection and other access attributes of a resource. Resource attribute modification, like resource inquiry, deals with operating system data structures. The user of these functions should be familiar with the format of these data structures. Also, it is recommended that user-supplied subroutines act as a common interface to the functions. In this way, the user is less sensitive to changes in system structures.

3.4 Resource Attributes

All logical resources have attributes. The attributes of resources control how the resources are managed and determine who can use them.

The operating system ensures that all logical resources are defined in directories; for example, by providing names. Protection is applied to resources to determine who may use them and how they can use them. Resources can be shared (used by more than one task at the same time). Resources can otherwise be declared as nonshared (used by only one task at a time). Another set of attributes determines how the resource can be accessed; for example, data can be read from but not written to the resource, only certain areas of the resource can be written to, the resource can be deleted, etc.

3.4.1 Protection

The protection provided by the operating system for logical resources is organized so that a resource can be managed from the perspective of three classes of users: the owner of a resource, a member of a specific group of users associated with a resource, and any other arbitrary user of a resource. A resource can be defined and managed as applicable to each class.

Protection is supplied for environments where desired. Since protection can be harmful when it is not administered properly, the user is advised to protect resources only to the appropriate level required. By default, owner and project group privileges are equal and all owners belong to the same project group. This default method of operation allows all users of the system to attach to all resources defined to the system.

When a task attempts to attach a resource, the system determines whether it is the owner, a member of a project group, or an other arbitrary user. On every attempt to attach a resource, the associated owner name is checked first, then the project group name. If the task does not match either of the first two checks, the task is given the access rights associated with an other arbitrary user. Otherwise, the task is given the access rights for the level that was matched.

3.4.1.1 Owner

An owner is the person creating a resource. When the resource is created, the owner establishes all attributes for the resource. The owner can specify which project group and others can access the resource and what their access capabilities are.

3.4.1.2 Project Group

The project group is the name of a group of users allowed to access the resource.

3.4.1.3 Others

The others group defines users of a resource who are not the owner or members of the project group.

3.4.2 Shareable Resources

A resource can be defined as shareable when it is created. Shareable resources can be attached to more than one task at the same time.

When a shareable resource is attached, the requesting task indicates how it wants to use the resource. A shareable resource can be used in three modes: exclusive, explicit, and implicit. The resource can only be attached in one of the three usage modes at any given point in time.

3.4.2.1 Exclusive Use

When a task requires exclusive use of a shared resource, the task can request it when attaching to the resource. A resource attached for exclusive use can only be used by the task that was granted the exclusive attachment. Once a resource has been attached for exclusive use, other tasks requesting attachment can optionally be denied or enqueued until the resource becomes available.

A task may require exclusive access to a resource it is already attached to and may be currently sharing with other tasks. In this case, the task must call the M.LOCK service. The M.LOCK service determines whether the caller is the only task attached to the resource. If so, the caller is immediately given exclusive access. Otherwise, the caller is optionally denied or enqueued until it receives exclusive access.

3.4.2.2 Explicit Use

When multiple tasks require simultaneous attachment to the same shareable resource, the tasks can attach to the resource for explicit use. With explicit use, the attached tasks can use the resource in a way that can destroy recorded data. It is the responsibility of tasks using a resource in this mode to ensure that the integrity of the data recorded on the resource is preserved.

Several mechanisms are provided to allow explicit users to preserve data integrity. A task can gain exclusive access to a resource by calling the M.LOCK service. In this case, the M.LOCK service will enqueue the caller until exclusive access can be granted. A set of tasks can synchronize on access to the resource by calling the M.SYNC and M.UNSYNC services. Synchronization locking does not guarantee exclusive access to a resource; it is merely a semaphore that is locked. The semaphore for the resource is returned when the resource is attached and must be used as an input parameter to the M.SYNC and M.UNSYNC services. Another type of semaphore, the resourcemark, can be used for synchronizing access to a resource or a set of resources. The use of semaphores requires that all tasks attached to a resource in explicit use mode cooperate to preserve the integrity of the resource.

3.4.2.3 Implicit Use

When a task attaches to a shared resource but does not explicitly declare exclusive or shared use, the resource is attached for implicit use. Implicit use is the default usage mode for all attachment to resources. In this mode, the operating system automatically allows the resource to be accessed by multiple tasks attached in compatible access modes. With compatible access modes, the following access combinations are allowed: multiple readers, multiple readers with a single writer, single writer only, and two simultaneous writers.

3.5 Resource Access Attributes

All resources have a set of attributes to determine how the resource can be accessed. This section defines the access attributes for each logical resource and defines the purpose of these attributes.

3.5.1 Access Attributes for Volumes

A volume does not have access attributes. Instead, volume management is determined by the type of volume - system, user, or multiprocessor.

3.5.1.1 System Volume

In order for a volume to be managed as a system volume, it must meet a minimum set of criteria:

- It cannot be mounted as a multiprocessor volume.
- It must be formatted by the Volume Formatter utility. For example, it contains a valid system bootstrap in blocks zero to three of the volume, it contains system image and device type dependent information in the volume descriptor, and it contains a valid system image.
- It must have a system directory created as the first directory on the volume by the Volume Manager utility.
- The system directory must contain the minimum subset of system files to constitute a viable system.

When a volume is mounted as a system volume, it is treated as a public volume. Public volumes can be used by anyone allowed access to the system. Additionally, public volumes are used to acquire temporary files, swap files, and spooled input/output files. Like all public volumes, the system volume cannot be dismounted from a running system.

3.5.1.2 User Volume

Any volume mounted after the system volume is a user volume. User volumes are not required to have bootstraps, system images, or any other requirements associated with system volumes. A user volume must be formatted by the Volume Formatter utility.

A user volume can be mounted as a public volume by the System Administrator (SA). A public volume has the following attributes:

- It can only be mounted by tasks or individuals having the System Administrator (SA) attribute.
- Once mounted, it cannot be dismounted from a running system.
- Once mounted, it can be used by any user allowed to logon the system.
- A mounted public volume does not have to be explicitly mounted by anyone that wants to use it.
- Public volumes are used for storage of temporary files, swap files, and spooled input/output files.

A user volume can be mounted as a nonpublic volume. A nonpublic volume has the following attributes:

- It can be mounted by anyone who logs on the system.
- It must be explicitly mounted by each user that wants to use it.
- It can be dismounted while the operating system is running. A nonpublic volume can be physically dismounted only when the associated use and assign counts indicate the volume is not in use.

3.5.1.3 Multiprocessor Volume

Any user volume can be mounted as a multiprocessor volume. Multiprocessor volumes allow tasks that operate in separate system environments to have concurrent access to any volume resource.

In order for a volume to be mounted as a multiprocessor volume, it must meet the following criteria:

- The disc drive volume is to be mounted on must have been hardware configured as a dual-ported disc drive (only model 8055 disc processors support dual-ported access. If a Cache Disc Accelerator (CDA) is used, it must have been hardware configured as multiported.
- The disc drive must have been identified as a multiported drive in the appropriate SYSGEN DEVICE directive.
- A SYSID parameter must have been specified with the mount request, identifying the software port for the caller's operating environment.
- Cannot be the SYSTEM volume.

3.5.2 Access Attributes for Directories

Directories have attributes determining how they are managed. These attributes can be specified for each user class (owner, project group, and others). The access attributes are defined when a directory is created. Access attributes for directories are described in this section.

3.5.2.1 Read Access

A directory with read access can be attached like a file with read-only access. A user assigning a directory in the read mode must be familiar with the format of directory entries in order to extract meaningful information. Read access allows the contents of a directory to be presented by the LOG service or directive with wild card characters. If read access is not allowed, a directory cannot be logged.

3.5.2.2 Add Entry Access

New directory entries can be added to a directory with add entry access.

3.5.2.3 Delete Entry Access

Directory entries can be deleted from a directory with delete entry access.

3.5.2.4 Delete Directory Access

A directory can be deleted when it does not contain any active directory entries if it has delete directory access. Utilities, directives, and services are provided to delete entries from directories.

3.5.2.5 Traverse Access

A directory can be searched when a pathname is being executed by the system if it has traverse access. For example, a resource can be located in a directory. The contents of a directory cannot be presented with the LOG service or directive if wild card characters are used and the user only has traverse access to the directory.

3.5.3 Access Attributes for Files

Files have attributes determining how they are managed. These attributes can be specified for each class of user; owner, project group, or other arbitrary users. The access attributes are defined when a file is created. This applies for both temporary and permanent files. Access attributes for files are described in this section.

3.5.3.1 Read Access

A file with read access can be attached for read-only access.

3.5.3.2 Write Access

A file with write access can be attached for read/write access. This mode establishes all new data contents for a new or existing file. In this mode, unused extensions to a file are automatically deleted when the file is closed.

3.5.3.3 Modify Access

A file with modify access can be attached for read/write access. This mode modifies the data contents of an existing file. When accessed in this mode, files cannot be automatically extended.

3.5.3.4 Update Access

A file with update access can be attached for read/write access. This mode modifies the data contents of an existing file and appends new data to the file. When accessed in this mode, files can be automatically extended.

3.5.3.5 Append Access

A file with append access can be attached for read/write access. This mode appends new data contents to an existing file. When accessed in this mode, files can be automatically extended.

3.5.3.6 Delete Access

A file with delete access can be deleted if the directory containing the file's directory entry allows delete entry access.

3.5.4 Access Attributes for Memory Partitions

Memory partitions have attributes determining how they are managed. They can be specified for each class of user: owner, project group, or other. The access attributes are defined when a memory partition is created. Access attributes for memory partitions are described in this section.

3.5.4.1 Read Access

A memory partition with read access can be attached (included) for read-only access.

3.5.4.2 Write Access

A memory partition with write access can be attached (included) for read/write access.

3.5.4.3 Delete Access

A memory partition with delete access can be deleted if the directory containing the partition's directory entry allows delete entry access.

3.6 Management Attributes

All logical resources have a set of applicable management attributes regardless of which user class is attached. Management attributes for resources are described in this section.

3.6.1 Extension Attribute

Extension attributes apply only to permanent and temporary files.

3.6.1.1 Manual Extension Attribute

Manual extension attributes apply only to files. They enable a file to be manually extended by the extend service or directive.

3.6.1.2 Automatic Extension Attribute

Automatic extension attributes apply only to files. They enable a file to be extended automatically when data is written to the file. Automatic extension is subject to restrictions inherent to access modes. For example, a file attached in the modify mode cannot be extended.

3.6.2 Contiguity Attribute

Contiguity attributes apply only to extendible files. This attribute informs the operating system that the file should be contiguous. When a file has this attribute and is to be extended, the operating system attempts to allocate the new segment contiguous to the last segment. If the contiguous extension fails, the system attempts to allocate the new segment at any available location on the same volume. The Volume Manager RESTORE directive can be used to attempt to restore a discontiguous file contiguously if it contains the contiguous attribute.

3.6.3 Maximum and Minimum Extension Attributes

The maximum and minimum extension attributes can only be described by a detailed explanation of the extension algorithm. When a file is to be extended, the presence of a dynamically user-supplied value, a minimum increment, and the maximum increment is verified. The following chart shows the results of this verification.

Supplied Value	Minimum Increment	Maximum Increment	Result
No	No	No	The file is not extended
No	No	Yes	Maximum extension attempted
No	Yes	No	Minimum extension attempted
Yes	No	No	Supplied value attempted
No	Yes	Yes	Maximum extension attempted first, minimum attempted second
Yes	No	Yes	Supplied value attempted first, maximum extension attempted second
Yes	Yes	No	Supplied value attempted first, minimum extension attempted second
Yes	Yes	Yes	Supplied value attempted first, maximum extension attempted second, minimum extension attempted third

3.6.4 Maximum File Size Attribute

Maximum file size attributes apply only to files and can be specified when a file is created. If specified, a file will not become larger than the size specified. If a maximum file size is not specified, a file can be extended a maximum of 31 times or until file space cannot be acquired, whichever occurs first.

3.6.5 Shared Attribute

Shared attributes apply to files and directories and can be specified when the resource is created. If specified, the resource can be attached and accessed by more than one task. If shared is not specified, the resource can only be attached and accessed by one task at a time. Memory partitions are always given the shared attribute.

3.6.6 End-of-file Management Attribute

End-of-file management applies only to files and can be specified when a file is created. When a file has this attribute, the system records the highest file relative block number written to on the file and does not allow reading or modifying beyond this point. For files being used in a sequential manner, this attribute is recommended. In cases of random access or multiple writers on a file, this attribute is not desirable or predictable.

When a file does not have this attribute, the end-of-file block is the last block allocated to the file.

3.6.7 Fast Access Attribute

Fast access applies only to files and can be specified when a file is created. This attribute enables files defined on a volume to be attached in one disc access through the file identifier (RID). The RID identifies the volume where the file resides, the block number of the file's Resource Descriptor (RD), and the creation date and time of the file. Files created with the fast access attribute will always retain their original RID as long as they remain defined on the volume. A file explicitly deleted and subsequently recreated is assigned a new RID.

System services using a Resource Create Block (RCB) can create files with the fast access attribute by setting the appropriate bit in the RCB (see Section 5.14.2.6).

When a Volume Manager RESTORE or COPY directive is performed on a file created with the fast access attribute, the file will retain its original RID. See MPX-32 Reference Manual Volume 2, Chapter 3.

3.6.8 Zero Attribute

Zeroing applies only to files and can be specified when a file is created. File space will be prezeroed when a file is created or extended with this attribute.

3.6.9 File Type Attribute

File type applies only to files. This is a two-digit hexadecimal number that can arbitrarily classify resources. File types 00 through 9F are available for customer usage. A0 through FF are reserved for Gould CSD development's usage. Known file type codes are:

42	Toolkit blocked mode
43	Toolkit card mode
46	Toolkit formatted mode
50	Toolkit packed mode
55	Toolkit unblocked mode
B0	Base mode object file
BA	Base mode shared image (or BASIC save file)
BB	Base mode object library file
BC	Base mode macro library file
BE	Base mode load module file
C0	Spooled output file
CA	Cataloged load module
D0	Memory disc save task (J.MDSAVE) file
DB	Symbolic debugger command file
ED	Saved text editor file
EE	Stored text editor file
FE	Text editor work file
FF	SYSGEN generated file

3.6.10 No-save Attribute

No-save applies only to files and can be specified when a file is created. A file with this attribute cannot be saved by the Volume Manager SAVE directive unless the SAVN parameter is Y.

3.7 Operating System Memory Allocation

Unless extended execution space is specified, MPX-32 occupies the lower portions of each task's address space, thereby allowing MPX-32 to run mapped with each task.

MPX-32 maintains lists of available memory for allocation to tasks and for I/O that requires intermediate buffering.

3.7.1 I/O Buffer and I/O Queues

The system memory pool is an area of memory which is contiguous to the resident system and has a size specified at SYSGEN by the POOL directive. The entire memory pool is write-protected from the unprivileged task and is intended for use exclusively by system services. The memory pool is mapped into the address space of each task and is allocated in doublewords. The maximum size of any entry is 192 words. Typical system uses of the memory pool area are:

- I/O queues - Approximately 26 words are allocated when IOCS queues a request and deallocated when post-I/O processing is complete.
- Message or run-request buffers - Up to 192 words are allocated by the M.SMSGR or M.SRUNR services and deallocated when receiver processing is complete.

3.7.2 Blocking Buffers for Blocked I/O

File assignments for permanent files and devices optionally specify that a file is blocked or unblocked. The default is blocked. If blocked, blocking buffers for the files are allocated at load time in the Task Service Area (TSA). The Catalog BUFFERS directive may be used to provide additional blocking buffer space for dynamically allocated, blocked files.

3.7.2.1 Large Buffers for Blocked Files

Large blocking buffers contain two or more 192-word blocks. The total number of buffers should not exceed 247.

When using the TSM $ASSIGN directive with the BBUF parameter, large blocking buffers can be allocated at load time in the Task Service Area (TSA). The BBUF parameter is only valid with a TSM $ASSIGN directive.

User-supplied large blocking buffers in multiples of 192-word blocks can also be created using a 16-word expanded FCB. Byte zero of word 15 contains the number of 192-word blocks within the large blocking buffer. The address of the buffer is placed in bytes one through three. When byte zero does not specify a number of blocks, one blocking buffer is automatically allocated.

3.8 Memory Classes

When a nonbase mode task is cataloged, the user can specify the class of memory required at run time. Memory classes are established at SYSGEN time, and there are no limitations for the positioning or sizes of the classes. The Cataloger ENVIRONMENT directive indicates the class of memory with the following parameters:

Parameter	Result
S	Execution delayed until class S, H, or E is available (default)
H	Execution delayed until class H or E is available
E	Execution delayed until class E is available

In situations where a CONCEPT/32 system does not have memory installed of the class the user task requests, the first available lower class is allocated to that task. If an excessive request is made, the requestor is aborted.

If there is no ENVIRONMENT parameter for memory class, tasks are loaded into any memory class available.

Base mode tasks are loaded into any memory class available.

3.9 Memory Allocation for Tasks

The unit of memory allocation is called a map block and is 2KW on a CONCEPT/32. All user tasks are discontiguously loaded into a whole number of physical map blocks, utilizing the mapping mechanism to create their contiguous logical address space. No partial map blocks are allocated.

This scheme allows user tasks to dynamically expand and contract their address space by using the memory management service calls described in Chapter 6.

The unit of memory protection is called a protection granule and is 512 words. Thus, it is possible to protect a task's TSA even though it is in the same map block as the data section (DSECT or read/write).

3.9.1 Static Memory Allocation

The Cataloger determines the size in protection granules of a cataloged load module. The Cataloger ALLOCATE directive may be used to specify additional bytes of memory. The size of the TSA is determined at activation time and rounded up to a number of protection granules. This value is added to the cataloged requirement to determine task size. Additionally, the TSM $ALLOCATE directive may be used to specify the total task size of the DSECT. The final sum is rounded up to a map block increment.

3.9.1.1 Static vs. Dynamic Shared Memory

Characteristics	Static Partitions	Dynamic Partitions	Shared Image
Logical addresses	Fixed at SYSGEN	Fixed by Volume Manager	Determined CREATE COMMONat link time
Physical addresses	Fixed at SYSGEN	Variable	Variable
Allocation unit	2K words	2K words	2K words
Time of allocation	SYSGEN	Run time by M.INCLUDE	Automatic by activation or M_INCLUDE
Time of deallocation	Never	When allocation count = 0	When allocation count = 0
Inclusion	Automatic by	Run time by M.INCLUDE activation or M.INCLUDE	Automatic by activation or M_INCLUDE
Exclusion	Automatic by exit	Automatic by exit or or M.EXCLUDE	Automatic by M.EXCLUDEexit or M_EXCLUDE
Owner names or task numbers	None	Established by M.INCLUDE	None caller
Swapping	Never swapped	Swappable when user count = 0	Swappable when user count = 0

3.9.1.2 Memory Partition Applications for Nonbase Mode Tasks

Characteristics	Global	Datapool	Extended Common	CSECTs
Cataloger resolves references	Yes	Yes	No	Yes
Compiler resolves references through extended bases	No	No	Yes	N/A
Must be logically below 128KW	Yes	No	No	Yes
Variables are order dependent	Yes	No	Yes	N/A
Static	Yes	Yes	Yes	No
Dynamic	Yes	Yes	Yes	Yes

3.9.2 Dynamic Address Space Expansion/Contraction (M.GE, M.FE, M.GD, M.FD, M.MEMB, M.MEMFRE)

A nonbase mode task can expand and contract both its execution and extended data space through the system services. The M.GE service appends a map block to the user's execution space starting at the end of his DSECT. M.GE can be used more than once to obtain additional map blocks, as long as they are available in the task's logical address space. The M.FE service frees the most recently obtained map block; for example, it works in the opposite order of M.GE. The M.GD service appends a map block to the user's extended indexed data space, starting from 128KW. Like M.GE, it can be used more than once. The M.FD service frees the most recently obtained map block from the extended indexed data space; for example, it works in the opposite order of M.GD. The M.GE and M.GD services do not apply to base mode tasks.

The M.MEMB service provides dynamic memory allocation capabilities for the user. Memory is allocated in byte increments on doubleword boundaries. Allocation starts at the next doubleword boundary following the user's last loaded address (T.END). Any number of bytes can be specified up to the maximum available in the user's logical address space. Each call to the service provides contiguous allocation for the requested amount. Areas allocated by subsequent calls are not contiguous with previously allocated areas. Allocated areas may or may not be in extended memory. The user should operate in extended mode (SEA) when addressing these areas. The M.MEMFRE service frees memory, in random order, obtained from the M.MEMB service. These two services cannot be used with the M.GE, M.GD, M.FE, and M.FD services.

3.9.3 Extended Indexed Data Space for Nonbase Mode Tasks

MPX-32 provides limited support for logical addresses above 128KW for nonbase mode tasks. The following restrictions apply to the use of this address space:

- Instructions cannot be executed in this logical space
- The user must reference this space by index registers. Negative offsets are invalid in the word address field of any instruction as long as the indexed addressing mode is active.
- No data initialization facilities are provided for this logical space
- The user must dynamically request this logical space to be mapped with the memory management system services
- Global and Datapool are not supported in extended data space

3.9.4 Intertask Shared Global Memory and Datapool Memory (M.INCLUDE, M.EXCLUDE)

Intertask shared memory is provided under MPX-32 through Global and Datapool memory partitions and shared images. There are up to one hundred Global regions (GLOBAL00 - GLOBAL99) plus a Datapool partition available to the user.

Global and Datapool partitions can be defined either by SYSGEN or the Volume Manager utility. Shared images are created by the LINKER/X32. See Section 4.12 for shared image information.

Partitions created at SYSGEN are considered permanently allocated and are assigned both physical and logical memory attributes applying to any task that references the partitions. This type of allocation is called static allocation. The static Global and Datapool partitions are defined in integral numbers of protection granules.

Both Global and Datapool partitions are located in an integral number of physical map blocks starting on a map block boundary. Areas are mapped into user space when required. See Sections 3.9.1.1 and 3.9.1.2.

Write protection is available to prevent the user from storing into a common area where he does not have write access.

Alternatively, the user can define dynamic shared memory partitions with the Volume Manager. There are several key distinctions between statically and dynamically allocated common:

- Statically allocated common is fixed in physical memory even when no task is sharing it through its map. Dynamically allocated common is deallocated when its allocation count equals zero.
- Statically allocated common is invoked on a system-wide basis. Dynamically allocated common is based on a subsystem concept, where all tasks issue an M.INCLUDE request. A particular common partition, such as Datapool, can be defined concurrently in several such subsystems. However, each subsystem has a physically unique partition.

- Dynamically allocated common can be excluded from a task by the M.EXCLUDE system service. The user can elect to subsequently include another dynamically allocated common area by M.INCLUDE. Statically allocated partitions are not supported by the Volume Manager.

- All logical references to common, whether statically or dynamically allocated, are resolved by the cataloger. The logical address of a system common partition is fixed when the partition is defined.

- Load modules from one MPX-32 configuration are compatible with another even if Global or Datapool are allocated different physical addresses. The only compatibility requirement is that both systems employ the same logical conventions.

Figure 3-1 illustrates a relatively complex view of the relationship between logical address spaces and statically and dynamically allocated common partitions. The figure also introduces the allocation considerations for shared procedures, which are described in the section which follows.

In brief,

- Tasks A, B, and C reference a static Datapool partition

- Tasks A and B use an M.INCLUDE service for dynamic GLOBAL10

- Tasks A, B, and C use M.INCLUDE for GLOBAL02

- Task D shares no memory or code with other tasks. Map blocks seven and up are available to the task.

- Tasks A and B are shared. They have CSECT mapped at the same location in each logical address space.

3.9.5 Shared Procedures for Nonbase Mode Tasks

MPX-32 supports shared procedures. A catalog parameter specifies a shared procedure that must consist of a CSECT (pure code and data) and a DSECT (any impure data). When a shared procedure is activated for the first time, the system loader reads both the shared procedure section and the impure data section into memory. The shared procedure section is mapped like an M.INCLUDE service request. Every subsequent activation of the shared procedure causes only the impure data section of the shared procedure to be loaded since the procedure section is already in memory, has not been altered, and can be mapped like an M.INCLUDE service request.

Shared procedures, such as shared memory areas, have an allocation count to prevent premature deallocation of the memory they occupy and a user count to control the swapping of these partitions.

Shared procedure space is allocated in the highest available logical address. (See Figure 3-1.)

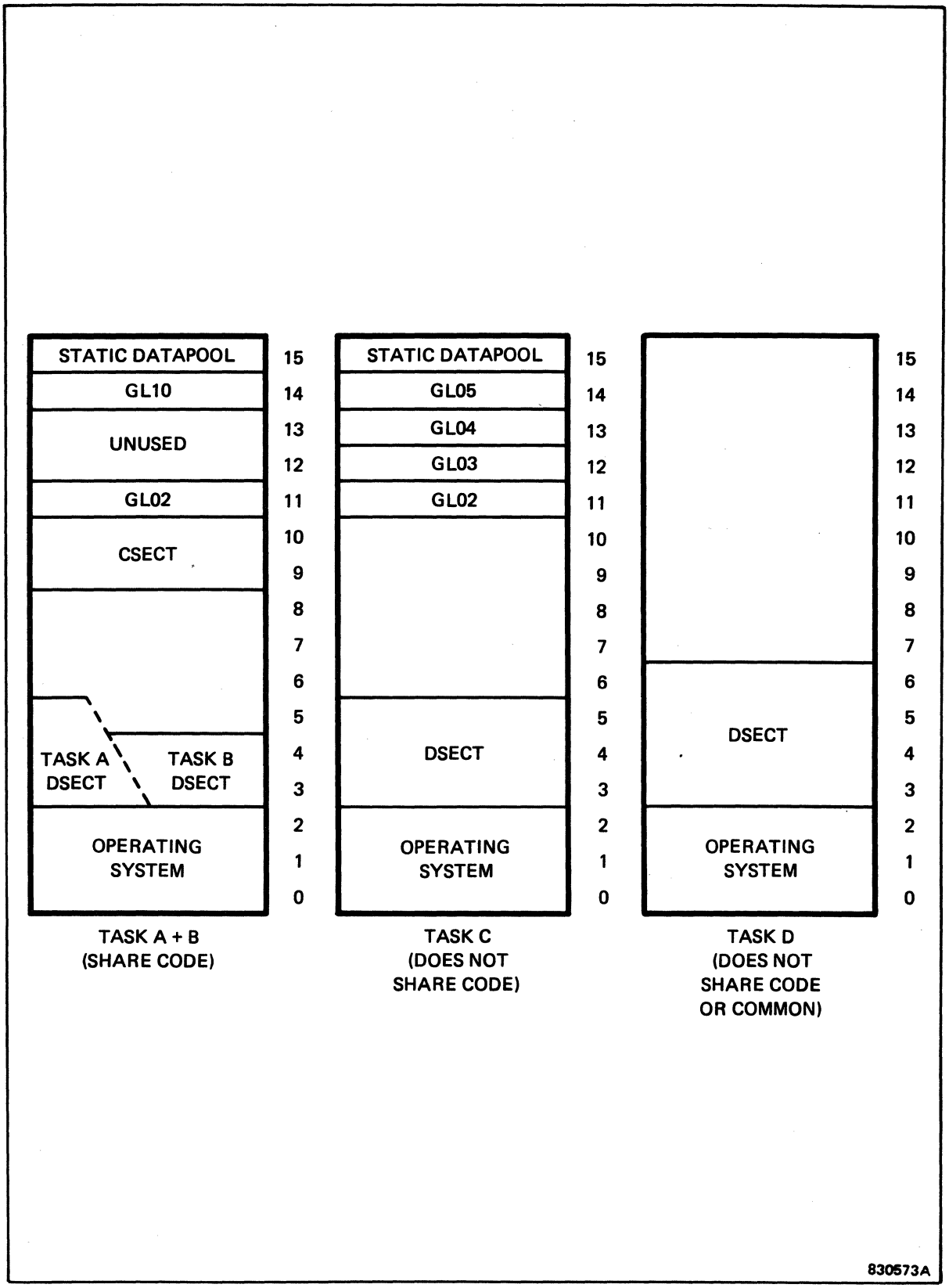


Figure 3-1. Sample Allocation of Common Memory Partitions and Common Code

3.9.6 Multiprocessor Shared Memory

Multiprocessor shared memory is memory that is shared between systems. This portion of memory must be managed by the user. If not properly SYSGENed, it is possible for MPX-32 nonresident tasks to be allocated memory in the shared memory sections. This can allow corruption of the nonresident tasks by the other systems.

It is recommended that multiprocessor shared memory be used as follows:

- For memory that will be shared between systems, create a static memory partition in the multiprocessor shared memory via SYSGEN. The partition can be any size.

- Any remaining memory can be allocated to MPX-32 tasks. This memory should be SYSGENed exclusively for a particular system. This memory can be divided between systems as long as each area can be accessed by only one system. To accomplish this, SYSGEN each area as present in one system and non-present in all other systems.

3.10 Extended MPX-32 (Expanded Execution Space)

Extended MPX-32 is an optional mode of operation that allows a portion of the MPX-32 operating system to be positioned into a task's extended memory. Extended MPX-32 creates a split image that divides the operating system into two sections:

- The nonextended section of the split image is nonbase mode code that is mapped into the lower 128KW of user task space below the TSA.

- The extended section of the split image is translated into base mode code. This allows part of the operating system to be removed from the lower 128KW of the user nonbase task space. The extended section can then be positioned in the extended address area where only data could previously be positioned.

As a result of the split image, additional user execution space is available in the lower 128KW of MPX-32 for nonbase program execution.

Control communication between the two sections is performed by adaptative code that is positioned in the lower memory area just above the communication region. See Figure 3-2. For an illustration of extended MPX-32 program flow control, see Figure 3-3.

Source modules that can operate in the extended mode have been modified. Their object modules have been compressed into OH.32_E, a SYSGEN input file. To function in the extended mode, the code and data sections of these modules are separted by SSECT and DSECT instructions. The DSECT area is data mapped in the lower 128KW of memory, just above the adaptive coding. The DSECT area must be bounded within the first 16KW of physical and logical memory to allow direct reference by the extended section. See Figure 3-2.

For users who have modified existing source modules, or who want to move their own modules to nonextended memory, those modules must be modified and re-assembled. See the Programming Considerations section.

The Macro Assembler can generate base and nonbase object code from the same source files. The instruction mode is set by assigning the PRE file to MPX_EXT. See the MPX-32 Technical Manual, Volume I, Chapter 5.

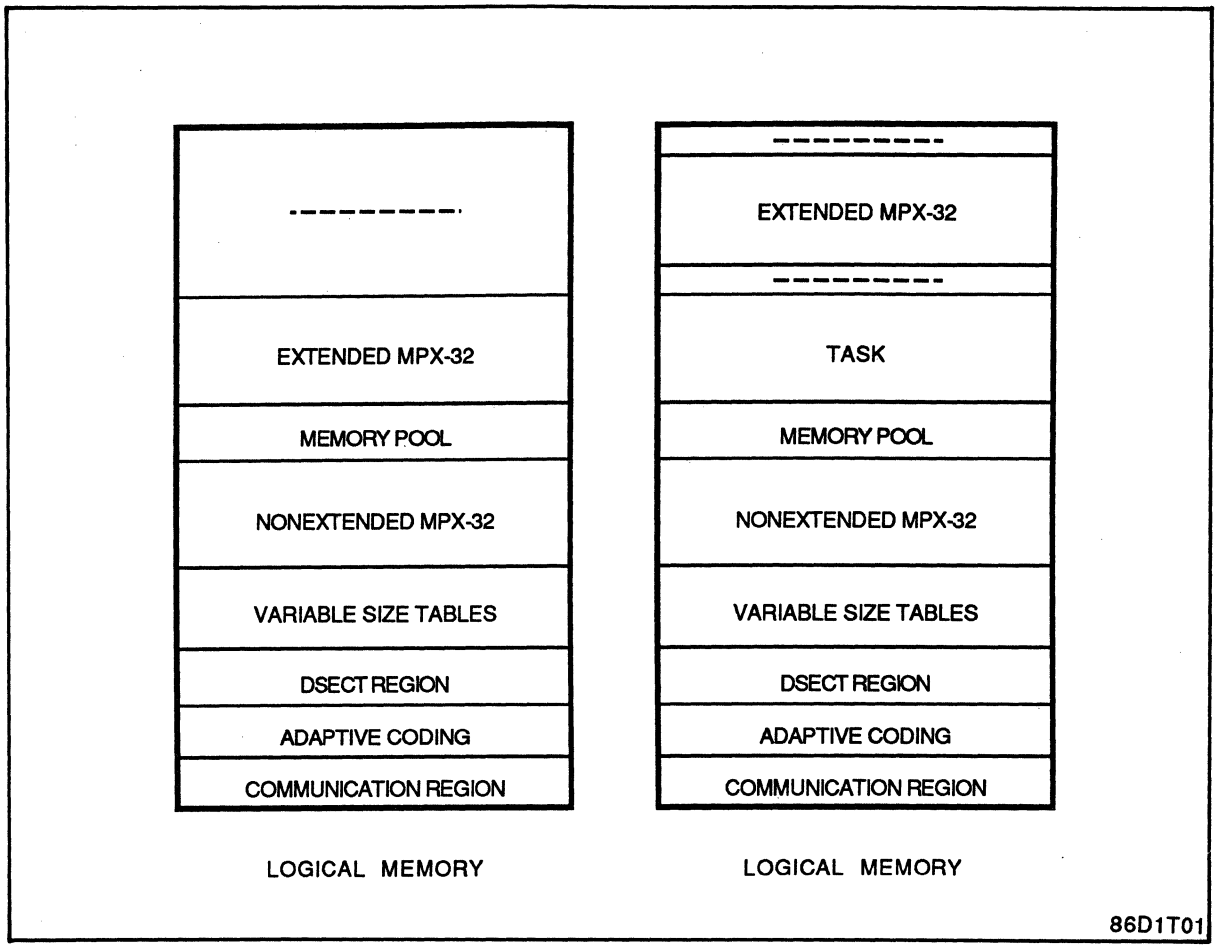


Figure 3-2. Extended MPX-32 Logical and Physical Memory Allocation

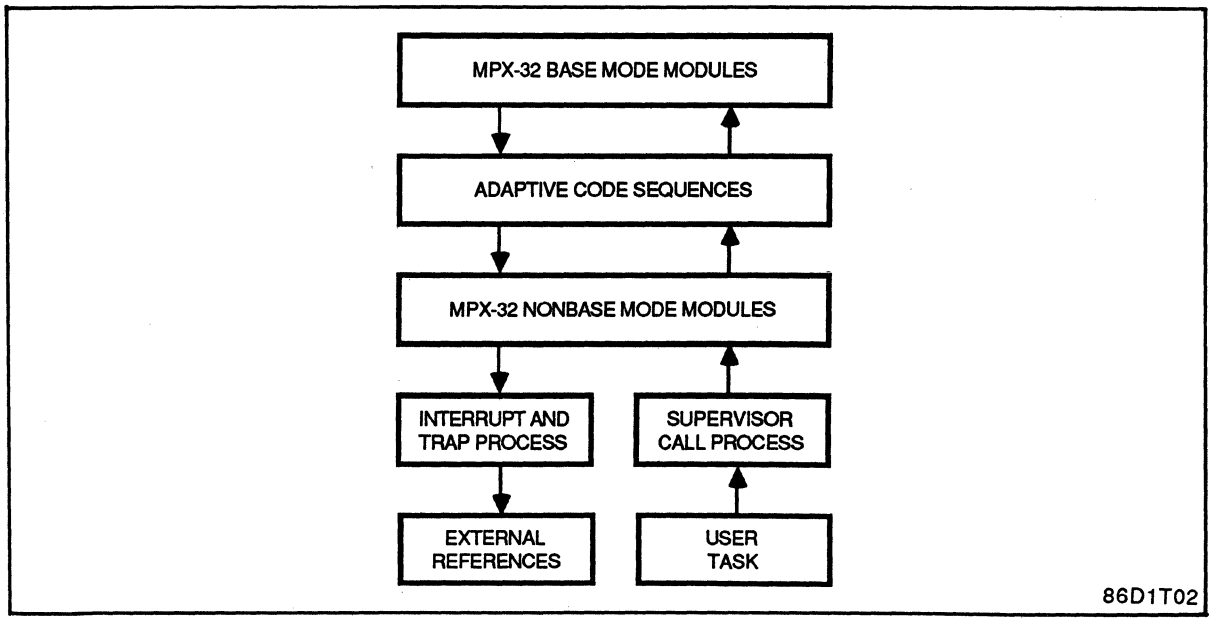


Figure 3-3. Extended MPX-32 Program Flow Control

A minimal performance degradation may occur when switching from nonextended to extended modules. Careful selection of the modules to reside in the extended area can reduce the performance impact in exchange for increased size in the nonextended area. For suggestions on improving performance, see the Extended MPX-32 Performance Considerations section.

WARNING: Because the CONCEPT 32/27 does not support the base mode instruction set, expanded execution space is not available on the CONCEPT 32/27. Attempts to boot expanded execution space on the CONCEPT 32/27 or a CONCEPT 32/87 that does not have base mode instruction capability result in a fatal abort.

3.10.1 Implementing Extended MPX-32

To initialize the extended mode, the following steps must be completed:

1. Any modules that are not in the OH.32_E object library and that meet the programming considerations for extended execution need to be compressed into file OH.32_E.
2. OH.32_E, the compressed extended object modules, must be assigned to logical file code OBR for input into SYSGEN. If OH.32_E is not assigned to OBR, the system cannot create a split image and the extended mode is not available.
3. OH.32 must be assigned to logical file code OBJ.
4. The SYSGEN EXTDMPX directive can be specified. This directive specifies the default starting logical map block number of the extended section. If EXTDMPX is not specified, the default places the extended mode coding in the task's TSA where it does not conflict with the task's use of its logical address space (MINADDR).
5. Run SYSGEN.

Example

```
TSM>ASSIGN DIR TO DIRECTIVES BLOC=Y
TSM>ASSIGN OBJ TO @SYSTEM(SYSTEM)OH.32 BLOC=Y
TSM>ASSIGN OBR TO @SYSTEM(SYSTEM)OH.32 E BLOC=Y
TSM>ASSIGN SLO TO SLO
TSM>SYSGEN
```

3.10.2 SYSGEN and the Expanded Execution Area

When SYSGEN is activated, it processes an object pre-scan, followed by an object load. The object in the file assigned to logical file code OBR is scanned, followed by the file assigned to the logical file code OBJ. This allows the extended mode object code to be processed before the nonextended object code, and prevents the module from being created in both modes.

When the files for extended execution have been properly assigned, SYSGEN completes the following:

- SYSGEN resolves all base mode references when a new image is created. The data sections of the affected modules are placed in the nonextended portion of MPX-32 while the code sections are placed in the extended portion. This allows all memory references, other than branch objects, to be referenced with a base register of zero.

- SYSGEN identifies the sections that require instruction modification to use a nonzero base register (i.e., all local branch instructions), and which registers to use.
- SYSGEN provides the linkage that is required to change from one instruction mode to the other. Additional code required to perform the linkage is placed in a separate section that is located in the first 16KW of memory.
- SYSGEN resolves all references between the extended and nonextended modules.

3.10.3 Using Expanded Execution Space

Extended memory is user transparent after SYSGEN has been performed, assuming that EXTDMPX has defaulted to MINADDR. The only extended MPX-32 directives that can be specified after the SYSGEN has been performed are the Cataloger EXTDMPX and TSM EXTDMPX directives. These directives specify where extended MPX-32 is located in the mapped logical address space of the task.

- Cataloger EXTDMPX directive -- overrides the SYSGEN EXTDMPX assignment. When building a load module, if the Cataloger EXTDMPX directive is not specified, the SYSGEN EXTDMPX directive remains in effect when the task is activated. For syntax information on the Catalog EXTDMPX directive, see the MPX-32 Utilities Manual.
- TSM EXTDMPX directive -- overrides the Cataloger and SYSGEN directives. The TSM EXTDMPX directive has no effect on base mode executable images or on nonbase tasks cataloged with ENVIRONMENT SHARED. If the TSM EXTDMPX directive is not specified and the Cataloger EXTDMPX directive was specified, the Cataloger EXTDMPX directive remains in effect. If the Cataloger EXTDMPX directive was not specified, the SYSGEN EXTDMPX directive is in effect. For syntax information, see Volume II of the MPX-32 Reference Manual.

Figure X-4 depicts the sectioned task area with the different EXTDMPX directive parameters.

3.10.4 Extended MPX-32 Performance Considerations

System	EXTDMPX parameter	Considerations
32/87	Unavailable	Extended memory cannot be configured on this system because the 32/27 does not support the base mode instruction set.
32/87	MINADDR, MAXADDR, or map block number=64, or any map block number between 64 and 256 that positions extended memory where no logical address usage conflicts occur	Task performance decreases if there is a greater span of valid mapping information. Default to MINADDR is suggested unless the task requires additional execution space. If additional execution space is required, EXTDMPX=64 is preferred, unless there are conflicts in the memory space usage.
32/67 and **32/97**	MINADDR, MAXADDR, or any map block number between 63 and 2048 that positions extended MPX-32 where no logical address usage conflicts occur	Translation buffers are used to load mapping information as needed. Task performance will not change as a result of a large span of valid mapping information.

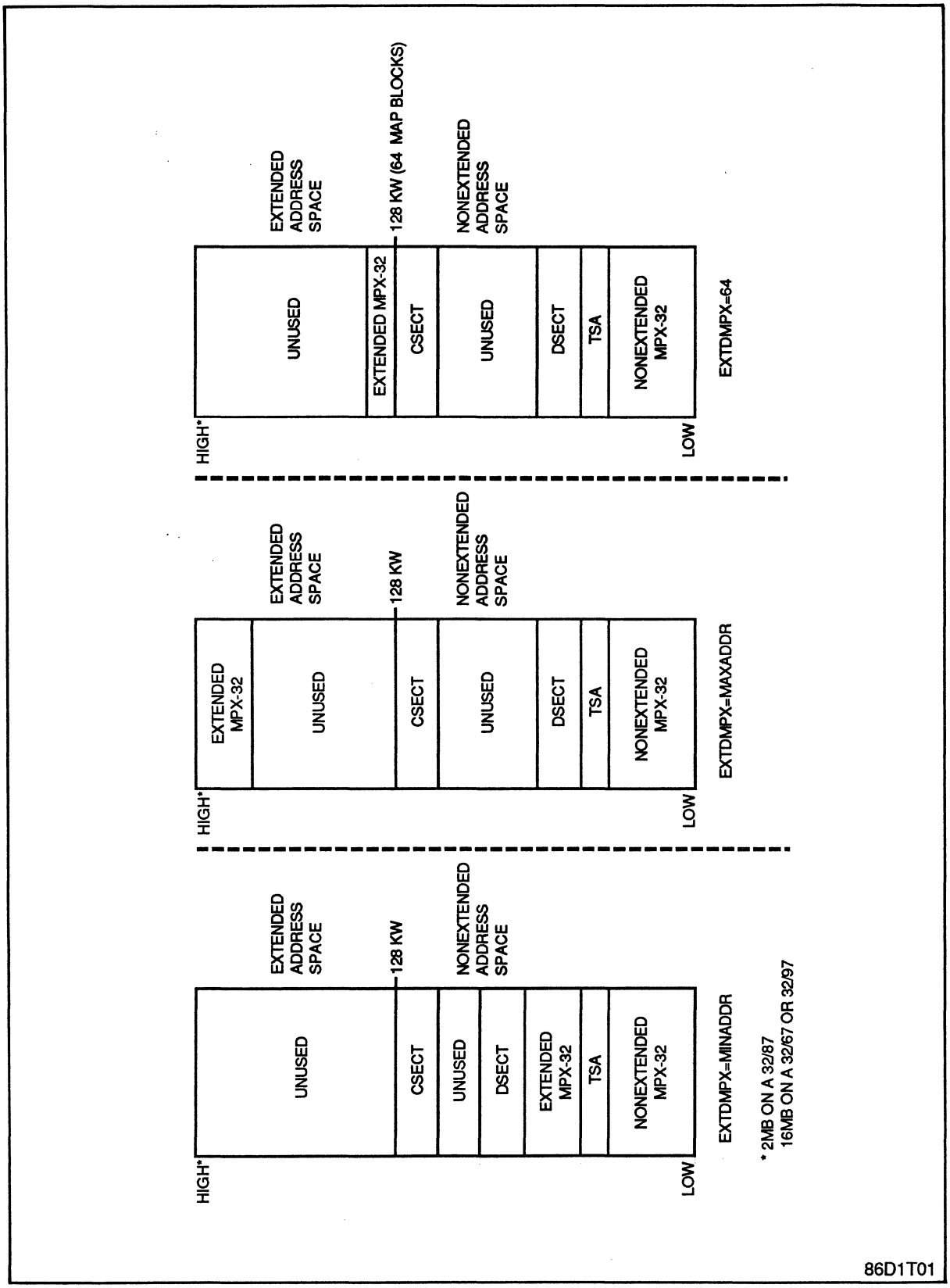


Figure 3-4. Sectioned Task's Logical Address Space Using EXTDMPX

3.10.5 Programming Considerations

Existing user modules, such as I/O handlers and service routines, can execute without code changes if they are positioned in the nonextended area of MPX-32. If user modules are positioned in the extended area of MPX-32, the following guidelines must be observed:

- SSECT and DSECT directives must be inserted around code and data areas in the source module. This causes the assembler to generate the required sectioned object code.

- The module must use the expanded execution macro calls (MBR_xxx) for intermodule communication interfaces. See the MPX-32 Technical Manual, Volume I, Chapter 1.

- The module must be reassembled with PRE assigned to MPX_EXT during assembly of code. Then, the modules can be compressed into the object library assigned to OBR for input to SYSGEN.

- The module cannot contain coding that includes:
  - subroutines that are callable from an interrupt level,
  - routines that depend on "logical equals physical" addressing,
  - nontransparent instruction addressing, such as passing return addresses to external routines or providing end-action addresses in file control blocks.

Extended MPX-32 modules cannot:

- User indirect addressing in any code area that will run in the extended code region.

- Use nonbase mode instructions, such as CEA, LEA, or SEA.

- Run unmapped.

- Use data parameter lists in which a branch and link is followed by data or address constant words in-line for external references.

When modifying a module to function in the extended mode, variable return points should not be used when designing external references. This can cause rapid expansion of the size of the adapter code.

3.10.6 Aborts and Error Messages

If extended execution errors are detected during SYSGEN, an SG37, SG38, or SG39 is generated.

Any extended execution errors generated during task execution return an SG98 abort, and an error message is displayed on the listed output and, if possible, on the user's terminal.

CHAPTER 4

VOLUME RESOURCE MANAGEMENT

4.1 Symbolic Resource Management

The MPX-32 operating system manages all of its major resources with a generalized resource management scheme. With this scheme, resources are named symbolically in directories. The directory entries establish the symbolic name to a physical resource relationship.

Each directory entry for a resource points to a resource descriptor. These resource descriptors are the central data structure for volume resource management. They contain all information which is required to manage the resource. They are stored on disc and some portions are brought into memory as needed for efficiency. Utilizing this important data structure, all primary resources are managed in the same way.

The symbolic directory entry and the resource descriptor are defined when the resource is created. A resource descriptor (RD) contains a unique identifier that can identify the resource internally. The resource identifier (RID) allows the name to be expressed and used by the system in a short and unambiguous manner.

The MPX-32 operating system also provides mechanisms which address the problems of static and dynamic resource management.

Static resource management defines the access, protection, and allocation attributes of a resource. To handle static management considerations adequately, all attributes are specifiable when the resource is defined (created).

Static resource management provides guarantees to tasks that are activated (i.e., requested for execution) ensuring the required resources will be available when the task begins execution.

Dynamic resource management provides methods that allow convenient dynamic attachment and access to a resource. To handle this requirement, the operating system provides methods to enable a task to attach or access a resource, to enqueue on a currently unavailable resource, to gain exclusive access to a shared resource, and to synchronize on the use of a shared resource.

This generalized resource management scheme means the same protection mechanism, the same resource management algorithms, and other resource operations work in a standard and predictable manner on all resources.

This general scheme does not imply users of the resources cannot detect differences between the types of resources but that, with the exception of memory attachment, the user may not have to be concerned with the differences. For users having specific needs or requirements for certain types of resources, a standard resource inquiry mechanism is available to report all information known about a resource.

4.1.1 Types of Resources

There are four major types of resources managed by the MPX-32 operating system: disc volumes, directories, files, and memory partitions.

4.1.2 Classes of Resources

There are two classes of resources managed by the operating system: shareable and non-shareable.

Shareable resources can be accessed by two or more concurrently executing tasks. For example, the resource can be attached to a task while another task is currently attached to it. Any major resource of the system can be declared as shareable when the resource is created; for example, defined to the system.

Nonshareable resources can only be used by a single task at any time. For example, the resource cannot be attached to a task while another task is currently attached to it. Disc volumes, directories and files can be declared as nonshareable when the resource is created; for example, defined to the system.

4.1.3 Classes of Resource Users

The MPX-32 operating system divides the users of resources into three distinct classes: the owner of the resource, any member of a group of users of the resource, and any other arbitrary user of the resource. These three classes are used for the purpose of controlling access rights to the resource.

- Resource Owners - The power to control access rights to the resource is given to the owner of the resource. The resource owner is determined at the time the resource is created. Generally, the resource owner is the creator of the resource. When the resource is created, the owner assigns the logical attributes and protection for the resource. These attributes can be changed after the resource is created, but this privilege is allowed only to the resource owner.

- Resource Project Groups - When a resource is created, the creator of the resource can define the name of a group of users and specify the resource's attributes and protection as it applies to all members of that group.

- Other Resource Users - A user that is not the owner or a member of the project group associated with the resource is also given a set of resource attributes and protection as it applies within this perspective.

4.1.4 Shareable Resource Control Mechanisms

Shareable resources can be attached and accessed in three different ways: exclusive, implicitly shared, and explicitly shared. These three uses are mutually exclusive of one another.

- Exclusive use - When a shareable resource is attached for exclusive use, the resource is not available for use by any other task until the resource is detached from the using task.

- Implicit shared use - When a shareable resource is attached for implicitly shared use, the resource can only be attached by another task that is attaching the resource in a compatible access mode.

- Explicit shared use - When a shareable resource is attached for explicit sharing, the resource can only be attached by another task attempting to attach the resource for explicit sharing. Tasks that are explicitly sharing a resource do not have to access the resource in compatible access modes but are expected to use the shared resource control mechanisms designed to preserve the integrity of the resource. See Section 4.10.13. Memory partitions are always attached for explicit shared use.

4.2 General Resource Control

The needs for resource control include the following:

- Time-critical task must be able to quickly attach and access a resource

- Task must be able to enqueue on the access or attachment of a resource

- Task must be able to gain exclusive use of a shared resource

- Task that is sharing a resource must be able to synchronize on the use of that resource

Potential users of resources must consider that conflicts for use of resources can occur. When conflicts occur, the operating system furnishes the user with mechanisms that aid in resolving the conflicts. These mechanisms are: the ability to attach a resource statically or dynamically, the ability to enqueue for attachment or access to a resource, the ability to attach a shareable resource for exclusive use, and the ability to synchronize on the use of an attached shareable resource.

4.2.1 Enqueue and Synchronous Notification Mechanism

When a task is attempting to dynamically attach a resource, the resource may not be available. If a denial return is furnished, the task is notified immediately and control is returned to the task. If a denial return is not furnished, the task is enqueued for the resource and removed from execution until the resource becomes available.

With the automatic enqueue mechanism, the task can optionally specify the length of time it is willing to wait for availability of the resource. If the resource is available within the prescribed time, the task is given normal completion status. If the resource is not available in time, abnormal completion status is reported to the task.

4.2.2 Dequeue Mechanism

When an unavailable resource is released, the highest priority task enqueued for the resource is attached to the resource and is made eligible for execution.

4.3 Shareable Resource Access Control

After a shareable resource is attached, the tasks using the resource need mechanisms for controlling its access. The following mechanisms provide lock control and enqueue capabilities at attachment and momentary synchronization lock control and enqueue capabilities at access.

4.3.1 Shareable Resource Locking

When a task desires to gain exclusive access to a shareable resource, the task can attach to the resource for exclusive use. Once attached, no other tasks in the system can attach to the resource. The resource remains attached until released.

Once a resource is attached, subsequent requestors of the resource may enqueue for the resource.

4.3.2 Shareable Resource Synchronization

If a set of tasks desire to synchronize access to a shareable resource, the tasks may all attach the resource for explicit shared usage and then employ the resource synchronization locking mechanism. A resource that is synchronization locked should not be accessed by any task other than those that have secured the synchronous lock, but the operating system will not prevent concurrent access by other tasks not using the lock.

Once a resource is locked, subsequent requestors of the lock may enqueue for the lock.

4.4 Standard Disc Structure

A uniform volume structure, coupled with a two-level directory structure is implemented in the MPX-32 operating system. The operating system supports removable disc pack volumes. These volumes are constructed and managed using a standard format (see Section 4.8).

4.4.1 Directory Structure

The operating system incorporates a two-level directory structure. See Section 4.9. This type of directory structure is well-suited for interactive program development because it gives each user of the system a perspective of being the only user of the system.

The functionality of the directory structure in conjunction with the overall volume organization provides fast access for the time-critical user. Specifically, a directory entry always points to a resource descriptor. The resource descriptor has associated with it a unique volume relative address. This fact causes all symbolic file names to be reduced to a short unambiguous name, for example, the file identifier. By furnishing the file identifier, time-critical tasks are guaranteed they acquire the file description in one access.

4.4.2 Root Directory

Each removable volume contains a master directory referred to as the root directory. This directory is the directory of all directories defined on a volume. The root directory can also contain the definition of other resources which are not directories. All named resources on the volume can be located through the root directory.

4.4.3 Current Working Directory

Associated with the two-level directory structure is the notion of a current working directory. The current working directory is a directory located on a mounted volume. From the perspective of this directory, a task can reference resources defined in the directory without specifying a complete pathname (see Pathname discussion next). In other words, a task can specify a one-to-sixteen character resource name and the operating system will prefix the current working directory and volume name to form a complete pathname in order to locate the resource.

In the interactive environment, a current working directory is associated at logon.

In the batch environment, a current working directory is associated when a job starts execution.

In the real-time (independent) environment, the current working directory is the same as the one associated with the activator.

All named resources on a volume are accessible from the root directory.

4.5 Pathnames

A pathname is a symbolic (ASCII) character string used to unambiguously refer to any named resource defined on a volume. The pathname contains a sequence of symbolic names. Each symbolic name is delineated by special characters embedded in the pathname string. The special characters enable the operating system to directly interpret the meaning of each symbolic name. The last symbolic name in the pathname string is the target of the pathname. A pathname target can be a file, directory or memory partition.

Utilizing a pathname, any named resource existing on a volume can be located for purposes of attachment or inquiry. To locate a resource, the operating system requires the identity of the volume and directory containing the resource (target). As previously mentioned, the special characters imbedded in the pathname string uniquely identify the required components. These required components can be specified explicitly or by implication.

A pathname explicitly stating each of the required components is referred to as a fully-qualified pathname. Fully-qualified pathnames are processed by examining the pathname string starting at the left and proceeding to the right or end of the string. As each special character is detected, the identified component is located in the appropriate system structure. An identified volume is found in the Mounted Volume Table (MVT), a directory is found in the root directory of the volume and the target resource is found in the specified directory. If any of the pathname components is not found, the processing of the pathname is terminated at that point.

Each active task in the operating system has an associated current working volume and directory. The current working volume and directory are associated with the task when it becomes active in the operating system. The current working volume and directory allow the task to reference resources defined in this association by implication. That is, the task is allowed to reference a resource with a pathname that is not fully-qualified. For example, such a pathname may indicate the pathname execution is to start at the root directory. In this case, the current working volume is implied. As another example, only the resource name is specified. In this case, the current working volume and

directory are implied. The task can view this operation as if the operating system were supplying the missing pathname components. To be precise, the task must consider the operating system is concatenating the current working volume or current working volume and directory to the pathname supplied by the task. This concatenated string must resolve to a fully qualified pathname in order to be successfully executed by the operating system.

4.5.1 Executing Pathnames

When a pathname is presented to a command or service, the operating system parses the pathname. As mentioned previously, the operating system interprets special characters imbedded in the pathname to have a specific meaning. Special characters that are allowed to be specified in a pathname are as follows:

Special character	Description
@	A volume
^	Root directory of a volume
(	Named directory on a volume
)	Named resource on a volume

4.5.2 Fully-qualified Pathnames

A fully-qualified pathname consists of all the information the operating system requires to locate and identify a resource. Whenever the following pathnames are presented to the log directive or service, only the specified resource will be logged.

Examples of fully-qualified pathnames

1. @volume^(directory)resource

   @ indicates a volume is being specified

   volume is the 1- to 16-character name of the volume where the required resource resides. The volume must be physically mounted on the system. The system determines the physical device where the specified volume is mounted.

   ^ indicates the root directory of the volume is being specified

   (indicates a directory is being specified

   directory is the 1- to 16-character name of the directory on the specified volume in which the required resource is defined.

) indicates a resource is being specified

   resource is the 1- to 16-character name of the resource to be located in the specified directory on the specified volume

2. @volume(directory)resource

@	indicates a volume is being specified
volume	is the 1- to 16-character name of the volume where the required resource resides
(	indicates a directory is being specified
directory	is the 1- to 16-character name of the directory on the specified volume where the required resource is defined. In this example, the optional special character to indicate the root directory is not used. However, this pathname is equivalent to the preceeding example.
)	indicates a resource is being specified
resource	is the 1- to 16-character name of the resource to be located in the specified directory on the specified volume

3 @volume^resource

@	indicates a volume is being specified
volume	is the 1- to 16-character name of the volume where the required resource resides
^	indicates the root directory of the volume is being specified
resource	is the 1- to 16-character name of the resource to be located in the root directory on the specified volume

4.5.3 Not Fully-qualified Pathnames

The operating system allows the use of pathnames that are not fully-qualified. The use of such a pathname causes the operating system to substitute symbolic names that are not directly specified when the pathname is presented to a directive or service.

The operating system makes the appropriate substitions based on the association with the user's current working volume and current working directory.

Examples of not fully-qualified pathnames

1. ^(directory)resource

^	indicates the root directory of the current working volume
(	indicates a directory is being specified
directory	is the 1- to 16-character name of the directory on the volume where the required resource is defined
)	indicates a resource is being specified

resource — is the 1- to 16-character name of the resource to be located in the specified directory on the specified volume

2. resource

indicates the root directory of the current working volume

resource — is the 1- to 16-character name of the resource defined in the root directory of the current working volume

4.5.4 Fully-qualified Pathnames for Directories Only

Any of the preceding pathnames can be used to locate or reference directories or any other resources defined on a volume. The following pathname formats can only be used when referencing directories. These formats are only allowed in the change directory and log resource commands and services.

1. @volume^(directory)
   @volume(directory)

@ — indicates a volume is being specified

volume — is the 1- to 16-character name of the volume where the required directory resides

^ — indicates the root directory of the volume is being specified

(— indicates a directory is being specified

directory — is the 1- to 16-character name of the directory to be located in the specified volume

Usage:

These pathname formats are equivalent to each other. If this pathname format is used with a log resource directive, all files in the specified directory would be logged.

2. @volume ^directory

@ — indicates a volume is being specified

volume — is the 1- to 16-character name of the volume where the required directory resides

^ — indicates the root directory of the volume is being specified

directory — is the 1- to 16-character name of the directory to be located in the specified volume

Usage:

If this pathname format is used with the change directory directive, the directory name would be changed to the specified directory on the specified volume. If this pathname format is used with the log resource directive, the specified directory on the specified volume would be logged.

4.5.5 Not Fully-qualified Directory Pathnames

The operating system allows the use of pathnames that are not fully qualified. The use of such a pathname causes the operating system to substitute symbolic names that are not directly specified when the pathname is presented to a command or service.

1. ^(directory)

^	indicates the root directory of the current working volume
(	indicates a directory is being specified
directory	is the 1- to 16-character name of the directory to be located on the current working volume

Usage:

If this pathname format is used with the change directory directive, the directory would change to the specified directory. If this pathname format is used with the log resource directory, all files in the specified directory are logged.

2. @volume^

@	indicates a volume is being specified
volume	is the 1- to 16-character name of the volume where the required directory resides
^	indicates the root directory of the volume is being specified

Usage:

If this pathname format is used with the log resource directive and the ROOT= option is reset, all directories in the specified volume would be logged. If this pathname format is used with the LOG RESOURCE directive and the ROOT= option is set, the root directory in the specified volume would be logged. This pathname format is not valid with the change directory directive.

3. ^

^	indicates the root directory of the current working volume. Most often, this form of pathname determines the names of all directories defined on the current working volume.

Usage:

If this pathname format is used with the log resource directive, all directories in the root directory of the current working volume would be logged. This pathname format is not valid with the change directory directive.

4. ^directory

^	indicates the root directory of the current working volume
directory	is the 1- to 16-character name of the directory which resides on the current working volume

Usage:

If this pathname format is used with the change directory directive, the default working directory would be changed to the directory specified on the current working volume. If this pathname format is used with the log resource directive, the directory would be logged.

4.6 Resource Protection

Protection is supplied for environments where protection is desired. When a resource is created, the user can specify the protection attributes of the resource. Since protection can be harmful as well as helpful, the user is advised to only protect resources to the appropriate level required.

Each resource defined to the system has protection attributes that are unique and appropriate for the resource. For example, files are protected from being accessed in certain modes (see Section 4.10.12), directories are protected from being searched or modified (see Section 4.9.3), and memory partitions are protected similarly to files (see Section 4.11.2).

Any of the major resources managed by the operating system can be protected from the perspective of the owner of the resource, a member of a group of users of the resource, or an arbitrary user of the resource. This protection scheme is versatile and gives the owner of the resource a reasonable means to control the resource.

When a resource is created, the process requesting the creation of the resource is the resource owner unless otherwise explicitly stated at the time of creation of the resource. The owner of a resource assigns the attributes of all the levels of protection.

4.7 System Administration

The MPX-32 operating system incorporates the notion of a System Administrator (SA). This means a person can be designated to control certain aspects of the operating system thereby enabling global management of those aspects.

It is the responsibility of the SA to define the persons allowed to logon the system in the key (M.KEY) file. Persons defined in this file are referred to as users. Each person logging on the system has an associated name. This associated name is referred to as the owner name. During the process of using the system, the user will create temporary or permanent resources. The owner's name will be recorded in these resources to indicate their origin and to determine who is controlling the resources. Furthermore, the SA will assign in the key file the capabilities associated with each user.

During the course of using the system, the persons allowed to logon the system are associated with a project group. It is the responsibility of the SA to define the projects in the project (M.PRJCT) file. There is not a rigid relationship between the users of the system and the projects to which they belong. Simply, each user has an associated project group at log on time. Once a user is logged on, he may change his project group to any name contained in the project file. In some cases, the user must furnish a key in order to have the project group changed. This key is associated with the project group name that is wanted. Management of the project file is the responsibility of the SA and the system only allows tasks with the SA attribute to modify the key and project files.

The key and project files are optional. If these files are not present in the system configuration, any user can log on the system and can be a member of any project group.

In the context of the SA concept is the notion the SA is not restricted by any mechanism in the system. The SA can gain access to protected resources, can execute privileged system functions, etc.

4.8 Volumes

A volume is a disc pack with a standard format. The following capabilities are available with a volume:

- It is portable from drive to drive and from system to system
- Its contents can be accessed by name
- It can be processed in a known manner to promote efficient, predictable operation
- It can be protected

There two types of volumes: system and user. The purpose and nature of volumes and how to use them are described in this section.

4.8.1 Volume Assignment

Before any volume resource (file, directory or memory partition) can be assigned, the volume on which the resource resides must be attached to the task. This is accomplished via the mount operation.

When the Resource Management Module (H.REMM) receives a mount request for a volume that is not physically mounted, it assigns the volume a set of attributes. These attributes determine the nature of all subsequent mount or dismount operations performed on that volume. There are four attributes that apply to mounted volumes: system, public, OPCOM-mounted, and nonpublic.

4.8.1.1 System Volume

The system volume attribute is given to the volume mounted by the SYSINIT task at system IPL and initialization. This volume is always given the first Mounted Volume Table (MVT) entry. It is referred to within the operating system by the keyword, SYSTEM, a volume name reserved to MPX-32 for this purpose.

4.8.1.2 Public Volume

The public volume attribute is supplied as an option to the mount function. When this option is specified, the volume is mounted for public use. These volumes remain mounted as long as the system is running and are available for resource assignments by all tasks subsequently activated in the system. The system volume, swap volume (if different from the system volume) and other volumes designated by the System Administrator (SA) are automatically mounted as public volumes by the system initialization process. The task requesting the mount of a public volume must have the SA attribute or the mount request is denied.

4.8.1.3 OPCOM-Mounted Volume

The OPCOM-mounted volume attribute is assigned to any volume explicitly mounted via the Operator Communications (OPCOM) task. These volumes remain mounted until an explicit dismount request is issued for them with OPCOM. For example, they remain physically mounted even though their use count may go to zero when dismounted by a task other than OPCOM.

The public volume attribute takes precedence over the OPCOM-mounted attribute. Hence, if a volume is mounted by OPCOM with the public option specified, the volume will be mounted as a public volume.

4.8.1.4 Nonpublic Volume

The nonpublic volume attribute is assigned to any volume not mounted with the public option. These volumes are assigned specifically to the tasks that mount them. The Resource Management Module maintains use and resource assign counts on these volumes for system accounting. A nonpublic volume is physically dismounted only when the associated use and resource assign counts go to zero as the result of a dismount request.

The OPCOM-mounted attribute is compatible with nonpublic volumes. However, a nonpublic, OPCOM-mounted volume is physically dismounted only when the dismount requirements of both attributes are satisfied.

4.8.2 Mounting Formatted Volumes

Volumes are mounted implicitly, as a result of task activation, or explicitly, through the M.MOUNT service. When a task is activated, all public volumes known to the system and the task's default working volume are implicitly mounted for the task by the Resource Management Module. The task is not required to explicitly mount these volumes.

The default working volume is the volume designated by the system M.KEY file for tasks activated by the TSM environment (interactive or batch). Tasks activated in the real-time environment assume the working volume of the parent task at activation.

If a task wishes to allocate resources on volumes other than those implicitly mounted as the result of task activation, it must issue a mount request for each additional volume with the M.MOUNT service. This may be accomplished as a static assignment or as a dynamic mount request during task execution.

The mounting of a volume establishes a task as a user of that volume. If the volume is not physically mounted at the time of the mount request, the nonresident task, J.MOUNT is invoked for the requesting task. J.MOUNT issues a mount message at the operator's console and waits for the operator to inform it a volume has been installed on the requested disc drive. This message may be inhibited, if desired, by SYSGEN directive or mount option. The format of the mount message for formatted volumes is:

```
MOUNT DISC VOLUME volname ON DRIVE devmnc---
{TASK} taskname,taskno REPLY R,H,A OR DEVICE:
{jobno}
```

volname — is the 1- to 16-character left-justified, blank-filled name given to the volume when it was formatted by the Volume Formatter

devmnc — is the device mnemonic for the unit selected in response to the assignment. If a specific channel and subaddress are supplied in the assignment, the specific drive is selected and named in the message; otherwise, a unit is selected by the system and its complete address is named in the message.

jobno — if the task is part of a batch stream or job, identifies the job by job number

taskname — is the name of the task requesting the volume mount

taskno — is the task number supplied to this task by the system

R,H,A,DEVICE — the device listed in the message can be allocated and the task resumed (R), a different device can be selected (DEVICE), the task can be aborted (A), or the task can be held with the specified device deallocated (H)

When the volume has been physically mounted, J.MOUNT reads the volume descriptor, initializes a memory-resident Mounted Volume Table (MVT) entry for the volume, and verifies the volume integrity.

Upon successful mounting of a nonpublic volume, the Resource Management Module allocates a Volume Assignment Table (VAT) entry in the task's TSA for the volume. This, in effect, attaches the volume to the task for subsequent resource assignments and system accounting.

4.8.3 Automatic Mounting at System Boot

The task SYSINIT automatically mounts the system volume as part of the IPL process. The volume mounted by SYSINIT is always given the system and public volume attributes.

SYSINIT mounts the swap volume, if requested by the operator in response to a system prompt. After the swap volume is mounted, it is used by the system swapper (J.SWAPR) for swap file space allocations. A new swap volume may be established whenever the system is booted. There is no requirement to generate a new system image via SYSGEN just to change the swap volume.

If a specific swap volume is not requested at IPL, the system volume is used as the swap volume. The swap volume is always given the public volume attribute.

4.8.4 Dismounting Formatted Volumes

When a task no longer requires the use of a volume, it detaches itself from the volume via the dismount operation. This may be performed explicitly through the M.DMOUNT service or implicitly through the task exit process.

Upon receiving a dismount request for a nonpublic volume, the Resource Management Module disassociates the task from the volume by updating its Volume Assignment Table (VAT) and the associated Mounted Volume Table (MVT) entry. If the MVT indicates there are no other users of the volume, the volume descriptor is updated, and a dismount message is displayed on the operator's console provided the issuance of messages has not been inhibited, and the volume is dismountable.

Public volumes are not considered dismountable, and a dismount message is never issued for them. OPCOM-mounted volumes are not considered dismountable unless the dismount request is issued by OPCOM. When this occurs and the MVT indicates there are other users of the volume at the time of the OPCOM dismount request, a dismount message is not issued, but all subsequent mount requests are denied for that volume. A dismount message is then issued when the last task attached to the volume detaches itself through task exit or an explicit dismount request.

When a task exits, the Resource Management Module implicitly dismounts all volumes assigned in the task's Volume Assignment Table.

4.8.5 Components of a Volume

A volume is comprised of several components which are used for particular functions. The way that the structure is defined and the way that it is used determines its total functionality. The structures and formats defined in this section achieve the functions described in previous sections.

The component structures are:

- boot block
- volume resource descriptor
- volume root directory descriptor
- resource descriptors
- resource descriptor allocation map
- volume root directory
- space allocation map

The boot block and volume descriptor must reside in absolute fixed locations; other portions are located in either relative fixed or nonfixed locations.

4.8.5.1 Boot Block

System volumes are mounted and used for IPL. When this is done, the boot block is read into the system memory and executed. This brings in the resident system image. The process is called a bootstrap. The boot block consists of a fixed number of the first consecutive blocks on the volume. They are located at the beginning of the volume to simplify the IPL process. For standard disc devices, the boot block begins at head 0, track 0, sector 0.

The number of blocks dedicated to the boot block is determined as some common denominator between the currently used sectoring (192W) and sectoring which is power-of-two related. The boot block size will be large enough to facilitate the bootstrap process currently required.

A bootstrap always occupies the boot block. The difference between system versus user volumes is that system volumes have a system image.

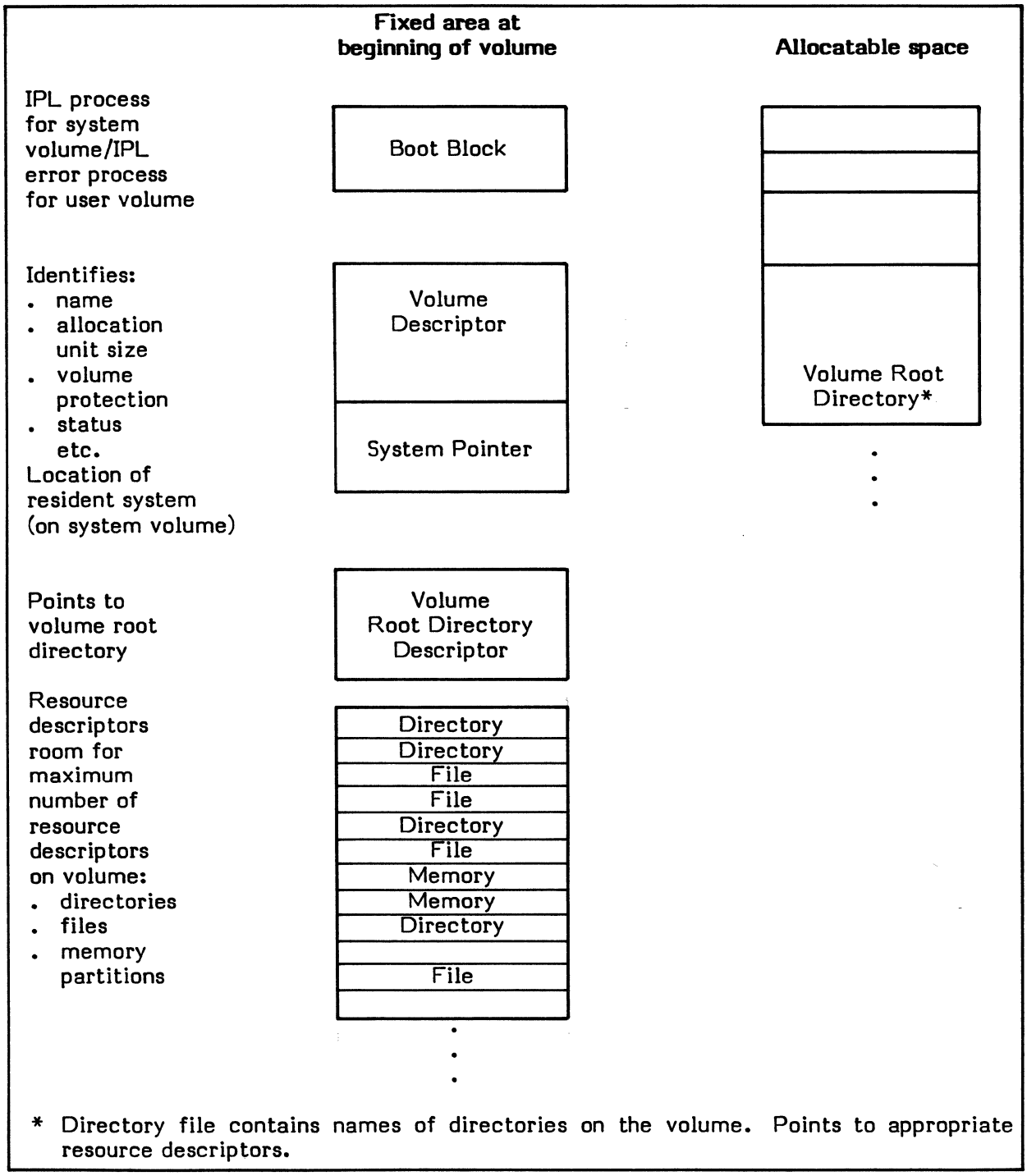


Figure 4-1. Volume Format

4.8.5.2 Volume Descriptor

The volume descriptor is a form of resource descriptor used to define the volume on which it resides. Its contents define protection and access privileges of the volume. It identifies the name of the volume, whether it can be mounted as a system volume, the volume's granularity, and other attributes, all of which are specified when the volume is created. The volume descriptor is in a fixed location and is the next consecutive block following the boot block.

The volume descriptor also points to key structures on the volume by giving their location (starting block). The volume descriptor identifies the location of the resource descriptor segments. Two segments are allowed, the first of which is obtained when the volume is created. A segment descriptor is contained within the volume which points to this segment and describes the amount of space allocated for descriptors (number of descriptors/blocks). One additional entry is provided to enable obtaining additional space for descriptors.

The volume descriptor contains a bootstrap descriptor which contains information required by the bootstrap about the system image and the device on which it resides. The bootstrap descriptor also describes the amount of space required for the system image. Entry space is also provided in the bootstrap descriptor to hold a pointer to the descriptor of a file having an alternate system image. Either image can be used at system IPL.

Other information required by the mounting/dismounting process is kept in the volume descriptor. Some of this information is also brought into an in-memory table when the volume is attached (mounted). The volume descriptor is structured so the information required for in-memory use (once the volume has been mounted) can be moved into memory simply.

4.8.5.3 Resource Descriptors (RDs)

The resource descriptor describes a particular resource. Its contents define attributes of the resource, its protection, requirements, limitations, etc.

Different types of descriptors are used to describe the generic resources made available and managed by the operating system. The different descriptor tables are referred to as directory resource descriptors, file resource descriptors, volume resource descriptors, and memory resource descriptors.

The different types of resource descriptors are intermixed within the space allocated for resource descriptors and identified by type by a fixed position field within the resource descriptor. The initial amount of space reserved for obtaining resource descriptors is specified when a volume is created. The creator also specifies whether the space obtained for resource descriptors can be increased.

Resource descriptors are allocated and deallocated from the resource descriptor segment and tracked via a bit map. Each segment contains its own bit map located at the end of the segment. When the first segment no longer contains any free descriptors, a second segment is obtained, if allowed. The descriptor is then obtained from a second segment. All subsequent requests are then obtained from the second segment until resource descriptors from the first become available.

All resource descriptors have three basic sections: a section containing information common to all types of resource descriptors, a section containing information different for each type, and a section for user supplied information. Certain resource descriptor information can only be changed by the system.

Copies of permanent file resource descriptors can reside in the Memory Resident Descriptor Table (MDT). This eliminates the disc access necessary to read the RDs, and results in a reduction in the amount of time spent when allocating files. Refer to the Rapid File Allocation Utility (J.MDTI) Chapter in Volume II for details.

4.9 Directories

A directory is a list of names of resources, where the entry for each name points to a resource descriptor (RD) that defines the basic characteristics of the resource (protection, starting/ending sectors, etc.). The MPX-32 operating system directory management is based on a two-level structure. Directories are:

- Created on a volume
- Named and protected as specified by the user when he creates them
- Associated with a user at logon (each user has a current working directory associated with his logon owner name)
- Changeable; for example, the user can change his current working directory

Figure 4-2 illustrates the two-level directory concept. It deals with a disc volume containing directories and files.

In the figure, alphabetic characters are used to represent directory names, numbers are used to represent file names.

The following basic concepts are related to the operating system and the two-level directory structure:

- Volume root directory
- User directories
- Access by pathnames
- Protection

Conceptually, user directories are all the directories originating under the volume root directory at the top of the figure. All directory access begins at a volume root directory. The user can locate to a different directory or file.

Protection is the means of restricting a user's access to a directory or file. For example, if directory Y is protected, the user may or may not be able to access file 40 and if file 40 is protected, the user may or may not be able to modify file 40 or perform other types of operations on the file.

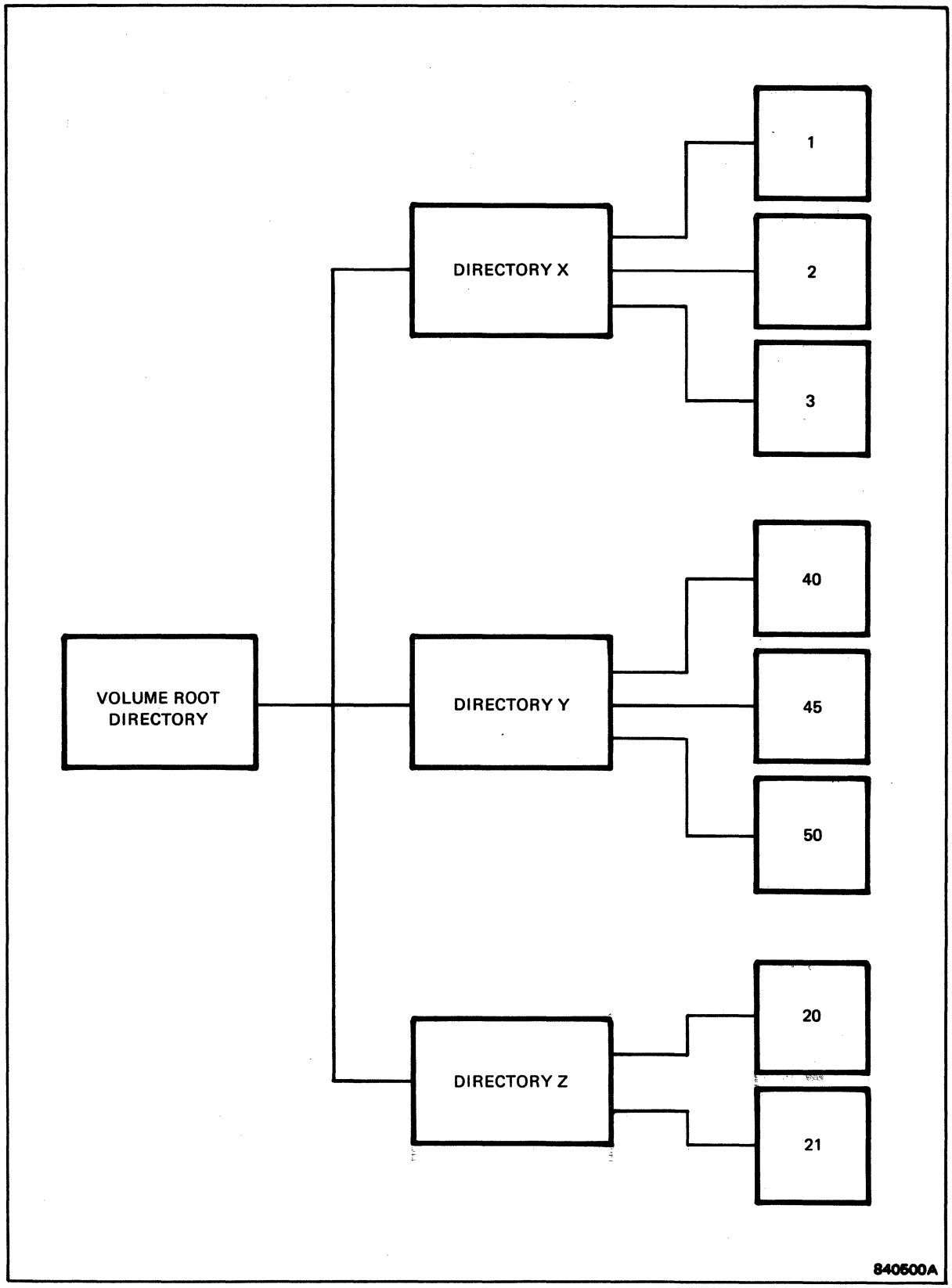


Figure 4-2. A Sample Hierarchical Directory Structure

4.9.1 Volume Root Directory

The MPX-32 operating system provides the ability to put all files on a system volume and also allows users to dismount disc packs (as user volumes). This is the concept of a root directory for a volume.

A volume root directory is maintained on every volume (see Figure 4-1) and lists the names of the directories and resources (normally files) on the volume. To get from one directory to another, the user starts from the volume root directory by using the special character uparrow (^) to go to the current volume root directory.

4.9.2 Creating Directories

The create directory function creates a directory and defines its protection and other attributes. Figure 4-3 illustrates the directory function where a user of directory X wants to access file 3. (Also see Figure 4-2).

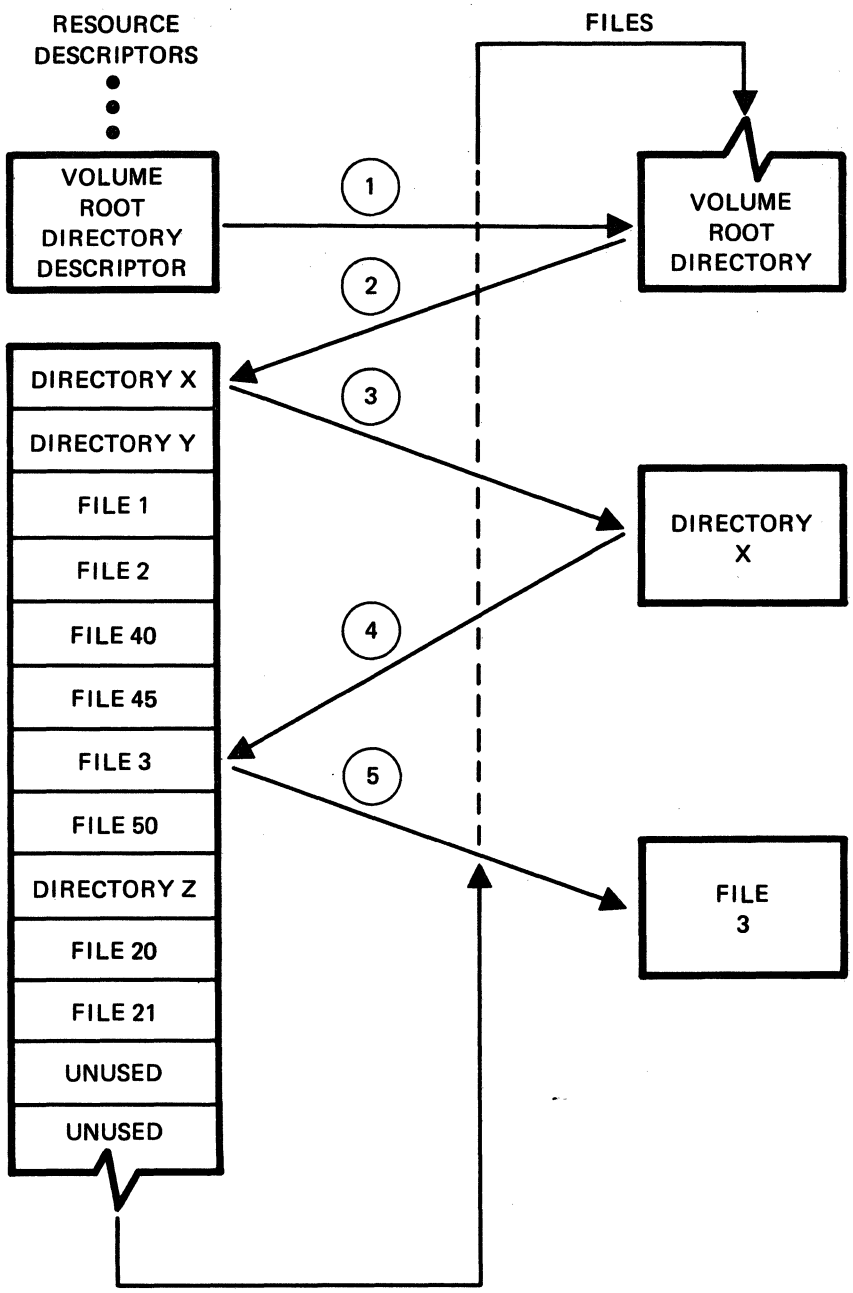
```
@volume^(directory)file
^(directory)file
```

The user must have the ability to traverse the volume root directory and to add entries in the volume root directory. The user gains access to the volume root directory as its owner, as a member of its defined project group, or as other.

To create a directory, the owner provides information which is stored in the resource descriptor for the directory:

- Owner name - can specify other than his logon owner name as the owner of the directory
- Project group name - the name of a group of users identified by the owner to have specific access privileges to the directory
- Protection - the set of operations allowed separately for the owner, the defined project group name, and all others

Once a directory has been created, entries for files or memory partitions are defined using the create function.



NOTES:

1. THE ROOT DIRECTORY DESCRIPTOR POINTS TO THE VOLUME ROOT DIRECTORY.
2. THE VOLUME ROOT DIRECTORY CONTAINS THE NAME OF DIRECTORY X AND POINTS TO ITS RESOURCE DESCRIPTOR.
3. THE RESOURCE DESCRIPTOR FOR X POINTS TO DIRECTORY X.
4. DIRECTORY X CONTAINS THE NAME OF FILE 3 AND POINTS TO THE RESOURCE DESCRIPTOR FOR THE FILE.
5. THE RESOURCE DESCRIPTOR FOR FILE 3 POINTS TO FILE 3.

840501A

Figure 4-3. Locating a File on a Volume

4.9.3 Protecting Directories

The creator of a directory can allow or restrict the ability to read the resource descriptor for the directory or to delete the directory.

- Read - allows the directory to be read
- Delete - allows the directory to be deleted
- Traverse - allows the directory to be traversed via a pathname

These access rights can be applied separately to the owner, the project group, and all other users.

4.9.4 Protecting Directory Entries

The creator of a directory can allow or restrict the ability to add resources to or delete resources from the directory he creates.

- Add - allows additions to the directory
- Delete - allows deletions from the directory

These access rights can be applied separately to the owner, the project group, and all other users.

4.9.5 Using Directories

Since all files and memory partitions are located in directories, different directories are normally traversed in the process of creating and manipulating resources. Although the ability to associate a working directory with a user at logon relieves the naive user from having to know about directory access, many users will be accessing more than one directory.

Via a special system file and other files established by the System Administrator (SA), a working directory is associated with a user at logon. All files he creates are automatically located in this directory unless he specifies otherwise. To access a resource within the current working directory, only the name of the desired resource needs to be supplied.

The working directory associated with an owner name can be changed to a different directory by using the change directory function or service and supplying the pathname to the new working directory.

What the user can do in the new working directory depends on the protection defined for it and whether he owns it, supplies a project group name to gain project group access rights to it, or is another user (other).

To locate a file in a directory other than the current working directory, a directory pathname ending with the name of the file must be specified. To locate a directory other than the current working directory, a pathname ending with the name of the directory must be specified. See Section 4.5.

4.10 Files

Files are sets of information stored on a volume. A file is given a unique identity so it can be referenced as a single entity for processing.

Files can store data, transactions, executable code, command sequences, etc. This section considers files as resources and does not deal with the structure of information within files.

Two types of files are allowed by the operating system: permanent and temporary. Permanent files remain defined to the operating system until they are explicitly deleted and they can be referred to in two ways: by their given name or by a unique identifier (RID) assigned by the system which allows faster access to information about the file. Temporary files are defined in the operating system as long as the task requiring them is in execution and then they are automatically deleted.

Files are attached for use either statically or dynamically. Attachment for files is a two phase process, where the first phase is assignment. Once assigned, they can be opened for use in a particular access mode. The requestor then operates on the file according to the allowed access. Files can be extended dynamically to obtain additional space. Shareable files can be accessed by multiple users. Sharing is normally restricted to compatible access modes.

A fast access mechanism is supported by MPX-32 which enables referencing resources (directories, files, memory partitions) by their resource identifier (RID). All resources defined on a volume have an associated RID. The creator of the resource can obtain the resource's RID by specifying the appropriate address in the Resource Create Block (RCB) when the resource is created. See Chapter 5, Tables 5-12, 5-13 and 5-14. This fast access mechanism should not be confused with the fast access attribute which only applies to files. See Chapter 3, Section 3.6.7.

4.10.1 File Attributes

The basic attributes of a file are defined when it is created and include:

- Size
- Extendibility
- Access privileges
- Sharability
- Contiguity
- Fast access
- Protection

These attributes can only be altered while the file remains defined to the system.

4.10.2 Obtaining File Space

Space is obtained for a file out of unused space on a volume. The user requests space initially by the create function and subsequently by an extend function. A file can be extended automatically by setting up appropriate parameters when the file is created.

The space obtained for a file is marked used to prevent duplicate use of the same space. The space obtained for a file in any single allocation request is called a segment. Files containing a single segment are contiguous, while those containing multiple segments may not be contiguous.

There are certain efficiency tradeoffs resulting from obtaining space in different ways. The most efficient use of space is gained from obtaining contiguous space. This can be ensured by requesting the total amount of required space when the file is created and by declaring the file nonextendible. Efficiency gained by using contiguous space is most noticeable when randomly accessing a file. Files that need to be extendible can gain some efficiency by extending space in fixed length segments. Although this is not as efficient as creating a single contiguous file space, it is a significant improvement over extending in variable length segments.

4.10.2.1 Granularity

The space on a disc volume is measured in units known as blocks. A block is the logical sector size of all disc volumes the operating system supports. The block size used is 192 words. To minimize fragmentation of unused space resulting in inefficient usage of the available space, the space is obtained in allocation units consisting of a fixed number of blocks.

The allocation unit size is determined only at the creation of a volume and can be any number of blocks. The default size for an allocation unit is determined by the class of the device on which the volume is mounted at the time it is created. Once an allocation unit size is established for a given volume, it is fixed and remains so. Although the volume can be subsequently mounted on other devices of the same class, its allocation unit size remains unchanged.

4.10.2.2 Contiguity

For efficiency, it is desirable to obtain contiguous space for a file. In a contiguous file, space is comprised of a single segment. A file is defined as contiguous when it is created by indicating it is not extendible, and by not specifying multiple segment creation option.

4.10.2.3 Extendibility

Frequently, it is difficult to determine all the space a file requires at creation and it is necessary to be able to extend the file's space according to changing needs. If specified at creation, a file's space can be extended beyond its current size.

For extendible files, it is desirable to specify the lengths (number of blocks) in which a file will grow. The length is specified when the file is created and becomes the segment size to use whenever the file is extended. The segment size specified is rounded to the next highest allocation unit defined for the volume on which the file is created. If a segment size is not specified, a system default is applied. If the file is created with the zero option specified, any extensions to the file are first zeroed.

A file can be extended by either fixed or variable length segments. A fixed length segment has its maximum increment equal to its minimum increment. A variable length segment has its maximum increment greater than its minimum increment. For fixed length segments, if the contiguous space required is not available, the file is not extended. For variable length segments, the requested amount of space is obtained except where the request is greater than any available contiguous space. In this case, the largest available amount of contiguous space is obtained and the user is notified.

A file can be extended either manually or automatically. This is also defined at creation of the file. For automatic extensions, the user of the file need not be aware of requirements for additional space. When required, the file is extended without notification. For manual extensions, the user is notified when more space is needed. Optionally, the user extends the file via the extend function. If variable length extensions are requested, the requested size extension is attempted. If the requested size cannot be obtained, a second attempt is made for the maximum increment size. If the maximum increment size cannot be obtained, an attempt is made for minimum increment size. If none of the three sizes can be obtained, a denial is issued. For both automatic and manual extensions requiring fixed size segments, if the request exceeds the available contiguous space, the request is denied.

4.10.2.4 Size

A file's size is the amount of space obtained for it. The size of a file is determined at creation of the file or by extending the file.

The size specified when a file is created is the minimum space allocated to the file (initial space allocation). The close function will cause a truncation of the file's space to the file's initial space or its minimum space requirement.

The maximum size allowed for a particular file to be extended can be specified at creation of the file. The maximum size for the file can also be specified to prevent users from extending a file beyond prescribed limits.

4.10.3 File Names and Fast Access

It is desirable to reference files in a simple fashion. Files are given symbolic names at their creation and are then referenced by their name. Such files are called permanent files. Files created without specifying a name are called temporary files.

All temporary and permanent files are known internally by a unique file identifier (RID). This identifier is assigned internally at creation of the file and identifies the resource descriptor block which defines the respective file's attributes and space.

When referencing a file by its name, a given amount of time (overhead) is required to obtain its definition from a directory. This can be excessive for some time-critical applications. To eliminate the overhead of directory searches, a file may be referenced by its RID. All temporary and permanent files can be referenced by their RID. When a file is created and the fast access attribute is specified, the file's RID remains unchanged throughout any operation performed on the file. For further details on the fast access attribute, see Chapter 3, Section 3.6.7.

4.10.4 File Protection

In many applications, it becomes necessary to protect files against certain users and types of access. The users of a file are granted its use only in the ways allowed them. Protection can be applied to safeguard attaching and/or extending files.

Protection of files is established by the creator/owner at creation of the file. The owner of the file is the only user allowed to specify or modify its protection attributes.

4.10.5 Permanent Files

Permanent files are named files and are permanent to the system until explicitly deleted. They are known to the system by one directory entry and are defined to the system at their creation. There can be one and only one definition for a named file. This definition is maintained and known to the system via a resource descriptor.

Permanent files can be used either shared or nonshared. This is determined by the owner/creator. Permanent files are shared according to rules defined previously.

The allowed uses of a file can be altered at creation, assignment, and open. For files, attachment is comprised of the last two phases (assign and open). By the time a task accesses a file, all uses have been defined and verified. How each of these functions alters the context of file use is described in the following sections.

4.10.6 Creating Files

The create function allocates file space, defines the attributes of a file, and builds a resource descriptor which defines the file space. This resource descriptor also contains the attributes of the file.

For permanent files, the user-supplied file name associates the directory entry, resource descriptor (file attributes), and the file space. For temporary files and fast access files, a unique file identifier is supplied by the system. This provides an alternate method of association with a file and its attributes.

Files become known to the system and are, therefore, said to exist when created by the operating system. The creation of a file is accomplished as requested by the successful completion of an executed create function. Once a file is created, it can be attached and accessed by its name or identifier.

At creation, the attributes of a file are defined either explicitly, by providing them as specified parameters or implicitly, by omitting certain parameters, in which case reasonable defaults, values, or attributes are assumed. Where a reasonable default cannot be made for a parameter, the parameter must be supplied by the user.

4.10.7 Attaching Files

To secure a file for use, the file must be attached. When attachment of a file is requested, the requestor is granted attachment on the basis of both the file's defined allowances to the requestor and its availability. The two phases required for the attachment of files are assignment and open.

4.10.8 Assigning Files

File assigning by the assign function is an attachment phase that associates a resource with a task by a logical file code.

Nonshareable files allow only exclusive use. Once nonshareable files are assigned, they cannot be used by others until released. A requestor of a nonshareable file that is already assigned is enqueued or optionally denied on the request.

A requestor can also gain exclusive use of a shareable file by requesting its exclusive assignment. As in the previous case, the requestor is enqueued or optionally denied if the file is currently assigned to another task.

Shareable files can be shared in two ways: implicitly or explicitly. If shared use is requested when the file is assigned, the file is explicitly shared. Assignment to a file for explicit sharing is allowed only if there are no other tasks attached to the file or if all others who are attached are explicitly sharing the file.

If neither shared or exclusive is specified when assigning a shareable file, it is implicitly shared. This is allowed only if no other tasks are attached to the file or if all other tasks are implicitly sharing the file in a compatible access mode.

In summary, the first task that attaches a file establishes the context of use for subsequent requestors of the file. The context established by the first task can be changed only when the file is detached from the tasks.

The desired access mode(s) for the requested file may be specified when assigning the file. This defines the intended access for the file attachment.

A specific access mode or multiple access modes can be requested when assigning the file for implicitly shared use. The attachment is granted if the requested access mode(s) are compatible with those of users currently attached to the file.

Access modes can be omitted when assigning a file for implicit shared use. In this case, the only access ensured is read access. Other access modes can be requested when later opening the file. As a result, the requestor may be enqueued for the request since there is no guarantee the requested access mode will be compatible with other users.

The required access mode(s) need not be specified for explicitly shared files. For explicit sharing, there is no compatibility requirement since all sharers are expected to synchronize and use locking to maintain file integrity. It is not possible for explicit sharers to contend for file use as a result of specified access modes when opening the file. They are guaranteed use of the file in the requested mode.

Assignment parameters are defined explicitly by providing them as specified parameters or implicitly by default.

4.10.9 Opening Files

A file must be assigned before being opened. A file must be opened before any operations to the file are allowed. The desired file is referenced as a required parameter to the open function.

The access mode for a file is determined at open. The type of access allowed depends on the allowances or restrictions associated with a file at creation and assignment.

The access mode in which the file is opened determines the position within the file. The open function performs logical connections of control table information between the file, system, and its requestor. Also, if required, a device handler is initialized for the device where the file resides.

4.10.10 File Operations

The operating system provides a set of operations which can be performed on files. The data structure within the file itself is of no concern to the operating system. The lowest structure recognized by the operating system is the block.

Files can be accessed either sequentially or randomly. The intended access is specified when opening the file. Files opened as sequential are operated on in a sequential manner. Subsequent operations advance one block from the previous position in the file. Files opened as random are processed or operated on in a random manner. Each operation supplies a specific file relative block number to which the operation is performed.

The access method is determined at open by examining the random access indicator contained in the File Control Block (FCB). If the random access indicator is not set, the access method of the file is determined to be sequential access.

Three general types of operations are provided for use with files:

- Read - transfer data from a file to memory
- Write - transfer data from memory to a file
- Position - move to an indicated position in a file

For read and write operations, parameters are supplied denoting: the memory address to or from which the data is to be transferred, the number of data bytes to transfer and the position in the file (implied for sequential access) at which the operation is to commence. Position can also be specified independently of read or write operations, in which case, no data transfer operations are performed during the position function.

4.10.10.1 Sequential Access

Sequential access gives the user the ability to transfer data to or from a file in a sequential manner. The user is allowed to specify a buffer address aligned on an arbitrary byte boundary and specify an arbitrary transfer count in bytes. The transfer granularity to disc files is 192 words. This means transfers to or from a file are executed in multiples of 192 words, for example, disc blocks. During output operations to a disc file, only the requested number of bytes are output to the disc file. Any bytes remaining to acquire the next highest 192 word boundary are automatically zeroed by the disc controller.

The operating system recognizes file granularity. It does not recognize the data format inside a file, therefore, the file system is not sensitive to record boundaries. A file must be read in a form that is compatible with the way the file was written.

4.10.10.2 Random Access

If the random access indicator is set in the File Control Block (FCB) when a file is opened, the access method of the file is determined to be random access. This means the user must specify the file relative block number (192 words) where the requested read or write operation is to begin. As with sequential access, the user is allowed to specify a buffer address that starts on an arbitrary byte boundary and an arbitrary transfer count in bytes. Also, as in sequential access, the operating system is only cognizant of the 192 word granularity of a file, therefore, data formats denoting record boundaries are not detectable.

The operating system supports extendible random access files. Using auto-extendibility on random access files can cause the file to become discontiguous; therefore, the efficiency of a program performing random access disc I/O might be impaired. To prevent this, create the file with sufficient size to allow for possible extension. If automatic file extension is to be inhibited, the file must be created with the appropriate attributes through VOLMGR. If autoextendibility is inhibited and an attempt is made to access beyond the file, an EOM indicator is set.

It is necessary to initialize the file with a known data pattern and detect null records-areas within the file that contain the initial data pattern. As an option, a file may be sequentially initialized with a pattern of binary zeroes at creation.

4.10.11 File Positioning

File positioning provides the capability for moving within a file without transferring data to or from the file. The rules for file positioning are dependent on which access method is in effect on the file. Two types of file positioning are allowed:

- Absolute - the ability to position to the beginning or end-of-file
- Relative - the ability to move to a location in a file with respect to the current position in the file

4.10.11.1 Absolute File Positioning Operations

Absolute positioning allows the user to position to the beginning-of-file or to the end-of-file without regard to the current position in the file. Absolute positioning is used with sequential and random access methods. Three operations are provided for absolute positioning in a file:

- Rewind file - position to the beginning-of-file and indicate beginning-of-medium
- Backspace file - position to the beginning-of-file. Same as rewind.
- Advance file - position to the end-of-file and indicate end-of-file. Any attempt to advance a file beyond end-of-file causes an end-of-medium to be indicated.

4.10.11.2 Relative File Positioning Operations

Relative positioning allows the user to position to the beginning-of-record or to the end-of-record with respect to the current position in the file. Relative positioning can be used only with the sequential access method. Two operations are provided for relative positioning in a file:

- Backspace Record - backspace a file block (192 words). The beginning-of-medium indicator is set if this condition is detected.

- Advance Record - advance a file block (192 words). The end-of-file indicator is set if this condition is detected. Additionally, the end-of-medium indicator is set if positioning beyond end-of-file is attempted.

4.10.12 File Access Modes

File access modes control allowed access methods and combinations of operations allowed to a file. The operating system defines five allowable access modes: read, write, modify, update and append. The mode in which a file is to be accessed is specified when the file is opened. Only one access mode can be specified at open.

The following chart shows the access modes and the conditions determined when the file is opened.

Requested Mode	Allowed Operation	Allowed Access Methods	File Position at Open	EOF Position at Open	EOF Position at Close
Read	Read, ABS Position, REL Position	Sequential, Random	First block of file	Highest sequentially written block number + 1	Same position as at open
Write	Read, Write, ABS Position, REL Position	Sequential only	First block of file	First block of file	Highest sequentially written block number + 1
Modify	Read, Write, ABS Position, REL position	Sequential, Random	First block of file	Highest sequentially written block number + 1	Same position as at open
Update	Read, Write, ABS Position, REL Position	Sequential, Random	First block of file	Highest sequentially written block number + 1	Equal to position at open or new highest sequentially written block number + 1 if data was appended to the file
Append	Read, Write, ABS Position, REL Position	Sequential only	* Highest sequentially written block number	Highest sequentially written block number + 1	New highest sequentially written block number + 1

- EOF refers to the actual end-of-file block (or defined end-of-file), not to any software end-of-file (either blocked or unblocked) contained within the file.
- The EOF block is equal to the first block of a file when the file is created.
- The EOF block is equal to the last block of a file if the file is optionally initialized with the binary zero pattern when the file is created.
- * For blocked files, the file position at open is placed at the last blocked software end-of-file before the defined end-of-file block.

4.10.12.1 Read Mode

A file opened in read mode allows read-only access to a file. The position of the file after opening is the first block of the file. Read operations operate from the first block of the file to the defined end-of-file.

Sequential and random access methods are allowed in the read mode. Additionally, absolute and relative positioning is allowed with respect to the restrictions appropriate to blocked files.

4.10.12.2 Write Mode

A file opened in write mode allows sequential read and write access to a file. The position of the file after opening is the first block of the file. Write operations operate from the first block of the file to the last block of the file's allocated space (EOM). With extendible files, additional file space can be automatically allocated when the EOM condition is detected and the file is being written.

The write mode is provided to write the initial data contents of a newly created file. write mode can also establish new data contents for a file (i.e., file rewrite). When a file is opened in write mode, the end-of-file (EOF) indicator is automatically set to the first block of the file. This step effectively discards the current data contents of the file. As the file is sequentially written, the end-of-file indicator is moved and logically exists at the end of all data that has been recorded in the file. Operating in this manner allows the file, if shared, to be read while a single writer is establishing new data contents for the file. This method of operation is only allowed when the writer is the first task to open the file. Readers opening the file after the writer are able to read all data the writer has written. Attempts by any reader to read data beyond the writer's current position in the file cause that reader to be suspended (i.e., blocked from execution), until the writer has established additional new data in the file. The results of this method of operation are only predictable when all sharers of the file are accessing it sequentially. If the writer cannot be certain of sequential access, the writer should attach (assign or open) the file for exclusive use.

When a file opened in the write mode is closed, the end-of-file position is recorded in the resource descriptor for the file. This enables determination of the required size of the file, and all existing and unused extensions to the space of the file are returned to the pool of allocatable space on the volume.

Only the sequential access method is allowed in the write mode. Additionally, absolute and relative positioning is allowed but discouraged if the file is to be concurrently shared with readers on the file.

4.10.12.3 Modify Mode

A file opened in modify mode enables read and write access to a file. The position of the file after opening is to the first block of the file. Modify operations operate from the first block of the file to the defined end-of-file.

The modify mode is provided to allow modifications to be made to the existing data contents of a file.

For blocked files, modify operations can rewrite or change the position of the blocked software end-of-file. However, modify operations are still constrained to operate within the original range of the first block of the file to the defined end-of-file block.

Sequential or random access methods are permitted in the modify mode. Since all or portions of the existing data in the file can be modified, other tasks are not able to gain concurrent access to the area of the file operated on by the modify mode. Additionally, absolute and relative positioning is allowed with respect to the restrictions appropriate to blocked files.

4.10.12.4 Update Mode

A file opened in update mode enables read and write access to a file. The position of the file after opening is to the first block of the file. Update operations operate from the first block of the file to the last block of the files allocated space (EOM). With extendible files, additional file space can be automatically allocated when the EOM condition is detected and the file is being written.

The update mode is provided to allow modifications to be made to the existing data contents of a file and to append new data to the file. Sequential and random access methods are allowed in the update mode. Since existing data in the file can be modified and new data can be appended, other tasks cannot gain concurrent access to any portion of the file.

Although sequential and random access methods are permitted, appending new data to a file must be done sequentially regardless of which access method is in effect on the file. Additionally, absolute and relative positioning is allowed with respect to the restrictions appropriate to blocked files.

4.10.12.5 Append Mode

A file opened in append mode allows new data to be appended to existing data in a file. The position of the file after opening is at the end of existing data in the file. Append operations operate from the file position at open to the last block of the file's allocated space (EOM), i.e., rewind cannot go past the files position at open. With extendible files, additional file space can be automatically allocated when the EOM condition is detected and the file is being written.

As new data is sequentially appended to an existing file, the end-of-file indicator is moved and logically exists at the end of all data that has been appended to the file. Operating in this manner allows the file, if shared, to be read by concurrently executing tasks while new data is being appended to the file. This method of operation is only allowed when the appender is the first task to open the file. Readers opening the file after the appender are able to read all data the appender has written. Attempts by any reader to read beyond the appender's current position in the file cause that reader to be suspended (i.e., blocked from execution) until the appender has appended additional new data to the file. The results of this method of operation are only predictable when all sharers of the file are accessing it sequentially. If the appender cannot be certain of sequential access, the appender should attach (assign or open) the file for exclusive use.

When a file opened in the append mode is closed, the end-of-file is recorded in the resource descriptor for the file. Append mode, unlike write mode, does not return unused space at the end-of-file to the pool of allocatable space on the volume.

Only the sequential access method is allowed in the append mode. Additionally, absolute and relative positioning is allowed but discouraged if the file is to be concurrently shared with readers on the file.

4.10.13 Sharing Files

Shared access allows simultaneous access for users of differing access modes but places restrictions on certain combinations for implicit sharing. Users who implicitly share a file must open the file with access modes compatible with all other users of the file. The following summarizes the compatible modes for any single access mode:

Access Mode	Compatible Access Mode
Read	Read, write*, or append
Write	Read*
Modify	Append+
Update	None
Append	Read or modify+

* Read and write are compatible only if the writer is the first task attached to the file. If not, the writer must wait until the reader closes the file.

\+ Append and modify are not compatible for blocked files.

Explicit sharing of a file allows the user to intermix all access modes, some combinations of which are considered incompatible for implicit sharing. Synchronization and file locking functions can be used to ensure locking out simultaneous accesses to files when multiple writers/readers are sharing a file explicitly and could thus yield undefined results. Explicit sharers do so knowingly, and therefore, must perform their own synchronization and locking control.

4.10.14 Closing Files

Closing a file prohibits the requestor from subsequent operations to the file. The file is then closed from the perspective of the requestor. For shared files, the file does not become closed to other sharers of the file.

When a user closes a file after implicitly sharing it, the access modes available to others for the same file can change. This is determined at the close of the file. Closing can then allow the system to complete other requests for assigning or opening the file, not formerly allowed.

In some access modes, the end-of-file is determined at its close. This also allows determination of the required size of a file and enables the return of unused space. For implicitly shared files, it can also mean that areas not previously accessible to other sharers become accessible after any single-sharer closes.

4.10.15 Detaching Files

A file can be detached by requesting the deassign function. This frees the file and returns it as an available resource to the system. The freeing of the file is from the perspective of the requestor of the detachment. If the file is being shared and is attached by other sharers, the sharers maintain attachment of the file. In all cases, when a file is detached, its use count is decremented. The file is completely returned to the system when the use count is decremented to zero (file not in use). The file then becomes available to other requestors who may have been suspended due to the file having been in use.

4.10.16 Deleting Files

Permanent files are deleted by using a delete function. When the file is deleted, the space obtained for the file is returned to the volume by marking its previously occupied allocation units as available in the volume's allocation map. The entry for the file is removed from its associated directory.

4.10.17 Temporary Files

Temporary files are a type of file resource specified and defined when they are created by the create function. They are deleted from the system when the creator of the file terminates execution.

Temporary files do not have names associated with them. They are referenced by a unique assigned identifier called a resource identifier that contains an integer index pointing directly to the file's resource descriptor.

4.10.17.1 Creating Temporary Files

Temporary files are created by executing a create function requesting a temporary file. All parameters allowed for creating permanent files are also allowed for temporary files except fast access. Because temporary files have no name and must be referenced by their assigned identifier, they are already fast access. For example, their resource descriptor can be found in one disc access. Temporary files can also be created by the assign function.

Typically, the majority of parameters allowed for creating a temporary file are not required. When a temporary file is to be made permanent, these parameters establish the attributes of the file to use when the file becomes permanent.

4.10.17.2 Assigning Temporary Files

Temporary files must be assigned by the assign function before they can be opened. Assignment establishes the tables required to use the file and reserves the file for use by the requestor.

Existing temporary files, those having previously been created by a create function, are assigned (assign function) by using their resource identifier. The resource identifier is given to the requestor when the file is created. The requestor also gives a logical file code (three characters) which becomes logically equated to the resource identifier. Once the file has been assigned, it can be referenced by its logical file code.

Temporary files not created by the create function can be both created and assigned by the assign function. For such cases, the initial space allocated is specified by the assigner or is a fixed number of allocation units.

Temporary files created at assignment are extendible. Reasonable defaults are assumed for parameters normally specified when creating a temporary file. The device where the temporary file is to be created can be specified indirectly by volume name.

4.10.17.3 Opening and Accessing Temporary Files

Temporary files are accessed initially by executing an open function. The access modes that the file can be opened in are specified when assigning the file. The rules for this function are the same rules that apply to permanent files.

4.10.17.4 Deleting and Detaching Temporary Files

The space used by a temporary file is freed by executing a deassign function. Deletion of a temporary file is implied when the file is detached.

Because all files are detached at termination of a task, temporary files are then implicitly deleted upon termination of the using task, whether the termination is normal or not. The task has the ability to make a temporary file permanent before it terminates.

4.10.17.5 Making Temporary Files Permanent

Temporary files can be made permanent through the use of the M.TEMPER system service. Execution of this function creates an entry in the directory specified by the pathname. See Directories and Pathnames. The entry points to the resource descriptor for the temporary file.

4.11 Memory Partitions-Nonbase Mode of Addressing

Memory partitions are named areas of physical memory that can be shared by concurrently executing nonbase mode tasks. Each memory partition has a relationship with physical memory and with the logical address space associated with a task. There are two types of memory partitions: static and dynamic.

Static partitions are created when the operating system is generated using the system generation (SYSGEN) process. When static partitions are created, their location and size in physical memory is specified. Additionally, their relationship to the user process logical memory is specified. Partitions declared in this manner permanently reserve the specified physical area of memory and this area of memory remains reserved until the system is regenerated. Static memory partitions cannot be deleted.

Dynamic memory partitions are created by system utilities. When the memory partition is defined, the user specifies the partition's relationship to the task's logical address space. The partition's relationship to physical memory, for example, its size is also specified, but the physical memory is not allocated until the dynamic partition is allocated to a task.

4.11.1 Creating Memory Partitions

When memory partitions are created, the attributes of the partitions are defined. These attributes include:

- Name
- Protection
- Size
- Location in logical address space

When a partition name is used to attach the partition, the size, location, and other attributes are validated.

4.11.2 Protecting Memory Partitions

The protection allowed to memory partitions is the same as the protection allowed to any other resource managed by the operating system, for example, owner, project group, and other. In addition to the common forms of resource protection, partitions can also be protected from write access.

4.11.3 Attaching Memory Partitions

To attach a partition, the partition must have been created. The requestor is granted access to the partition on the basis of the partition's access rights defined for the requestor. Valid access rights for partitions are delete, read and write. For example, a particular user cannot attach a partition that is protected from the user.

Memory partitions are always attached to a task for explicit shared use. A task is not denied attachment of a shared partition if the user for whom the task is executing has the proper access rights.

Once a partition has been attached, the nonbase mode task gains access to the partition via the M.INCLUDE and M.EXCLUDE system services. The M.INCLUDE service maps the partition into the task's logical address space, providing the space for the partition is not currently allocated.

4.11.4 Accessing Memory Partitions

Once mapped into the task's logical address space, the task accesses the space of the partition by using memory reference instructions. If the user associated with the task does not have write access to the partition, the task will be prevented from modifying the contents of the partition.

4.11.5 Detaching Memory Partitions

The M.EXCLUDE service allows the task to map a memory partition out of the task's logical address space. This makes the space available to include another partition or to use the space in some other manner. A partition that has been excluded from the task's address space cannot be referenced until it is again included.

Detaching memory partitions informs the system that the task no longer requires a guarantee that the partition will remain available for access. If the partition is mapped into the task's logical address space at the time of the detachment request, the partition is excluded from the task's address space.

Detachment of a static memory partition does not release the physical memory assigned to the partition nor does it modify the contents of the partition.

Detachment of a dynamic memory partition releases the physical memory assigned to the partition if no other tasks currently have the partition attached. When the physical memory used by a dynamic partition is released, the contents of the memory locations are made available for any type of physical memory request.

4.11.6 Deleting Memory Partitions

Static memory partitions cannot be deleted. Dynamic memory partitions can be deleted using the M.DELETE service.

4.11.7 Sharing Memory Partitions

Memory partitions can be attached and accessed by concurrently executing nonbase mode tasks. The users of shared partitions have the use of shared resource synchronization features.

Additionally, these users can develop their own protocol for sharing the resources.

4.12 Shared Images

Shared images are named areas of physical memory that can be shared by concurrently executing base mode tasks. Shared images can be absolute or position independent.

Absolute shared images have fixed logical addresses within a task's logical address space.

Position independent shared images do not contain any relocatable address references. Any references outside of the shared image are relative to a base used at execution time. The logical address of a position independent shared image is defined to the task at link time.

A shared image can contain both read-only and read/write program image sections. At link time, a specification can be made to activate the shared image as single copy or multicopy. A single-copy shared image (the default) has only one copy of both the read only and the read/write sections in memory. Both sections are shared by all tasks requesting inclusion of that image.

A multicopy shared image has a separate copy of the shared image for each including task. A multicopy-shared shared image has a single copy of the read only section for all tasks and a separate read/write section for each task.

4.12.1 Created Shared Images

Shared images are created by the LINKER/X32 when the atttributes of the shared images are defined. These attributes include:

- Name
- Protection
- Size
- Location in logical address space

The size, location, and other attributes are validated when a shared image name attaches the shared image.

4.12.2 Protecting Shared Images

The protection allowed to shared images is defined at link time and is the same as the protection allowed to any other resource managed by the operating system. This protection is the owner, project group, and other scheme. In addition to the common forms of resource protection, shared images can also be protected from write access.

4.12.3 Attaching Shared Images

To attach a shared image, the shared image must have been created. The requestor is granted access to the shared image on the basis of the shared image's access rights defined for the requestor. Valid access rights for shared images are read and write.

The access mode is requested at link time and a denial is made at include time if the requested mode is incompatible with the shared image definition.

Shared images can be included into a task's address space by preassignment or dynamic inclusion. A preassigned image is loaded and/or mapped into the referencing task's logical address space at activation time and remains there until completion or until the task excludes it via the M_EXCLUDE or M.EXCLUDE system service. A dynamic image is loaded and/or mapped upon request by the M_INCLUDE or M.INCLUDE system service.

4.12.4 Accessing Shared Images

All shared images to be included by a task, whether preassigned or dynamic, must be defined by the user at link time.

When a shared image is linked, a version number and compatibility level can be specified. This information is copied into the preamble for each task referencing the shared image at link time, and is used to verify that the shared image requested at activation is compatible with the shared image defined at link time.

The physical address of a shared image can be specified at link time enabling systems with shared memory to share the image. This feature should be used with caution because a task is given an immediate denial if the requested physical memory is allocated to another task.

A shared image can be defined as resident at link time. A resident shared image is then loaded into memory using the OPCOM INCLUDE directive, and remains in memory until removed by the OPCOM EXCLUDE directive.

4.12.5 Detaching Shared Images

The M_EXCLUDE and M.EXCLUDE services allow the task to remove a shared image from the task's logical address space. This function makes the space available to include another shared image or to use the space in some other manner. A shared image that has been excluded from the task's address space cannot be referenced until it is again included.

If write back is requested, the read/write section is written back to the disc. If write back is not requested, there is no write back. Write back is not performed until all users have detached the shared image. The physical memory occupied by the shared image is then available to other tasks, providing the image is not included as resident.

4.13 Multiprocessor Shared Volumes

The multiprocessor shared volume is an MPX-32 feature that allows tasks, operating in separate system environments, to obtain concurrent directory and file access. The operating system maintains resource integrity against incompatible access or usage modes on these resources within the scope of volume management described in this chapter.

A volume is treated as multiprocessor only if it has been software mounted as multiprocessor on a multiported drive. A multiported drive is defined to be any disc drive that is hardware configured with the ability to communicate concurrently with up to sixteen independent processors. The hardware characteristics of the disc drive are defined by the appropriate DEVICE directive supplied to the SYSGEN utility. The software characteristics of the volume are defined by the presence or absence of a SYSID parameter when the volume is mounted.

The synchronization mechanism for multiprocessor resources is maintained by software information kept in the resource descriptor (RD). Therefore, consideration must be given to system performance and access restrictions on these resources. The specific performance and restriction issues applying to multiprocessor resources are discussed in sections 4.13.2 and 4.13.4.

WARNING: The system volume cannot be a multiprocessor volume.

4.13.1 Multiprocessor Resources

A multiprocessor resource is defined as a volume resource residing on a volume mounted as a multiprocessor. Permanent files, temporary files, directories, the volume descriptor map (DMAP) and the volume space map (SMAP) can be multiprocessor resources. Memory partitions and space definitions are never treated as multiprocessor resources regardless of where they reside.

Because the resource synchronization mechanism for multiprocessor files must be kept on disc (in the RD) rather than in memory, additional system overhead is incurred in the areas of create, delete, assign, open, close and deassign resource on multiprocessor volumes. After a file is allocated, the actual number of I/O operations performed is not affected by the multiprocessor characteristics of the resource.

4.13.2 Multiprocessor Resource Access

Whenever the allocation status of a multiprocessor resource changes in any system environment, the resource accounting information maintained by the system on that resource is updated in its associated RD. The updating of this information is synchronized by a multiprocessor lock maintained in the last word of the RD. This word is reserved to MPX-32 for this purpose and should not be used by any customer applications. If this lock is set, access is not allowed to any other task in other system environments until the task that set the lock releases it. The setting and releasing of these locks is performed by the system and is transparent to the task.

If the system cannot obtain the multiprocessor lock for a task because it is set by another task in the CPU, the task will try/suspend for the number of times specified by the SYSGEN DPTRY directive. If DPTRY is one, the task is issued an immediate denial. If DPTRY is zero or not specified, the try/suspend cycle is repeated indefinitely until the lock is obtained.

The amount of time the task is suspended is determined by the SYSGEN DPTIMO directive. This directive accepts the number of timer units to be applied as the interval for all multiprocessor delays. If DPTIMO is not specified as a SYSGEN directive, a one second delay is established as the default by the Resource Management Module (H.REMM).

When access to a multiprocessor resource is denied due to assignment access mode or usage incompatibilities with another task, the requesting task is enqueued or suspended as appropriate (provided that DPTRY has indicated that it will wait for the resouce to become available). If the incompatibility is due to a task in the same CPU, the requesting task is enqueued. If the incompatibility is due to a task in the other CPU, the requesting task is suspended. If suspended, the try/suspend cycle as for multiprocessor locks is performed.

The following conditions can determine if a task can be enqueued rather than suspended for a multiprocessor resource:

1. The resource is exclusively locked, and the lock owner is in the same system environment as the requesting task.
2. Incompatible access modes are encountered on an implicitly shared resource in which the writer is known to be in the same system environment as the requesting task.
3. A synchronous resource lock cannot be obtained because the lock is owned by another task in the same system environment.

4.13.3 Mounting Multiprocessor Volumes

Mounting a multiprocessor volume is signified by the presence of SYSID in the mount request. The format for the mount request of a multiprocessor volume is:

MOUNT volname ON devmnc SYSID= [MPn / DPx]

n is zero through F
x is zero or one

Multiprocessor volumes can be mounted as public or nonpublic. However, the system volume cannot be mounted as a multiprocessor volume.

When a multiprocessor volume is mounted, J.MOUNT prompts the operator for permission to perform volume cleanup if the volume descriptor indicates the volume was not previously dismounted. When volume cleanup is performed, no regard is given to access from any other port. All resource descriptors are purged of any software multiprocessor information. Therefore, it is the responsibility of the operator to ensure the integrity of the mount process from all system environments prior to allowing volume cleanup.

If the operator indicates volume cleanup is not to be performed, and J.MOUNT detects the port associated with the SYSID specification has not been previously dismounted, the following message is displayed on the system console:

```
J.MOUNT - WARNING - VOLUME SHOWS PORT DESIGNATOR MP(DP)n ALREADY ALLOCATED
J.MOUNT - REPLY C TO CONTINUE, A TO ABORT:
```

4.13.4 Multiprocessor Resource Restrictions

Some features of volume management provided by MPX-32 are restricted when applied to multiprocessor resources. This is a result of the additional system overhead associated with the processing of these resources. The following sections describe some of the more significant restrictions and potential conflicts that can apply when resources are shared concurrently from separate system environments.

4.13.4.1 EOF Management

Dynamic End-Of-File (EOF) information is not available to tasks sharing a multiprocessor resource from separate system environments. The updated EOF information is not available until the writer closes the resource and the reader reallocates the file. However, tasks sharing multiprocessor resources within the same system environment have access to the full range of EOF management allowed to nonmultiprocessor resources. In this context, writer means any task accessing an implicity or explicitly shared resource in write, update or append access mode.

4.13.4.2 EOM Management

Dynamic End-of-Medium (EOM) information is not available to tasks sharing a multiprocessor resource from separate system environments. The updated EOM information is not available until the extender has completed the service, and the other task has reallocated the file. Tasks sharing multiprocessor resources within the same system environment have access to the full range of EOM management allowed to nonmultiprocessor resources. In this context, extender means any task accessing an implicitly or explicitly shared resource in update or append access mode, as well as any task requesting a manual extension of an extendible file.

4.13.4.3 Resource Deadlocks

When a task obtains exclusive use of a resource in such a way that it will not (or cannot) release it, any task waiting to gain access to that resource is indefinitely postponed. This situation results whenever a system failure occurs while the multiprocessor RD lock has been set for a task accessing a multiprocessor resource from that system. This usually is the situation when a task (attempting to gain access to a multiprocessor

resource in the operational system) appears to be cycling between a suspended and ready-to-run state for an extended period of time.

Under these circumstances, the multiprocessor lock remains in effect until the volume is remounted. This multiprocessor lock can be removed by using the OPCOM UNLOCK directive from the operational system. Once J.UNLOCK has completed, the volume can be remounted from the failed system, but volume cleanup must be inhibited.

4.13.4.4 Reserve/Release Multiported Disc Services (M.RESP/M.RELP)

The multiprocessor features of MPX-32 do not use the reserve and release multiported disc services. The M.RESP service causes the drive to be exclusively reserved to the system environment from which the request was issued. The disc remains reserved until explicitly released by the M.RELP service. If a volume is mounted on the multiported disc drive at the time of a reserve request, it is inaccessible from the other system environment until explicitly released.

MPX-32 performs an implicit reserve of the multiported disc that remains effective for the duration of the IOCL used to set or release the software lock in the appropriate resource descriptor.

The M.RESP and M.RELP services should not be used with the multiprocessor features of MPX-32, or the result can be unpredictable system behavior.

4.13.4.5 Volume Status

The information displayed by OPCOM in regard to currently available space on a volume status inquiry is not accurate for volumes mounted as multiprocessor. OPCOM obtains this information from a resident system structure rather than the volume space map. Therefore, it does not reflect any volume space allocated or deallocated from a separate system environment. The status displayed by OPCOM on these volumes only reflects space changes initiated from within the system environment making the inquiry.

This restriction only applies to the status displayed by OPCOM. Volume space is managed by the volume space map which is treated as a multiprocessor resource by the system envionments and always reflects the accurate state of the volume space.

4.13.5 Optimum Use of Multiprocessor Resources

As a result of the restrictions imposed on multiprocessor resources, the following steps can be taken on resources shared by tasks in separate system environments:

1. Do not rely on dynamic EOF management. When creating multiprocessor files, specify EOF management is not in effect. For example:

   ```
   CREATE F filename SIZE=nn EOFM=N
   ```

   This sets EOF to the size of the file (EOM). In this way, concurrent access by multiple tasks do not result in EOF detection until the physical EOF is reached.

2. Have the sharing tasks assign the file for explicit shared use. For example:

   (Task 1) ASSIGN LFC TO filename ACCESS=(R) SHARED=Y
   and
   (Task 2) ASSIGN LFC TO filename ACCESS=(W) SHARED=Y

   This allows concurrent access to the file by the reader and writer. Synchronization can then be performed by the record structure in the file (which results in less I/O overhead) or through the synchronous resource lock services provided by MPX-32.

3. To avoid unnecessary I/O overhead associated with multiprocessor resources do not direct the creation of temporary or swap files on a multiprocessor volume unless absolutely necessary. If the current working volume is a multiprocessor volume, then additional I/O overhead is associated with processing SLO, SBO, and SGO files on the volume.

4. Whenever a system failure occurs in one system, it is recommended that J.UNLOCK is activated from the running system. After J.UNLOCK completes, the failed system can be rebooted. With volume cleanup inhibited, remount the volume from the port where the system failure occurred.

5. Mount PUBLIC rather than NONPUBLIC.

CHAPTER 5

RESOURCE ASSIGNMENT/ALLOCATION AND I/O

This chapter provides an overview of the user interfaces provided for the assignment/allocation of resources to a task and the subsequent I/O services available. It assumes the reader is familiar with the terms and concepts presented in Chapter 4.

The MPX-32 Resource Management Module (H.REMM) performs all operations necessary to obtain the physical resources required to execute a task. The MPX-32 I/O control system (IOCS) receives and processes device-independent I/O requests from both user tasks and the MPX-32 operating system.

5.1 Introduction

The following sections discuss the MPX-32 concepts of logical I/O, wait I/O, no-wait I/O, device-dependent I/O, error processing and status posting conventions. Major I/O concepts in the context of the interface between IOCS, standard device handlers and the operating system are also presented. Finally, the handling of special system files and how to set up a File Control Block (FCB) and Type Control Parameter Block (TCPB) is covered along with a description of the I/O services available to users of MPX-32.

5.2 MPX-32 Logical Device-independent I/O

MPX-32 provides the user with versatile logical device-independent I/O capabilities. The user can code references to logical files and request an MPX-32 Input/Output Control System to perform I/O.

Several important advantages are gained by performing logical file I/O:

- The user need not be aware of specific device handling requirements
- Unprivileged tasks can perform I/O (the I/O instructions are part of the privileged instruction set)
- Tasks that perform logical I/O are easier to debug and modify

To provide MPX-32 with sufficient information to create the necessary linkages between the user's logical files and the actual peripheral devices or disc files, the user must:

- Identify logical files with logical file codes (LFCs)
- Describe logical file attributes with File Control Blocks (FCBs)
- Associate logical files with their target physical devices or disc files with logical file code assignments

5.2.1 Logical File Codes

Logical file codes (LFCs) are user defined one- to three-character ASCII codes that identify logical files within tasks.

Logical file codes are configured into corresponding File Control Blocks (FCBs). See Section 5.9 for FCB format.

5.2.2 File Control Blocks

A File Control Block (FCB) must be set up by the user to describe each logical file within a task, and to describe certain attributes of each logical I/O operation.

Certain information collected by IOCS following each I/O operation is made available to the user by the corresponding FCB. Certain space in the FCB is reserved for use by IOCS.

The FCB format is decribed in detail in Section 5.9.

5.2.2.1 Logical I/O Initiation

To initiate a logical I/O operation, the user must code into his task a call to one of the data transfer or device access services accompanied by the address of a corresponding FCB.

5.2.3 Assignment vs. Allocation

The attachment of a task to a resource progresses through two phases: assignment and allocation. The current phase of a particular resource attachment depends on the amount of information supplied by the requestor up to that time.

Assignment is the process of associating a logical file code (LFC) with a system resource. This action informs the system of a task's intention to use a resource, but does not describe the usage (exclusive use, explicit shared, implicit shared) or the intended access mode (read, write, modify, update or append). Hence, the resource is still susceptible to allocation by other tasks, and no guarantee is made that the assigning task can obtain the resource in any specific usage or access mode.

Allocation is the process of securing a resource for a specific usage and access mode for the requesting task. At this point, the task has defined all of its intentions and can perform logical I/O operations on the resource for the usage and access requested.

5.2.3.1 Determination of Resource Allocation

When an LFC is assigned to a system resource, the task can indicate the mode in which it intends to use the resource (exclusive or explicit shared). If this is done, the assignment becomes an allocation because these usage modes imply that the task is allowed any access mode authorized to it by the resource creator. The task is guaranteed access to the resource when logical I/O is initiated. If a usage mode is not indicated at LFC assignment, implicit shared use of the resource is assumed by default.

When an LFC is attached to a resource with implicit shared use, the resource is not allocated until a specific access mode is indicated. If this occurs at LFC assignment, the assignment becomes an allocation or is deferred until the resource is opened. In the latter case, the resource is not allocated until it is opened, and there is no guarantee the specific access mode is obtained because other tasks may have allocated the resource for implicit shared use in an incompatible access mode.

5.2.3.2 Assign/Open Resource Allocation Matrix

Specified Usage At Assign	Specified Usage At Open	Point At Which Access Is Specified	Is Resource Allocated At Assign	Allocation Action At Open
Exclusive Shared Use	Exclusive Shared Use	N/A	Yes	None
Exclusive Shared Use	Explicit Shared Use	N/A	Yes	Reallocate and Dequeue
Exclusive Shared Use	Implicit Shared Use	N/A	Yes	None
Explicit Shared Use	Exclusive Shared Use	N/A	Yes	Deallocate and Allocate*
Explicit Shared Use	Explicit Shared Use	N/A	Yes	None
Explicit Shared Use	Implicit Shared Use	N/A	Yes	None
Implicit Shared Use	Exclusive Shared Use	Assign	Yes	Deallocate and Allocate*
Implicit Shared Use	Exclusive Shared Use	Open	No	Allocate*
Implicit Shared Use	Exclusive Shared Use	Assign	Yes	Deallocate and Allocate*
Implicit Shared Use	Exclusive Shared Use	Open	No	Allocate*
Implicit Shared Use	Implicit Shared Use	Assign	Yes	None
Implicit Shared Use	Implicit Shared Use	Open	No	Allocate*

* No guarantee that a specific access mode (read, write, modify, update, append) is available at open.

5.2.4 Logical File Code Assignment

Before executing a logical I/O request, the task must associate the appropriate logical file code (LFC) with the target peripheral device or disc file. This is accomplished by logical file code assignment.

At this time, the requestor can specify one or more access modes that apply to this assignment. This set of access modes defines the set of allowable I/O operations that can be performed on the resource for the duration of this assignment (read, write, modify, update, and append). The access modes specified at assignment must not allow more access than that allowed to this user by the resource creator or the assignment will be denied. If no access modes are specified, the default access modes in the resource descriptor for this user class are allowed.

A specific resource usage (exclusive or explicit shared) may also be declared at LFC assignment. If not supplied, the resource is assumed to be assigned for implicit shared use. In this case, the resource is not allocated to the requesting task at assignment, unless only one access mode is allowed.

Logical file codes can be assigned to specific peripheral devices or files when a task is cataloged (static assignment) or during task execution by the M.ASSN service (dynamic assignment).

For tasks that run under TSM control (interactive or batch oriented), static assignments can also be made by the user at run time. For such assignments, if the logical file code matches one assigned at catalog time, it replaces the cataloged assignment. If the file code assigned at run time does not match any cataloged assignment, it is added to the cataloged assignments.

Dynamic assignments cannot override cataloged or run-time assignments, and any attempt to do so is treated as an error. To accomplish dynamic override, the user task must first deallocate (deassign) the static assignment by the M.DASN service.

The maximum number of assignments is 245. There is additional space reserved for assignments needed by the operating system.

5.2.4.1 Making Assignments via Resource Requirement Summary (RRS)

The Resource Requirement Summary (RRS) is a structure that defines the assignment requirements of a resource to the Resource Management Module (H.REMM). It is supplied by the Cataloger, LINKX32, or TSM for static assignment of resources to a task or as an argument for the dynamic assignment of a particular resource. There are distinct types of RRS entries recognized by REMM corresponding to the resource modes and allocation mechanisms available. RRS entries are variable length structures with the first 4 words generally common for all entries, and the remaining number of words dependent on the RRS type. RRS entries always begin on a doubleword boundary. They must contain an even number of words for static assignments made by parameter task activation or load module activation where pathnames are applied. RRS entries are presented to H.REMM in the following formats.

Unless specified, the first four words of an RRS entry contain the following:

Word 0	Byte 0	cleared, and a one- to three-character, left-justified, blank-filled logical file code (LFC) in bytes 1,2, and 3

Word 1 Byte 0 specifies the RRS type with the following va significance:

Value	Description
1	Assign by pathname (RR.PATH)
2	Assign to temporary file (RR.TEMP)
3	Assign to device (RR.DEVC)
4	Assign to LFC (RR.LFC2)
5	Assign by segment definition (RR.SPACE)
6	Assign by resource ID (RR.RID)
7	Reserved for future use
8	Reserved for future use
9	Mount by device mnemonic (RR.MTDEV)
10	Assign to ANSI labeled tape (RR.ANS)
11	Assign to shadow memory (R.R.SHRQ)
12-255	Reserved

Byte 1 specifies the size of this RRS entry in words

Bytes 2 and 3 vary depending on the RRS type

Restriction: A value must be specified in the RRS type field. There are no defaults applied to this portion of the RRS.

Word 2 Access specification field specifies the access restrictions to be applied to the allocation of this resource. The bit interpretations are as follows:

Bit	Meaning if Set
0	Allow read access (RR.READ)
1	Allow write access (RR.WRITE)
2	Allow modify access (RR.MODFY)
3	Allow update access (RR.UPDAT)
4	Allow append access (RR.APPND)
5-15	Reserved
16	Explicit shared use requested (RR.SHAR)
17	Exclusive use requested (RR.EXCL)
18	Assign as volume mount device (R.R.MNT)

Restrictions:

1. The bit pattern specified in bits 0 to 4 must not allow more access than specified in the resource descriptor for this user.

2. Only one of bits 16 to 17 can be set to indicate the intended usage mode. Successful allocation of a resource for exclusive use implies the setting of an exclusive resource lock on that resource.

Defaults:

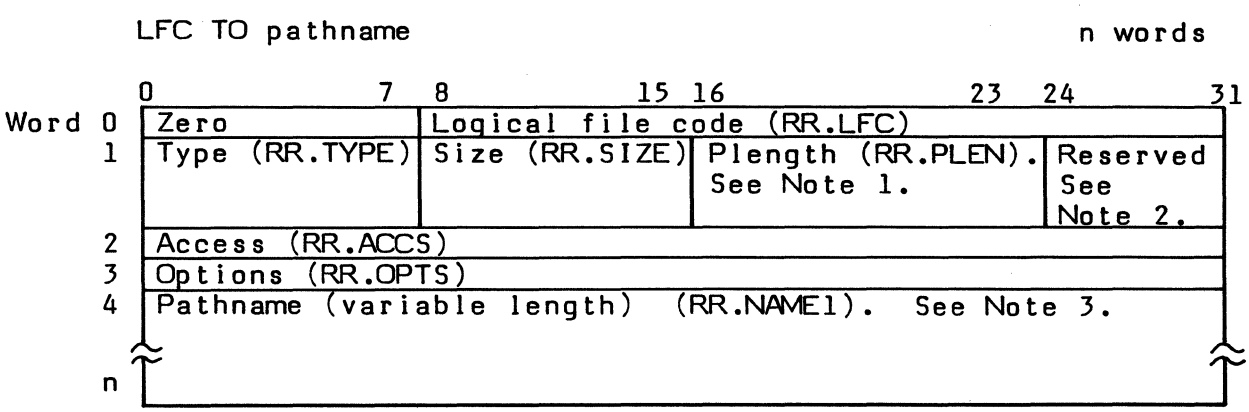
If the access specification field is zero, the default access contained in the resource descriptor for this user is used, and the resource is allocated for implicit shared use.

Word 3 Options specification field specifies the allocation options that are in effect for this assignment. The bit interpretations are as follows:

Bit	Meaning if Set
0	Treat as SYC file (TSM/JOB only) (RR.SYC)
1	Treat as SGO file (TSM/JOB only) (RR.SGO)
2	Treat as SLO file (RR.SLO)
3	Treat as SBO file (RR.SBO)
4	Explicit blocked I/O (RR.BLK)
5	Explicit unblocked I/O (RR.UNBLK)
6	Inhibit mount message (RR.NOMSG)
7	Reserved for system use
8	Automatic open requested (RR.OPEN)
9	User buffer address to be supplied at open (RR.BUFF)
10-11	Reserved for system use
12	Mount with no-wait (RR.NOWT)
13	Mount as a public volume (RR.PUBLC)
14	By H.VOMM for special case handling of VOMM assignments (RR.VOMM)
15	Spool file when deallocated (RR.SEP)
16	Mount as ANSI tape (RR.ANSI)
17-31	Reserved

Word 4-n Is RRS type dependent.

Type 1 (Assign by pathname)



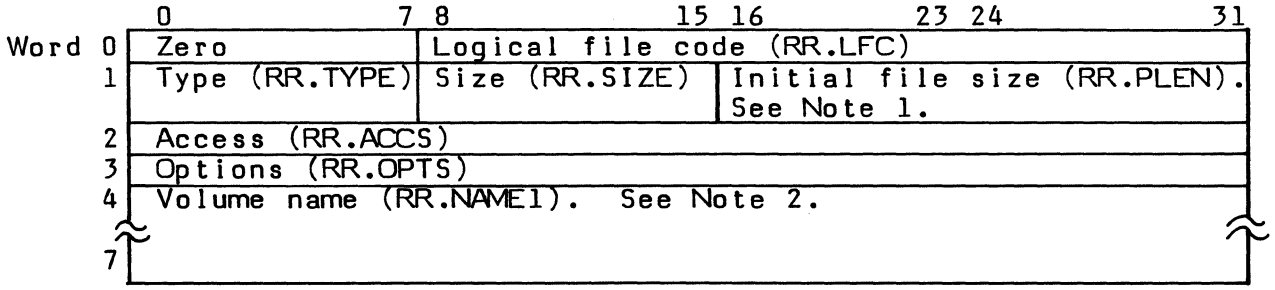
Notes:

1. RR.PLEN contains character count of pathname/pathname block
2. Byte 3 is zero. This field is used by MPX-32 for big blocking buffers.
3. RR.NAME1 is the resource pathname or pathname block (PNB). See Section 5.14.1 for format.

Type 2 (Assign to temporary file)

LFC TO TEMP [=(volname)] 4-8 words

Temporary files created in this manner cannot be made permanent unless they are created on a volume which has a valid directory established for this user.

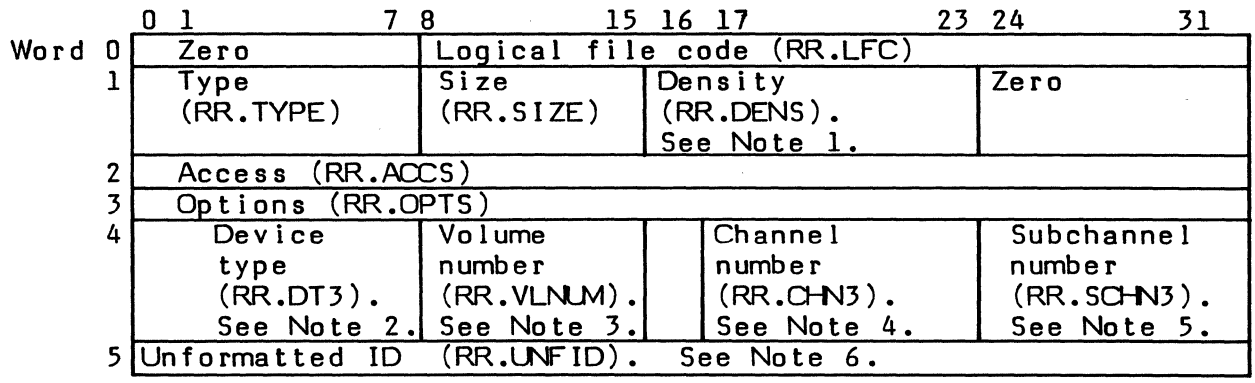


Notes:

1. RR.PLEN contains the initial file size in logical blocks. If a size is not supplied, the system default is used. See Section 5.2.4.2.
2. RR.NAME is a 1- to 16-character left-justified, blank-filled volume name. This element of the RRS entry is optional. If supplied, the temporary file will be created on the specified volume. Otherwise the file will be created on the task's current working volume or any available public volume.

Type 3 (Assign to device)

LFC TO DEV=devmnc 6 words



Notes:

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

Bit	Meaning if Set
0	Indicates 800 bpi Nonreturn to Zero Inverted (NRZI)
1	Indicates 1600 bpi Phase Encoded (PE)
6	Indicates 6250 bpi Group Coded Recording (GCR)

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

2. RR.DT3 contains the device type code associated with this device. See Table 5-2. The device type code is the only required portion of this word. If a channel or subchannel is supplied, bits 0 and 16 are set respectively to indicate the presence of these portions of the device address. The Get Device Type Code (M.DEVID) service can be used to obtain the correct value for byte 0 from the appropriate device mnemonic.

3. RR.VLUM contains the volume number for multivolume media.

4. RR.CHN3 contains the logical channel number associated with this device.

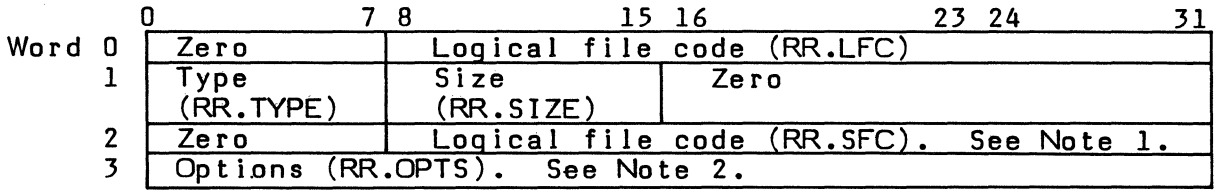
5. RR.SCHN3 contains the logical subchannel to be applied with the logical channel number.

6. RR.UNFID is a one- to four-character identifier to be associated with an unformatted media (magnetic tape, disc, or floppy disc). If nonzero, this name appears on the mount/dismount messages; otherwise, SCRA is used as the default identifier.

Type 4 (Assign to LFC)

LFC TO LFC=lfc — 4 words

An LFC to LFC assignment inherits all the access and assignment restrictions specified by the original LFC assignment. The allowable access modes and blocking status cannot be changed by assigning a second LFC to a resource.



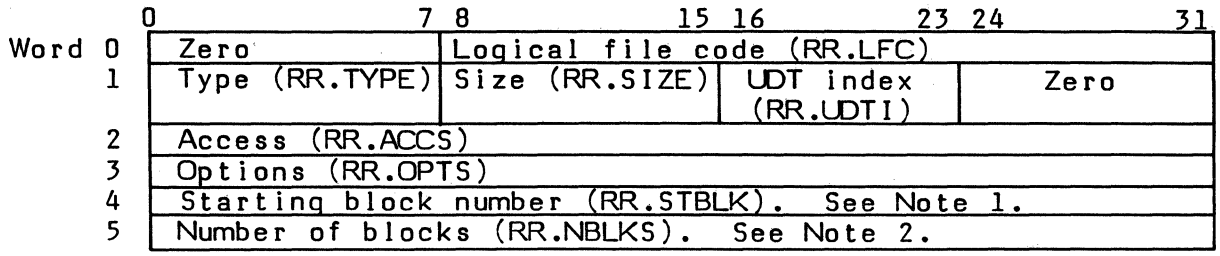
Notes:

1. RR.SFC contains a one- to three character, left-justified, blank-filled logical file code. This LFC must have been previously assigned to a resource.

2. RR.OPTS is the options specification field. Automatic OPEN may be selected by specifying the option in this field.

Type 5 (Assign by segment definition)

Dynamic allocation only — 6 words



Notes:

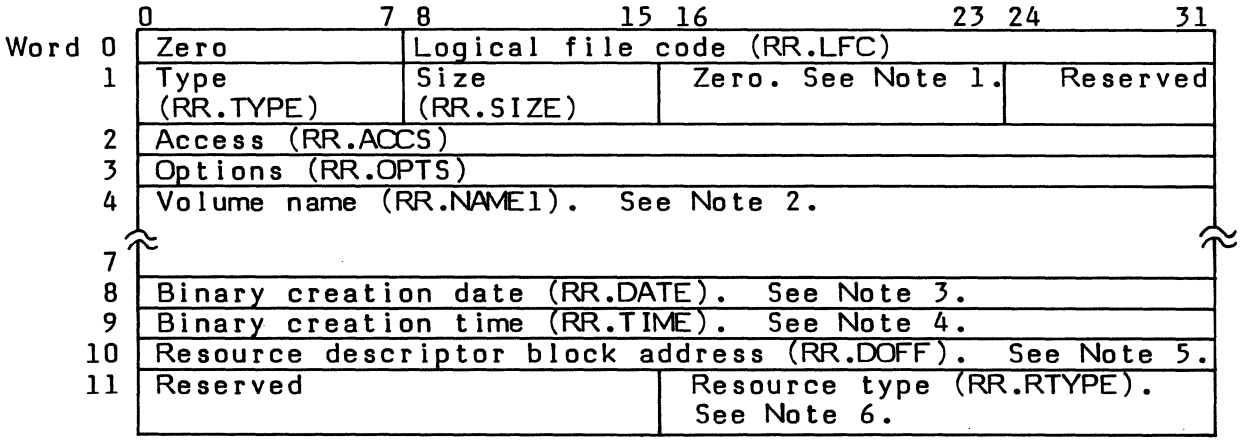
1. RR.STBLK is the starting block address of segment definition.

2. RR.NBLKS is the number of contiguous blocks in definition. This type of RRS is only valid for a contiguous volume resource. Extendible files must be allocated by other means.

Restriction: Allocation by segment definition is a privileged assignment mode. An error condition is generated if this form of allocation is attempted by an unprivileged task.

Type 6 (Assign by resource ID)

LFC TO RID=(resid) 12 words



Notes:

1. Word 1, byte 2 is zero. This field is used by MPX-32 for big blocking buffers.
2. RR.NAME1 is a 1- to 16-character, left-justified, blank-filled volume name.
3. RR.DATE is the binary creation date.
4. RR.TIME is the binary creation time.
5. RR.DOFF is the block address of resource descriptor.
6. RR.RTYPE is the resource type value (right-justified).

Type 7 Reserved for Future Use

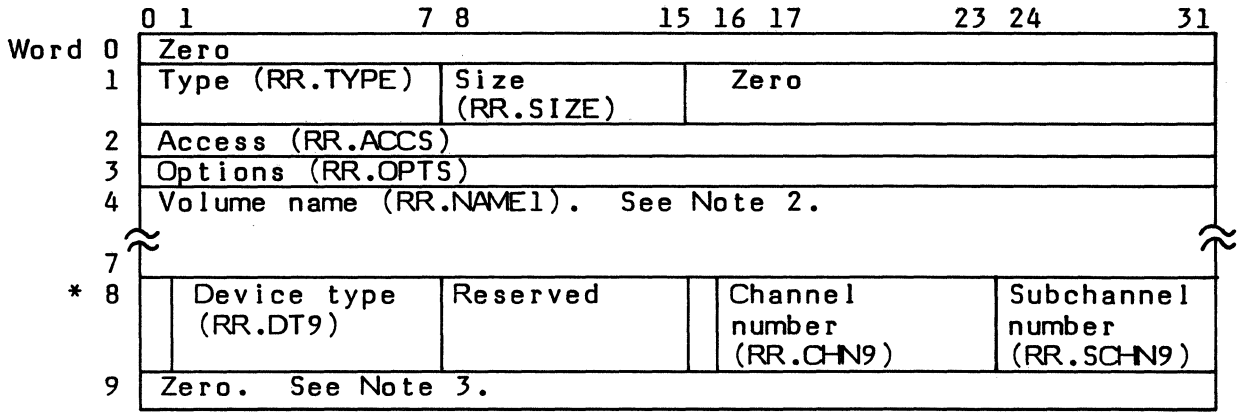
Type 8 Reserved for Future Use

Type 9 (Mount by device mnemonic)

MOUNT volname ON devmnc 10 words

If the public volume option is selected, the volume is mounted for public use and remains mounted until the system is halted.

For a description of the dynamic assignment/allocation services, M.ASSN and M.DASN, see Chapter 6.



Notes:

1. Word 0 is zero, or a three character SYSID for dual-port volumes.

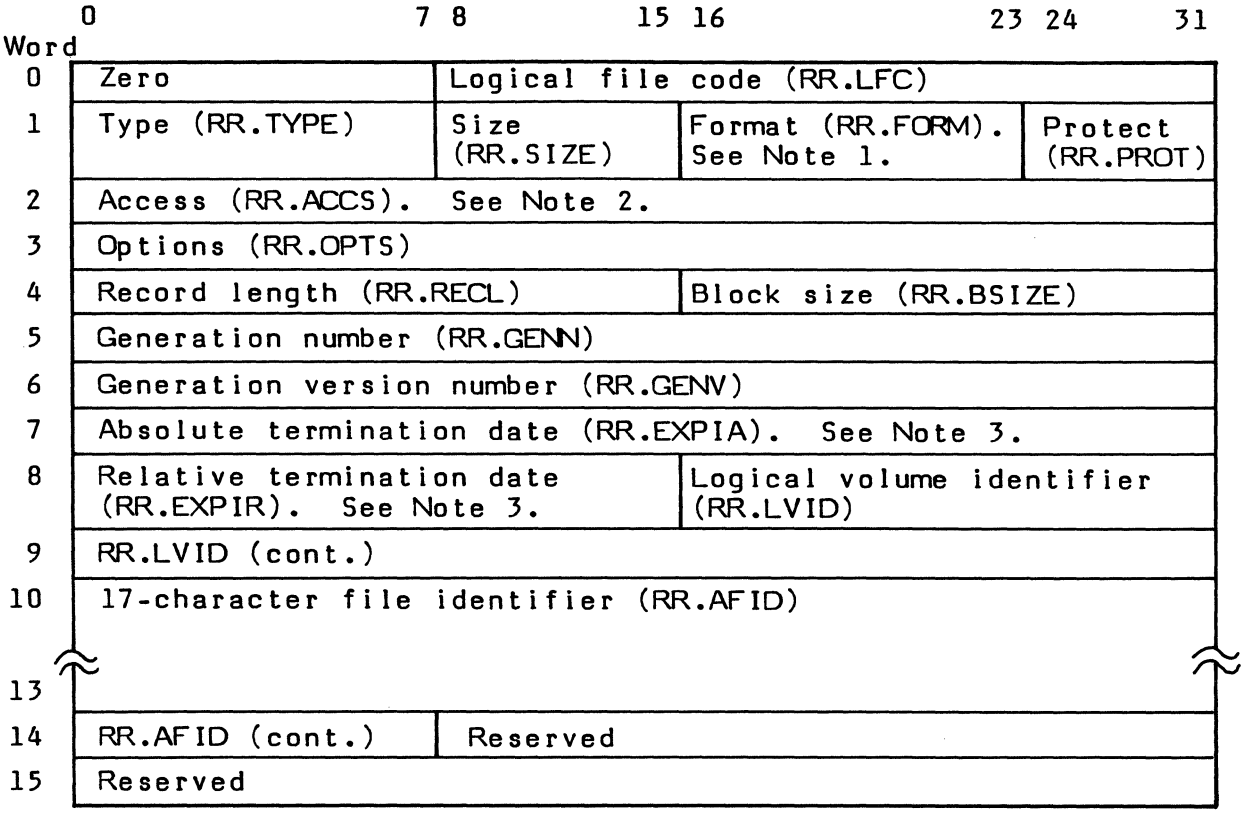
2. RR.NAME1 is a 1- to 16-character, left-justified, blank-filled volume name.

3. Word 9 is zero (reserved for system use).

*Word 8 is the Device specification word. Format same as for type 3, word 4. This element describes the device where the volume is to be mounted. If a complete address is specified, an attempt is made to mount the volume on that device only. Otherwise, the volume is mounted on any device that matches the portion of the specification word supplied in the RRS.

Type 10 (Assign to ANSI tape)

LFC TO @ANSITAPE (lvid) filename



Notes:

1. RR.FORM values are as follows:

Value	Meaning
Numeric 0	Use default format
ASCII F	Fixed length record format
ASCII D	Variable length record format
ASCII S	Spanned record format

2. RR.ACCS accesses a resource in the read, write, update, or append mode.

3. For the absolute termination date, word 7 is one blank followed by a two-digit number that specifies the year and the first digit of the 3-digit day. Word 8, bytes 0 and 1 contain the remaining two digits of the day.

   For the relative termination date, word 7 is zeroed; word 8, bytes 0 and 1 contain the binary relative termination date.

Type 11 (Assign to shadow memory)

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

Notes:

1. RR.SHAD contains the shadow flags that qualify the start and end addresses, or specify that the entire task is to be shadowed. The bit interpretations are as follows:

Bit	Meaning if Set
0-9	Reserved
10	Shadow the stack (RR.SHST)
11	Shadow memory is required (RR.SHRQ)
12	Shadow the entire task (RR.SHALL)
13	Absolute address (RR.ABS)
14	Relative to the code section origin (RR.CREL)
15	Relative to the data section origin (RR.DREL)

2. RR.SADD is the starting logical byte address of the memory space to be put in shadow memory. The address is considered relative to the start of the task (T.BIAS), unless qualified with shadow flags in word 1.

3. RR.EADD is the ending logical byte address of the memory space to be put in shadow memory. The address is considered to be relative to the start of the task (T.BIAS), unless qualified with shadow flags in word 1.

The start and end addresses are specified as a byte address in the TSM SHADOW directive and the RRS. The range of address space specified is inclusive. Internal to MPX-32, these addresses are map block bounded. Due to the map block granularity of 2KW, more address space than requested can be shadowed, but never less.

5.2.4.2 Temporary File Assignments

Temporary files are created explicitly by the Create Temporary File (M.TEMP) service and assigned by the resource identifier (RID) obtained when the file was created.

Temporary files are implicitly created as the result of an LFC assignment, such as

```
ASSIGN OUT TO TEMP=(volume)
```

When a temporary file is created in this manner, the resource is given the following characteristics:

- All access modes are allowed
- The file is shareable
- The file is automatically and manually extendible
- The initial file size is 16 blocks, unless a size is specified in the RRS
- File extensions are made in a minimum of 32 block increments

If blocked or unblocked is not specified as an assignment option to a temporary file, all I/O is blocked by default.

5.2.5 Opening a Resource for Logical I/O

Before any service requested by the user to initiate an I/O operation is completed, the user's logical file must be opened. This operation establishes the access mode for subsequent I/O operations. The specified access mode must have been allowed by the resource assignment or an error occurs.

If an access mode is not specified, the resource is opened for the appropriate default access, unless only a specific access mode was allowed at assignment. In that case, the resource is opened for the allowed access mode. The default access modes are read access for files and update access for peripheral device assignments.

A resource usage mode, exclusively or explicitly shared, is also specified at open. When this is done, the supplied usage overrides that specified at LFC assignment. This results in the attachment to the resource reverting to an assignment, if allocated. If the LFC assignment allocated the resource for a shareable usage, there is no guarantee that the resource is available for reallocation when the usage specification is overridden at open. A previously allocated resource can not be available at open in this case.

The task performs the function directly with the M.OPENR service, automatically during assignment, or automatically by IOCS when an I/O initiation service request is made and the logical file is not open. In the latter case, the default access mode in effect for that assignment applies.

When the logical file is properly opened, the FCB and FAT are linked together to complete the I/O structure for IOCS.

5.3 Resource Conflicts and Error Handling by Caller Notification Packet (CNP)

In the course of processing resource assignments, conditions exist that require additional information from the task requesting the resource. MPX-32 allows the user the option of waiting for resources that are temporarily not available. The requesting task can also dictate the manner in which error or denial status is presented.

The Caller Notification Packet (CNP) is the mechanism used by the Resource Management Module (H.REMM) and the Volume Management Module (H.VOMM) for handling abnormal conditions that result during resource requests. This structure is a standardized parameter optionally supplied to many of the services provided by REMM and VOMM.

The CNP consists of a five- to six-word area containing the following information, all or part may be used by the particular service being called:

Word 0 Wait request time-out value interpreted as follows:

Value	Description
>0	Return immediately with a denial code if the service cannot be completed
0	Place the requesting task in a wait state until the designated service can be completed
-n	Place the requesting task in a wait state until the designated service can be completed or until the expiration of n timer units, whichever occurs first

Word 1 Abnormal return address - if an error condition or denial condition occurs, control is transferred back to the task at this address

Word 2 Options field (bytes 0 and 1) - A bit sequence and/or value used to provide additional information that is necessary to fully define the calling sequence for a particular service

Status field (bytes 2 and 3) - A right-justified, numeric value identifying the return status for this call

Word 3 Reserved for future use

Word 4 Reserved for future use

Word 5 Parameter link - Required only for a resource assignment where automatic open has been specified in the option word of the Resource Requirement Summary (RRS). If specified, this word must contain the address of a valid File Control Block (FCB) for this assignment.

5.3.1 Status Posting and Return Conventions

The services that accept a CNP as a calling parameter adhere to a common status posting and return convention. Status is always posted as a numeric value that represents one of three conditions:

- Successful - CC1 not set indicates the service completed without any abnormalities.
- Error - Indicates a condition occurred during execution that precluded any continuation of the service. This condition has a positive numeric value identifying an error condition specific to the service.
- Denial - Indicates the service was not completed due to resource nonavailability or other system constraints which do not necessarily preclude the continuation of the service at a later time. This condition is also represented by a numeric value specific to the service, but is posted only if the caller has indicated that the task is not to wait for the condition to be alleviated. Denial conditions are returned only by the Resource Management Module (H.REMM).

The following conventions apply to the posting of status and the return sequence applied to service calls that accept a CNP as an input parameter:

- CC1 is set to indicate the posting of a nonzero status value
- The task is never aborted as the result of posting a status value
- The location of status and return sequence for denials and errors is dependent on the presence or absence of a Caller Notification Packet (CNP):

  If CNP is supplied :

  - Status is posted in the status field of the CNP
  - For all error conditions, the return is made by the abnormal return address if supplied. Otherwise, a normal return occurs.
  - For all denial conditions, the task is enqueued if requested; otherwise, the return sequence is the same as an error condition.

  If CNP is not supplied:

  - Status is posted in register seven
  - For all error conditions, a normal return occurs
  - For all denial conditions, the task is enqueued until the service can be completed

The following is a summary of the status codes returned by the Resource Management Module (H.REMM) for resource allocation:

Value	Description
0	Successful completion
1	Unable to locate resource (invalid pathname or memory partition definition)
2	Specified access mode not allowed
3	FPT/FAT space not available
4	Blocking buffer space not available
5	Shared Memory Table (SMT) entry not found
6	Volume Assignment Table (VAT) space not available
7	Static assignment to dynamic common
8	Unrecoverable I/O error to volume
9	Invalid usage specification
10	Dynamic partition definition exceeds memory limitations
11	Invalid Resource Requirement Summary (RRS) entry
12	LFC logically equated to an unassigned LFC
13	Assigned device not in system
14	Resource already allocated by requesting task
15	SGO or SYC assignment by real-time task
16	Common memory conflicts with task's address space
17	Duplicate LFC assignment attempted

18	Invalid device specification
19	Invalid Resource ID (RID)
20	Specified volume not mounted
21	J.MOUNT run request failed
22	Resource is marked for deletion
23	Assigned device is marked off-line
24	Segment definition allocation by unprivileged task
25	Random access not allowed for this access mode
26	User attempting to open SYC file in a write mode
27	Resource already opened by this task in a different access mode
28	Invalid access specification at open
29	Specified LFC is not assigned to a resource for this task
30	Invalid allocation index
31	Close request issued for an unopened resource
32	Attempt to release an exclusive resource lock that was not owned by this task, or a synchronous lock that was not set
33	Attempt to release an exclusive resource lock on a resource that has been allocated for exclusive use
34	Attempt to mount a public volume without the System Administrator attribute
35	Attempt to exclude memory partition that is not mapped into requesting task's address space
36	Reserved
37	Invalid J.MOUNT request
38	Time out occurred while waiting for resource to become available
39	Reserved
44	Writeback requested and shared image has no writeback section
45	Loading error during inclusion of read-only section of shared image
46	Unable to obtain resource descriptor lock (multiport only)
48	Incompatible load address for shared image
49	Excessive multicopied shared images with no read only section

Status codes in the range 50 to 63 represent denial conditions that result in suspension of the task if a CNP is not supplied or enqueue is indicated by the time-out value in the CNP:

Value	Description
50	Resource is locked by another task
51	Shareable resource is allocated by another task in an incompatible access mode
52	Volume space is not available
53	Assigned device is not available
54	Unable to allocate resource for specified usage
55	Allocated Resource Table (ART) space is not available
56	Reserved
57	Volume is not available for mount with requested usage
58	Shared Memory Table (SMT) space is not available
59	Mounted Volume Table (MVT) space is not available
60-79	Reserved
80	Shared image version level is not compatible with executable image
81-255	Reserved

5.4 MPX-32 Volume Resource Access

The MPX-32 Volume Management Module (H.VOMM) maintains a Resource Descriptor (RD) list on disc for each mounted volume in the system. A descriptor exists for every currently active volume resource (permanent files, temporary files, directories and memory partitions). Each resource descriptor contains all the access, accounting, and space definition information pertaining to the associated resource.

When a volume resource is assigned for I/O, the MPX-32 Resource Management Module (H.REMM) reads the associated resource descriptor. The usage and access specifications supplied at assignment are verified against those allowed by the resource descriptor for protection and resource integrity. If a discrepancy exists, the task is aborted (static assignment) or the appropriate error code is returned (dynamic assignment). In either case, the assignment is denied.

The description of a valid volume resource is placed in the File Assignment Table (FAT). IOCS uses the FAT entry to map the appropriate disc and file. Access to disc files is sequential, starting with the first relative block in the file, unless the user has indicated random access in the FCB. However, random access is not allowed in the write and append access modes.

Permanent files are accessed by pathname or resource identifier (an eight-word unique identifier obtained when the resource was created). Temporary files are accessed by resource identifier only.

5.4.1 Volume Resource Space Management

The MPX-32 Volume Management Module (H.VOMM) provides space management for all currently mounted volumes in the system. Disc space is allocated to files from the available space not dedicated to volume management, an area whose size is determined when the volume is formatted. Volume space is allocated from this area of the disc in units, each consisting of an integral number of 192 word blocks. See Section 5.14.1.5.

5.4.2 Temporary vs. Permanent Files

Temporary files have all the access and protection attributes associated with permanent files. However, temporary files are automatically deallocated, and their volume space is returned to the pool of available space when a task exits or aborts.

Temporary files created by the Create Temporary File (M.TEMP) service can be shared by tasks that received the associated Resource Identifier (RID) from the creating task. In this case, the volume space allocated to the temporary file is not deallocated until the last task assigned to it deallocates the file or exits.

Temporary files are made permanent if they are created on a specific volume in a valid directory for the user making the request.

Permanent files and memory partitions remain defined on the volume until they are explicitly deleted by a user who has access to them.

5.4.3 System Directory

The system directory is a special directory that resides on the system volume. It is identified within the pathname structure by the keyword SYSTEM, a name reserved for MPX-32.

All executable images or load modules with the System Administrator attribute must reside in the system directory.

Any other files created on a volume can reside in the system directory if desired. However, no special significance is given to these files.

5.5 MPX-32 Device Access

When a device is assigned for I/O, the MPX-32 Resource Management Module verifies that the device is available, allocates blocking buffers required for blocked I/O to disc, magnetic tape, or floppy disc, and identifies the device for IOCS in the File Assignment Table (FAT). If a device is not available (not included in the SYSGEN configuration of a system, off-line, etc.), the Resource Management Module aborts the task (static assignment) or returns an error code (dynamic assignment).

Throughout the reference manual, the generic descriptor devmnc indicates that a device can be specified.

Under MPX-32, device addresses are specified using a combination of three levels of identification: device type, device channel/controller address, and device address/subaddress.

Using the generic device type only results in allocation of the first available device of the type requested.

Using the generic device type and specifying the channel/controller address results in allocation of the first available device of the type requested on the specified channel or controller.

Specifying the device type, channel/controller, and device address/subaddress allocates a particular device.

5.5.1 Magnetic Tape

A multivolume magnetic tape is a set of one or more physical reels of magnetic tape processed as a continuous reel (255 maximum). Multivolume magnetic tape processing is supported under MPX-32 by the Multivolume Magnetic Tape Management Module (H.MVMT).

The multivolume magnetic tape can be used to transfer data to any MPX-32 Revision 2 or 3 system. It must be used when data to be transferred cannot fit on a single tape.

MPX-32 Source System Release*	MPX-32 Destination System Release**	Result
Pre-3.3	Pre-3.3	Possible data loss at end of tape
Pre-3.3	3.3 or later	H.MVMT treats magnetic tape as pre-3.3; possible data loss at end of tape.
3.3 or later	Pre-3.3	The pre-3.3 H.MVMT treats the magnetic tape as pre-3.3; possible data loss at end of tape.
3.3 or later	3.3 or later	H.MVMT detects the new header and expects a trailer.

* The source system generates the multivolume magnetic tape
** The destination system reads the multivolume magnetic tape

Figure 5-1. Multivolume Magnetic Tape Data Transfers Between Different Operating Systems

Multivolume tape reels have the following format:

192 Word header	Data	End of tape marker	Remaining data	End of file marker	192-word trailer

The 192-word header is a tape label produced as the first record for each volume of a multivolume file. The volume number and the reel identifier portions of this record are verified during subsequent read operations. The header has the following format:

Words	Contents
0	Reel ID (four-character ASCII entry)
1	Volume number 1 to 255 in binary format
2-3	Date (ASCII) in Gregorian format (MM/DD/YY)
4	Time (byte binary) as follows:

Byte	Meaning
0	Hour
1	Minutes
2	Seconds
3	Intervals (1/60th of a second)

5	Multivolume revision number (ASCII) (0001)
6-7	'HEADERØØ'
8-9	'FORØMULT'
10-11	'IVOLUMEØ'
12-13	'MAGNETIC'
14-15	'ØTAPESØF'
16-17	'ORØMPX-3'
18-19	'2ØØØØØØØ'
20-191	Reserved (zero)

The format for the trailer is as follows:

Words	Contents
0	Reel ID (four-character ASCII entry)
1	Volume number 1 to 255 in binary format
2-3	Date (ASCII) in Gregorian format (MM/DD/YY)
4*	Time (byte binary) as follows:

Byte	Meaning
0	Hour
1	Minutes
2	Seconds
3	Intervals (1/60th of a second)

5	Multivolume revision number (ASCII) (0001)
6-7	'TRAILERØ'
8-9	'FORØMULT'
10-11	'IVOLUMEØ'
12-13	'MAGNETIC'
14-15	'ØTAPESØF'
16-17	'ORØMPX-3'
18-19	'2ØØØØØØØ'
20-191	Reserved (zero)

* The date and time stored in the trailer is when the trailer was built; it indicates the date and time stored in the header and should be identical to the date and time of the header.

Multivolume magnetic tape processing mode is invoked whenever an assignment to a magnetic tape unit is encountered where multivolume is indicated by a volume number.

Multivolume processing is automatic for magnetic tape operations in the forward direction, such as read, write, advance, and erase. For reverse direction operations, such as rewind or backspace, the user is required to provide processing logic for proper reel manipulation and positioning requests.

Volume numbers in the range of 1 to 255 are provided as an operations aid in mounting and dismounting physical reels of magnetic tape. MPX-32 treats the volume numbers in circular fashion (for example, volume 1 follows volume 255 in the numbering scheme).

A scratch tape (SCRA) is not assigned if the multivolume mode is specified. Also, a unit applicable for multivolume magnetic tape operations cannot be designated a shared (SHR) device.

For multivolume magnetic tape processing, MPX-32 cannot pass end-of-medium (EOM) indicators to the user by the FCB. If a read, write, or advance operation is requested when tape is at end-of-medium, the system issues a dismount message to the console teletypewriter for the current volume (reel), followed by a mount request for the next sequential volume number before performing the requested operation.

Depending on the MPX-32 revision of the source svstem. read and write requests complete the following for a multivolume magnetic tape:

Source system*	Request	Result
Pre-3.3	Read	No check for expanded header
	Write	No expanded header or trailer exists
3.3 or later	Read at BOT	Checks for expanded header. If expanded header is detected, bit 2 of DFT.FLGS is set.
	Write	Writes expanded header and trailer
* The source system generates the multivolume magnetic tape.		

The system does not recognize multivolume mode specification when a magnetic tape is positioned at load point and a rewind or backspace operation is requested.

For Volume Manager restore operations, special processing occurs if the file being restored resides on two or more reels of a multivolume magnetic tape.

5.5.2 Unformatted Media

When a task is activated that has a device assignment to magnetic tape, disc, or floppy disc, the resource is treated as an unformatted medium. This implies that a medium must be mounted on the drive prior to the initiation of any I/O operations. The device is treated, in effect, as one continuous file for sequential I/O operations. Table 5-1 lists the physical dimensions of the medium for the various disc drives supported by MPX-32.

If the blocked or unblocked option is not selected at LFC assignment, all I/O to unformatted mediums is blocked by default.

When the resource is opened for I/O, a mount message is displayed on the operator's console. This message is inhibited, if desired, by the SYSGEN directive or assignment option. A mount message is not issued for assignments to shared devices. The format of the mount message for unformatted media is:

```
MOUNT reel VOL volume ON devmnc
TASK  taskname,taskno REPLY R,H,A or DEVICE:
jobno
```

reel — specifies a one- to four-character identifier for the reel. If not specified, the default is SCRA (Scratch).

volume — identifies the volume number to mount if multivolume tape

devmnc — is the device mnemonic for the tape unit selected in response to the assignment. If a specific channel and subaddress are supplied in the assignment, the specific tape drive is selected and named in the message; otherwise, a unit is selected by the system and its complete address is named in the message.

jobno — if the task is part of a batch job, identifies the job by job number

taskname — is the name of the task the unformatted medium is assigned to

taskno — is the task number assigned to this task by the system

DEVICE,R,A,H — the device listed in the message can be allocated and the task resumed (R), a different device can be selected (DEVICE), the task can be aborted (A), or the task can be held with the specified device deallocated (H). If an R response is given and a high speed XIO tape drive is being used, its density can be changed when the software select feature is enabled on the tape unit front panel. If specified, it overrides any specification made at assignment. Values are:

Value	Description
N or 800	Indicates 800 bpi Nonreturn to Zero Inverted (NRZI)
P or 1600	Indicates 1600 bpi Phase Encoded (PE)
G or 6250	Indicates 6250 bpi Group Coded Recording (GCR). Default.

Example usage: RN, R1600, etc.

Note: Do not insert blanks or commas.

Response:

To indicate that the drive specified in the mount message is ready and proceed with the task, mount the appropriate medium on the drive and type R (resume), optionally followed by a density specification if the drive is a high speed XIO tape unit. To abort the task, type A (abort). To hold the task and deallocate the specified device, type H (hold). The task is then resumed by the CONTINUE directive. At this time, a drive is selected by the system and the mount message is redisplayed.

To select a drive other than the drive specified in the message, enter the mnemonic of the drive you want to use. Any of the three levels of device identification can be used. The mount message is reissued. Mount the tape and type R (resume) if satisfactory, or, if not satisfactory type, abort, override, or hold as just described.

Table 5-1
Disc Description Table

SYSGEN Disc Code	Unit Size	Type	Sector Size	Sectors/ Track	Number of Heads	Max. Cylinders	Sectors/ Cylinder	Total Sectors	Max. Data Byte Capacity
Extended I/O Devices									
FL001	Floppy	Moving Head	256B*	26	2	77	52	4,004	1.02MB
FH005	5MB	Fixed Head**	192W	20	4	64	80	5,120	3.93MB
CD032	32MB	Cartridge Module	192W	20	1 + 1	823	40	32,920	25.28MB
MH040	40MB	Moving Head	192W	20	5	411	100	41,100	31.56MB
MH080	80MB	Moving Head	192W	20	5	823	100	82,300	63.20MB
MH160	160MB	Moving Head	192W	20	10	823	200	164,600	126.41MB
MH300	300MB	Moving Head	192W	20	19	823	380	312,740	240.15MB
MH340	340MB	Moving Head	192W	20	24	711	480	341,280	262.10MB
MH600	600MB	Moving Head	192W	20	40	843	800	674,400	517.94MB

* Smallest accessible block size = 192W
** Captive media, moving head disc

Sectors/Cylinder = (Sectors/Track) x (Number of Heads)
Total Sectors = (Sectors/Cylinder) x (Max. Cylinders)
Max. Data Byte Capacity = (Max. Sectors) x 768 bytes/sector

Note: The 32MB Cartridge Module Contains removeable and captive media within a single cabinet. Software treats the removeable and captive medias as separate devices, each with its own device subaddress. Figures represent total drive capacity.

5.5.3 Examples of Device Identification Levels

Examples of the three forms of device specification follow:

Form 1 - Generic Device Class

```
ASSIGN OUT TO DEV=M9 MULTIVOL=1 ID=SRCE
```

In this example, the device assigned to logical file code (LFC) OUT is any 9-track tape unit on any channel. The multivolume reel number is one. The reel identifier is SRCE and the tape is blocked.

Form 2 - Generic Device Class and Channel/Controller

```
ASSIGN OUT TO DEV=M910 ID=SRCE2 BLOCKED=N
```

In this example, the device assigned to logical file code (LFC) OUT is 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. The reel identifier is supplied. The tape is unblocked and is not multivolume.

Form 3 - Specific Device Request

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

In this example, the device assigned to logical file code (LFC) OUT is 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. The tape reel identifier is SCRA. The tape is blocked and is not multivolume.

5.5.4 GPMC Devices

GPMC/GPDC device specifications are similar to the general structure previously described. For instance, the terminal at subaddress 04 on GPMC 01 whose channel address is 20 would be identified as follows:

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

5.5.5 NULL Device

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

5.5.6 System Console

Logical file codes are assigned to the system console by using the device type CT.

5.5.7 Special File Attributes

There are two special file attributes that can be applied to resources at LFC assignment, print and punch. These attributes are supplied as options on the TSM $ASSIGN statement. For example:

```
$ASSIGN OUT TO OUTPUT PRINT                (permanent file)
$ASSIGN OUT TO TEMP=(volname) PRINT        (temporary file)
```

See Section 5.8 for a description of the handling of special file attributes.

5.6 Samples

A description of device selection possibilities are as follows:

Disc

DC	Any disc except memory disc
DM	Any moving head or memory disc
DM08	Any moving head disc on channel 08
DM0801	Moving head disc 01 on channel 08
DM0002	Memory disc 02 on channel 00
DF	Any fixed head disc
DF04	Any fixed head disc on channel 04
DF0401	Fixed head disc 01 on channel 04
FL70F0	Floppy disc 0 on controller F channel 70

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

CR	Any card reader
CR78	Any card reader on channel 78
CR7800	Card reader 00 on channel 78

Line Printer

LP	Any line printer
LP7A	Any line printer on channel 7A
LP7A00	Line printer 00 on channel 7A

Table 5-2
MPX-32 Device Type Codes

Dev Type Code	Device	Device Description
00	CT	Operator console (not assignable)
01	DC	Any disc unit except memory disc
02	DM	Any moving head or memory 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*
08	CR	Any card reader
0A	LP	Any line printer
0B	PT	Any paper tape reader-punch
0C	TY	Any teletypewriter (other than console)
0D	CT	Operator console (assignable)
0E	FL	Floppy disc
0F	NU	Null device
10	CA	Communications adapter (binary synchronous/asynchronous)
11	U0	Available for user-defined applications
12	U1	Available for user-defined applications
13	U2	Available for user-defined applications
14	U3	Available for user-defined applications
15	U4	Available for user-defined applications
16	U5	Available for user-defined applications
17	U6	Available for user-defined applications
18	U7	Available for user-defined applications
19	U8	Available for user-defined applications
1A	U9	Available for user-defined applications
1B	LF	Line printer/floppy controller (used only with SYSGEN)
N/A	ANY	Any nonfloppy disc except memory disc

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

5.7 Device-independent I/O Processing Overview

A task starts I/O operations (reads, writes, etc.) by issuing service calls to IOCS. IOCS validates the logical address of the task's data buffer (defined in the Transfer Control Word of the FCB), and links an I/O request containing the TCW information to a queue for the appropriate device handler. The I/O requests are queued by the software priority (1 to 64) of requesting tasks.

The handler issues appropriate instructions to the device controller (command device, test device, or start I/O).

The controller performs the I/O. For example, it reads a record into the task's data buffer. When the requested I/O is complete, the controller issues a Service Interrupt (SI).

If the handler is passing the address of a list of directives or data (IOCL) for a controller to operate on, the SI is returned from the controller when all operations specified in the list have been completed.

The processing that occurs when the handler receives the SI interrupt from the controller depends on whether the user has established a wait I/O or no-wait I/O environment by the FCB.

5.7.1 Wait I/O

If wait I/O is indicated in the FCB, the task waits (suspends) until the I/O operation completes. IOCS returns to the task at the point following the I/O service call.

5.7.1.1 Wait I/O Errors

If an error occurs during an I/O operation to tape or disc, the handler automatically retries the operation. If retry operations fail, the handler passes the error to IOCS and IOCS posts status in the FCB (word 3, and optionally words 11 or 12). The task can take action or not as described in the next section. When an error is detected on a card reader, line printer, or other device that does not have automatic retry, and operator intervention is applicable, the handler passes the error to IOCS and IOCS issues a message on the system console indicating the device is not operating:

```
*devmnc INOP: R,A?
```

Sample criteria for the INOP message are: the printer runs out of paper or jams, a card reader jams, etc. IOCS allows the console operator to correct the condition or abort. If the device is fixed, the operator types R (retry). IOCS re-establishes the entry conditions for the handler (passes the TCW with the initial transfer count, etc.) and calls the handler to retry the I/O operation that was in error. The error status is cleared and I/O proceeds normally. If the operation aborts, IOCS starts abort processing.

5.7.1.2 Wait I/O Exit and I/O Abort Processing

If error processing is not applicable (disc or magnetic tape) or if the operator has responded A (abort) to the INOP message, IOCS either aborts the task or transfers control to the task at the address specified in word 6 of the FCB. If the task gains control, it examines the contents of word 3 and optionally words 11 and 12 as applicable, and performs its own exit or abort processing.

If the user has not defined an error return address for wait I/O in the FCB, IOCS aborts the task and displays an abort message on the system console:

I/O ERR DEV: devmnc STATUS statusword LFC:lfc mm/dd/yy hh:mm:ss

IOCS displays status word 3 from the FCB on the system console.

For I/O to XIO devices, a second line is displayed along with the I/O ERR message:

XIO SENSE STATUS = senseword

where senseword is the sense information returned in FCB.IST1 of an extended FCB.

5.7.1.3 Error Processing and Status Inhibit

The user sets word 2, bit 1 of the FCB to bypass error processing by handlers and IOCS. On error, the status word of the FCB is still set by IOCS (unless bit 3 is set as described below), and IOCS transfers control to the task normally. The task must perform any error processing.

If the user sets word 2, bit 3 of the FCB, he inhibits handlers from checking status in any respect. No error status is returned to IOCS. All I/O appears to complete without error.

5.7.2 No-wait I/O

If the user has indicated no-wait I/O in the FCB, IOCS returns control to the task immediately after an I/O request is queued. The task continues executing parallel with I/O. When the handler fields an SI interrupt for the specified I/O operation, it notifies the MPX-32 Executive. The executive links the I/O queue entry to a software interrupt list for the task (a task interrupt). When the task is to gain control, the executive passes control to IOCS. An unprivileged user is limited to five no-wait I/O requests.

5.7.2.1 No-wait I/O Complete Without Errors

IOCS checks the address specified by the user in the FCB for no-wait I/O end action processing. If I/O completes successfully, it routes control to the address provided by the user for normal end processing (word 13). If the user has not specified this address in the FCB, IOCS returns control to the executive and the task continues executing at the point where the task interrupt occurred.

5.7.2.2 No-wait I/O Complete with Errors

When IOCS gains control on a task interrupt and an error occurs, IOCS routes control to the task at the user-supplied error return address in word 14 of the FCB. If the user has not supplied this address, control is returned to the executive and then to the task so that it continues execution where the task interrupt occurred.

The FCB (word 3 and optionally words 11 and 12) indicates the cause of an error. The task is responsible for examining the word(s) and for any recovery procedure. IOCS makes no attempt to recover from an error condition through operator intervention when a task uses no-wait I/O.

5.7.2.3 No-wait End-action Return to IOCS

A task using no-wait I/O end-action processing should return to IOCS by an SVC 1,X'2C' after normal or error processing is complete. IOCS returns control to the executive, which returns control to the task where the task interrupt occurred. In any end-action processing, register one points to the FCB address.

5.7.3 Direct I/O

Within a task, the user can temporarily bypass normal IOCS and handler functions by coding his own handler and attaching it to a specific channel (service interrupt level). Direct I/O is totally under the user's control and acquires data at rates that prohibit IOCS overhead.

To perform direct I/O, the task passes a TCW directly to a device, completely bypassing the IOCS. The task is responsible for any control structures related to the I/O operation (it does not, for example, have use of a FCB) and it must field interrupts on its own. The user must be familiar with the hardware and its response to software instructions, and must implement all support pertaining to I/O for the device.

Before connecting his own handler, the user issues a reserve channel call to IOCS. IOCS then holds all outstanding or subsequent requests for the specified channel until the task issues a release channel call. When IOCS receives the release request, it resumes normal processing on the channel with the standard handler.

Direct I/O is not the same as developing an I/O handler and linking it into the system at SYSGEN. Direct I/O is a privileged operation.

5.7.4 Blocked I/O

Blocked I/O processing is supported under MPX-32 by the Blocked Data Management Module (H.BKDM).

I/O to disc files and magnetic tapes is blocked or unblocked depending on the device assignment or the setting of word 2, bit 5 of the FCB at open. If the user does not set bit 5, I/O is blocked or unblocked as specified by the device assignment. If bit 5 is set at open, I/O is blocked. The FCB definition overrides any assignment specification of unblocked (specified explicitly at assignment).

For blocked I/O, REMM automatically allocates and uses a 192-word blocking buffer that provides intermediate buffering between a task's data buffer and a device. A block is 192 words long and records that can be packed into the block are a maximum of 254 bytes long. Longer records are truncated. The particular buffer used for blocked I/O can be user-supplied by the FCB at open or allocated from the buffer area of the TSA.

On input, IOCS transfers a block of records from a device into the blocking buffer and moves them into the task's data buffer one logical record at a time. On output, IOCS transfers one logical record at a time from the task's data buffer into the blocking buffer and outputs the accumulated records in 192-word blocks.

Reads and writes to the same blocked file or tape by the same task are not mixed because they interfere with the blocking buffer operations being performed by IOCS. A read attempted while IOCS is writing out a group of records to the blocking buffer (or a write when reading) is not executed and IOCS aborts the task.

Special care must be taken when writing to a blocked file in modify mode. Blocked files are modified on a block-by-block basis rather than record-by-record. MPX-32 clears each block of a blocked file before writing records to the block. The user may rewrite any number of records in the cleared block. However, since MPX-32 does not preread the block before clearing it, the user must preread any records in a block which will be modified and rewrite them, if necessary.

5.7.4.1 Assign/Open Block Mode Determination Matrix

Form of Assignment	Open by M.OPENR			Open by M.FILE
	No CNP Options Set	CNP Word 2 Bit 10 Set	CNP Word 2 Bit 11 set	FCB Word 2 Bit 5 set
AS LFC TO pathname	Blocked	Unblocked	Blocked	Blocked
AS LFC TO pathname BLOCK = Y	Blocked	Unblocked	Blocked	Blocked
AS LFC TO pathname BLOCK = N	Unblocked	Unblocked	Blocked	Blocked
Al LFC = filename	Blocked	Unblocked	Blocked	Blocked
Al LFC = filename,,U	Unblocked	Unblocked	Blocked	Blocked

Notes:

When bit 10 is set, the resource is opened in unblocked mode, overriding any specification made at assignment.

When bit 5 or bit 11 is set, the resource is opened in blocked mode, overriding any specification made at assignment.

If both bit 10 and bit 11 are set, the resource is opened in blocked mode.

5.7.5 End-of-file and End-of-medium Processing

Table 5-3 describes the different types of end-of-file (EOF) and end-of-medium (EOM) processing supported under MPX-32. EOF only occurs when the beginning of the record to be transferred meets the specified condition. If EOF occurs in other than the first block of the transfer, the transfer is truncated up to the EOF block with the actual transfer count set to the truncated amount and no EOF indication. On the next sequential transfer after the truncated record, the task receives the EOF indication.

5.7.6 Software End-of-file for Unblocked Files

MPX-32 provides a software end-of-file (EOF) capability for an unblocked file in the sequential mode of operation. A request to write an EOF causes a write of a 192-word physical EOF record which begins with X'0FE0FE0F'. When reading, EOF is reported when X'0FE0FE0F' is detected as the first word of the first block read. If a read request spans across an EOF block, the read request is truncated to the amount of data up to but not including the EOF block, and the actual record length reflects that amount of data transferred. The next read request after the truncated request produces an EOF indication.

Software EOF reporting is inhibited by specifying data formatting inhibited. If data formatting inhibited is specified when a request to write EOF is issued, the request is returned complete but no EOF record is written to the file. EOF reporting is automatically inhibited when reading in random access mode.

A software EOF should be the last record written to an unblocked file prior to closing the file. This precaution is necessary in case the file is subsequently opened in append mode. The I/O facilities expect the last record on an unblocked file to be an EOF record. In append mode, the record pointer is positioned just in front of the last record so that the next record written is appended to the end of the previous data and not to the EOF record. If an EOF has not been written, then the last record from the previous write session is overwritten and lost.

Table 5-3
EOF and EOM Description

		EOF	EOM
Blocked	Read	EOF occurs when the record with the EOF bit set in the block buffer control word is read.	EOM occurs when a record is read from a block that is past the last block written to the file.
	Write	Not Applicable	EOM occurs when a record is written to a block that is one or more blocks past the end of the file space for a file that cannot be extended.
Unblocked with Data Formatting	Read	EOF occurs if the first word of data read has the data pattern X'0F' 0F' in it. EOF will also occur if a read starts 1 block past the last block written to the file (EOF block in the FAT).	EOM occurs when a read starts from two or more blocks past the last block written in the file.
	Write	Not Applicable	(same as Blocked Write EOM)
Unblocked with Data Formatting Inhibited	Read	EOF occurs if a read starts 1 block past the last block written to the file (EOF block in the FAT).	(same as Unblocked Read with Data Formatting EOM)
	Write	Not Applicable	(same as Blocked Write EOM)

5.8 Spooled Output with Print or Punch Attribute

Real-time (not job related) permanent and temporary files are assigned with either the print or punch attribute. When the file is deassigned, spooled output is automatically generated on the auto-selectable print or punch device. If the file is temporary, it is deleted when the output process is complete. If the file is permanent, it is not deleted when the output process is complete.

5.9 Setting Up File Control Blocks for Device Independent I/O

Section 5.2 describes the function of an FCB. Parts of the FCB are required and must be defined by the user:

- A logical file code
- A Transfer Control Word (TCW) indicating transfer count and data buffer address for I/O operations controlled by this FCB

IOCS assumes the following if no other special I/O characteristics are defined in the FCB:

- Wait I/O - IOCS returns to the calling task only when a requested operation on the file or device assigned to this FCB is complete
- Automatic retry on error by IOCS and some handlers, as described in Section 5.7
- Device-dependent output and input are handled using standard techniques
- Status information is returned in the FCB
- File and device access is sequential

Areas of the FCB define:

- No-wait I/O - Immediate return to the calling task after I/O operation is queued. The user can define the return address to return to in the task when processing is complete (normal or error).
- Error processing inhibit - Only status is returned by handlers. No error processing is performed by IOCS or handlers.
- Special device output characteristics
- Random access for disc files
- Expanded Transfer Control Word (TCW)

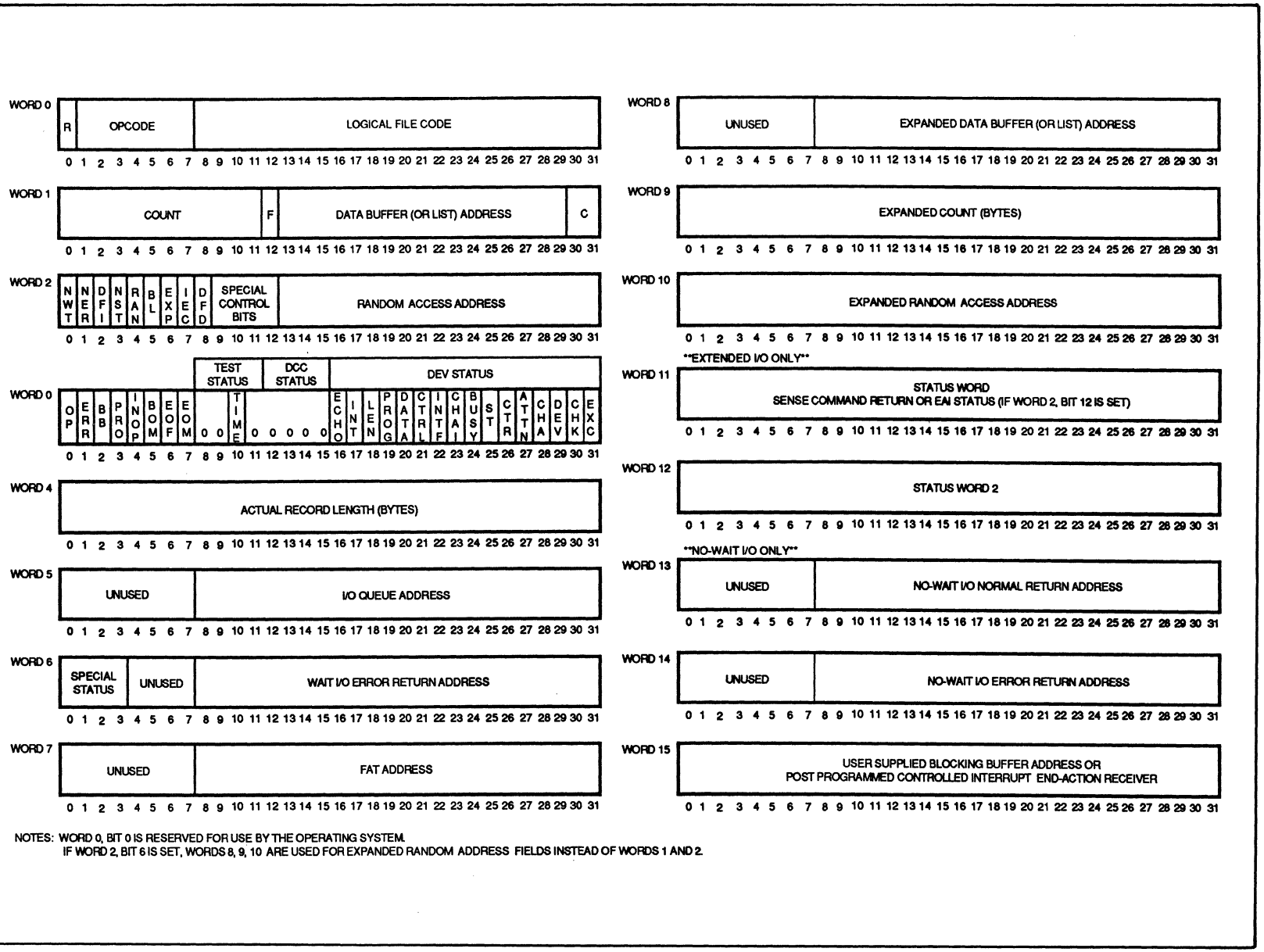


Figure 5-2. File Control Block

NOTES: WORD 0, BIT 0 IS RESERVED FOR USE BY THE OPERATING SYSTEM.
IF WORD 2, BIT 6 IS SET, WORDS 8, 9, 10 ARE USED FOR EXPANDED RANDOM ADDRESS FIELDS INSTEAD OF WORDS 1 AND 2.

Table 5-4
Nonextended I/O Device Status
(2000 Level)

FCB Word 3 Bits	16	17	18	19	20	21	22	23	24
All F-Class Devices	ECHO	PPCI	INCORRECT LENGTH	PROGRAM CHECK	DATA CHECK	CONTROL CHECK	INTERFACE CHECK	CHANNEL CHAINING CHECK	BUSY
High Speed Data Interface (Generic Handler) D-Class	CD TERMI-NATION	ERROR STATUS FORMAT (See Word 3 description)							
GPMC	DEVICE DEPENDENT								

FCB Word 3 Bits	25	26	27	28	29	30	31
ALL F-Class Devices	STATUS MODIFIER	CONTROLLER END	ATTENTION	CHANNEL END	DEVICE END	UNIT CHECK	UNIT EXCEPTION
High Speed Data Interface (Generic Handler) D-Class	EXTERNAL TERMINATION	IOCB ADDR ERROR	ERROR ON TI ADDR FETCH	DEVICE EOB	EP5 ERROR PRE-CLUDED REQUEST QUEUEING	NONEXECUTIVE CHANNEL PROGRAM IOCB TYPE IN ERROR 00-DATA TRANSFER 01-DEV STATUS 10-COMMAND TRANSFER	
GPMC	DEVICE DEPENDENT	TRANS-MISSION ERROR	INCORRECT LENGTH	UNUSUAL END	ILLEGAL ORDER	INTERRUPT PENDING	CHANNEL END

Some areas of the FCB are defined by IOCS. IOCS stores the opcode each time the task specifies a particular FCB, stores status returned by handlers, tracks actual record length in bytes for each transfer, and builds and maintains I/O queue and File Assignment Table (FAT) addresses. Figure 5-2 describes the FCB layout. All but words 0 and 1 are optional. The user should initialize all portions of the FCB that IOCS or handlers set up or that IOCS must handle to zero.

5.9.1 FCB Word Descriptions

5.9.1.1 Word 0

IOCS defines the I/O operation indicated by the task (read, write, etc.) in terms of the operation allowed on an assigned device or file; the user defines the logical file code (LFC) used externally to assign a file or device for the operation.

Bits	Description
0	Reserved
1-7	Operation Code - IOCS uses a single hexadecimal digit to indicate the type of function for the device handler. Allowable functions and their definitions are unique to each peripheral device. See Tables 5-5 and 5-6.
8-31	Logical File Code - Any combination of three ASCII characters is acceptable and must be supplied by the user.

5.9.1.2 Word 1

This word (or words 8 and 9 if bit 6 of word 2 is set) supplies a Transfer Control Word (TCW) that accesses a data buffer or IOCL for I/O. If no TCW definition is supplied, the transfer buffer defaults to location zero of the task's logical address space and is 4096 words (4KW) maximum.

Bits	Description
0-11	Count - Three hexadecimal digits specify the number of units (bytes, halfwords, or words) to be transferred to or from a device or file. The count must include a carriage control character, if applicable. The units the count relates to are determined by the data buffer address in bits 12 to 31. The maximum value of this field is 4096 words. If zero, the maximum transfer count defaults to 4096 words.

The F bit (12) and C bits (30 and 31) of the data buffer address are set by IOCS according to the following definitions:

Bits	Description
12, 30/31	These format code bits specify byte, halfword, or word addressing for data transfers. They are interpreted as follows:

Type of Transfer	F (Bit 12)	CC (Bits 30-31)
Byte 0	1	00
Byte 1	1	01
Byte 2	1	10
Byte 3	1	11
Left halfword	0	01
Right halfword	0	11
Word	0	00

If a halfword or word transfer is specified and a device accepts only bytes, IOCS adjusts the count accordingly (times two or four). If a byte transfer is specified and a device accepts only halfwords or words, IOCS checks to see that the number of bytes in the buffer is an even multiple of the requested transfer and the data buffer address is on an acceptable boundary. If these conditions exist, IOCS adjusts the count accordingly and initiates the transfer. If the conditions are not met, the request is treated as a specification error. Table 5-7 indicates transfers that are acceptable for various devices. Addresses that produce specification errors are flagged with an asterisk.

Note: IOCS operations described above enable the user to specify byte transfers beginning on a word boundary or word transfers on any device, whether the device operates on bytes, halfwords, or words.

Doubleword addressing is not allowed; IOCS aborts the task.

13-29 The data buffer address specifies the start address of a data buffer reserved for reads and writes by the user. Bounding requirements are indicated in Table 5-7.

Table 5-5
Device Functions (Standard Devices) Page 1 of 3

Operation	IOCS Op Code	Line Printer (LP)	Card Reader (CR)	Mag Tape (MT)	Disc (DM/DF/DC/Floppy)	High Speed Data (HSD) Interface
Open (M.FILE)	0	IOCS opens	NOP	IOCS Opens	IOCS Opens	NOP-intended for user device specific processing.
Rewind (M.RWND)	1	Eject; set BOM bit (3/5) in FCB	Set BOM bit	Rewind Tape	Set current block address to zero (FAT)	Send device command FCB.
Read Record (M.READ)	2	Spec error	Read card (80 max ASCII) (120 max BIN)	Read to data buffer as described in TCW	Read to data buffer as described in expanded TCW	Read to data buffer as described in expanded TCW
Write Record (M.WRIT)	3	Write from data buffer described in TCW	Spec error	Write from data buffer described in TCW; if blocked, writes 'n' data buffers to blocking buffer before output	Write from data buffer described in TCW; if blocked, IOCS writes 'n' data buffers to blocking buffer before output	Write from data buffer described in expanded TCW
Write EOF (M.WEOF)	4	NOP	Spec error	Write EOF	If blocked, IOCS writes EOF. If unblocked, writes X'OFE0FE0F'	Send device command
Execute Channel (Extended I/O or GPMC or HSD)	5	Spec error	Spec error	Execute Channel Program	Execute Channel Program	Logical or physical Start I/O format request

850254-1

Table 5-5
Device Functions (Standard Devices) Page 2 of 3

Operation	IOCS Op Code	Line Printer (LP)	Card Reader (CR)	Mag Tape (MT)	Disc (DM/DF/DC/Floppy)	High Speed Data (HSD) Interface
Advance Record (M.FWRD)	6	Spec error	Skip card	Advance record	If blocked, advance record; if unblocked, advance one 192W block.	Send device command
Advance File (M.FWRD)	7	Spec error	Advance file (past X'OF' - pseudo EOF)	Advance file (past EOF)	Spec error	Send device command
Backspace Record (M.BACK)	8	Spec error	Spec error	Backspace record	If blocked, backspace record; if unblocked, backspace one 192W block	Send device command
Backspace File (M.BACK)	9	Spec error	Spec error	Backspace file to previous EOF	Spec error	Send device command
Upspace (M.UPSP)	A	Upspace	NOP	Multivolume only. If BOT, writes volume record. If EOT, performs ERASE, writes EOF, and issues MOUNT message.	Spec error on F-class discs. For floppy only: Format diskette. New diskettes must be formatted prior to normal usage.	Send device command
Erase or Punch Trailer Not user IOCS/handler provide call automatically	B	NOP	NOP	Multivolume only. Same as upspace above. Erases 4" of tape before writing.	NOP	Send device command

850254-2

Table 5-5
Device Functions (Standard Devices) Page 3 of 3

Operation	IOCS Op Code	Line Printer (LP)	Card Reader (CR)	Mag Tape (MT)	Disc (DM/DF/DC/Floppy)	High Speed Data (HSD) Interface
Eject/Punch Leader	C	Eject to top of form	NOP	Write dummy record with eject control character as first character	NOP	Send device command
Close (M.CLSE)	D	IOCS closes	IOCS closes	IOCS closes	IOCS closes	NOP, complementary function to open
Reserve FHD Port	E	Spec error	Spec error	Spec error	Reserve port - 4 MB disc only; else, spec error Reserve Dual Ported Disc	Send device command
Release FHD Port	F	Spec error	Spec error	Spec error	Release port - 4 MB disc only; else, spec error Reserve Dual Proted Disc	Send device command

850264-3

Table 5-6
Device Functions (Terminals, Handler Action Only)

Operation	IOCS Op Code	Handler = H.ASMP (ALIM)	Handler = F8XIO (8-Line)
Open M.FILE	0	NOP*	Initialize IOP channel if necessary
Rewind M.RWND	1	NOP*	SENSE operation
Read record M.READ	2	Read to data buffer	Read to data buffer
Write record M.WRIT	3	Write record to terminal	Write record to terminal
Write EOF M.WEOF	4	NOP*	NOP*
Execute channel	5	Execute channel	Execute channel
Advance record M.FWRD	6	Connect communications channel	Set data terminal ready
Advance file M.FWRD	7	Disconnect communications channel	Reset data terminal ready
Backspace record M.BACK	8	Initialize device and set time-out value	Used by J.TINIT to initialize terminals
Backspace file M.BACK	9	Clear break status flag word	Reset request to send
Upspace M.UPSP	A	Spec error**	Set request to send
Erase/punch trailer	B	Transmit break	Set/reset break (depends on flags in FCB)
Eject/punch leader M.EJECT	C	Spec error**	Define special character
Close M.CLSE	D	NOP*	NOP*
Reserve FHD port	E	Spec error**	Set single-channel operation (default)
Release FHD port	F	Spec error**	Set dual-channel operation

* NOP = No operation performed
** Spec Error = Illegal operation code

Table 5-7
Acceptable/Nonacceptable Device Transfers
Specified in TCW Word 1, Bits 12 and 30/31

FCB Format/Bounding	Device Transfers					
	Words		Halfwords		Bytes	
	Bits		Bits		Bits	
	12	30/31	12	30/31	12	30/31
Word	0	00	0	00	0	00
Left Halfword	0	01	0	01	0	01
Right Halfword	0	11*	0	11	0	11
Byte, first even	1	00	1	00	1	00
Byte, third even	1	10*	1	10	1	10
Byte, any odd	1	x1*	1	x1*	1	x1*

* Write to/reads from device result in IOCS abort with this FCB definition.

5.9.1.3 Word 2

Word 2 provides optional control specifications for I/O.

Bits	Description

0-8 Operational specification - These nine bits enable special operations to be performed by IOCS. The meaning of each bit is as follows:

Bit	Definition	
0	NWT	No-wait I/O (words 13 and 14). Default: wait I/O.
1	NER	No error return processing
2	DFI	Data format inhibit. Use format in bits 8 to 12. If not specified, IOCS provides formats. See Table 5-8.
3	NST	No status check by handler; no status returned. All I/O appears to complete without error.
4	RAN	Random access (user supplies address in bits 13 to 31). Default: sequential, IOCS supplies address.
5	BL	Blocked I/O, disc/tape only. Used only by the M.FILE service. Default: based on assignment.
6	EXP	Use words 8, 9, and 10 for expanded random address fields rather than the fields in words 1 and 2. Default: fields in words 1 and 2.
7	IEC	Reserved for internal IOCS use. The user's FCB is being used for physical I/O during blocked data handling and the FCB parameters are in the task's scratchpad.
8	DFD	Device format definition. Special definitions for some devices are indicated in bits 9 to 12. See Table 5-8.

9-12 Device control specification - These bits contain control specifications unique to certain devices. Device handlers interpret and process the specifications. A bit setting is meaningful only when a particular type of device is assigned as indicated in Table 5-8, columns two and three. Column one indicates default control.

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

Note: If bit 6 of word 2 is set, the expanded random access address in word 10 is used instead of bits 13 to 31 above.

Table 5-8
Default and Special Device Formatting (Page 1 of 2)

Device	Default (Bit 2=0)	Override (Bit 2=1)	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12
Card Reader (CR)	Read Auto Select Mode. First column - Rows 2-5, all punched = binary, no translation, max. 120 bytes. Not all punched = ASCII, translate, max 80 bytes. EOF = column 1, Rows 2-5 only punched. Binary, no translation (X'0F'). Max 120 bytes.	See Bit 8	0=ASCII read 1=Binary read				
Line Printer (LP)	Interpret first character as carriage control	No carriage control interpretation					
Console or Terminal (CT or TY)	Interpret first character as carriage control	No carriage control		Read: 0 = Echo 1 = No echo			
High Speed Data (HSD) Interface (Generic Handler)			See Word 2 definition	See Word 2 definition	See Word 2 definition	See Word 2 definition	See Word 2 definition
Paper Tape Reader (PT)	Read in formatted mode, skipping leader	Read unformatted	0=Do not skip leader 1=Skip leader				
Paper Tape Punch (PT)	Punch in formatted mode	Punch unformatted					

850255-1

Table 5-8
Default and Special Device Formatting (Page 2 of 2)

Device	(Bit 2=0)	(Bit 2=1)	Bit 8	Bit 9	Bit 10	Bit 11	Bit 12
Mag Tape (MT, 7-track only)	Binary, ODD parity, 800 BPI	See Bits 8 to 10	0=Interchange (binary coded decimal) 1=Packed (binary)	If Bit 8=0: 0=EVEN parity 1=ODD parity	0=800 BPI 1=556 BPI		
(MT 9-track only)	N/A						
Discs (DM,DF,FL)	Report EOF if X'0FE0FE0F' encountered in word 0 of 1st block during read of unblocked record	No EOF reporting on unblocked reads					
ALIM (Asynchronous Line Interface Module) Terminals (TY)	Read: receive data (bytes) defined for transfer count. Write: formatted.	Read: Bit 2: 0 / Bit 8: 1 / Bit 9: 0 =Blind mode reset Read: Bit 2: 0 / Bit 8: 0 / Bit 9: 1 =Echo on read Read: Bit 2: 1 / Bit 8: N/A / Bit 9: N/A =Receive data Read: Bit 2: 0 / Bit 8: 0 / Bit 9: 0 =Receive data Write: Bit 2: 0 / Bit 8: N/A / Bit 9: 0 =Formatted write Write: Bit 2: 0 / Bit 8: N/A / Bit 9: 1 =Initialize device Write: Bit 2: 1 / Bit 8: N/A / Bit 9: N/A =Unformatted write			On Read: 1= Inhibit conversion of lower case characters to upper case 0= Convert		
8-Line Asynchronous Communications Multiplexer (TY)	Read: perform special character formatting. Write: first character is for form control.	Read: if formatmatting is inhibited, reads n bytes. Write: n bytes without form control.	Transmit break (erase, punch) (trailer): 0=Stop transmitting break 1=Start transmitting break. Read: 1=ASCII control character detect.	Read: 0=Echo by controller 1=No echo by controller. Write: 0=Normal write 1=Initialize device (load UART parameters).	Read (if Bit 2=0): 0=convert lower case character to upper case 1=Inhibit conversion	Read: 0=no special character detect 1=special character detect. Write 0=normal write 1=write with input subchannel monitoring plus software flow control.	Read: 0=do not purge type ahead buffer 1=purge type ahead buffer

850255 -2

For High Speed Data (HSD) Interface applications, word 2 bit meanings are as follows:

Bits	Meaning if Set
8	Request device status after transfer - Indicates an IOCB should be added to the IOCL to retrieve device specific status after the data transfer has completed.
9	Send device command prior to data transfer - Indicates an IOCB should prefix the data transfer to transmit a device command word to the device. The value sent is the 32-bit expanded random access address.
10	Disable time-out for this request - Indicates that the operation takes 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.
11	Set UDDCMD from least significant byte of word 2 - Indicates that the UDDCMD byte in the data transfer IOCB should be set to the least significant byte of the random access field of the FCB. This provides the ability to pass additional control information to the device without modifying the device driver.
24-31	If bit 11 is set, these bits define the UDDCMD field of the generated IOCB, overriding the default value from a handler table.

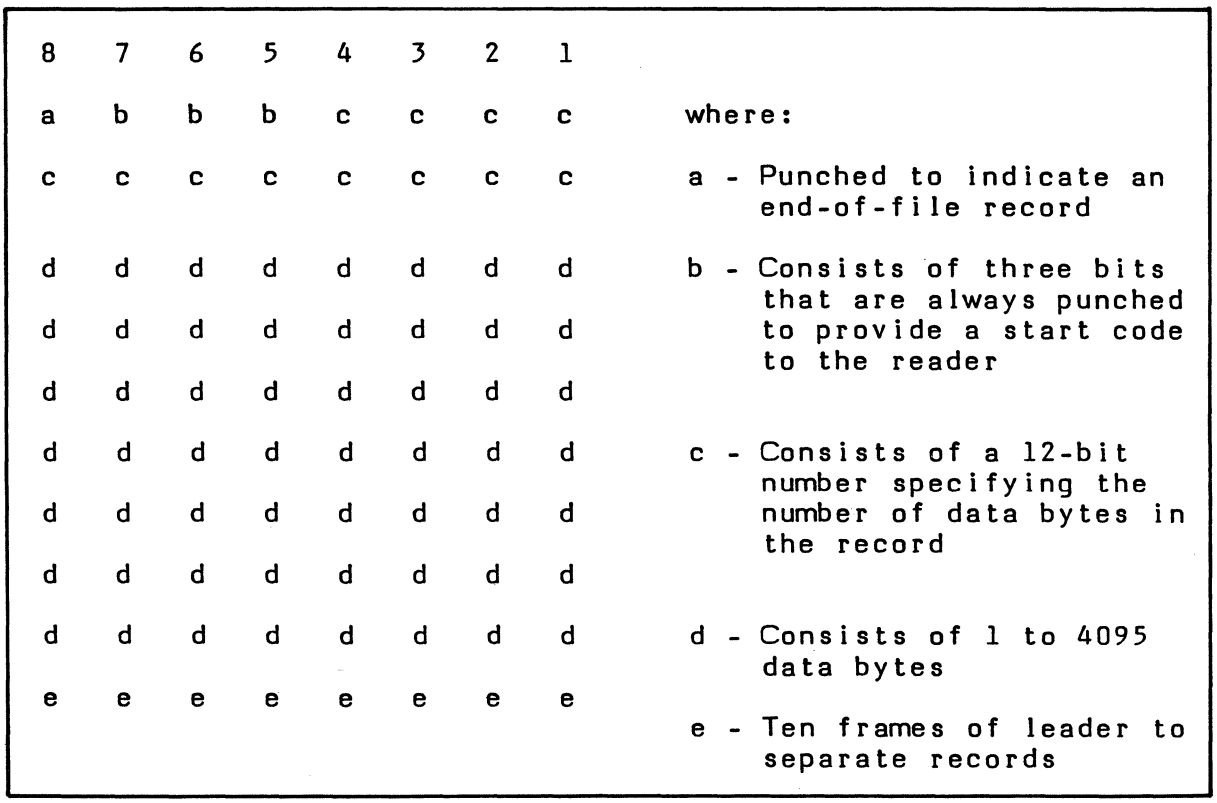


Figure 5-3. Punched Tape Format

Table 5-9
Standard Terminal and Line Printer
Carriage Control Characters and
Interpretation

Control Character	Hexadecimal Value	Result on a Terminal	Result on a Line Printer
Blank	20	One linefeed/carriage return before write	Single space before print
0	30	Two linefeed/carriage returns before write	Double space before print
1	31	Five linefeed/no carriage returns before write	Page eject (slew) before print
+	2B	No linefeed/carriage return before write (line append)	No space before print (overprint)
-	2D	Five linefeed/carriage returns before write	Page eject, save and print up to three user supplied title lines. See Notes 1 & 2.
<	3C	One linefeed/carriage return before write	Set inhibit spooler title line in this file. See Note 2.
>	3E	One linefeed/carriage return before write	Set enable spooler title line in this file. See Note 2.
=	3D	One linefeed/carriage return before write	Page eject and clear up to three user-supplied title lines in this file. See Note 2.

Notes:

1. User-supplied title lines have the same effect as this character. Supplying a fourth title line clears the first three, but only one page is ejected. User-supplied titles are retained by the spooler and are repeated at the top of each page until cleared or the spool file ends.
2. If the line printer is directly allocated, this character is processed as a blank and the spooler title line is not produced.

5.9.1.4 Word 3

Word three returns I/O status. IOCS uses 32 indicator bits to return the status, error, and abnormal conditions detected by handlers during the previous or current device operation. The task can examine these bits as needed. Individual bit assignments for bits 0 - 7 apply to any device. Bits 8 - 31 mean different things depending on the device. For extended I/O devices, individual bits are shown in the FCB word 3 area of Figure 5-2. For nonextended I/O devices, bit fields are shown above the FCB word 3 area of Figure 5-1 and device status is returned as described in Table 5-5. Bits in word 3 have the following meanings:

Bit	Description	
0	OP	Operation in progress (I/O request has been queued). Bit is reset after I/O post processing is complete
1	ERR	Error condition found
2	BB	Invalid blocking buffer control pointers encountered in blocking or deblocking
3	PRO	Write protect violation
4	INOP	Device is inoperable
5	BOM	Beginning-of-medium (load point) or illegal multivolume number on magnetic tape
6	EOF	End-of-file (TSM also sets this bit when CTRLC entered on terminal)
7	EOM	End-of-medium (TSM also sets this bit when any other than a carriage return is entered at bottom of screen (end-of-tape, end of disc file)

Status for Extended I/O Devices

Bit	Description	
10	TIME	Last command exceeded time-out value and was terminated
16	ECHO	Echo on channel. Invalid for IPS devices.
17	INT	Post-task controlled interrupt on channel
18	LEN	Incorrect record length on channel
19	PROG	Channel program check
20	DATA	Channel data check
21	CTRL	Channel control check
22	INTF	Interface check

Bit	Description	
23	CHAI	Channel chaining check.
24	BUSY	Controller/device busy
25	ST	Controller/device status modified.
26	CTR	Controller end.
27	ATTN	Attention
28	CHA	Channel end
29	DEV	Device end
30	CHK	Unit check
31	EXC	Unit exception

Status for Nonextended I/O Devices

Bits	Description	
8-11	Test Status	Testing status as received from a 8000 level test device (TD) issued by handler
12-15	DCC Status	Controller status (DCC) as received from a 4000 level test device (TD) instruction
16-31	Device Status	Device status as received from a 2000 level test device (TD) instruction. Not applicable for PT, CR, or TY. See Table 5-4.

For High Speed Data (HSD) Interface applications, word 3 error status bits have the following meanings:

Bits	Description
17-18	Unused
19	Record length error
20	Parity error
21	Nonpresent memory (NPM)
22	Invalid opcode in IOCB word 0
23	Device inoperable
24	Data buffer overflow

5.9.1.5 Words 4 and 5

Word four defines record length. This word is used by IOCS to indicate the actual number of bytes transferred during a read or write.

Word five defines I/O queue address in bits 8-31. This field is set by IOCS to point to the I/O queue for an I/O request initiated from this FCB.

5.9.1.6 Word 6

Word six contains special I/O return status in bits 0 - 7 as follows:

Bits	Description
0	No-wait I/O normal end-action address not executed
1	No-wait I/O error end-action address not executed
2	KILL command. I/O was not issued.
3	Exceptional condition occurred in I/O request
4	Software read flow control required
5-7	Not used

Word six defines a wait I/O error return address in bits 8 - 31. Specify the address to transfer control to if an unrecoverable error occurs. If this field is not defined, an unrecoverable error is detected, and the user has not set bits 1 and 3 of word 2 to inhibit error processing, IOCS aborts the task.

5.9.1.7 Word 7

Word seven defines the File Assignment Table (FAT) entry associated with all I/O performed for this FCB. The FAT address is supplied by IOCS. This word must not be defined by the user.

5.9.1.8 Word 8

Word eight begins expanded TCW definition. This area of the FCB can define transfers larger than 4095 words, as for extended I/O devices.

Bits	Description
8-31	Expanded data buffer address specifies the start address of a data buffer reserved by the user for reads or writes. This must be a logical byte address with no format bit present. Word bounding is required for some devices if unblocked. or Expanded data/command chain list address is the word address that points to the data or command chain list (IOCL) if using Execute Channel service, SVC 1,X'25'.

5.9.1.9 Word 9

Word nine continues the expanded TCW definition.

Bits	Description
0-31	Expanded transfer count - Eight digits specify the number of bytes to be transferred. Note that the transfer count supplied here is always in byte units. Must include the carriage control character, if applicable.

5.9.1.10 Word 10

Word ten defines the expanded random address. This word 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.

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

5.9.1.11 Word 11

Word 11 returns status.

Bits	Description
0-31	Status word one - For extended I/O. If an error is detected during an I/O operation, these 32 bits are returned by the SENSE command. or Status EAI - For communications adapter (CA) interface, returns external asynchronous interrupt (EAI) status if bit 12 of word 2 is set.

5.9.1.12 Word 12

Word 12 returns status word 2. This is the second status word returned from extended I/O hardware.

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

5.9.1.13 Word 13

Word 13 defines normal return address for no-wait I/O. Bits 8 - 31 specify the transfer control address when a no-wait I/O operation is complete. The code at this address must be terminated with a call to IOCS for postprocessing service (SVC 1,X'2C'). See Section 5.7.2.

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

5.9.1.14 Word 14

Word 14 defines an error return address for no-wait I/O. Bits 8 - 31 specify (optionally) the transfer control address when no-wait I/O completes with an error. The code at this address must be terminated with a call to IOCS for postprocessing service (SVC 1,X'2C'). See Section 5.7.2.

5.9.1.15 Word 15

Word 15 defines a user-supplied blocking buffer address for device-independent I/O or a postprogrammed controlled interrupt end-action receiver (see Section 5.11.2.3) for device-dependent I/O. If the user-supplied blocking buffer is a large blocking buffer, byte zero contains the number of 192W blocks within the large buffer.

5.9.2 Macros (M.DFCB/M.DFCBE)

The M.DFCB or the M.DFCBE macro can define an FCB or an expanded FCB, respectively.

There is no base mode equivalent service for M.DFCB. The base mode equivalent service for M.DFCBE is M_CREATEFCB.

Syntax:

```
M.DFCB[E] label, lfc, count, buffer, [error]
          , [random] , [NWT] , [NER] , [DFI] , [NST]
          , [RAN] , [ASC] , [LDR] , [INT] , [EVN]
                    [BIN]   [NLD]   [PCK]   [ODD]
          , [556] , [nowait], [nowaiterror],[buffaddr]
            [800]
```

label — ASCII string to use as symbolic label for address of this FCB

lfc — Logical file code. See word 0, bits 8 - 31.

count — Transfer count. See word 1, bits 0 - 11. Specify in number of bytes. For expanded FCB, see word 9, bits 0 - 31.

buffer — Start address of data buffer. (Reserve on word boundary.) See word 1, bits 12 - 31. For expanded FCB, see word 8, bits 8 - 31.

error — Error return address for wait I/O. See word 6, bits 8 - 31.

random — Random access address. See word 2, bits 12 - 31. For expanded FCB, see word 10, bits 0 - 31.

NWT/NER/ DFI/NST/ RAN/ — See word 2, bits 0 - 4

ASCII/BIN — For CR or CP, see Table 5-8

LDR/NLD	For PT (reader), see Table 5-8
INT/PCK	For 7-track mag tape, see Table 5-8
EVN/ODD	For parity, see Table 5-8
556/800	For bits per inch density, see Table 5-8
nowait	M.DFCBE (expanded FCB) only. Specifies address for normal no-wait I/O end action return.
nowaiterror	Same as above. Specifies address for no-wait I/O error end action return.
buffaddr	Specifies the address of a 192W user-supplied blocking buffer to be used for blocked I/O (applies to expanded FCB only)

5.9.3 Sample FCB Set-up Nonmacro

User defines an FCB for terminal message output:

```
TERM        DATAW     G'UT  '
            GEN       12/B(TY.LEN),20/B(MESSAGE)
            REZ       6W
MESSAGE     EQU       $
            DATAB     C' "M"J CREATE FAILED. ERRTYPE   '
TY.LEN      EQU       $-MESSAGE
ERRTYPE     EQU       $-1B
```

Notes: The source code uses TERM with M.WRIT to access this FCB. The logical file code is UT. The Transfer Control Word built with the GEN directive has a transfer count equal to the message (TY.LEN EQU $-MESSAGE) computed by the Assembler.

The buffer start address is at MESSAGE (address is supplied by the Assembler). In the actual message, "M indicates carriage return and "J indicates line feed before output. The last byte of the message comes from register five when an error occurs as defined by:

```
C.MSG       EQU       $
            M.CONBAD
            STB       R7,ERRTYPE
            M.WRIT    TERM
```

5.9.4 Sample FCB Set-up Macro

```
            M.DFCB    TERM, UT, TY.LEN, MESSAGE
MESSAGE     EQU       $
            DATAB     C' "M"J CREATE FAILED. ERRTYPE   '
TY.LEN      EQU       $-MESSAGE
ERRTYPE     EQU       $-1B
```

5.10 Setting Up Type Control Parameter Blocks (TCPBs) for the System Console

Messages are sent from a task to the system console and a response is optionally read back by a TCPB. The TCPB sets up task buffer areas for messages output by the task and reads back from the console.

The TCPB is comprised of a write and an optional read transfer control word defined like the TCW in word one of the FCB. If read back from the terminal is not desired, the read TCW must be zero. The user must perform his own carriage return and line feed.

The size of the read buffer should include space for both an input character count preceding the message (provided by IOCS) and the carriage return (end-of-record) character typed by the user at the end of an input line.

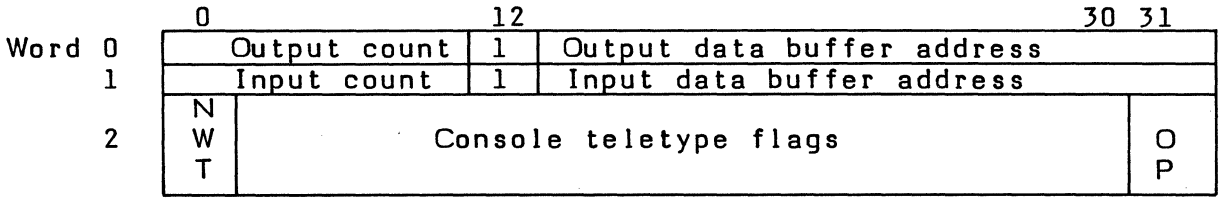
The count of input characters actually typed is placed by IOCS in the first byte of the read buffer. This input count does not include the carriage return character.

The byte transfer count is normally used by a task as maximum allowed input before termination. If the operator types in the maximum count without having typed a carriage return, the read is terminated.

The end-of-record character (carriage return) is normally allowed for in both the read and write buffer transfer count. If five characters are expected, the read transfer count would typically be for six characters. This allows the operator to type in and verify the accuracy of all five characters before terminating the input message by typing a carriage return.

Message transfers are always in bytes, so the buffer address must be a byte address (F-bit setting - bit 12 as shown in Figure 5-4).

If the NWT bit is set (word 2, bit 0), IOCS returns immediately to the calling task after the message is queued. The task can subsequently examine the OP indicator set by IOCS in word 2, bit 31 to see if the requested transfer is complete (0) or in process (1).



Bit Interpretations

NWT (word 2, bit 0) is set to define no-wait I/O
OP (word 2, bit 31) is set by IOCS if an operation is in progress

Figure 5-4. Type Control Parameter Block

5.11 MPX-32 Device-dependent Input/Output

MPX-32 provides the user with the ability to perform direct channel I/O to F-class I/O devices. Direct channel I/O is accomplished by issuing a request to execute a logical or physical channel program that was built by the user.

A logical channel program allows the user to use the specific physical attributes of the device without having to be privileged or work with physical addressing. The user can assign a logical file code to a physical device, but must be concerned with the physical characteristics of the device. This gives the unprivileged user a way to use the explicit functionality provided by the device that may not be provided by device-independent I/O.

A physical channel program allows a privileged user to issue an I/O command list directly to a channel/controller/device. The I/O command list is issued to the channel without any modification by the system. This allows a time critical user to issue physical I/O command lists that do not require the normal overhead of logical I/O.

5.11.1 Device-dependent I/O Processing Overview

A task can issue device-dependent I/O by issuing an Execute Channel Program (EXCPM) request to IOCS for any XIO device allocated in the channel program access mode. This enables the task to pass an I/O channel program that it has built to a specified device with minimal IOCS overhead.

EXCPM support provides the user with the ability to perform direct channel I/O by using either a logical channel program or a physical channel program. A logical channel program is an I/O Command List (IOCL) built using logical addresses inside the building task's logical address space. The operating system will change all addresses to physical and resolve all physical address discontinuities by the use of data chaining before the IOCL is executed by the device. On completion of the I/O, control is returned to the task in the same manner as any I/O request.

A physical channel program is an IOCL ready to be executed by the device and is pointed to by a logical address. The area in memory containing the physical IOCL must be made unswappable by the user. All addresses in a physical channel program are physical addresses and discontinuities in the logical to physical mapping have been resolved by the user. A task must be privileged to execute a physical channel program.

A physical channel program started in the no-wait mode may have a Post Program Controlled Interrupt (PPCI) EA receiver associated with it. This special EA receiver may be specified along with the normal EA receiver for no-wait I/O.

The EXCPM support applies only to extended I/O F-class devices.

The console does not have channel program capabilities.

5.11.2 Operational Description of Execute Channel Program (EXCPM)

The task must specify the logical address and a time-out value for the channel program in the FCB. The time-out value must be specified in seconds and placed in the second halfword of word 2 in the FCB. If the value equals zero, no time out is set for the EXCPM request. The logical address of the channel program must be placed in word 8 of the FCB. The task starts the I/O operation by issuing an SVC 1,X'25' or an M.CALL H.IOCS,10.

A SENSE buffer can be specified with an EXCPM request by placing the size of the buffer, in bytes, in FCB.SSIZ and the address of the buffer in FCB.SENA. If a SENSE buffer is specified, when the channel program terminates (normally or abnormally), a SENSE command is issued to the device and the information received is placed in the buffer specified. The size of the sense buffer must be equivalent to the number of bytes

of sense status that a particular device will return. If the buffer size is zero, no sense information is returned and a buffer is not needed.

5.11.2.1 Logical Channel Program

The task builds a channel program inside its logical address space using the standard XIO commands and logical addressing. A logical IOCL must be contiguous in logical memory and TIC commands cannot be used to create loops.

5.11.2.2 Physical Channel Program

The task must be privileged to execute a physical channel program. It must build a physical IOCL using standard XIO commands and all related addresses must be physical. It must also resolve all data area discontinuities with the use of data chaining. After the physical IOCL is built, the task places the logical address of the IOCL into word eight of the expanded FCB and sets bit two in word two of the FCB to indicate this is a physical IOCL. The user must insure that the memory containing the physical IOCL is unswappable.

5.11.2.3 Post Programmed Controlled Interrupt (PPCI) End-action Receiver

A privileged task may specify that a PPCI end-action receiver be used with a physical channel program issued in no-wait mode. This PPCI receiver will be entered when a PPCI occurs during the execution of the IOCL by the device. The logical address of the PPCI receiver must be placed in word 15 of the expanded FCB and bit 8 of word 2 of the FCB must be set.

The system builds a Caller Notification Packet (CNP) to transfer control to the task's PPCI receiver. The address of the CNP is passed to the task's PPCI receiver in register three. The CNP contains pointers to a status queue where PPCI status is posted. The status queue will be located immediately after the notification packet in memory. The default size of the status buffer is four doublewords. The user can specify the number of doublewords desired in the status buffer in the first byte of word 15 in the FCB.

When a PPCI is received by the handler's SI entry point, bit eight of IOQ.CONT is checked to see if a PPCI receiver is present. If a PPCI receiver is not present, the interrupt level is exited without any status being posted. If a PPCI receiver is present, the status received from the PPCI is posted in the status buffer that follows the notification packet. When status is posted into the queue, the total PPCI received count (NOT.STAR) is incremented by one and the address of next status location to use (NOT.STPT) is incremented by two words. If NOT.STPT points past the end of the status buffer, NOT.STPT is reset to point to the start of the buffer.

After status is posted, the handler checks to see if the notification packet is on the task interrupt queue in the task's DQE. If it is, the SI level is exited without issuing an end-action request. If the notification packet is not on the task interrupt queue, a request for end-action processing is made and the SI level is exited. If a task is at end-action level, it is possible that PPCI interrupts can occur which will cause status to be posted and another end-action request to be issued.

It is the task's responsibility to keep track of the number of PPCIs previously processed by the end-action routine. The task must also keep track of the position in the status

buffer of the last status that was previously processed. This information will allow the end-action routine to know where the present status starts and whether any status was overwritten in the buffer (status was overwritten if the interrupts received minus the interrupts processed are greater than the length of the buffer).

The task uses the no-wait end-action return (SVC 1,X'2C') to exit the PPCI end-action level in the same manner that any no-wait end-action routine is exited.

5.11.2.4 Restrictions

The task must be privileged to execute a physical channel program and it must execute a physical channel program in no-wait mode to be able to have a PPCI end-action receiver.

The use of the TIC command to create loops is prohibited for logical channel programs. The use of the TESTSTAR and READ BACKWARD commands are also prohibited in logical channel programs.

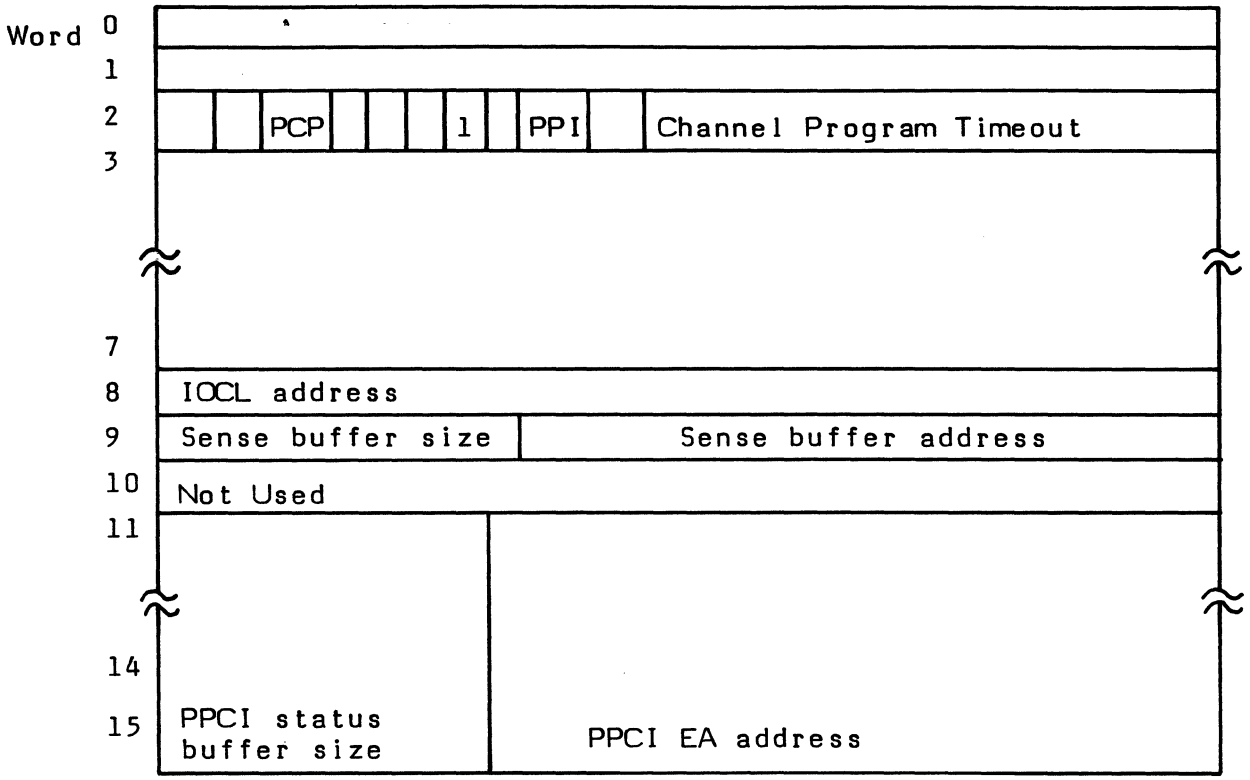
If the task does not want device I/O between channel programs, it must reserve the device exclusively while it is using it. This can be done by assigning the device for exclusive use or requesting an exclusive resource lock after the device is allocated. With this method there is no way to reserve a controller or channel.

If the task wants SENSE information to be returned on an error condition, it must set up a SENSE buffer to store the information. The SENSE buffer address must be placed in bits 8 - 31 of word 9 of the expanded FCB being used for the I/O. The length of the SENSE buffer, in bytes, must be placed in bits 0 - 7 of word 9 of the expanded FCB. If SENSE information is not wanted on error, word 9 of the FCB must be zero.

5.11.3 Setting Up File Control Blocks for EXCPM Requests

The differences between an FCB used for normal device-independent I/O and an FCB used for an EXCPM request are shown in Table 5-10. The EXCPM FCB must be an expanded FCB.

Table 5-10
Execute Channel Program FCB Format



Word 2:

Bit	Description
2	Contains the Physical Channel Program PCP) if set. If not set, contains the Logical Channel Program.
8	Contains the PPCI end-action receiver if set. If not set, there is no PPCI end-action receiver.

Word 8 contains the address of channel program to be executed

Word 9:

Bits	Description
0-7	Contain the size of user-supplied sense buffer
8-31	Contain the address of user-supplied sense buffer

Word 15:

Bits	Description
0-7	Contain the size of the PPCI status buffer
8-31	Contain the address of the PPCI end-action receiver

5.11.4 Postprogram Controlled Interrupt Notification Packet

If a task sets up a PPCI end-action receiver to check status during execution of its channel program, the status is returned in a notification packet. The address of the notification packet is contained in register three upon entering the task's PPCI end-action receiver. The notification packet is described in Table 5-11.

Table 5-11
Notification Packet Layout for PPCI Receiver

	0 7	8 15	16 23	24 31
Word 0	String forward address (NOT.SFA)			
1	String backward address (NOT.SBA)			
2	Link priority (NOT.PRI)	NOT.TYPE. See Note 1	Reserved	
3	FCB address (NOT.CODE)			
4	PSD 1 of task's PPCI receiver (NOT.PSD1)			
5	PSD 2 of task's PPCI receiver (NOT.PSD2)			
6	Number of PPCIs received since last buffer clear (NOT.STAR)		Number of status doublewords in status buffer (NOT.STAS)	
7	Address of PPCI status buffer (NOT.STAA)			
8	Address of buffer storing next status doubleword (NOT.STPT)			
9	Reserved			
10	PPCI status buffer			
n				

Notes:

1. NOT.TYPE - Set to one for asynchronous notification.
2. Words zero to nine are updated by the operating system and must not be changed by the user.

5.11.5 Macros (M.FCBEXP)

The M.FCBEXP macro can define a File Control Block (FCB) for an Execute Channel Program request.

Syntax:

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

label	ASCII string to use as symbolic label for address of FCB
lfc	Logical file code; word 0, bits 8 - 31 of the FCB
cpaddr	Logical address of channel program to be executed
tout	Time-out value specified in seconds
PCP	Specifies physical channel program
NWI	Specifies no-wait I/O request
NST	No status checking. All I/O appears to complete without error.
ssize	Size of user-specified sense buffer
sbuffer	Address of user-specified sense buffer
nowait	Normal no-wait end-action return address
nowaiterror	No-wait end-action error return address
waiterror	Wait I/O end-action error return address
psize	Size of PPCI status buffer to use
ppciaddr	PPCI end-action address

5.12 Resident Executive Services (H.REXS)

The H.REXS module replaces H.MONS. It provides interfaces for all major system services.

The following seven H.MONS entry points do not have equivalent H.REXS entry points and, therefore, are still valid by their H.MONS calls.

SVC	Macro	Entry Point	Description
1,X'40'	M.ALOC	H.MONS,21	Allocate file or peripheral device
1,X'41'	M.DALC	H.MONS,22	Deallocate file or peripheral device
1,X'42'	M.PDEV	H.MONS,1	Physical device inquiry
		H.MONS,2	Reserved
1,X'61'	M.CDJS	H.MONS,27	Submit job from disc file
1,X'73'	M.LOG	H.MONS,33	Permanent file log
1,X'74'	M.USER	H.MONS,34	User name specification

These entry points are included for compatibility purposes only and are described in Section 6.4. The H.REXS services, run faster than these H.MONS services and support more capabilities available in MPX-32.

All other H.REXS entry points perform the equivalent function as their H.MONS counterparts transparent to the user. See Section 6.2 for H.REXS system service descriptions.

5.13 Resource Management (H.REMM)

The H.REMM module replaces H.ALOC. It performs the allocation and assignment of all system resources and is responsible for maintaining proper access compatibility and usage rights for these resources, such as files, directories, partitions, devices, and memory. The mechanisms for coordinating concurrent access to shared resources are also contained within this module.

Whenever H.REMM services are called by their macro names, any optional parameter not specified in the call is handled as follows:

- The appropriate register is assumed to have been previously loaded
  (or)
- The appropriate register will be zeroed

See the calling sequence description of each service to determine applicability of missing parameter handling.

The following five H.ALOC entry points do not have equivalent H.REMM entry points and, therefore, are still valid by their H.ALOC calls:

SVC	Macro	Entry Point	Description
1,X'1F'	M.SMULK	H.ALOC,19	Unlock and dequeue shared memory
1,X'71'	M.SHARE	H.ALOC,12	Share memory with another task
1,X'72'	M.INCL	H.ALOC,13	Get shared memory (include)
1,X'79'	M.EXCL	H.ALOC,14	Free shared memory (exclude)
N/A	N/A	H.ALOC,17	Allocate file by space definition

These entry points are included for compatibility purposes only and are described in Section 6.4. The H.REMM services run faster than these H.ALOC services and support more capabilities available in MPX-32.

All other H.REMM entry points perform the equivalent function as their H.ALOC counterparts transparent to the user. See Section 6.2 for H.REMM system service descriptions.

5.14 Volume Management Module (H.VOMM)

The Volume Management Module (H.VOMM) is responsible for manipulating the MPX-32 volume resident and related memory resident data structures in order to allow for creation, deletion, and maintenance of user and system resources that reside on MPX-32 volumes.

Resources that reside on MPX-32 volumes include temporary files, directories, and the permanent files and memory partition definitions associated with the directories.

The volume resident data structures manipulated by H.VOMM include resource descriptors (RDs), the resource descriptor allocation map (DMAP), the file space allocation map (SMAP), the volume's directories, and their directory entries. The memory resident data structures include the Mounted Volume Table (MVT) and the File Assignment Table (FAT).

5.14.1 H.VOMM Conventions

5.14.1.1 Entry Point Conventions

Entry points 1 through 16 and 23 are provided as user level services and are accessible through unprivileged SVC calls.

Entry points 17 through 22, 24, and 25 are meant for MPX-32 system usage, including internal H.VOMM usage, and are accessible only to privileged callers with M.CALL.

5.14.1.2 Pathnames

Pathnames are used to uniquely identify a volume resident resource by explicitly or implicitly describing the volume, one or more directories, and the resource name. Pathnames consist of variable length ASCII character strings that are resolved on-line by H.VOMM in order to locate a resource.

H.VOMM supports a two level directory structure per volume including the root directory (level 1), and user directories (level 2). In the examples which follow, VOL means one- to sixteen-character volume name, DIR means a one- to sixteen-character directory name, and FILE means a one- to sixteen-character resource name.

Example 1

```
@VOL(DIR)FILE
```

The @VOL component defines the volume where the resource FILE is located and implies that directory DIR should be located by the named volume's root directory. Furthermore, resource FILE should be located by directory DIR.

Example 2

```
FILE
```

The missing volume and directory components imply that resource FILE should be located by the user's current working volume and directory. The user's default current working volume and directory are established when the user enters the system and can be changed by the user.

Example 3

```
^(DIR)FILE
```

The ^ implies that the directory DIR should be located by the root directory of the user's current working volume, and that resource FILE should be located by user directory DIR.

Example 4

```
@SYSTEM(SYSTEM)FILE
```

The volume name SYSTEM and directory name SYSTEM are reserved names that identify the current system (IPL) volume and the special system directory on that volume.

5.14.1.3 Pathname Blocks

The pathname block is an alternative form of a pathname which may be used interchangeably with pathnames. Its most important characteristic is that, because of its structure, it can be parsed faster than a pathname. The pathname block is a doubleword bounded, variable length ASCII character string which H.VOMM can distinguish from a pathname since the pathname block always starts with an exclamation point.

H.VOMM provides a service to convert a pathname to a pathname block. The examples which follow illustrate common pathnames and their corresponding pathname blocks.

Example 1

@VOL1(DIR1)FILE1

WORD	
0	! V O L
1	blank
2	V O L 1
3	blank
4	blank
5	blank
6	! D I R
7	R O O T
8	D I R 1
9	blank
10	blank
11	blank
12	! R E S
13	blank
14	F I L E
15	1 Ø Ø Ø
16	blank
17	blank

Example 2

FILE1

WORD	
0	! V O L
1	W O R K
2	! D I R
3	W O R K
4	! R E S
5	blank
6	F I L E
7	1 Ø Ø Ø
8	blank
9	blank

Example 3

(DIRECTORY)MYFILE

WORD	
0	! V O L
1	W O R K
2	! D I R
3	R O O T
4	D I R E
5	C T O R
6	Y Ø Ø Ø
7	blank
8	! R E S
9	blank
10	M Y F I
11	L E Ø Ø
12	blank
13	blank

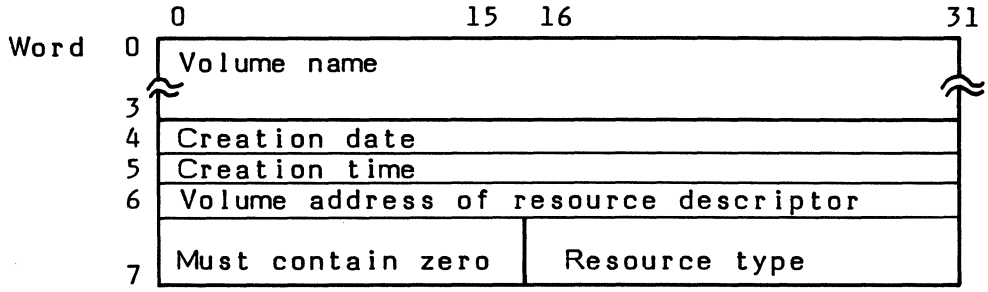
Example 4

@SYSTEM(SYSTEM)LOADMOD

WORD	
0	! V O L
1	S Y S T
2	! D I R
3	S Y S T
4	! R E S
5	blank
6	L O A D
7	M O D ⌀
8	blank
9	blank

5.14.1.4 Resource Identifiers

The fastest means of locating a volume resource (once created) is by its resource identifier (must be on a doubleword boundary). The resource identifier has the following format:



Since the resource identifier contains the volume address of the resource descriptor, the resource descriptor (which points to and describes the resource) can be accessed directly without going through the various directories which would otherwise have to be traversed.

Given a valid pathname defining a resource, the corresponding resource descriptor may be retrieved by the H.VOMM locate resource service. The first eight words of a resource descriptor consist of the resource identifier.

5.14.1.5 Allocation Units

File space is allocated in allocation units. Allocation units are integral multiples of 192-word disc blocks. Allocation unit size is a volume parameter which is specified when the volume is formatted.

User calls to unprivileged H.VOMM entry points that allocate file space expect the amount of file space to be specified in terms of 192-word disc blocks. H.VOMM then rounds the number of blocks requested up to the nearest number of allocation units.

5.14.1.6 File Segment Definitions

The space associated with a file consists of one or more file segments. A file segment is a set of contiguous allocation units on a volume. When a file is created, its space consists of one or more file segments. As the file is expanded, new file segments become associated with the file. Different segments within one file are almost always discontiguous.

A file segment is defined by a two word file segment definition whose format follows:

Word 0	Absolute 192W block volume segment address
1	Segment length in 192W blocks

Up to 32 file segment definitions can be stored in one resource descriptor. A file cannot expand beyond 32 segments.

5.14.2 Calling/Return Parameter Conventions

5.14.2.1 Unused Register

No calling arguments are passed to an H.VOMM entry point by register three. Therefore, register three is an unused register which is preserved across all H.VOMM entry point calls.

5.14.2.2 Specifying a Volume Resource

Whenever an H.VOMM entry point requires a volume resource to be identified (except H.VOMM,9) only one register is required regardless of whether pathname, pathname block, resource identifier, or logical file code (if the resource is assigned and opened) is supplied. Therefore, the following conventions are assumed by H.VOMM:

Identifier Supplied	Calling Register Format
Pathname Vector	0 … 7 | 8 … 31: Pathname length in bytes | Pathname address
Pathname Block Vector	0 … 7 | 8 … 31: Pathname block length in bytes | Pathname Block Address
Resource ID Vector	0 … 7 | 8 … 31: Resource ID length = 32 | Resource ID Address
Logical File Code	0 … 7 | 8 … 31: Must be zero | LFC
File Control Block	0 … 7 | 8 … 31: Must be zero | FCB Address

5.14.2.3 Status Codes

All H.VOMM entry points supply a status code to the caller indicating whether the entry point operation was successful and, if not, why not. A status code summary follows:

Status Code	Indication
0	Operation successful
1	Pathname invalid
2	Pathname consists of volume only
3	Volume not mounted
4	Directory does not exist
5	Directory name in use
6	Directory creation not allowed at specified level
7	Resource does not exist
8	Resource name in use
9	Resource descriptor unavailable
10	Directory entry unavailable
11	Required file space unavailable
12	Unrecoverable I/O error while reading DMAP
13	Unrecoverable I/O error while writing DMAP
14	Unrecoverable I/O error while reading resource descriptor
15	Unrecoverable I/O error while writing resource descriptor
16	Unrecoverable I/O error while reading SMAP
17	Unrecoverable I/O error while writing SMAP
18	Unrecoverable I/O error while reading directory
19	Unrecoverable I/O error while writing directory
20	Project group name or key invalid
21	Reserved
22	Invalid FCB or LFC
23	Parameter address specification error
24	Resource descriptor not currently allocated
25	Pathname block overflow
26	File space not currently allocated
27	Change defaults not allowed
28	Cannot access resource in requested mode or default system image file cannot be deleted
29	Operation not allowed on this resource type
30	Required parameter was not specified
31	File extension denied; segment definition area full
32	File extension denied; file would exceed maximum size allowed
33	I/O error occurred when resource was zeroed
34	Replacement file cannot be allocated
35	Invalid directory entry
36	Directory and file not on same volume
37	An unimplemented entry point has been called
38	Replacement file is not exclusively allocated to the caller
39	Out of system space
40	Cannot allocate FAT/FPT when creating a temporary file
41	Deallocation error in zeroing file
42	Resource descriptor destroyed or the resource descriptor and the directory entry linkage has been destroyed
43	Invalid resource specification

Status Code	Indication
44	Internal logic error from Resource Management Module (H.REMM). Abort task, try a different task. If it fails, reboot system.
45	Attempted to modify more than one resource descriptor at the same time or attempted to rewrite resource descriptor prior to modifying it
46	Unable to obtain resource descriptor lock (multiprocessor only)
47	Directory contains active entries and cannot be deleted
48	A resource descriptor's link count is zero
49	Attempting to delete a permanent resource without specifying a pathname or pathname block vector
50	Resource descriptor contains unexpected resource descriptor type

In some cases, H.VOMM also displays H.REMM abort conditions. If a user calls an H.VOMM service that calls an H.REMM service for processing and an abort condition occurs within the H.REMM processing, the abort condition is returned to H.VOMM. H.VOMM displays it to the user in the format 10xx where xx is the specific H.REMM abort condition. For example, abort condition 1026 indicates H.REMM error 26 has occurred. The TSM $ERR directive can determine the reason for the error, for example $ERR RM26.

5.14.2.4 Caller Notification Packet (CNP)

The CNP format is described in the H.REMM documentation. If a call to an H.VOMM entry point is accompanied by a CNP, the entry point status is always posted in the CNP. In addition, the caller can use the CNP to supply a denial return address that is to be taken if status is nonzero. If a CNP accompanies an H.VOMM call, status is nonzero, and no denial return address is supplied, a normal return is taken.

A CNP can be used to specify one or more options which are unique to the H.VOMM entry point called. To determine whether options apply, see the individual H.VOMM entry point descriptions.

If a CNP is not supplied, entry point status is always posted in register seven, a normal return is always taken, and no options may be specified.

In any case, whether a CNP is supplied or not, CC1 is always set if return status is nonzero.

5.14.2.5 Pathnames/Pathname Blocks

Whenever an H.VOMM entry point returns a nonzero status when it cannot completely resolve a pathname or pathname block, the address of the first unresolvable item within the pathname or pathname block is returned to the caller.

5.14.2.6 Resource Create Block (RCB)

Each H.VOMM entry point that creates a permanent file, a temporary file, a memory partition, or a directory may receive a resource create block (RCB) in order to fully define the attributes of the resource that is created. RCB formats are described in Tables 5-12, 5-13, and 5-14.

If an RCB is not supplied by the caller, the resource is created with the default attributes described in Chapter 4.

Table 5-12.
Permanent and Temporary File Resource Create Block (RCB)

Word	
0 1	File owner name (RCB.OWNR)
2 3	File project group name (RCB.USER)
4	Owner rights specifications (RCB.OWRI). See Note 1.
5	Project group rights specifications (RCB.UGRI). See Note 1.
6	Other's rights specifications (RCB.OTRI). See Note 1.
7	Resource management flags (RCB.SFLG). See Note 2.
8	Maximum extension increment (RCB.MXEI). See Note 3.
9	Minimum extension increment (RCB.MNEI). See Note 4.
10	Maximum file size (RCB.MXSZ). See Note 5.
11	Original file size (RCB.OSIZ). See Note 6.
12	File starting address (RCB.ADDR). See Note 7.
13	File ID location (RCB.FAST). See Note 8.
14	Option flags (RCB.OPTS). See Note 9.
15	Default override (RCB.FREE). See Note 10.

Notes:

1. Rights specifications are optional:

Bit	Description
0	Read access allowed (RCB.READ)
1	Write access allowed (RCB.WRIT)
2	Modify access allowed (RCB.MODI)
3	Update access allowed (RCB.UPDA)
4	Append access allowed (RCB.APPN)
9	Delete access allowed (RCB.DELE)

2. Resource management flags. For any bit not set, system defaults apply and, in some cases, the default is the equivalent of the bit being set (optional):

Bit	Description
0-7	Resource type, equivalent to file type code, interpreted as two hexadecimal digits, 0 - FF (RCB.FTYP)
11	File EOF management required (RCB.EOFM)
12	Fast access (RCB.FSTF)
13	Do not save (RCB.NSAV)
23	Zero file on creation/extension (RCB.ZERO)
24	File automatically extendible (RCB.AUTO)
25	File manually extendible (RCB.MANU)
26	File contiguity desired (RCB.CONT)
27	Shareable (RCB.SHAR) (owner rights spec only)
31	File data initially recorded as blocked (RD.BLOCK)

3. Maximum extension increment is desired file extension increment specified in blocks (optional). Default is 64 blocks.

4. Minimum extension increment is the minimum acceptable file extension increment specified in blocks (optional). Default is 32 blocks.

5. Maximum file size is the maximum extendible size for a file specified in blocks (optional).

6. Original file size is the original file size specified in blocks (optional). Default is 16 blocks.

7. File starting address is the disc block where the file should start, if possible. If the space needed is currently allocated, an error is returned (optional).

8. File ID location is the location in the file creator's task where the eight word resource identifier (RID) is to be returned if fast access is required (optional).

9. Option flags bits are as follows:

Bit	Description
0	Owner has no access rights (RCB.OWNA)
1	Project group has no access rights (RCB.USNA)
2	Others have no access rights (RCB.OTNA)

10. Default override - If set, these bits override any corresponding bit set in RCB.SFLG and the system defaults (optional):

Bit	Description
0-7	Must be zero
11	File EOF management not required
12	Fast access not required
13	Resource can be saved
23	Do not zero file on creation/extension
24	File is not automatically extendible
25	File is not manually extendible
26	File contiguity is not desired
27	Resource is not shareable
31	File data initially recorded as unblocked

Table 5-13.
Directory Resource Create Block (RCB)

Word	
0	Directory owner name (RCB.OWNR)
1	
2	Directory project group name (RCB.USER)
4	Owner rights specifications (RCB.OWRI). See Note 1.
5	Project group rights specifications (RCB.UGRI). See Note 1.
6	Other's rights specifications (RCB.OTRI). See Note 1.
7	Resource management flags (RCB.SFLG). See Note 2.
8	Reserved
10	
11	Directory original size (RCB.OSIZ). See Note 3.
12	Directory starting address (RCB.ADDR). See Note 4.
13	Directory ID location (RCB.FAST). See Note 5.
14	Option flags (RCB.OPTS). See Note 6.
15	Default override (RCB.FREE). See Note 7.

Notes:

1. Rights specifications bits are as follows:

Bit	Description
0	Read access allowed (RCB.READ)
8	Directory may be traversed (RCB.TRAV)
9	Directory may be deleted (RCB.DELE)
10	Directory entries may be deleted (RCB.DEEN)
11	Directory entries may be added (RCB.ADEN)

2. Resource management flags are optional:

Bit	Description
13	Do not save (RCB.NSAV)
27	Shareable (RCB.SHAR)

3. Directory original size is the number of entries required (optional).

4. Directory starting address is the disc block number where the directory should start, if possible. If the space needed is currently allocated, an error is returned (optional).

5. Directory ID location is the location in the directory creator's task where the eight word resource identifier (RID) is to be returned if fast access is required (optional).

6. Option flags are as follows:

Bit	Description
0	Owner has no access rights (RCB.OWNA)
1	Project group has no access rights (RCB.USNA)
2	Others have no access rights (RCB.OTNA)

7. If default override is set, these bits override any corresponding bit set in RCB.SFLG and the system defaults (optional).

Bit	Description
0-7	Must be zero
13	Resource can be saved
27	Resource is not shareable

Table 5-14.
Nonbase Mode Memory Partition Resource Create Block (RCB)

Word	
0 1	Partition owner name (RCB.OWNR)
2 3	Partition project group name (RCB.USER)
4	Owner rights specifications (RCB.OWRI). See Note 1.
5	Project group rights specifications (RCB.UGRI). See Note 1.
6	Other's rights specifications (RCB.OTRI). See Note 1.
7	Resource management flags (RCB.SFLG). See Note 2.
8	Reserved
≈	≈
10	
11	Partition original size (RCB.OSIZ). See Note 3.
12	Partition starting address (RCB.ADDR). See Note 4.
13	Partition ID location (RCB.FAST). See Note 5.
14	Option flags (RCB.OPTS). See Note 6.
15	Default override (RCB.FREE). See Note 7.

Notes:

1. Rights specifications are optional:

Bit	Description
0	Read access allowed (RCB.READ)
1	Write access allowed (RCB.WRIT)
9	Delete access allowed (RCB.DELE)

2. Resource management flags are optional:

Bit	Description
13	Do not save (RCB.NSAV)

3. Partition's original size is the number of protection granules required.

4. Partition's starting address is a 512-word protection granule number in the user's logical address space where the partition is to begin.

5. Partition's ID location is a location in the partition creator's task where the eight word resource identifier (RID) is to be returned if fast access is required (optional).

6. Option flags are optional:

Bits	Description
0	Owner has no access rights (RCB.OWNA)
1	Project group has no access rights (RCB.USNA)
2	Others have no access rights (RCB.OTNA)
24-31	Define memory class. Values are:

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

7. If set, these bits override any corresponding bit set in RCB.SFLG and the system defaults (optional):

Bits	Description
0-7	Must be zero
13	Resource can be saved

5.14.3 Bad Block Handling

The process of initializing a volume with the media diagnostic and verification program produces information describing any sections of a disc that may not be used. This information may describe defective areas on the disc or areas that are reserved for use by the diagnostic hardware built into some disc drives.

This information is written to the disc and later read by J.VFMT and J.MOUNT. Using this information, these two programs mark all allocation units containing defective or reserved areas as allocated. This prevents H.VOMM from allocating space in a defective or reserved area.

5.14.4 Services

Whenever H.VOMM services are called by their macro names, any optional parameter not specified in the call is handled as follows:

- The appropriate register is assumed to have been previously loaded
  (or)
- The appropriate register will be zeroed

See the calling sequence description of each H.VOMM service in Section 6.2 to determine applicability of missing parameter handling.

CHAPTER 6

NONBASE MODE SYSTEM SERVICES

6.1 General Description

MPX-32 offers a set of resident nonbase mode system service routines designed to perform frequently required operations with maximum efficiency. Using the Supervisor Call instruction, tasks running in any environment can call these routines.

All system service routines are reentrant. Thus, each service routine is always available to the task which is currently active.

System service routines are provided as standard modular components of the Mapped Programming Executive. The open-ended design of the system, however, gives each user freedom to add whatever service routines are required to tailor MPX-32 to a specific application.

System services enable tasks to:

- Activate, suspend, resume, abort, terminate and hold task execution
- Change a task's priority level
- Create, test, and delete timers
- Interrogate system clocks
- Allocate and deallocate devices and files
- Obtain the characteristics of a device or file
- Communicate with other tasks with messages and status words
- Load and execute overlays
- Obtain information on the memory assigned to a task
- Connect tasks to interrupts
- Interrogate the arithmetic exception and option word status for a task

MPX-32 services are implemented as SVC traps. There are several ways of accessing services:

- By macro calls, with parameter passing as indicated. The expansion code in the system macro library is then accessed automatically during assembly to provide Assembly language setup of appropriate registers and instructions including SVCs, in the user's code.

- By setting up appropriate registers and instructions directly and using appropriate SVCs.

- By following the course above but issuing an M.CALL request to the entry point of the system module that provides the service.

The first two access paths are described for each system service. The third access path is privileged, and is indicated primarily to provide the appropriate system module names and entry point numbers for cross-reference to other documentation when needed.

Special operations performed for a task are:

- Open - If not issued by the task, IOCS opens the file or device for the default access in effect at that time.
- Close - If not issued by the task, the file is closed automatically and a device is deallocated automatically during task termination.

All system services are arranged alphabetically by their macro name. System services that are not macro callable, but are available to the user, are described in Section 6.3.

Appendix B provides cross-reference charts.

6.1.1 RTM System Services Under MPX-32

MPX-32 accepts call monitor (CALM) instructions that are syntactically and functionally equivalent to the RTM CALMs. These CALMs are implemented for compatibility purposes only. It is recommended that the MPX-32 system services be used instead of CALMs because they run faster and support the capabilities available in MPX-32. Mixing CALMs and SVCs in the same program element is not encouraged.

Generally, RTM CALMs operate under MPX-32 without any change in syntax or function. A few seldom-used CALMs have been deleted, and others can have additional restrictions applied to them. In general, however, the changes to the user's source code should be minimal in the conversion from RTM to MPX-32.

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 CALMs have been deleted in MPX-32:

- CALM X'62' — Unlink Dynamic Job Queue Entry (not required in MPX-32) M.DDJS or CALL M:UNLKJ
- CALM X'63' — Activate with Core Append (replaced by memory expansion and contraction services of MPX-32) (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 re-entrant run time library)

All Random Access Calls (MPX-32 does not support H.DRAH):

- CALM X'59' Random Access OPEN (CALL OPEN)
- CALM X'5A' Random Access READ (CALL READ)
- CALM X'5B' Random Access WRITE (CALL WRITE)
- CALM X'5C' Random Access File (CALL DEFINE)
- CALM X'5D' Random Access File (CALL FIND)

TSS CALM:

MPX-32 replaces TSS with TSM, an on-line support package. Therefore, all TSS CALM's X'80' - X'84' have been deleted.

On a CONCEPT/32 computer, 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 is generated when a doubleword operation code is used with an incorrectly bounded operand; therefore, coding changes are required when a trap occurs.

6.1.2 System Services - Syntax Rules and Descriptions

All system services can be called by their macro name, their SVC number, or their module entry point number. It is recommended that whenever possible the macro name be used. When a macro name is used, any optional parameter not specified in the call is handled as follows:

- the appropriate register is assumed to have been previously loaded

  (or)

- the appropriate register will be zeroed

See the Calling Sequence description of each service to determine applicability of missing parameter handling.

Defaults for optional parameters are documented in the description of the individual services.

When a required parameter is not specified or an invalid parameter is specified, an error message is displayed in the listing regardless of the listing controls in effect.

6.1.3 IPU Executable Nonbase Mode System Services

Once a task has gained entry into the IPU, there is a limited set of system services that the IPU can execute. These are memory reference only system services, since the IPU can not execute any I/O instructions. The following nonbase mode system services are executable in the IPU:

SVC	DESCRIPTION
M.ADRS	Memory Address Inquiry
M.BBTIM	Acquire Current Date/Time in Byte Binary Format

SVC	DESCRIPTION
M.BTIM	Acquire Current Date/Time in Binary Format
M.CMD	Get Command Line
M.CONABB	Convert ASCII Date/Time to Byte Binary Format
M.CONADB	Convert ASCII Decimal to Binary
M.CONAHB	Convert ASCII Hexadecimal to Binary
M.CONASB	Convert ASCII Date/Time to Standard Binary
M.CONBAD	Convert Binary to ASCII Decimal
M.CONBAF	Convert Binary Date/Time to ASCII Format
M.CONBAH	Convert Binary to ASCII Hexadecimal
M.CONBBA	Convert Byte Binary Date/Time to ASCII
M.CONBBY	Convert Binary Date/Time to Byte Binary
M.CONBYB	Convert Byte Binary Date/Time to Binary
M.CTIM	Convert System Date/Time Format
M.DATE	Date and Time Inquiry
M.DEVID	Get Device Mnemonic or Type
M.DSMI	Disable Message Task Interrupt
M.DSUB	Disable User Break Interrupt
M.ENUB	Enable User Break Interrupt
M.ENVRMT	Get Task Environment
M.GTIM	Acquire System Date and Time in any Format
M.PGOW	Task Option Word Inquiry
M.QATIM	Acquire Current Date/Time in ASCII Format
M.SYNCH	Set Synchronous Task Interrupts
M.TDAY	Time-of-Day Inquiry
M.TSTE	Arithmetic Exception Inquiry
M.TSTT	Test Timer Entry

6.2 Macro-Callable System Services

All nonbase mode system services are described in detail in the pages which follow, arranged alphabetically by their macro name. System services which are supported for nonbase mode tasks are prefaced by the characters 'M.'.

6.2.1 M.ACTV - Activate Task

The M.ACTV service activates a task. The task assumes the owner name of the caller. When a load module is supplied as input, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied as input.

The base mode equivalent service is M_ACTV.

Entry Conditions

Calling Sequence:

```
M.ACTV    loadmod

(or)

LD        R6,loadmod
SVC       1,X'52' (or) M.CALL   H.REXS,15
```

loadmod is either a left-justified blank-filled doubleword containing the one- to eight-ASCII character name of the load module for which an activation request is queued (must be a system file), or R6 is zero and R7 contains the pathname vector or RID vector which points to the load module to be activated.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6	Equals zero if the service could be performed
R7	Contains the task number of task activated by this service

(or)

R6	Equals one if invalid attempt to multicopy a unique load module
R7	Task number of existing task with same name

(or)

R6 Value	Description
2	Load module file not in directory
3	Unable to allocate load module
4	File is not a valid load module
5	DQE is not available
6	Read error on resource descriptor
7	Read error on load module
8	Insufficient logical/physical address space for task activation

Abort Cases:

None

Output Messages:

None

6.2.2 M.ADRS - Memory Address Inquiry

The M.ADRS service provides the beginning and ending logical addresses of the memory allocated to a task. The beginning address is the location into which the first word was loaded and is a word address. The ending address is also a word address and defines the last word allocated to the task.

This service can be executed by the IPU.

The base mode equivalent service is M_ADRS.

Entry Conditions

Calling Sequence:

```
M.ADRS

(or)

SVC     1,X'44' (or) M.CALL  H.REXS,3
```

Exit Conditions

Return Sequence:

```
M.IPURTN  6,7
```

Registers:

R6	Logical word address of the first location of the task's DSECT. This address is always on a page boundary.
R7	Logical word address of the last location available for loading or expansion of the task's DSECT. This address is always on a map block boundary minus one word.

Abort Cases:

None

Output Messages:

None

6.2.3 M.ANYW - Wait for Any No-Wait Operation Complete, Message Interrupt, or Break Interrupt

The M.ANYW service places the currently executing task in a state waiting for the completion of any no-wait request, for the receipt of a message, or for a break interrupt. The task is removed from the associated ready-to-run list, and placed in the any-wait list. A return is made to the program location following the SVC instruction only when one of the wait conditions has been satisfied or when the optional time-out value has expired.

The base mode equivalent service is M_ANYWAIT.

Entry Conditions

Calling Sequence:

```
M.ANYW timel

(or)

LW      R6,timel
SVC     1,X'7C' (or) M.CALL  H.REXS,37
```

timel contains zero if wait for an indefinite period is requested. Otherwise, timel contains the negative number of time units to elapse before the wait is terminated.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

MS31	User attempted to go to an any wait state from an end-action routine

Output Messages:

None

6.2.4 M.ASSN - Assign and Allocate Resource

The M.ASSN service associates a resource with a Logical File Code (LFC) used by a process and to allocate the resource. This function creates a FAT/FPT pair within the user's TSA and associates an Allocated Resource Table (ART) entry for system administration and control of the resource while allocated. When implicit sharing is indicated by the absence of a specified usage mode, the appropriate linkage is established to coordinate concurrent access. The option is provided to allocate and open a resource via a single call to this function.

The base mode equivalent service is M_ASSIGN.

Entry Conditions

Calling Sequence:

```
M.ASSN  rrsaddr [,cnpaddr]

(or)

LA      R1,rrsaddr
LA      R7,cnpaddr  (or)  ZR   R7
SVC     2,X'52'  (or)  M.CALL   H.REXS,21
```

rrsaddr is the address of an RRS entry (Type 1 - 6).

cnpaddr is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, options field, status field, and parameter link.

The option field contains an access and usage specification for opening of this resource. This field is used only if the automatic open flag is set in the option word of the RRS. See M.OPENR service.

If automatic open is indicated in the Resource Requirement Summary, word 5 of the CNP must contain the address of a valid File Control Block (FCB) for this assignment. See M.OPENR service.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R5	M.RTRN R5
(or)	(or)
M.RTRN R5 (CC1 set)	M.RTRN R5,R7 (CC1 set)

Registers:

R5 Allocation index; a unique 32-bit integer number associated with the allocated resource. This index can set and release resource locks for exclusive or synchronous access.

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
1	Unable to locate resource (invalid pathname)
2	Specified access mode not allowed
3	FAT/FPT space not available
4	Blocking buffer space not available
7	Static assignment to dynamic common
8	Unrecoverable I/O error to volume
9	Invalid usage specification
11	Invalid RRS entry
12	LFC logically equated to unassigned LFC
13	Assigned device not in system
14	Resource already allocated by requesting task
15	SGO or SYC assignment by real-time task
17	Duplicate LFC assignment attempted
19	Invalid resource ID
20	Specified volume not assigned
22	Resource is marked for deletion
23	Assigned device is marked off-line
24	Segment definition allocation by unprivileged task
25	Random access not allowed for this access mode
27	Resource already opened in a different access mode
28	Invalid access specification at open
29	Specified LFC is not assigned to a resource for this task
38	Time out occurred while waiting for resource to become available
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task
51	Shareable resource is allocated in an incompatible access mode
54	Unable to allocate resource for specified usage
55	Allocated Resource Table (ART) space not available

Note: Status values 25-29 are returned only when auto-open is indicated.

Wait Conditions

When the resource is not available (status values 50-63), the task is placed in a wait state, as appropriate, if specified by a CNP.

6.2.5 M.ASYNCH - Set Asynchronous Task Interrupt

The M.ASYNCH service resets the asynchronous task interrupt mode back to the default environment.

The base mode equivalent service is M_ASYNCH

Entry Conditions

Calling Sequence:

```
M.ASYNCH

(or)

SVC     1,X'1C'     (or)     M.CALL     H.REXS,68
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

CC1 set	Asynchronous task interrupt already set

Abort Cases:

None

Output Messages:

None

6.2.6 M.BACK - Backspace Record or File

The M.BACK service is not applicable for SYC files or unblocked files. This service performs the following functions for backspacing records:

- If the file is output active, a purge is issued prior to the backspace function. After the specified number of records are backspaced, control returns to the user.
- Backspaces specified number of records.

M.BACK performs the following functions for blocked files:

- If the file is output active, an end-of-file and purge are issued prior to performing the backspace file function. Records are backspaced until an end-of-file record is found.
- The specified number of files are backspaced.
- The read/write control word then points to the end-of-file just encountered.

The base mode equivalent service is M_BACKSPACE.

Entry Conditions

Calling Sequence:

```
M.BACK  fcb, [R],[number]

(or)

LA      1,fcb
LNW     4,number
SVC     1,X'35' or M.CALL H.IOCS,9  (or)
SVC     1,X'36' or M.CALL H.IOCS,19
BIW     4,$-1W
```

fcb — is the FCB address

R — backspaces by record (SVC1,X'35'); else backspaces by file (SVC1,X'36')

number — address of word containing four times the number of records or files to backspace, or the contents of register four if not supplied

$-1W — branches back to SVC until reaches last word of register four

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cells for blocked file
IO09	Illegal operation on the SYC file

Output Messages:

None

6.2.7 M.BATCH - Batch Job Entry

The M.BATCH service submits a batch job stream located in a disc file. The disc file is described by the calling parameter in register one. Prior to calling this service, the specified disc file should be rewound to purge the contents of the blocking buffer if it has been dynamically built.

The base mode equivalent service is M_BATCH

Entry Conditions

Calling Sequence:

```
M.BATCH arga [,cnpaddr]

(or)

LW      R1,arga
LA      R7,cnpaddr (or) ZR R7
SVC     2,X'55' (or) M.CALL H.REXS,27
```

arga is a PN vector, PNB vector, or RID vector for a permanent file; or an LFC or FCB address for a temporary file

cnpaddr is a CNP address or zero if CNP is not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN R7 (CC1 set)

(or)

M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, denial address

Status:

CC1 set

Posted in R7 or in the status field of the CNP

Value	Description
0	Operation successful
1	Pathname invalid
2	Pathname consists of volume only
3	Volume not mounted
4	Directory does not exist
5	Disc file has not been previously opened

Value	Description
6	Unable to activate J.SSIN2, batch job not submitted
7	Resource does not exist
14	Unrecoverable I/O error while reading resource descriptor
18	Unrecoverable I/O error while reading directory

Abort Cases:

None

Output Messages:

None

6.2.8 M.BBTIM - Acquire Current Date/Time in Byte Binary Format

The M.BBTIM service acquires the system date and time in byte binary format. The date and time are returned in a two word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_BBTIM.

Entry Conditions

Calling Sequence:

```
M.BBTIM   addr

(or)

LA        R1,addr
ORMW      R1,=X'02000000'
SVC       2,X'50'  (or)  M.CALL   H.REXS,74
```

addr is the address of a two-word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.9 M.BORT - Abort Specified Task, Abort Self, or Abort with Extended Message

M.BORT - Specified Task

This service allows the caller to abort another task. If the named task has been swapped out, it is not aborted until it regains CPU control. See Chapter 2, Table 2-2. If the specified task is not in execution, the request is ignored.

The base mode equivalent service is M_BORT.

Entry Conditions

Calling Sequence:

```
M.BORT    abcode,task
(or)
LW        R5,abcode
ZR        R6          } (or) LD R6,taskname
LW        R7,taskno   }
SVC       1,X'56'     (or) M.CALL  H.REXS,19
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared.

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — zero if any of the following conditions exist:

- the specified task was not found
- the specified task name was not single copied
- the owner name of the task requesting the abort is not privileged and is restricted from access to tasks with a different owner name (by the M.KEY file)
- the task is in the process of exiting the system

Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

M.BORT - Self

This service aborts the calling task by issuing an abort message, optionally performing a post-abort dump, and performing the functions common to the normal termination service as described in Chapter 2.

Entry Conditions

```
M.BORT    abcode
(or)
LW        R5,abcode
SVC       1,X'57' (or) M.CALL  H.REXS,20
```

abcode contains the four-character ASCII abort code

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

task number ABORT AT: XXXXXXXX - YYYYY mm/dd/yy hh:mn:ss ZZZZ

task is the one-to eight-character name of the task being aborted

number is the task number of the task being aborted

XXXXXXXX is the location the abort occurred

YYYYY is the beginning of the DSECT

mm is the two-character decimal number of the month between 01 and 12

dd is the two-character decimal number of the day between 01 and 31

YY is the two-character decimal number of the year between 00 and 99

hh is the two-character decimal number of the hour between 00 and 23

mn is the two-character decimal number of the minutes between 00 and 59

ss is the two-character decimal number of the seconds between 00 and 59

ZZZZ is the four-character abort code

M.BORT - With Extended Message

A call to this service results in an abort of the specified task.

Entry Conditions

Calling Sequence:

```
M.BORT    abcode,task,extcode

(or)

LD        R2,extcode
LW        R5,abcode
LI        R6,0          } (or) LD R6,taskname
LW        R7,taskno     }
SVC       1,X'62' (or) M.CALL  H.REXS,28
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

extcode — contains the extended abort code message consisting of one to eight ASCII characters, left-justified and blank-filled

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — zero if any of the following conditions exist:

- the specified task was not found
- the specified task name was not single copied
- the owner name of the task requesting the abort is not privileged and is restricted from access to tasks with a different owner name (by the M.KEY file)
- the task is in the process of exiting the system

Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

6.2.10 M.BRK - Break/Task Interrupt Link/Unlink

The M.BRK service allows the caller to clear the user's break receiver or to establish the address of a routine to be entered whenever another task or the operator activates his task interrupt by an M.INT service.

The base mode equivalent service is M_BRK.

Entry Conditions

Calling Sequence:

```
M.BRK     brkadd

(or)

LA        R7,brkadd
SVC       1,X'6E' (or) M.CALL  H.REXS,46
```

brkadd is the logical word address of the entry point of the task's break/task interrupt routine or zero to clear the break receiver

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.11 M.BRKXIT - Exit from Task Interrupt Level

The M.BRKXIT service must be called at the conclusion of executing a task interrupt routine. It transfers control back to the point of interruption.

The base mode equivalent service is M_BRKXIT.

Entry Conditions

Calling Sequence:

```
M.BRKXIT

(or)

SVC     1,X'70' (or) M.CALL  H.REXS,48
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.12 M.BTIM - Acquire Current Date/Time in Binary Format

The M.BTIM service acquires the system date and time in binary format. The date and time are returned in a two word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_BTIM.

Entry Conditions

Calling Sequence:

```
M.BTIM    addr

(or)

LA        R1,addr
ORMW      R1,=X'01000000'
SVC       2,X'50'  (or)  M.CALL   H.REXS,74
```

addr is the address of a two-word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.13 M.CLOSER - Close Resource

The M.CLOSER service terminates operations in the current access mode on a resource. The resource is marked closed in the FPT. The user-count in the appropriate Allocated Resource Table (ART) entry is decremented if implicit shared use is in effect. For access modes other than read, the resource descriptor is updated. When the closing of a file implies a change of use or access mode for that resource, any tasks waiting for access to the resource in a compatible access mode are dequeued. If any logically equivalent resources are open, no further action is taken. For blocked files, any active output blocking buffer is purged. A close request to a resource that is already closed will result in an immediate return with the appropriate status posted.

The base mode equivalent service is M_CLOSER.

Entry Conditions

Calling Sequence:

```
M.CLOSER [fcbaddr] [,cnpaddr]

(or)

LA      R1,fcbaddr
LA      R7,cnpaddr (or) ZR  R7
SVC     2,X'43'  (or)  M.CALL   H.REMM,22
```

fcbaddr is the address of a File Control Block (FCB)

cnpaddr is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP

Value	Description
8	Unrecoverable I/O error to volume
29	Logical file code associated with FCB does not exist
31	Resource was not open
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

6.2.14 M.CLSE - Close File

The M.CLSE service marks a file closed in the File Pointer Table (FPT) and the count of open files (DFT.OPCT) is decremented. If any logically equivalent files (ASSIGN4) are open no further action is taken, for example, if count after decrementing is not equal to zero.

If the file is a system file or blocked file, purges any active output blocking buffer. The file is marked closed (open bit cleared in FAT).

For files assigned to SYC or SGO, the current disc address updates the Job Table for Job Control.

This service issues an EOF prior to purging system files SLO and SBO which were opened for read/write. Also issues an EOF prior to purging for blocked files which are output active.

Close requests to a file that is already closed are ignored.

The base mode equivalent service is M_CLSE.

Entry Conditions

Calling Sequence:

```
        M.CLSE      fcb [,[ EOF] [,REW ] ]
        (or)
        LA          1,fcb
       [SVC         1,X'38'  (or) M.CALL  H.IOCS, 5]
       [SVC         1,X'37'  (or) M.CALL  H.IOCS, 2]
        SVC         1,X'39'  (or) M.CALL  H.IOCS,23
```

fcb is the FCB address

EOF writes EOF (SVC 1,X'38'). See M.WEOF description.

REW rewinds file or device (SVC 1,X'37'). See M.RWND description.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO38	Task attempted to write to a file opened in read-only mode

Output Messages:

None

6.2.15 M.CMD - Get Command Line

The M.CMD service returns the portion of the command line between the program name and the end of the line if the program name is specified on the command line. If data does not exist or the command line has already been issued, a null string is returned.

This service can be executed by the IPU.

The base mode equivalent service is M_CMD.

Entry Conditions

Calling Sequence:

```
M.CMD

(or)

SVC     2,X'61' (or) M.CALL H.REXS,88
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN  R6,R7
```

Registers:

R6	Contains the length of the string in bytes, if found; otherwise, zero
R7	Contains the first byte address of the string, if found; otherwise, zero

Abort Cases:

None

Output Messages:

None

6.2.16 M.CONABB - Convert ASCII Date/Time to Byte Binary Format

The M.CONABB service converts the system date and time from ASCII format to byte binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONABB.

Entry Conditions

Calling Sequence:

```
M.CONABB  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'06000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL  H.REXS,75
```

addr1 is the address of a four-word buffer containing the ASCII-formatted date and time

addr2 is the address of a two-word buffer where the byte binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.17 M.CONADB - Convert ASCII Decimal to Binary

The M.CONADB service converts ASCII decimal doublewords to their binary equivalent.

An all blank doubleword converts to zero.

This service can be executed by the IPU.

The base mode equivalent service is M_CONADB.

Entry Conditions

Calling Sequence:

```
M.CONADB [n]

(or)

LD    R6,n
SVC   1,X'28' (or) M.CALL H.TSM,7
```

n is the address of a left-justified, doubleword-bounded, ASCII-coded decimal number, blank-filled. If not specified, contents of registers six and seven are converted.

Exit Conditions

Return Sequence:

```
M.IPURTN        6,7
```

Registers:

R6	Zero if a character is nonnumeric
R7	Binary equivalent of input

6.2.18 M.CONAHB - Convert ASCII Hexadecimal to Binary

The M.CONAHB service converts ASCII hexadecimal doublewords to their binary equivalent.

An all blank doubleword converts to zero.

This service can be executed by the IPU.

The base mode equivalent service is M_CONAHB.

Entry Conditions

Calling Sequence:

```
M.CONAHB [n]

(or)

LD    R6,n
SVC   1,X'29' (or) M.CALL  H.TSM,8
```

n is the address of a left-justified, doubleword-bounded, ASCII-coded decimal number, blank-filled. If not specified, contents of registers six and seven are converted.

Exit Conditions

Return Sequence:

```
M.IPURTN        6,7
```

Registers:

R6	Zero if a character is not hexadecimal
R7	Binary equivalent of input

6.2.19 M.CONASB - Convert ASCII Date/Time to Standard Binary

The M.CONASB service converts the system date and time from ASCII format to binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONASB.

Entry Conditions

Calling Sequence:

```
M.CONASB  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'05000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL   H.REXS,75
```

addr1 is the address of a four-word buffer containing the ASCII formatted date and time

addr2 is the address of a two-word buffer where the binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.20 M.CONBAD - Convert Binary to ASCII Decimal

The M.CONBAD service converts binary words to their ASCII decimal equivalent.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBAD.

Entry Conditions

Calling Sequence:

```
M.CONBAD [n]

(or)

LW    R5,n
SVC   1,X'2A' (or) M.CALL  H.TSM,9
```

n the address of a positive binary number

Exit Conditions

Return Sequence:

```
M.IPURTN        6,7
```

Registers:

R6,R7 ASCII result, right-justified with leading ASCII zeros

6.2.21 M.CONBAF - Convert Binary Date/Time to ASCII Format

The M.CONBAF service converts the system date and time from binary format to ASCII format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBAF.

Entry Conditions

Calling Sequence:

```
M.CONBAF  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'2000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL   H.REXS,75
```

addr1 is the address of a two-word buffer containing the binary formatted date and time

addr2 is the address of a four-word buffer where the ASCII formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.22 M.CONBAH - Convert Binary to ASCII Hexadecimal

The M.CONBAH service converts binary words to their ASCII hexadecimal equivalent.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBAH.

Entry Conditions

Calling Sequence:

```
M.CONBAH [n]

(or)

LW   R5,n
SVC  1,X'2B' (or) M.CALL  H.TSM,10
```

n is the address of a binary number

Exit Conditions

Return Sequence:

```
M.IPURTN        6,7
```

Registers:

R6,R7 ASCII result, right-justified with leading ASCII zeros

6.2.23 M.CONBBA - Convert Byte Binary Date/Time to ASCII

The M.CONBBA service converts the system date and time from byte binary format to ASCII format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBBA.

Entry Conditions

Calling Sequence:

```
M.CONBBA  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'04000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL   H.REXS,75
```

addr1 is the address of a two-word buffer containing the byte binary formatted date and time

addr2 is the address of a four-word buffer where the ASCII formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.24 M.CONBBY - Convert Binary Date/Time to Byte Binary

The M.CONBBY service converts the system date and time from binary format to byte binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBBY.

Entry Conditions

Calling Sequence:

```
M.CONBBY  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'01000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL   H.REXS,75
```

addr1 is the address of a two-word buffer containing the binary formatted date and time

addr2 is the address of a two-word buffer where the byte binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.25 M.CONBYB - Convert Byte Binary Date/Time to Binary

The M.CONBYB service converts the system date and time from the byte binary format to binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_CONBYB.

Entry Conditions

Calling Sequence:

```
M.CONBYB  addr1,addr2

(or)

LA        R1,addr1
ORMW      R1,=X'03000000'
LA        R2,addr2
SVC       2,X'51'  (or)  M.CALL   H.REXS,75
```

addr1 is the address of a two-word buffer containing the byte binary formatted date and time

addr2 is the address of a two-word buffer where the binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.26 M.CONN - Connect Task to Interrupt

The M.CONN service indirectly connects a task to an interrupt level so that when the interrupt occurs, the specified task will be scheduled for execution (resumed). If the specified task is not active, M.CONN will preactivate it. If preactivation is required, but the actual interrupt connection is denied, M.CONN deletes the residual task because the task would continue in the suspended state indefinitely.

The base mode equivalent service is M_CONN.

Entry Conditions

Calling Sequence:

```
M.CONN      task,intlevel

(or)

LW          R5,intlevel
LI          R6,0      } (or) LD R6,task (or)      LW R6, PNV }
LW          R7,taskno }                           LI R7, 0   }
SVC         1,X'4B' (or) M.CALL  H.REXS,10
```

task — the address of a doubleword containing the left-justifed blank-filled one- to eight-character ASCII name of the task (system file only); or zero in word zero and the task number in word one; or pathname or RID vector in word zero and zero in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

intlevel — is the hardware priority level where the task is to be connected

Exit Conditions

Return Sequence:

```
M.RTRN      6,7
```

Registers:

R6 — Denial Code:

Value	Description
1	Task already connected to an interrupt
2	Another task connected to the specified interrupt
3	Interrupt not SYSGEN specified indirectly connectable
4	Specified task not found in dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (by the M.KEY file).

R7 — Zero if task not connected to interrupt; otherwise, contains the task number

Abort Cases:

None

Output Messages:

None

6.2.27 M.CPERM - Create Permanent File

The M.CPERM service creates a permanent file. Permanent files are given names in directories and remain known to the operating system until explicitly deleted.

This service allocates a resource descriptor and the initial file space requirements for the file. Next, the specified attributes of the file are recorded in the resource descriptor. As a final step, the name of the file is established in the indicated directory.

When a directory entry is established, the directory entry is linked to the resource descriptor of the file. This link makes the relationship of the name of the file to the other attributes of the file. Typical file attributes are:

- The file's name
- The file's resource identifier (RID)
- The file's protection attributes
- The file's management attributes
- The file's initial space requirements

To create with possible multiple segments, the CNP address must be supplied. Byte 0 of the CNP option field contains the maximum segment numbers allowed at creation time.

The base mode equivalent service is M_CREATEP.

Entry Conditions

Calling Sequence:

```
M.CPERM  arga [,cnpaddr] [,rcbaddr]

(or)

LW          R1,arga
LA          R2,rcbaddr  (or)  ZR  R2
LA          R7,cnpaddr  (or)  ZR  R7
SVC         2,X'20'  (or)  M.CALL  H.VOMM,1
```

arga	contains a PN vector or PNB vector
cnpaddr	is a CNP address or zero if CNP not supplied
rcbaddr	is a RCB address or zero if default attributes are desired

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

6.2.28 M.CTIM - Convert System Date/Time Format

The M.CTIM service converts the system date and time from one of three standard formats (see Appendix H for time formats) to either of the other two formats. This service is callable from specific case macros that provide the function code in the macro call itself.

This service can be executed by the IPU.

The base mode equivalent service is M_CTIM.

Entry Conditions

Calling Sequence:

```
M.CTIM  funct,addr1,addr2

(or)

LA      R1,addr1
ORMW    R1,funct
LA      R2,addr2
SVC     2,X'51'  (or)  M.CALL  H.REXS,75
```

funct — is the address of a word that contains the function code (see chart below) in byte zero (most significant) and zeros in bytes one, two, and three

addr1 — is the address of a two- or four-word buffer where the user provides the date and time, in any of the three standard formats, for the system to convert

addr2 — is the address of a two- or four-word buffer where the system returns the converted date and time values in the format requested by the user

Funct code	Input format	Return format	Buffer in	Length out
1	Binary	Byte binary	2W	2W
2	Binary	Quad ASCII	2W	4W
3	Byte binary	Binary	2W	2W
4	Byte binary	Quad ASCII	2W	4W
5	Quad ASCII	Binary	4W	2W
6	Quad ASCII	Byte binary	4W	2W

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.29 M.CWAT - System Console Wait

The M.CWAT service suspends operation of the calling program until the specified I/O transfer is complete.

The base mode equivalent service is M_CWAT.

Entry Conditions

Calling Sequences:

```
M.CWAT      tcpb

(or)

LA          R1,tcpb
SVC         1,X'3D' (or) M.CALL  H.IOCS,26
```

tcpb is the address of a Type Control Parameter Block (TCPB). See Section 5.10.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.30 M.DASN - Deassign and Deallocate Resource

The M.DASN service deallocates a resource and disassociates it from a logical file code. When a device associated with any unformatted media is detached, a message is issued to inform the operator to dismount the medium, unless inhibited by user request or system constraints. Deallocation of a nonshared resource makes it available to other tasks. Deallocation of a shared resource makes the resource available, if the caller is the last task to deallocate it or the access mode changes as a result of the deallocation to allow other compatible tasks to attach to the resource. Deallocation of SLO and SBO files result in their definitions being passed to the system output task for processing. If the specified logical file code has been equated to other logical file codes in the system, only the specified lfc is deallocated. If close has not been issued, the resource is also closed. This function can also issue a dismount message for an unformatted medium with no resource deallocation.

The base mode equivalent service is M_DEASSIGN.

Entry Conditions

Calling Sequence:

```
M.DASN  arga [,cnpaddr]

(or)

LW          R1,arga
LA          R7,cnpaddr  (or)  ZR  R7
SVC         2,X'53'  (or)  M.CALL  H.REXS,22
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address, options field, and status field.

The options field of the CNP has the following bit significance when set:

0 - issue dismount message with no resource deallocation

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or in the status field of the CNP

Value	Description
8	Unrecoverable I/O error to volume
29	Logical file code associated with FCB not assigned
30	Invalid allocation index
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

6.2.31 M.DATE - Date and Time Inquiry

The M.DATE service returns to the caller the date (in ASCII), calendar information (century, year, month and day), and a count of the number of real-time clock interrupts since midnight. To aid in converting the interrupt count to time-of-day, counts of the number of interrupts per second and the number of interrupts per time unit are also returned.

This service can be executed by the IPU.

The base mode equivalent service is M_DATE.

Entry Conditions

Calling Sequence:

```
M.DATE pbaddr

(or)

LA      R7,pbaddr
SVC     1,X'15' (or) M.CALL  H.REXS,70
```

pbaddr is the logical word address of the first location of a parameter block formatted as follows:

- Words 0 and 1 contain the current date in the format entered at IPL time.
- Word 2

Bytes	Contents in Binary
0	Century
1	Year
2	Month
3	Day

- Word 3 contains the number of clock interrupts since midnight
- Word 4 contains the number of clock interrupts per second (initialized by SYSGEN)
- Word 5 contains the number of clock interrupts per time unit (initialized by SYSGEN)

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.32 M.DEBUG - Load and Execute Interactive Debugger

The M.DEBUG service causes one of the following events to occur:

- If the interactive debugger is currently loaded at the time the service is called, control is transferred to the debugger.
- If the interactive debugger is not currently loaded at the time the service is called, the debugger is loaded as an overlay segment, then control is transferred to the debugger.

MPXDB or SYMDB is loaded with nonbase mode tasks.

If the task is nonbase and has a shared CSECT, the debugger is not loaded and error code four is returned. To debug a nonbase shared CSECT task, request the debugger at task activation.

The base mode equivalent service is M_DEBUG.

Entry Conditions

Calling Sequence:

```
M.DEBUG

(or)

SVC     1,X'63' (or) M.CALL   H.REXS,29
```

Exit Conditions

Normal Return Sequence to Debugger:

M.RTRN	7	Contains the transfer address of the debugger if the debugger was loaded by this service call. Register seven contains zero if the debugger was already loaded at the time this service was called.

Abnormal Return Sequence to Caller:

R7 Value	Description
2	Debugger load module not found
4	Invalid preamble
5	Insufficient task space for loading
6	I/O error on resource descriptor
7	I/O error on resource
8	Loading error

Abort Cases:

None

Output Messages:

None

6.2.33 M.DEFT - Change Defaults

The M.DEFT service changes the caller's working directory or project group protection or any combination of the two attributes of the caller. The caller can change current working directory only or project group protection only or both.

Typically, the caller should invoke this service with two separate calls as though changing project group protection and changing the current working directory were two distinct services.

In cases where it is desirable to change both attributes with a single call, the caller should be aware of the effects. When both project group and working directory are specified, the project group is changed prior to attempting to change the current working directory. After changing the project group, if the attempt to establish the current working directory fails, the new project group protection will remain in effect and the caller will be notified via an error status code that the current working directory request failed. It is the responsibility of the caller to determine whether or not to continue with the new project group or to reestablish another project group.

The base mode equivalent service is M_DEFT.

Entry Conditions

Calling Sequence:

```
M.DEFT [[ARGA=]arga][,[CNP=]cnpaddr][,[PRJADR=]prjaddr][,[KEYADR=] keyaddr]

(or)

LW      R1,arga
LA      R4,prjaddr (or) ZR  R4
LA      R5,keyaddr (or) ZR  R5
LA      R7,cnpaddr (or) ZR  R7
SVC     2,X'27' (or) M.CALL  H.VOMM,8
```

arga — contains a PN vector, PNB vector, or RID vector

cnpaddr — is a CNP address or zero if CNP not supplied

prjaddr — is the address of the new project group name or zero if no change to project group name

keyaddr — is the address of the new project group key or zero if not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

6.2.34 M.DELR - Delete Resource

The M.DELR service explicitly deletes volume resources. This service must be used to delete directories, files, and memory partitions. The caller cannot delete a resource to which the caller does not have delete access.

As the first step, this service deletes the directory entry for the specified resource. Next, the volume space requirements are released. As a final step, the resource descriptor is released.

If the resource is allocated at the time of the delete request, only the directory entry is deleted. The volume space requirements and the resource descriptor for the resource will be released when the last assignment to the resource is removed.

To delete a permanent file or memory partition, the pathname or pathname block must be supplied. To delete a directory, the pathname or pathname block must be supplied and all files which were defined in the directory must have been previously deleted.

To delete a temporary file, the caller can provide the resource identifier (RID), or specify the logical file code (LFC), or the address of a File Control Block (FCB).

The base mode equivalent service is M_DELETER.

Entry Conditions

Calling Sequence:

```
M.DELR          [arga][,cnpaddr]
(or)
LW              R1,arga
LA              R7,cnpaddr (or) ZR  R7
SVC             2,X'24' (or) M.CALL  H.VOMM,5
```

arga contains a PN vector, PNB vector, LFC, or FCB address

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

6.2.35 M.DELTSK - Delete Task

The M.DELTSK service forces I/O completion and immediately aborts the specified task. See Task Termination Sequencing in Chapter 2. This service should only be used when abort fails to remove a task or when a task is queued for a resource. File integrity can be affected because operations are not allowed to complete normally. To preserve system integrity, the kill directive is processed as an abort for the amount of time specified by KTIMO (Sysgen). If this does not remove the task, it is killed.

The base mode equivalent service is M_DELTSK.

Entry Conditions

Calling Sequence:

```
M.DELTSK        abcode,task,extcode

(or)

LD              R2,extcode
LW              R5,abcode
LI              R6,0        } (or) LD R6,taskname
LW              R7,taskno   }
SVC             1,X'5A' (or) M.CALL H.REXS,31
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

extcode — contains the extended abort code message consisting of one to eight ASCII characters, left-justified and blank-filled

Exit Conditions

Return Sequence:

```
M.RTRN          7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (by the M.KEY file); otherwise, contains the task number.

Abort Cases:

None

Output Messages:

Modifies abort message to:

task number ABORT AT: xxxxxxxx - yyyy mm/dd/yy hh:mn:ss zzzzzzzzzz

task	is the one- to eight-character name of the task being aborted
number	is the task number of the task being aborted
xxxxxxxxxx	is the location the abort occurred
yyyy	is the bias to the logical start of the task
mm	is the two-character decimal number of the month between 01 and 12
dd	is the two-character decimal number of the day between 01 and 31
yy	is the two-character decimal number of the year between 00 and 99
hh	is the two-character decimal number of the hour between 00 and 23
mn	is the two-character decimal number of the minutes between 00 and 59
ss	is the two-character decimal number of the seconds between 00 and 59
zzzzzzzzzz	is the extended message code supplied with the call to this service

6.2.36 M.DEVID - Get Device Mnemonic or Type Code

The M.DEVID service allows the user to pass a device mnemonic or a generic device type code and receive the corresponding type code or mnemonic. See Appendix A for device mnemonic and device type codes.

This service can be executed by the IPU.

The base mode equivalent service is M_DEVID.

Entry Conditions

Calling Sequence:

```
M.DEVID         id

(or)

LW              R2,id
SVC             1,X'14' (or)  M.CALL   H.REXS,71
```

id contains either a device mnemonic in the right halfword with the left halfword zero or a device type code in byte 3 with zero in bytes 0-2

Exit Conditions

Return Sequence:

```
M.IPURTN        2
```

Registers if input was a device mnemonic:

R2 Bytes 0-2 contain zeros. Byte 3 contains the corresponding device type code.

Registers if input was a device type code:

R2 Left halfword contains zero. Right halfword contains the corresponding device mnemonic.

Registers if input was a mnemonic or device type code not in the system Device Type Table (DTT):

R2 Bit 0 is set. Bits 1-31 are unchanged.

Abort Cases:

None

Output Messages:

None

6.2.37 M.DIR - Create Directory

The M.DIR service creates a permanent directory. Permanent directories are given names in the root directory and remain known to the operating system until explicitly deleted.

Directories are used to contain the names of permanent files and memory partitions that are created in the directories. In this way, directories are used to classify permanent files as to whatever classification category is determined by the creator of the directories.

This service allocates a resource descriptor and the volume space requirements for the directory. Next, the indicated attributes of the directory are recorded in the resource descriptor. As a final step, the name of the directory is established in the indicated previous level (parent) directory.

When the directory is established, the directory entry is linked to the resource descriptor of the new directory. This link makes the relationship of the name of the new directory to the other attributes of the new directory. Typical directory attributes are:

- The directory's name
- The directory's resource identifier (RID)
- The directory's protection attributes
- The directory's management attributes
- The directory's volume space requirements

The base mode equivalent service is M_DIR.

Entry Conditions

Calling Sequence:

```
                    [[ARGA=]arga][,[CNP=]cnpaddr][,[RCB=]rcbaddr]
M.DIR               [arga][,cnpaddr][,rcbaddr]
(or)
LW                  R1,arga
LA                  R2,rcbaddr (or) ZR R2
LA                  R7,cnpaddr (or) ZR R7
SVC                 2,X'23' (or) M.CALL H.VOMM,4
```

arga — contains a PN vector or PNB vector

cnpaddr — is a CNP address or zero if CNP not supplied

rcbaddr — is a RCB address or zero if default attributes are desired

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

6.2.38 M.DISCON - Disconnect Task from Interrupt

The M.DISCON service disconnects a task that has previously been centrally connected to an interrupt level.

The base mode equivalent service is M_DISCON.

Entry Conditions

Calling Sequence:

```
M.DISCON task

(or)

LI      R6,0        (or)    LD R6,taskname
LW      R7,taskno
SVC     1,X'5D'     (or)    M.CALL  H.REXS,38
```

task — the address of a doubleword containing the name of the task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Normal Return Sequence:

```
M.RTRN          7
```

Registers:

R7 Contains the task number

Abnormal Return Sequence:

```
M.RTRN          6,7
```

Registers:

R6 Contains denial code as follows:

Value	Description
1	Task not found in dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (by the M.KEY file).
2	Task not indirectly connected
3	Task connected to invalid interrupt

R7 Zero if the task was not previously connected to an interrupt level

Abort Cases:

None

Output Messages:

None

6.2.39 M.DLTT - Delete Timer Entry

The M.DLTT service resets the timer for the specified task so that its specified function is no longer performed upon time out. Deletion of the timer entry does not delete the associated task. One-shot timers are deleted on expiration.

The base mode equivalent service is M_DLTT.

Entry Conditions

Calling Sequence:

```
M.DLTT      timer

(or)

LW          R7,timer
SVC         1,X'47'  (or)  M.CALL   H.REXS,6
```

timer right-justified two-character ASCII name of a timer

Exit Conditions

Return Sequence:

```
M.RTRN

CC1 set     Timer entry not found
```

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.40 M.DMOUNT - Dismount Volume

The M.DMOUNT service decrements the number of established users of a volume. If there are no remaining users of the volume, the Mounted Volume Table Entry (MVTE) associated with the volume is deallocated, and the mount device made available. If appropriate, a dismount message is displayed on the console to inform the operator that the volume should be dismounted.

The base mode equivalent service is M_DISMOUNT.

Entry Conditions

Calling Sequence:

```
M.DMOUNT   voladdr  [,cnpaddr]

(or)

LA         R1,voladdr
LA         R7,cnpaddr  (or)  ZR  R7
SVC        2,X'4A'  (or)  M.CALL  H.REMM,19
```

voladdr is the doubleword-bounded address of the volume name

cnpaddr is the address of a Caller Notification Packet if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)		(without CNP)	
M.RTRN		M.RTRN	
(or)		(or)	
M.RTNA	(CC1 set)	M.RTRN	R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or in the status field of the CNP

Value	Description
14	Caller has outstanding resource assignments on this volume
20	Volume not assigned to this task or volume is public

Wait Conditions

None

6.2.41 M.DSMI - Disable Message Task Interrupt

The M.DSMI service disables the task interrupts for messages sent to the calling task. M.DSMI is useful for synchronization gating of the task message interrupts.

This service can be executed by the IPU.

The base equivalent service is M_DSMI.

Entry Conditions

Calling Sequence:

```
M.DSMI

(or)

SVC     1,X'2E' (or) M.CALL  H.REXS,57
```

Exit Conditions

Return Sequence:

```
M.IPURTN
```

Registers:

```
CC1     Task interrupts were already disabled
```

Abort Cases:

None

Output Messages:

None

6.2.42 M.DSUB - Disable User Break Interrupt

The M.DSUB service deactivates the user break interrupt (see M.ENUB) and allows user breaks by the terminal BREAK key to be acknowledged.

This service can be executed by the IPU.

The base equivalent service is M_DSUB.

Entry Conditions

Calling Sequence:

```
M.DSUB

(or)

SVC     1,X'12' (or) M.CALL  H.REXS,73
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1 set	User break already disabled

Abort Cases:

None

Output Messages:

None

6.2.43 M.DUMP - Memory Dump Request

The M.DUMP service provides a dump of the caller's Program Status Doubleword (PSD), General Purpose Registers, and specified memory limits. The output is to a SLO file in side-by-side hexadecimal with ASCII format, with the PSD and registers preceding the specified memory limits. The PSD and registers are extracted from the first level of push-down of the calling task. Optionally, register five can specify the address of a ten word block containing registers zero through seven and the PSD to be dumped, respectively. Any task can request a memory dump.

The base mode equivalent service is M_DUMP.

Entry Conditions

Calling Sequence:

```
M.DUMP    start,end [,mem3]
(or)
ZR        R5 (or) LA  R5,mem3
LW        R6,start
LW        R7,end
SVC       1,X'4F' (or) M.CALL  H.REXS,12
```

start — contains the low logical word address requested in dump

end — contains the high logical word address requested in dump

mem3 — is the optional address of ten consecutive words containing registers zero through seven and a PSD, respectively. If register five is zero, the registers and PSD dumped are taken from the first level of push-down.

Note: Start and end are truncated to the nearest eight-word boundaries and memory is dumped between the truncated limits.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6	Value	Description
	1	High dump limit less than low limit
	4	No FAT or FPT space available
	5	Request made with insufficient levels of push-down available
	6	Cannot allocate SLO file
	7	Unrecoverable I/O error

R7 — Zero if dump could not be performed

Abort Cases:

None

Output Messages:

None

6.2.44 M.EAWAIT - End Action Wait

The M.EAWAIT service waits for the completion of any no-wait request or I/O end action if any are queued. If there aren't any outstanding, the service returns immediately to the user. This service is similar to the M.ANYW service.

The base mode equivalent service is M_AWAITACTON.

Entry Conditions

Calling Sequence:

```
M.EAWAIT timel

(or)

LW      R6,timel
SVC     1,X'1D'  (or)  M.CALL  H.EXEC,40
```

timel contains zero if wait for an indefinite period is requested; otherwise, timel contains the negative number of time units to elapse before the wait is terminated.

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

MS31	User attempted to go to an any wait state from an end-action routine

Output Messages:

None

6.2.45 M.ENMI - Enable Message Task Interrupt

The M.ENMI service enables task interrupts for messages sent to the calling task. It removes an inhibit condition previously established by invoking the M.DSMI service.

The base mode equivalent service is M_ENMI.

Entry Conditions

Calling Sequence:

M.ENMI

(or)

SVC 1,X'2F' (or) M.CALL H.REXS,58

Exit Conditions

Return Sequence:

M.RTRN

Registers:

CC1 set Task interrupts were already enabled

Abort Cases:

None

Output Messages:

None

6.2.46 M.ENUB - Enable User Break Interrupt

The M.ENUB service activates the user break interrupt and causes further user breaks from the user terminal break key to be ignored.

This service can be executed by the IPU.

The base mode equivalent service is M_ENUB.

Entry Conditions

Calling Sequence:

```
M.ENUB

(or)

SVC     1,X'13' (or) M.CALL  H.REXS,72
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1 set	User break already enabled

Abort Cases:

None

Ouput Messages:

None

6.2.47 M.ENVRMT - Get Task Environment

The M.ENVRMT service obtains more information on the task environment than what is provided in the Task Option Word.

This service can be executed by the IPU.

The base mode equivalent service is M_ENVRMT.

Entry Conditions

Calling Sequence:

```
M.ENVRMT

(or)

SVC     2,X'5E'  (or)  M.CALL  H.REXS,85
```

Exit Conditions

Return Sequence:

```
M.RTRN  R7
```

Registers:

R7 contains the task environment word as follows:

Bit	Definition
0	0 if batch task 1 if interactive or real-time task
1	0 if option NOCOMMAND is set 1 if option COMMAND is set
2	0 if option NOERR is set 1 if option ERROR is set
3	0 if cataloged or linked unprivileged 1 if cataloged or linked privileged
4	0 if currently unprivileged 1 if currently privileged
5-31	Reserved

Abort Cases:

None

Output Messages:

None

6.2.48 M.EXCLUDE - Exclude Memory Partition

The M.EXCLUDE service allows a nonbase task to dynamically deallocate any common area previously included. This service causes the assign count and user count to be decremented. The partition area is deleted and its resources returned to the free list when the assign count goes to zero. This service is also called by the exit processor (H.REMM,3) whenever a task aborts or comes to an unnatural end while associated with a memory partition. The partition is identified by the allocation index obtained when the partition was included, or the one- to eight-character name used to create it along with the one- to eight-character owner name or task number used to include it.

The base mode equivalent service is M_EXCLUDE.

Entry Conditions

Calling Sequence:

```
M.EXCLUDE [arga],[cnpaddr][,argb]

(or)

LW      R1,arga
LD      R4,argb
LA      R7,cnpaddr (or) ZR R7
SVC     2,X'41' (or) M.CALL H.REMM,14
```

arga — contains a PN vector, PNB vector, the address of the one- to eight-character left-justified, blank-filled system partition name, or the allocation index obtained when the partition was included.

argb — is a doubleword address containing a left-justified task number in word 0 and word 1 zero
(or)
a one- to eight-character left-justified, blank-filled owner name used to include the partition
(or)
not required, if an allocation index is used

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)		(without CNP)
M.RTRN		M.RTRN
(or)		(or)
M.RTNA	(CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
5	Shared Memory Table (SMT) entry not found for this partition
30	Invalid allocation index
35	Attempt to exclude memory partition that is not mapped into requesting task's address space
39	Unable to write back data section
58	Shared memory table space is not available

Wait Conditions

None

6.2.49 M.EXIT - Terminate Task Execution

The M.EXIT service performs all normal termination functions required of exiting tasks. See Chapter 2. All devices and memory are deallocated, related table space is erased, and the task's Dispatch Queue entry is cleared.

The base mode equivalent service is M_EXIT.

Entry Conditions

Calling Sequence:

M.EXIT

(or)

SVC 1,X'55' (or) M.CALL H.REXS,18

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

RX92	Task has attempted normal exit with messages in its receiver queue

Output Messages:

None

6.2.50 M.EXTD - Extend File

The M.EXTD service allows the space of a file to be manually extended. The caller can specify the size of the requested extension or choose to use the default file extension parameters defined when the file was created. If the specified size cannot be obtained, M.EXTD tries to extend by the maximum extension size that was specified at resource creation. If that size cannot be obtained, M.EXTD tries to extend by the minimum extension size that was specified at resource creation. A CC1 indicates that the requested extension size is not obtained. If the file was created with the zero option specified, the extension is zeroed.

This service only extends temporary or permanent files that are manually extendable. Directories and memory partitions cannot be extended. The caller must have write, update, or append access in order to extend the file.

The caller can extend a file regardless of whether the file is currently allocated. Additionally, the caller can supply any allowable resource specification, for example, pathname (PN), pathname block (PNB), resource ID (RID), logical file code (LFC) or address of a File Control Block (FCB).

The base mode equivalent service is M_EXTENDFILE.

Entry Conditions

Calling Sequence:

```
M.EXTD      [arga],[cnpaddr] [,blocks]
(or)
LW          R1,arga
LW          R6,blocks (or) ZR  R6
LA          R7,cnpaddr (or) ZR  R7
SVC         2,X'25' (or) M.CALL  H.VOMM,6
```

arga	contains a PN vector, PNB vector, RID vector, LFC, or FCB address
cnpaddr	is a CNP address or zero if CNP not supplied
blocks	is an address containing the number of blocks to extend the file by or zero if RCB extension parameters specified during file creation are to be used

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R6	M.RTRN R6
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R6	Number of contiguous blocks file actually extended by
R7	Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.51 M.FD - Free Dynamic Extended Indexed Data Space

The M.FD service allows the task to deallocate the most recently acquired extended memory map block, thus contracting its address space.

Entry Conditions

Calling Sequence:

```
M.FD

(or)

SVC     1,X'6A' (or)  M.CALL  H.REMM,9
```

Exit Conditions

Return Sequence:

```
M.RTRN  R3
```

Registers:

R3	New upper limit of extended memory; zero if no extended memory left allocated

6.2.52 M.FE - Free Dynamic Task Execution Space

The M.FE service allows the task to dynamically deallocate the most recently acquired execution space map block, thus contracting its address space.

Entry Conditions

Calling Sequence:

```
M.FE
(or)
SVC     1,X'68' (or)  M.CALL  H.REMM,11
```

Exit Conditions

Return Sequence:

M.RTRN	R3 (or) abort user with RM76

Registers:

R3	New upper address of execution space

Abort Cases:

RM76	User has attempted deallocation of TSA

6.2.53 M.FWRD - Advance Record or File

The M.FWRD service is not applicable for SYC files or unblocked files. This service performs the following functions for advancing records:

- Verifies volume record if BOT on multivolume magnetic tape.
- Advances specified number of records.

M.FWRD performs the following functions for blocked files:

- Logical records are advanced until an end-of-file is found. The read/write control word points to the first record after the end-of-file.
- Verifies volume record if BOT on multivolume magnetic tape.
- Advances specified number of files.

The base mode equivalent service is M_ADVANCE.

Entry Conditions

Calling Sequence:

```
M.FWRD          fcb,[R],[number]

(or)

LA              R1,fcb
LNW             R4,number
SVC             1,X'33'   (or)  M.CALL  H.IOCS,7
SVC             1,X'34'   (or)  M.CALL  H.IOCS,8
BIB             R4,$-1W
```

fcb — is the FCB address

R — advance by record (SVC1,X'33'). If not specified, the default is advance by file (SVC1,X'34').

number — address of word containing the number of records or files to be advanced, or the contents of register four if not supplied

$-1W — branches back to SVC the number of times specified by the parameter "number"

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cells for blocked file
IO07	A task has attempted to perform an operation which is not valid for the device to which the user's file is assigned
IO09	Illegal operation on the SYC file
IO29	Advance file issued for a blocked file while writing
IO30	Illegal or unexpected volume number or reel ID encountered on magnetic tape

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

6.2.54 M.GADRL - Get Address Limits

The M.GADRL service returns the logical addresses associated with the boundaries of a nonbase mode task. The address returned in register 5 reflects any increments previously allocated by the M.GE service and the address returned in register 7 reflects any increments previously allocated by the M.GD service. These addresses do not reflect any increments previously allocated by the M.INCL, M.INCLUDE, or M.SHARE services.

Entry Conditions

Calling Sequence:

```
M.GADRL

(or)

SVC     1,X'65' (or) M.CALL  H.REXS,41
```

Exit Conditions

Return Sequence:

```
M.RTRN          3,4,5,6,7
```

Registers:

R3	Contains the logical word address of the first location of the task's DSECT (always on a page boundary)
R4	Contains the logical word address of the last location in the DSECT actually loaded by the loader
R5	Contains the logical word address of the last location currently available in the task's contiguously allocated DSECT (always a map block boundary minus one word)
R6	Contains the logical word address of the first location of the task's CSECT or COMMON allocation (always a map block boundary)
R7	Contains the logical word address of the last location currently available in the task's contiguously extended indexed data space (always a map block boundary minus one word).

Abort Cases:

None

Output Messages:

None

6.2.55 M.GD - Get Dynamic Extended Data Space

The M.GD service allows the nonbase mode task to dynamically acquire an additional map block of memory in its extended area. The memory is of the same type specified when the task was cataloged. It is mapped in a logically contiguous manner, with the first request map starting at 128KW. The task can call this service up to 192 times, if sufficient memory exists, to expand its extended indexed data space. Alternately, the task can choose to deallocate this space in the reverse order by M.FD. The task is suspended until the allocation is successful.

Memory is allocated in 2KW increments.

Entry Conditions

Calling Sequence:

```
M.GD

(or)

SVC     1,X'69' (or) M.CALL  H.REMM,8
```

Exit Conditions

Return Sequence:

```
M.RTRN  R3,R4
```

Registers:

R3	Logical address of allocated memory
R4	Ending logical word address of allocated memory or error code if register three is zero

Error Conditions

R3 Equals zero

R4

Value	Description
1	Attempted allocation of an excessive number of map blocks
2	Attempted allocation exceeds physical memory configured
3	M.MEMB service in use

6.2.56 M.GE - Get Dynamic Task Execution Space

The M.GE service allows the nonbase mode task to dynamically expand its memory allocation in map block increments, starting at the end of its DSECT up to the top of its logical address space. The additional memory is of the same type specified when the task was cataloged. The task is mapped in a logically contiguous manner up to the start of its CSECT or Global common, or 128KW, whichever occurs first. The task is suspended until the allocation is successful.

Memory is allocated in 2KW increments.

Entry Conditions

Calling Sequence:

```
M.GE

(or)

SVC     1,X'67' (or) M.CALL  H.REMM,10
```

Exit Conditions

Return Sequence:

```
M.RTRN  R3,R4
```

Registers:

R3	Starting logical address of new map block
R4	Ending logical address of new map block

Error Conditions

R3 Equals zero

R4

Value	Description
1	Excessive DSECT allocation attempted
2	Attempted allocation exceeds physical memory configured
3	M.MEMB service is in use

6.2.57 M.GETDEF - Get Function From Terminal Definition

The M.GETDEF service allows the user to specify the LFC of a terminal currently open and allocated, the buffer address, and a terminal ability function request. The service will place the appropriate string of bytes in the buffer the user specifies and indicate the length on the string. For this service to operate, the partition TERMPART must exist and have been initialized by J.TDFI.

Entry Conditions

Calling Sequence:

```
M.GETDEF        [ibaddr]
(or)
LA              R1, ibaddr
SVC             2,X'7A' (or) M.CALL H.TSM,15
```

ibaddr is the logical 24-bit word address of the first word of the GETDEF parameter block formatted as follows:

Word	Description
0	Open and allocated terminal's LFC
1	User buffer 24 bit address for returned information
2	Half word 0 is the Requested Function (2 ASCII alphanumeric characters). Half word 1 is reserved.
3	Half word 0 is the User Buffer length in bytes. Half word 1 is the length in bytes of string returned by the service to the user's buffer.
4	Optional X coordinate for cursor positioning functions.
5	Optional Y coordinate for cursor positioning functions

If ibaddr is not supplied then register 1 is assumed to contain the address

Exit Conditions

Return sequence:

M.RTRN — On normal completion the string for the requested function is in the user's buffer, and the length of this string is in the parameter block string length field.

CC1 set

Error detected. The string length in the parameter block is set to 0 and the function contains the error number with the following meanings:

function=

Error Number	Description
1	Invalid LFC supplied
2	Unknown terminal type
3	User buffer is too large (>2K)
4	Cannot include partition
5	Undefined function requested
6	User buffer is too small
7	Partition data integrity suspect
8	Invalid terminal type supplied
9	Invalid user buffer address
10	Function is invalid for this terminal
N/A	Info block address is invalid (CC1 set only)

Registers:

R1

Unchanged. CC1 set if an error is detected.

Abort Cases:

MF01
SV03

If supplied address are invalid
If the entry point has been sysgened out (i.e. C.NOTDEF, C.BIT is set).

Output Message:

None

6.2.58 M.GMSGP - Get Message Parameters

The M.GMSGP service is called from the message receiver routine of a task that has received a message interrupt. It transfers the message parameters into the designated receiver buffer, and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB). For description of the PRB, see Chapter 2.

The base mode equivalent service is M_GMSGP.

Entry Conditions

Calling Sequence:

```
M.GMSGP    prbaddr

(or)

LA         R2,prbaddr
SVC        1,X'7A'  (or)  M.CALL  H.REXS,35
```

prbaddr is the logical address of the Parameter Receive Block (PRB)

Exit Conditions

Return Sequence:

```
M.RTRN     6
```

Registers:

R6 Contains the processing status error code:

Value	Description
0	Normal status
1	Invalid PRB address
2	Invalid receiver buffer address or size detected during parameter validation
3	No active send request
4	Receiver buffer length exceeded during transfer

Abort Cases:

None

Output Messages:

None

6.2.59 M.GRUNP - Get Run Parameters

The M.GRUNP service is called by a task that is executing for a run request. It transfers the run parameters into the designated receiver buffer, and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB). See Chapter 2.

The base mode equivalent service is M_GRUNP.

Entry Conditions

Calling Sequence:

```
M.GRUNP   prbaddr

(or)

LA        R2,prbaddr
SVC       1,X'7B'  (or)  M.CALL  H.REXS,36
```

prbaddr is the logical address of the Parameter Receive Block (PRB)

Exit Conditions

Return Sequence:

```
M.RTRN    6
```

Registers:

R6 Contains the processing status error code:

Value	Description
0	Normal status
1	Invalid PRB address
2	Invalid receiver buffer address or size detected during parameter validation
3	No active send request
4	Receiver buffer length exceeded during transfer

Abort Cases:

None

Output Messages:

None

6.2.60 M.GTIM - Acquire System Date/Time in Any Format

The M.GTIM service acquires the system date and time in any one of the three standard formats described in Appendix H. The user can also get the system date/time by using any of the three specific case macros. These macros generate the same SVC call but the function code is provided by the macro.

This service can be executed by the IPU.

The base mode equivalent services is M_GTIM.

Entry Conditions

Calling Sequence:

```
M.GTIM  funct,addr

(or)

LA      R1,addr
ORMW    R1,funct
SVC     2,X'50'  (or)  M.CALL   H.REXS,74
```

funct — is the address of a word containing the function code (see chart below) in byte zero (most significant) and zeros in bytes one, two, and three

addr — is the address of a buffer where the service places the date and time in the format requested by the user. This buffer is two or four words in length depending on the format desired.

Funct	Return format	Buffer length
1	Binary	2W
2	Byte binary	2W
3	Quad ASCII	4W

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.61 M.HOLD - Program Hold Request

The M.HOLD service makes the specified task ineligible for CPU control by setting the hold bit in the CPU Dispatch Queue. The specified task remains in the hold state until the operator issues the OPCOM CONTINUE directive. If the specified task is not in the CPU Dispatch Queue, the request is ignored.

The base mode equivalent service is M_HOLD.

Entry Conditions

Calling Sequence:

```
M.HOLD          task

(or)

LI              R6,0        } (or) LD R6,taskname
LW              R7,taskno   }
SVC             1,X'58  (or) M.CALL  H.REXS,25
```

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with the M.KEY file; otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

6.2.62 M.ID - Get Task Number

The M.ID service allows the user to pass the address of a parameter block containing any of the following: (a) task number, (b) task load module name, (c) owner name, or (d) task pseudonym, and the service provides the missing items if a matching entry is found. Initially, the caller passes zero (0) as the index value following the parameter block address. If more than one task in the Dispatch Queue satisfies the given parameters, the service returns to the caller with an index value in register five for retrieval of further entries. The caller is responsible for updating the index with the contents of register five and reissuing M.ID until all tasks that meet specifications have been identified or register five equals zero.

The base mode equivalent service is M_ID.

Entry Conditions

Calling Sequence:

```
M.ID        pbaddr,index

(or)

LW          R5,a variable equal to 0 or an index
LA          R7,pbaddr
SVC         1,X'64' (or) M.CALL  H.REXS,32
```

pbaddr is the logical word address of the first location of a parameter block formatted as follows:

Word	Contents
0	Task activation sequence number
1-2	Task load module name
3-4	Owner name
5-6	Pseudonym

The user supplies those items that are known and zeros the other words.

index a variable equal to zero for initial call, then previous DQE address for each subsequent call

Exit Conditions

Return Sequence:

M.RTRN 5

Normal Return Registers:

R5 Bit 0 is set if more than one task satisfies the given parameters. Bits 1-31 contain the DQE address of the first matching task found. If no entry satisfies the given parameters, register five equals zero. Register five may be used as input for subsequent calls.

Abnormal Return Registers:

CC1 set Invalid parameter block address. Register five remains unchanged.

Abort Cases:

None

Output Messages:

None

6.2.63 M.INCLUDE - Include Memory Partition

The M.INCLUDE service allows a nonbase task to include dynamic or static memory partitions, or shared images into its address space. The task is suspended until the inclusion is complete. If the resource has not been included by another task, an Allocated Resource Table (ART) and Shared Memory Table (SMT) entry is established for the resource. The resource is automatically allocated for explicit shared use. If inclusion is successful, the assign and user counts are incremented for the resource. The partition is identified by an eight-character partition name or a resource identifier (RID) that was defined when the partition was created. If the partition is dynamic, it is identified by an eight-character owner name that is used to associate this copy of the partition with a particular set of users. The shared image is identified by a resource pathname or a resource identifier (RID) that was defined when the image was created. Prezeroing of partitions is not performed by this service. The resource is swappable, if the user count goes to zero, and remains allocated until the assign count is zero. The option is provided to lock the resource for exclusive use. However, the resource remains locked until: the owner of the lock terminates, the Release Exclusive Lock service is explicitly called, or the resource is excluded by the task.

If a partition has been included by a task, subsequent includes by that task are ignored.

The base mode equivalent service is M_INCLUDE.

WARNING: To ensure proper inclusion, the first eight characters of a shared image file name or partition name should be unique within the system.

Entry Conditions

Calling Sequence:

```
M.INCLUDE  [arga], [argb] [,cnpaddr]
(or)
LW         R1,arga
LD         R4,argb
LA         R7,cnpaddr (or) ZR   R7
SVC        2,X'40' (or) M.CALL   H.REMM,12
```

arga — is an address containing the address of a resource ID (RID), obtained when the partition was created
(or)
an address containing a pathname vector, containing the character count in byte 0 and the address of the pathname string in bytes 1, 2, and 3

argb — applies to dynamic partitions only; when supplied, specifies a doubleword address containing the left-justified task number of the original owner of the partition in word 0; word 1 is zero.
(or)
a one- to eight-character left-justified, blank-filled owner name

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, options field, and status field.

The option field has the following bit significance when set:

Bit	Meaning if Set
0	Read/write access requested
1	Reserved
2	Set exclusive resource lock
3	Reserved for MPX-32
4-15	Reserved

Defaults:

If a CNP is not supplied, the partition is included as read-only with no lock established.

Exit Conditions

Return Sequence:

(with CNP)		(without CNP)	
M.RTRN	R3,R5	M.RTRN	R3,R5
(or)		(or)	
M.RTNA	(CC1 set)	M.RTRN	R7 (CC1 set)

Registers:

R3	Starting logical address of the shared memory partition
R5	Allocation index, a unique 32-bit integer number that may be used to set and release exclusive or synchronous locks on the partition while it is allocated. It contains the nonzero biased SMT index in the first byte and the address of the associated ART entry in the next three.
R7	Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP

Value	Description
1	Memory partition definition not found
2	Specified access mode not allowed
8	Unrecoverable I/O error to volume
10	Dynamic partition definition exceeds one megabyte
16	Memory requirements conflict with task's address space
38	Time out occurred while waiting for shared memory to become available
50	Partition is exclusively locked
55	Allocated Resource Table (ART) is full
58	Shared Memory Table (SMT) space unavailable
80	Shared image version level is not compatible with executable image

Wait Conditions

When the partition is not available (status values 50 to 58), the task is placed in a wait state, as appropriate, if specified in the CNP.

6.2.64 M.INQUIRY - Resource Inquiry

The M.INQUIRY service obtains information specific to a resource allocated by a nonbase mode task. The information is returned in the form of a series of pointers to the various data structures within the system that describe the resource. The resource must have been previously allocated (included for memory partitions) by the caller. Resources are identified by: a logical file code obtained when the resource was allocated, a memory partition name defined when the partition was created, or an allocation index obtained when the resource was allocated/included. If not supplied as an argument, the caller is provided with the unique allocation index which can set and release exclusive or synchronous locks on the resource while it remains allocated. It is the caller's responsibility to interpret the information in the identified structures as the application dictates. It is recommended that this be performed by a user-supplied subroutine which acts as a common interface between application programs and this service. In this way, resource inquiries are less sensitive to changes in system structures.

The base mode equivalent service is M_INQUIRER.

Entry Conditions

Calling Sequence:

```
M.INQUIRY  [arga],[argb] [,cnpaddr]

(or)

LA         R1,arga
LD         R4,argb
LA         R7,cnpaddr  (or)  ZR  R7
SVC        2,X'48'  (or)  M.CALL  H.REMM,27
```

arga — is the address of an eight-word parameter description area where the pointers to the appropriate system structure entries corresponding to this resource are to be returned

argb — is a doubleword address containing byte 0 cleared and a one- to three-character (left-justified, blank-filled) LFC in bytes 1, 2, and 3 of word 0 with word 1 zero

(or)

a doubleword address where word 0 contains the address of a one- to eight-character (left-justified, blank-filled) memory partition name and word 1 contains the left-justified task number or address of a one- to eight-character (left-justified, blank-filled) owner name

(or)

a doubleword address in which word 0 is zero and the allocation index obtained when the resource was assigned is in word 1

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Calling Sequence:

(with CNP)	(without CNP)
M.RTRN R5	M.RTRN R5
(or)	(or)
M.RTNA R5 (CC1 set)	M.RTRN R5,R7 (CC1 set)

Registers:

R5	Allocation index if not supplied as an argument, or zero if resource is undefined
R7	Return status if a CNP is not supplied; otherwise, unchanged

The interpretation of each word in the parameter description area and some of the more pertinent information that can be extracted from each structure is as follows:

Word 0 - Allocated Resource Table (ART) address:
- Number of tasks assigned to this resource
- Number of tasks currently using resource
- Exclusive lock owner (DQE index)
- Synchronous lock owner (DQE index)
- Current allocation usage mode
- Current allocation access mode (implicit shared)
- Shared relative EOF block number (implicit shared)
- Shared relative EOM block number (implicit shared)

Word 1 - File Assignment Table (FAT) address:
- Relative EOF block
- Relative EOM block
- Number of segments in file
- Current segment number
- Current access mode
- Relative file block position
- Volume number (unformatted media only)
- Unformatted ID (unformatted media only)
- Assigned access restrictions
- File attribute and status flags

Word 2 - Unit Definition Table (UDT) address:
- Device type code
- Logical channel number
- Logical subchannel number
- Physical channel number (if different)
- Physical subchannel number (if different)
- Sectors per block (disc/floppy)
- Sectors per allocation unit (disc/floppy)
- Sectors per track (disc/floppy)
- Number of heads (disc/floppy)
- Total number of allocation units (disc/floppy)
- Sector size (disc/floppy)

Characters per line (TTY/terminal)
Lines per screen (TTY/terminal)
Tab size (TTY/terminal)
Tab settings (TTY/terminal)

Word 3 - Device Type Table (DTT) address:
Number of controller entries for device
ASCII device mnemonic
Device type code

Word 4 - Controller Definition Table (CDT) address:
Controller I/O class
Number of devices on controller
Device type code
Interrupt priority level
Logical channel number
Logical subchannel number of first device
Address of interrupt handler
Interrupt vector location
Controller definition flags

Word 5 - Shared Memory Table (SMT) address. Applies only to memory partitions:
Number of tasks queued to partition
Starting map register number
Memory type
Starting page number (static only)
Total number of pages (static only)
Number of map image descriptors
Map image descriptor list

Word 6 - File Pointer Table (FPT) address:
Logical file code associated with resource

Word 7 - Mounted Volume Table (MVT) address. Applies only to volume resources:
Volume name
Current number of users of volume
Volume definition flags
Root directory resource ID
Number of descriptors available on volume
Number of allocation units available
Volume access restrictions

Notes:

1. A value of zero returned in any word of the parameter description area implies the corresponding structure does not apply to the resource for which the inquiry was made. For example, only words 0 and 5 apply for memory partitions.

2. For volume resources, words 2 through 4 pertain to the device upon which the volume is mounted.

3. The MPX-32 Technical Manual contains a complete description of the various system structure contents.

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
5	Shared memory table entry not found for partition
29	Logical file code not assigned
30	Invalid allocation index

Wait Conditions

None

6.2.65 M.INT - Activate Task Interrupt

The M.INT service allows the calling task to cause the previously declared break/task interrupt receiver routine of the specified task to be entered.

The base mode equivalent services is M_INT.

Entry Conditions

Calling Sequence:

```
M.INT     task

(or)

ZR        R6          }     (or)  LD R6,taskname
LW        R7,taskno   }
SVC       1,X'6F'  (or)  M.CALL  H.REXS,47
```

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6 — Unchanged if R7 is zero. If R7 is not zero, bit zero of R6 is one if the specified task was not set up to receive a pseudointerrupt; otherwise bit zero is zero. Bits 1-31 of R6 are zero in all cases.

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file. Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

6.2.66 M.IPUBS - Set IPU Bias

The M.IPUBS service allows the user to dynamically change the IPU bias state for the current task.

The base mode equivalent services is M_IPUBS.

Entry Conditions

Calling Sequence:

```
M.IPUBS bias
```

or

```
LW      R7,bias
SVC     2,X'5B'   (or)   M.CALL  H.REXS,82
```

bias is the IPU bias state requested as follows:

Value	Description
0	Nonbiased task; for example, can be executed by either the CPU or IPU
1	CPU only; for example, can be executed only by the CPU
2	IPU bias; for example, can be executed by either the CPU or IPU but is given priority status by the IPU

Exit Conditions

Return Sequence:

```
M.RTRN   R6,R7
```

Registers:

R6 contains execution status as follows:

Value	Description
0	Normal return
1	IPU is not configured in the system
2	IPU is currently marked off-line

R7 IPU bias state of the task before this service was issued as follows:

Value	Description
0	Nonbiased task
1	CPU only
2	IPU bias

Abort Cases:

None

Output Messages:

None

6.2.67 M.LOC - Read Descriptor

The M.LOC service reads a resource descriptor for a specified resource. This service can examine the attributes of any volume resource. It is the responsibility of the caller to be familiar with the fields of the resource descriptor in order to determine the recorded information. It is recommended that this service is called by a user supplied subroutine(s) which acts as a common interface between application programs and this service. In this way, application programs are less sensitive to changes in organization and content of these data structures.

The base mode equivalent service is M_RREAD.

Entry Conditions

Calling Sequence:

```
M.LOCK    [arga],[rdaddr] [,cnpaddr]

(or)

LW        R1,arga
LA        R6,rdaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2C' (or) M.CALL  H.VOMM,13
```

arga — contains a PN vector, PNB vector, RID vector, LFC, or FCB address

rdaddr — is a RD buffer address; doubleword bounded, 192W length

cnpaddr — is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R4	M.RTRN R4
(or)	(or)
M.RTNA R2 (CC1 set)	M.RTRN R7,R2 (CC1 set)

Registers:

R2	Address of last PN item processed (abnormal return)
R4	MVTE address for specified volume
R7	Return status if a CNP is not supplied

6.2.68 M.LOCK - Set Exclusive Resource Lock

The M.LOCK service allows a task to obtain exclusive allocation of a resource, as though it were nonshareable, for as long as the lock is owned. The resource must have been previously allocated (included for memory partitions), and is identified by either a logical file code (defined when the resource was assigned) or an allocation index (obtained when the resource was assigned or by a resource inquiry). The task may request immediate denial if the lock is not available, or wait for an indefinite or specified period of time. An exclusive resource lock may be obtained for any allocated resource that is not being shared by multiple tasks at the time of the call to this service.

The base mode equivalent services is M_LOCK.

Entry Conditions

Calling Sequence:

```
M.LOCK  [arga][,cnpaddr]

(or)

LW          R5,arga
LA          R7,cnpaddr  (or)  ZR   R7
SVC         2,X'44'  (or)  M.CALL   H.REMM,23
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains a LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
29	Specified LFC is not assigned
30	Invalid allocation index
38	Time out occurred while waiting to become lock owner
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task
51	Resource is allocated to another task

Wait Conditions

The task is placed in a wait state, as appropriate, if specified with the CNP.

6.2.69 M.LOGR - Log Resource or Directory

The M.LOGR service provides the nonbase mode task with a convenient interface to locate the directory entry and resource descriptor for a single resource or for all the resources defined in a specified directory.

The caller must specify a resource specification in the Resource Logging Block (RLB). The log service will evaluate the resource specification and determine whether to log a single resource or all the resources defined in a directory. Some resource specifications are ambiguous and require the caller to specify additional information so the type of log function requested can be determined. Each resource specification is described in more detail below.

In order to log all the resources defined in a specified directory, the M.LOGR service must be called repeatedly until the last resource in the directory has been logged. The user must set bit 0 to zero in RLB.TYPE to indicate the first call. The operating system automatically changes the contents of bit 0 to one to indicate recall. Once all resources on the directory are logged, the operating system automatically changes bit 0 back to zero to indicate all resources have been logged.

Note: The M.LOGR system service does not search the Memory Resource Descriptor Table (MDT) for resource descriptors.

Resource Specifications for Pathnames

The caller can specify any valid pathname that is recognized by the Volume Management Module. The log service recognizes all valid pathname variations. However, some pathnames are ambiguous within the context of this service and require special considerations in order for the service to function with the expected results.

Specifically, pathnames that end with a directory specification are interpreted to mean log the contents of the directory. Directories can be logged as resources utilizing one of two methods. Method 1 furnishes a pathname that specifies the directory as a resource. This specification is not ambiguous. Method 2 furnishes a pathname that ends with a directory specification. This type of pathname is ambiguous and requires special handling.

Example of Method 1

```
@VOLUME(DIRECTORY)RESOURCE
```

This type of pathname always logs the directory entry and resource descriptor for the specified resource.

Examples of Method 2

```
@VOLUME(DIRECTORY)
```

This type of pathname normally means log the contents of the specified directory. The meaning of this pathname can be changed by setting the log single flag (RLB.LS) bit in the RLB flag word (RLB.INT). When the RLB.LS flag is set, the directory entry and resource descriptor for the specified directory are returned.

@VOLUME DIRECTORY

This type of pathname means log the specified directory. The directory entry and resource descriptor for the specified directory are returned.

Resource Specifications for Pathname Blocks

Pathname blocks are processed in the same manner as pathnames.

Resource Specifications for a Resource Identifier

When a resource identifier (RID) is furnished, the log service assumes the indicated resource is a directory and attempts to log the indicated resource as a directory.

Resource Specifications for a Logical File Code (LFC), FCB Address, or Allocation Index

When this type of resource specification is provided the log service makes the following assumptions:

- The implied File Control Block (FCB) is assigned to a directory.
- The implied File Control Block (FCB) is opened.
- The buffer address contained in the FCB is the buffer to be used by the log service for locating directory entries.
- The transfer quantity contained in the FCB is the maximum size of the directory entry buffer.
- The FCB must be an extended FCB and must be opened in random access mode.
- The log service assumes the buffer is empty on the initial call and positions to the beginning of the directory and primes the supplied buffer. The directory is not read again until it is exhausted.

The caller should assign the directory in read mode so the directory can be searched by other users as it is being logged.

The base mode equivalent service is M_LOGR.

Entry Conditions

Calling Sequence:

```
M.LOGR    [rlbaddr] [,cnpaddr]

(or)

LA        R2,rlbaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'29' (or) M.CALL  H.VOMM,10
```

rlbaddr is the address of the Resource Logging Block (RLB)

cnpaddr is a CNP address or zero if not supplied

RLB Structure on Initial Call

Word		
0	PN vector or RID vector or zero (RLB.TGT). See Note 1.	
1	192-word buffer address or zero (RLB.BUFA). See Note 2.	
2	Reserved for system use	
3	Reserved for system use	
4	Type (RLB.TYPE) See Note 3.	zero (RLB.BOFF). See Note 3.
5	Length. See Note 4.	Directory return buffer address (RLB.DIRA). See Note 4.
6	User FCB address or zero (RLB.FCB). See Note 5.	
7	Flags (RLB.INT). See Note 6.	

Notes:

1. If the PN vector (length and address) specifies a resource, only one item is logged. If it does not end with a resource, but with a directory, the entire directory may be logged by repeated calls. A call by RID vector implies the RID is for a directory and all entries may be logged. A value of zero implies the entire contents of the current working directory.
2. If this field is zero, the RD is not returned.
3. The type value should be zero if the call is by PN vector (length and address) or zero to indicate working directory. Type should be one to indicate a call by RID. If all resources in a directory are to be logged, bit 0 of RLB.TYPE must be set to zero to indicate the first call.
4. This word contains the address of a buffer and its length in words. The buffer may be up to 16 words long. The log service will place the first n words of the logged directory entry into this buffer. This provides the user access to the file name and other attributes that exist only in the directory entry.
5. This service uses the system FCB by default. There is a potential for phasing problems as the directory to be logged must be deassigned between calls if multiple entries are desired. In many cases, the impact of having an entry deleted just after it has been logged, or having an entry appear after that spot in the directory has been scanned, will be small or nonexistent. In other cases, such as saving files in a directory, it may be major. To prevent these problems, the user may provide the address of a FCB that will be used to hold the directory while logging occurs.
6. Bits in this word are assigned as follows:

Bit	Description
0-1	Reserved
2	If set, directory entry and resource descriptor for specified directory are returned (RLB.LS)
3	Reserved
4	Used on return to indicate if resource was located (see RLB Structure on Return described below)
5-31	Reserved

Exit Conditions

(with CNP)	(without CNP)
M.RTNA (CC1 set)	M.RTRN R7

Registers:

R7 Return status if a CNP is not supplied; otherwise CNP address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

RLB Structure on Return

Word		
0	PN vector or RID vector or zero (RLB.TGT)	
1	192-word buffer address or zero (RLB.BUFA)	
2	MVTE address (RLB.MVTE). See Note 1.	
3	Disc address of directory RD (RLB.RDAD).	
4	Type (RLB.TYPE). See Note 2.	Byte offset of entry (RLB.BOFF).
5	Length	Directory return buffer address (RLB.DIRA)
6	User FCB address or zero (RLB.FCB)	
7	Flags (RLB.INT). See Note 3.	

Notes:

1. When all resources in a directory are to be logged, RLB.MVTE and RLB.RDAD are used by the operating system as input after the first call.

2. The operating system automatically changes the contents of bit 0 in RLB.TYPE as follows:

Value	Description
0	All resources in the directory have been logged; do not recall this service
1	Recall this service and log the next resource in the directory

3. Bits in this word are assigned as follows:

Bit	Contents
0-3	Reserved
4	Zero if resource was not located; one if resource was located
5-31	Reserved

6.2.70 M.MEM - Create Memory Partition

The M.MEM service creates permanent memory partition definitions. Permanent memory partition definitions are given names in directories and remain known to the operating system until explicitly deleted.

Memory partition definitions are the mechanism used by the operating system to establish the relationship of named globally accessible areas of memory to the tasks that require them.

This service allocates a resource descriptor and defines the memory requirements for the partition. Next, the attributes of the partition are recorded in the resource descriptor. As a final step, the name of the partition is established in the indicated directory.

When a directory entry is established, the directory entry is linked to the resource descriptor for the partition. This link makes the relationship of the name of the partition to the other attributes of the partition. Typical partition attributes are:

- The partition's name
- The partition's resource identifier (RID)
- The partition's protection attributes
- The partition's management attributes
- The partition's memory requirements

The base mode equivalent services is M_MEM.

Entry Conditions

Calling Sequence:

```
M.MEM     [arga],[rcbaddr][,cnpaddr]

(or)

LW        R1,arga
LA        R2,rcbaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'22' (or) M.CALL  H.VOMM,3
```

arga contains a PN vector or PNB vector

rcbaddr is the RCB address (required)

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.71 M.MEMB - Get Memory in Byte Increments

The M.MEMB service allows the task to dynamically expand its memory allocation in doubleword increments starting at the end of its DSECT up to the top of its logical address space. The additional memory will be of the same type specified when the task was cataloged. The task is mapped in a logically contiguous manner up to the end of its address space. The task is suspended until the allocation is successful. Repeated calls to this service are allowed. Allocation is not contiguous with previously allocated space.

This service cannot be used with the M.GE or M.GD services, or a call to H.MEMM,14.

The base mode equivalent service is M_GETMEMBYTES.

Entry Conditions

Calling Sequence:

```
M.MEMB  num

(or)

LW      R4,=num (or) LI R4,num
SVC     2,X'4B' (or) M.CALL  H.REMM,28
```

num is the number of bytes to allocate

Exit Conditions

Return Sequence:

```
M.RTRN  R3,R4
```

Registers:

CC1	Equals zero
CC2	Equals zero
R3	Contains the 24-bit starting logical doubleword address of allocated space
R4	Contains the number of bytes actually allocated (modulo 2W)

(or)

CC1	Equals zero
CC2	Equals zero
R3	Contains the 24-bit starting logical doubleword address of allocated space
R4	Contains the number of bytes actually allocated (modulo 2W). However, the number is less than requested.

Error Conditions

Allocation Denied:

CC1	Equals one
CC2	Equals one
R3	Equals zero
R4	Equals zero

6.2.72 M.MEMFRE - Free Memory in Byte Increments

The M.MEMFRE service allows the task to dynamically deallocate acquired memory. Deallocation can be random. The space address must have been previously obtained from the M.MEMB service. All of the space obtained from a given call is deallocated.

This service cannot be used with the M.FE or M.FD services.

The base mode equivalent service is M_FREEMEMBYTES.

Entry Conditions

Calling Sequence:

```
M.MEMFRE  addr

(or)

LW        R3,addr
SVC       2,X'4C' (or) M.CALL  H.REMM,29
```

addr is the starting address of a previously acquired dynamic space from the M.MEMB service

Exit Conditions

Return Sequence:

M.RTRN R3 (or) abort user with RM77

Registers:

R3 Equals zero if deallocation could not be performed. Deallocation address was not found in allocation table

Abort Cases:

RM77 A task has destroyed the allocation linkages in this dynamic expansion space

6.2.73 M.MOD - Modify Descriptor

The M.MOD service allows the owner of a resource to change or alter the protection or other resource management attributes of a resource. The owner can restrict or allow attributes with this mechanism.

Because of complications or confusion that can arise when certain attributes are changed, this service limits the information in a descriptor that can be changed. Information such as the volume space occupied by the resource cannot be changed as this would allow the caller to violate the integrity of the volume on which the resource resides.

The caller is allowed the following modifications:

- The protection fields of the descriptor
- The accounting fields of the descriptor
- The extension attribute fields of the descriptor
- User data field of the descriptor (words 160 through 175)
- The shared image field of the descriptor

This service is the first part of a two step operation. The caller is required to read the resource descriptor into memory in order to modify it. Once read into memory, the resource descriptor is locked (for example, protected from access) until the caller writes the modified descriptor back to the volume with the Rewrite Descriptor (M.REWRIT) service. The caller must issue the rewrite before modifying another descriptor.

Only the resource owner or the System Administrator are allowed to modify a resource descriptor. The format of the descriptor and the type of data to be modified must be known by the modifier.

The base mode equivalent service is M$\underline{\text{I}}$MOD.

Entry Conditions

Calling Sequence:

```
M.MOD     [arga],[rdaddr][,cnpaddr]

(or)

LW        R1,arga
LA        R6,rdaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2A' (or) M.CALL  H.VOMM,11
```

arga contains a PN vector, PNB vector, or RID vector

rdaddr is a RD buffer address (doubleword bounded, 192W length)

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.74 M.MODU - Modify Descriptor User Area

The M.MODU service allows users with write, update, append, or modify access to a resource to change or alter the user area of the resource descriptor of that resource.

Although the user is allowed to change all fields in the user area of the resource descriptor, words 176 through 190 are used by some utilities. Word 191 of the resource descriptor is a reserved location, and any changes to this word are ignored. For these reasons, the user is advised to use only words 160 through 175 of the resource descriptor.

This service is the first part of a two step operation. The caller is required to read the user area of the resource descriptor into memory in order to modify it. Once read into memory, the resource descriptor is locked (for example, protected from access) until the caller writes the modified user area back to the volume with the Rewrite Descriptor User Area (M.REWRTU) service. The caller must issue the rewrite before modifying another descriptor or descriptor user area.

The base mode equivalent services is M_MODU.

Entry Conditions

Calling Sequence:

```
M.MODU [arga],[uaaddr][,cnpaddr]

(or)

LW    R1,arga
LA    R6,uaaddr
LA    R7,cnpaddr
SVC   2,X'31'   (or)   M.CALL * H.VOMM,26
```

arga — contains a PN vector, PNB vector, or RID vector

uaaddr — is a user area buffer address (doubleword bounded, 32W length)

cnpaddr — is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.75 M.MOUNT - Mount Volume

The M.MOUNT service establishes a process as a user of a volume. If the volume has not been previously mounted, the operator is notified to mount the volume, and a Mounted Volume Table Entry (MVTE) is created for the volume. This entry remains memory resident as long as there are established users of the volume. A Volume Assignment Table (VAT) entry is allocated within the user's TSA for nonpublic volumes, provided the requested usage classification is compatible. Mount requests for an already mounted volume are ignored.

The base mode equivalent service is M_MOUNT.

Entry Conditions

Calling Sequence:

```
M.MOUNT [ rrsaddr] [,cnpaddr]

(or)

LA      R1,rrsaddr
LA      R7,cnpaddr  (or)  ZR   R7
SVC     2,X'49'  (or)  M.CALL   H.REMM,17
```

rrsaddr — is the address of a RRS entry (Type 9). See description of Resource Requirement Summary entries, Section 5.2.4.1.

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
1	Invalid volume name
2	Volume use not allowed to this user
6	Volume Assignment Table (VAT) space not available
8	Unrecoverable I/O error to volume
11	Invalid RRS entry
13	Mount device not in system
18	Invalid mount device specified
20	Unable to initialize volume (volume unsafe)
21	J.MOUNT run request failed
34	Attempt to mount a public volume without the System Administrator attribute
37	Invalid J.MOUNT request
38	Time out occurred while waiting for resource
39	Volume already mounted
53	Mount device unavailable
55	Allocated Resource Table (ART) space not available for mount device
57	Unable to mount volume for requested usage
59	Mounted Volume Table (MVT) space not available
73	File overlap occurred. Check system console.

Wait Conditions

When the volume is unavailable (status values 50-63), the task is placed in a wait state, as appropriate.

6.2.76 M.MOVE - Move Data to User Address

The M.MOVE service is used to move an arbitrary number of bytes from a location in the user's logical address space to a location in the user's read only memory section or the user's read/write memory section. This service can be used to set traps or modify data in the user's memory sections. This service cannot be used in shared read only sections.

The base mode equivalent service is M_MOVE.

Entry Conditions

Calling Sequence:

```
M.MOVE [BUFFER=]addr1 , [DESTADDR=]addr2 , [NUMBER=]num

(or)

LA      R1, addr1
LA      R2, addr2
LI      R4, num
SVC     2,X'62' (or) M.CALL H.REXS,89
```

addr1 is the byte address of the buffer to be moved

addr2 is the destination byte address

num is the number of bytes to be moved

Registers:

R1	Contains addr1
R2	Contains addr2
R4	Contains num

Exit Conditions

Normal Return Sequence:

```
M.RTRN
```

Abnormal Return Sequence:

```
M.RTRN  R7
```

Registers:

CC1 set Error condition

R7 Contains error condition (hexadecimal equivalent) as follows:

Value	Description
256	Invalid source buffer address
257	Destination buffer is in the operating system
258	Destination buffer is in the TSA
259	Invalid destination buffer address

6.2.77 M.MYID - Get Task Number

The M.MYID service allows the user to obtain status on the currently executing task.

The base mode equivalent services is M_MYID.

Entry Conditions

Calling Sequence:

```
M.MYID    pbaddr
(or)
ZR        R5
SBR       R5,0
LA        R7,pbaddr
SVC       1,X'64' (or)  M.CALL  H.REXS,32
```

pbaddr is the logical word address of the first location of a parameter block formatted as follows:

Word	Contents
0	Task activation sequence number
1-2	Task load module name
3-4	Owner name
5-6	Pseudonym
7-8	Current working directory, truncated to the first eight characters
9	Reserved
10	Scheduling flags (DQE.USHF)

Exit Conditions

Return Sequence:

```
M.RTRN    5
```

Registers:

CC1 set	Invalid parameter block address
R5,R7	Unchanged

Abort Cases:

RX32	Invalid DQE address

Output Message:

None

6.2.78 M.NEWRRS - Reformat RRS Entry

The M.NEWRRS service converts an RRS entry from MPX-32 Release 1.x format to the format acceptable for assignment processing by the Resource Management Module (H.REMM). See MPX-32 Release 1.x Technical Manual. It is intended for compatibility purposes and should only be used when internal recoding of the RRS entry to the new format is impractical. This service is automatically called during parameter task activation when an incompatible RRS format is encountered in the activation parameter block. This results in additional overhead during the activation of these tasks.

Entry Conditions

Calling Sequence:

```
M.NEWRRS  rrsaddr,newaddr [,cnpaddr]

(or)

LA        R1,rrsaddr
LA        R5,newaddr
LA        R7,cnpaddr  (or)  ZR  R7
SVC       2,X'54'  (or)  M.CALL  H.REXS,76
```

rrsaddr is the address of the RRS entry to be reformatted

newaddr is the address where the reformatted RRS entry is to be built

Restrictions:

The area where the reformatted RRS entry is to be built must be large enough to account for the expanded size of the new RRS. Use the following guidelines to determine the number of words required to accommodate the expansion:

MPX-32 1.x RRS type	Size if reformatted RRS
ASSIGN1	12 words
ASSIGN2	4 words
ASSIGN3 (disc)	4-8 words
ASSIGN3 (device)	6 words
ASSIGN4	4 words

cnpaddr is the address of a Caller Notification Packet (CNP) if notification is desired. See description of CNP in Chapter 5.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP

Value	Description
11	Invalid RRS entry
20	Specified volume not in system

Wait Conditions

None

6.2.79 M.OLAY - Load Overlay Segment

The M.OLAY service is provided for loading an overlay segment into a nonbase mode task. The actual loading is performed by the Task Management Module (H.TAMM) and control is returned to the caller upon completion. The named segment must have been defined to the Cataloger as an overlay.

Control can be transferred to the overlay segment at its cataloged transfer address rather than to the caller.

Entry Conditions

Calling Sequence:

```
M.OLAY    filename [ ,EXE]

(or)

LD        R6,filename
{SVC      1,X'50'     (or)    M.CALL  H.REXS,13 }(or)
{SVC      1,X'51'     (or)    M.CALL  H.REXS,14 }
```

filename — is a doubleword containing the name of the file from which the overlay segment is to be loaded, one to eight ASCII characters, left-justified and blank-filled

EXE — specifies transfer control to the overlay (SVC1,X'51'). If not specified, the default is loads overlay without transfer (SVC 1,X'50').

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Transfer address of the overlay segment or unchanged if the EXE parameter is specified

Abort Cases:

RX07 LD01-08	Cannot load overlay segment due to software checksum or data error
RX08	Overlay is not in the directory
RX10	Overlay has an invalid preamble
RX11	Unrecoverable I/O error during overlay loading
RX33	Overlay linkages have been destroyed by loading a larger overlay

Output Messages:

None

6.2.80 M.OPENR - Open Resource

The M.OPENR service prepares a resource for logical I/O and defines the intended access mode for subsequent operations on the resource. Protection is provided for both the requestor and the resource against indiscriminate access. If appropriate, additional FAT information is posted at this time. A blocking buffer can be allocated if not previously specified, explicitly or implicitly, during allocation of the resource. However, if a user-supplied buffer is specified in the FCB, it is used and any previously allocated blocking buffer is released. A mount message is issued as a result of this function when the I/O is to be performed to a device associated with unformatted media, and the message has not been inhibited by user request or previous open on the resource by another user. An open request to a resource that is already opened in the same access mode is ignored.

The base mode equivalent service is M_OPENR.

Entry Conditions

Calling Sequence:

```
M.OPENR  [fcbaddr] [,cnpaddr]

(or)

LA       R1,fcbaddr
LA       R7,cnpaddr  (or)  ZR   R7
SVC      2,X'42'  (or)  M.CALL   H.REMM,21
```

fcbaddr is the address of a File Control Block (FCB)

cnpaddr is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, options field, and status field.

The options field describes the access and usage specification with the following interpretation:

Byte 0 contains an integer value representing the specific access for which the resource is being opened. The following values are valid:

Value	Description
0	Open for default access
1	Open for read
2	Open for write (resource redefined)
3	Open for modify
4	Open for update
5	Open for append
6-255	Reserved

Byte 1 indicates the usage under which the resource is to be opened with the following significance:

Bit	Meaning if Set
0	Open for explicit shared use
1	Open for exclusive use
2	Open in unblocked mode (overrides any specification made at resource assignment)
3	Open in blocked mode (overrides any specification made at resource assignment)
4	Resource data blocked. If the file is actually written to in any access mode (append, modify, update, write), the data will be recorded as blocked in the resource descriptor at the time the file is closed, regardless of whether or not the I/O was actually performed in blocked mode.
5	Override with implicit shared use
6-7	Reserved

Restrictions:

1. An error condition is returned if an invalid integer value is present in byte 0.

2. Only one of bits 0 and 1 in byte 1 can be set. If set, any usage specified at the time the resource was assigned is overridden. This can result in a denial condition if the usage specified at open differs from that specified at assignment.

Defaults:

If a CNP is not supplied or the specification in the options field is zero: the resource is opened for read access (volume resource) or update access (device) unless only a specific access mode was allowed at assignment, in which case the resource is opened for that access; the usage is implicit shared or that specified at resource allocation, whichever is appropriate.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
2	Specified access mode not allowed
4	Blocking buffer not available
9	Invalid usage specification
25	Random access not allowed for this access mode
26	User attempting to open SYC file in a write mode
27	Resource already opened in a different access mode
28	Invalid access specification
29	No LFC which matches FCB file code
38	Time out occurred waiting for resource
46	Unable to obtain resource descriptor lock (multiprocessor only)
51	Shareable resource is allocated by another task in an incompatible mode
54	Unable to allocate resource for specified usage

Wait Conditions

If an access/usage specification is made at open which changes the nature of the resource allocation, the task may be placed in a wait state, as appropriate, if specified in the CNP.

6.2.81 M.PGOW - Task Option Word Inquiry

The M.PGOW service provides the caller with the 32-bit task option word (same as program option word).

This service can be executed by the IPU.

The base mode equivalent service is M_OPTIONWORD.

Entry Conditions

Calling Sequence:

```
M.PGOW

(or)

SVC   1,X'4C' (or) M.CALL   H.REXS,24
```

Exit Conditions

Return Sequence:

```
M.IPURTN   7
```

Registers:

R7 Contains the 32-bit task option word

Abort Cases:

None

Output Messages:

None

6.2.82 M.PNAM - Reconstruct Pathname

The M.PNAM service constructs and returns the pathname string that was used to assign a file. In most cases, this service acquires the complete name of a file that was statically assigned to a task. If a pathname component contains special characters, the component is returned enclosed within single quotes.

The base mode equivalent service is M_CONSTRUCTPATH.

Entry Conditions

Calling Sequence:

```
M.PNAM    [arga],[pnaddr] [,cnpaddr]

(or)

LW        R1,arga
LW        R4,pnaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2F' (or) M.CALL  H.VOMM,16
```

arga is a FCB address or LFC for the assigned volume resource

pnaddr is the PN address and maximum length

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R4	M.RTRN R4
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R4	Actual PN length and PN address
R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.83 M.PNAMB - Convert Pathname to Pathname Block

The M.PNAMB service converts a pathname to a form that can be more easily analyzed by software. In most cases, utility programs use this to syntax check a pathname that has been input on a directive line.

When called, this service parses the input pathname. If any errors are detected in the pathname syntax, this service is terminated and register 1 is updated to indicate the point where the error was detected.

The base mode equivalent services is M_PNAMB.

Entry Conditions

Calling Sequence:

```
M.PNAMB   [pnaddr], [pnbaddr] [,cnpaddr]
(or)
LW        R1,pnaddr
LW        R4,pnbaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2E' (or) M.CALL  H.VOMM,15
```

pnaddr contains a PN vector

pnbaddr contains a PNB vector

cnpaddr is a CNP address or zero if CNP not supplied

Applicable portions of the CNP for this function are time-out value, abnormal return address, option field, and status field. The options field of the CNP has the following interpretation:

Byte 0 is reserved

Byte 1 has the following significance when set:

Bit	Meaning if Set
0-6	Reserved
7	Parsing of the pathname includes only the volume and directory portions of the supplied pathname. This bit is usually set by J.TSM.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R1,R4	M.RTRN R1,R4
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R1	Address of first PN character not processed and remaining length
R4	PNB address and actual PNB length
R7	Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.84 M.PRIL - Change Priority Level

The M.PRIL service is provided to the privileged caller who wishes to dynamically alter the priority level of the specified task. Valid priority levels for real-time tasks are 1-54 inclusive. Valid priority levels for time distribution tasks are 55-64 inclusive. A real-time task cannot be changed to a time distribution priority level and a time distribution task cannot be changed to a real-time priority level. I/O continues to operate at base priority level of the cataloged task.

The base mode equivalent service is M_PRIL.

Entry Conditions

Calling Sequence:

```
M.PRIL    task,priority

(or)

LW        R5,priority
ZR        R6          }      (or)  LD R6,taskname
LW        R7,taskno   }
SVC       1,X'4A'  (or)  M.CALL   H.REXS,9
```

task — the address of a doubleword containing the name of the task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

priority — is the priority level to be assigned to the task (1-54 for a real-time task; 55-64 for a time-distribution task)

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found; otherwise, contains the task number

Abort Cases:

RX06 — Specified priority not in 1-64 range

Output Messages:

None

6.2.85 M.PRIV - Reinstate Privilege Mode to Privilege Task

The M.PRIV service allows a task that was cataloged as privileged to return to a privileged status. See M.UPRIV service.

The base mode equivalent service is M_PRIVMODE.

Entry Conditions

Calling Sequence:

```
M.PRIV

(or)

SVC     2,X'57'     (or)  M.CALL  H.REXS,78
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

CC1 set Successful operation

6.2.86 M.PTSK - Parameter Task Activation

The M.PTSK service is provided to the privileged caller who wishes to override specific task parameters (contained in the load module or executable image preamble) during activation the task name, optional resource requirements, and optional pseudonym are specified to the service call. When a task name is supplied in words two and three of the Parameter Task Activation Block, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied.

The base mode equivalent service is M_PTSK.

Entry Conditions

Calling Sequence:

```
M.PTSK      actaddr [,psbaddr]

(or)

LA          R1,actaddr
LA          R2,psbaddr (or) ZR  R2
SVC         1,X'5F' (or) M.CALL  H.REXS,40
```

actaddr is the logical word address of the first location of an activation parameter block formatted as shown below

PARAMETER BLOCK

(Words 0-12 are required. The RRS entries (words 13-n) are optional as indicated by byte 1.)

Word 0	Byte 0	Bit	Contents
		1	Reserved
		2	Terminal task
		3	Batch task
		4	Debug overlay required
		5	RTM resident (ESTABLISH)
		6	Command file active
		7	SLO assigned to SYC
	Byte 1		Number of resource requirements or zero if same as summary entries in the load module or executable image preamble
	Byte 2		Memory requirement equals number of 512-word pages exclusive of TSA or zero if memory requirements are to be taken from the preamble

Byte 3 — Memory class (ASCII E, H or S) or zero if memory class to be taken from the preamble

(or)

If the memory class is to be taken from the preamble, the caller has the option of specifying the task's logical address space in this field as follows:

Bits	Contents
0-3	Hexadecimal value 0 through F representing the task's logical address space in megabytes where zero is 1MB and F is 16MB
4-7	Zero

Word 1 — Byte 0 is the number of blocking buffers required or zero if same as the preamble

Byte 1 is the number of FAT/FPT pairs (files) to be reserved or zero if same as the preamble

Byte 2 is the priority level of task being activated or zero if same as the preamble

Byte 3 is the number of segment definition areas to be reserved for extendible files or zero if same as the preamble

Word 2 — Contains the first four bytes of the ASCII character left-justified, blank-filled load module or executable image name if word 3 contains the last four bytes; else zero

Word 3 — Contains the last four bytes of the ASCII character left-justified, blank-filled load module or executable image name if word 2 contains the first 4 bytes; else contains a pathname vector or RID vector if word 2 is zero

Word 4,5 — One- to eight-character ASCII left-justified, blank-filled pseudonym to be associated with task or zero if no pseudonym is desired. Pseudonyms are intended to be unique - the responsibility for uniqueness rests with the caller.

Word 6,7 — One- to eight-character ASCII left-justified, blank-filled owner name to be associated with task or zero if task to default to current owner name. Valid only when task has System Administrator attribute.

Word 8,9 — One- to eight-character ASCII left-justified, blank-filled project name to be associated with files referenced by this task or zero if same as LMIT

Word 10 — Byte 0 is the number of Volume Assignment Table (VAT) entries to be reserved for dynamic mount requests or zero if same as the preamble

Byte 1 is reserved

Bytes 2 and 3 contain the map block number for base mode MPX-32, or a negative two for MINADDR, or a negative one for MAXADDR

Word 11	Contains the initial value of the task option word or zero
Word 12	Contains the initial value of the task status word or zero
Words 13-n (optional)	Resource Requirement Summary List - each entry contains a variable length RRS. The RRS list is limited to a maximum of 384 words. Each entry is compared with the RRS entries in the LMIT. If the logical file code currently exists, the specified LFC assignment will override the cataloged assignment, otherwise it will be treated as an additional requirement (merged). If MPX-32 Release 1.x format of the RRS is specified, it is converted to the format acceptable for assignment processing by the Resource Management Module (H.REMM). See MPX-32 Release 1.x Technical Manual for format of the RRS.

psbaddr is the logical address of the Parameter Send Block (PSB) or zero if no parameters are to be passed. See Chapter 2 for PSB description.

Exit Conditions

Return Sequence:

M.RTRN 6,7

Registers:

R0	Destroyed
R6	Equals zero if the service could be performed
R7	Contains the task number of task activated by this service
(or)	
R0	Destroyed
R6	Equals one if invalid attempt to multicopy a unique task
R7	Task number of existing task with same name
(or)	
R0	Destroyed

R6

Value	Description
2	File specified in words two and three of the PSB not in directory
3	Unable to allocate file specified in words two and three of the PSB
4	File is not a valid load module or executable image
5	DQE is not available
6	Read error on resource descriptor
7	Read error on load module
8	Insufficient logical/physical address space for task activation
9	Calling task is unprivileged
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address

Value	Description
14	Memory pool unavailable
15	Destination task receiver queue full
16	Invalid PSB address
17	RRS list exceeds 384 words
18	Invalid RRS entry in parameter block

Abort Cases:

None

Output Messages:

None

6.2.87 M.QATIM - Acquire Current Date/Time in ASCII Format

The M.QATIM service acquires the system date and time in ASCII format. The date and time are returned in a four word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The base mode equivalent service is M_QATIM

Entry Conditions

Calling Sequence:

```
M.QATIM   addr

(or)

LA        R1,addr
ORMW      R1,=X'03000000'
SVC       2,X'50'  (or)  M.CALL   H.REXS,74
```

addr is the address of a four-word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

6.2.88 M.RADDR - Get Real Physical Address

The M.RADDR service allows unprivileged tasks to determine the physical memory address associated with a given logical address.

The base mode equivalent services is M_RADDR.

Entry Conditions

Calling Sequence:

```
M.RADDR [logicaladdr]

(or)

LA      R1, logicaladdr
SVC     1,X'0E'
```

logicaladdr is the logical address to be translated

Exit Conditions

Return Sequence:

Registers:

R7 Contains the physical address

Abort Cases:

None

Output Messages:

None

6.2.89 M.RCVR - Receive Message Link Address

The M.RCVR service allows the caller to establish the address of a routine to be entered for the purpose of receiving messages sent by other tasks.

The base mode equivalent services is M_RCVR.

Entry Conditions

Calling Sequence:

```
M.RCVR    recvaddr

(or)

LA        R7,recvaddr
SVC       1,X'6B' (or) M.CALL  H.REXS,43
```

recvaddr is the logical word address of the entry point of the receive message routine in the user's task

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 Contains zero if the receiver address was invalid, otherwise contains the receiver address

Abort Cases:

None

Output Messages:

None

6.2.90 M.READ - Read Record

The M.READ service performs the following functions:

- Provides special random access handling for disc files.
- Deblocks system files and blocked files.
- Reads one record into the buffer indicated by the Transfer Control Word (TCW) in the FCB.

The base mode equivalent service is M_READ.

Entry Conditions

Calling Sequence:

```
M.READ    fcb

(or)

LA        R1,fcb
SVC       1,X'31' (or) M.CALL H.IOCS,3
```

fcb is the FCB address. Appropriate transfer control parameters are defined in the TCW. See Section 5.9.1.2.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03	Nonprivileged user attempting transfer to a logical address outside legal boundaries
IO06	Invalid blocking buffer control cell for a system or blocked file
IO26	Read attempted for a system or blocked file while write in process
IO30	Illegal volume record. Either volume number or reel ID from volume record do not match FAT information.
IO32	Second attempt to read a $ statement in an SYC file
RM02	Attempt to read from SLO/SBO in update mode

Output Messages:

Dismount/mount messages if EOT and multivolume magnetic tape

6.2.91 M.RELP - Release Dual-ported Disc/Set Dual-channel ACM Mode

The M.RELP service applies to dual-port extended I/O discs and allows the privileged user to release a device from its reserved state. When issued to an eight-line that has been SYSGENed as full-duplex, this service can be used to set the eight-line from single-channel to dual-channel mode (applies to ACMs using the H.F8XIO handler only).

The base mode equivalent service is M_RELP.

Entry Conditions

Calling Sequence:

```
M.RELP      fcb

(or)

LA          R1,fcb
SVC         1,X'27'  (or)  M.CALL  H.IOCS,27
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.92 M.RENAM - Rename File

The M.RENAM service changes the name of an existing permanent file. This service can move a file from one directory to another directory on the same volume.

When called, this service creates the new name of the file in the specified directory and then deletes the old name of the file from the specified directory.

The base mode equivalent service is M_RENAME.

Entry Conditions

Calling Sequence:

```
M.RENAM   [arga],[newaddr][,cnpaddr]

(or)

LW        R1,arga
LW        R2,newaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2D' (or) M.CALL  H.VOMM,14
```

arga contains the old PN or PNB vector

newaddr contains the new PN or PNB vector

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA R7 (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.93 M.REPLAC - Replace Permanent File

The M.REPLAC service replaces the data contents of an existing permanent file with the data contents of an existing temporary file. At the completion of this service, the permanent file retains its original directory entry and resource descriptor.

This service is provided so utility programs can change the data contents of a file without changing any of the file's other attributes. In other words, this service maintains the integrity of a file's resource identifier and therefore provides the fast file mechanism.

This service can be used on any permanent file. At the completion of this service, the temporary file is deallocated and deleted. An error condition is returned if the permanent file is allocated to another task at the time of the service call, and bit 0 of the CNP option field is not set.

This service should only be used on files with the fast access attribute. For files which do not have the fast access attribute, this same functionality can be accomplished by using the Delete Resource (M.DELR) service followed by the Change Temporary File to Permanent File (M.TEMPER) service.

The base mode equivalent service is M_REPLACE.

Entry Conditions

Calling Sequence:

```
M.REPLAC [fcbaddr],[pnaddr] [,cnpaddr]

(or)

LA      R1,fcbaddr
LW      R2,pnaddr
LA      R7,cnpaddr  (or)  ZR      R7
SVC     2,X'30'     (or)  M.CALL  H.VOMM,23
```

fcbaddr — is the FCB or LFC address of the temporary file, or value in R1, if not supplied

pnaddr — is the pathname vector of the permanent file, or the value in R2, if not supplied

cnpaddr — is a CNP address or zero if CNP is not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

CC1 set	Error condition
R7	Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.94 M.RESP - Reserve Dual-ported Disc/Set Single-channel ACM Mode

The M.RESP service applies to dual-port extended I/O discs and allows the privileged user to reserve a device to the requesting CPU until such time as a release (M.RELP) is issued. When issued to an ACM that has been SYSGENed as full-duplex, this service can reset the ACM from dual-channel to single-channel mode (applies to ACMs using the H.F8XIO handler only).

The base mode equivalent service is M_RESP.

Entry Conditions

Calling Sequence:

```
M.RESP      fcb

(or)

LA          R1,fcb
SVC         1,X'26' (or) M.CALL  H.IOCS,24
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.95 M.REWRIT - Rewrite Descriptor

The M.REWRIT service writes a modified resource descriptor back to a volume and releases the modify lock on the descriptor. This is the last step of a two step operation (the first step is M.MOD).

When this service is invoked, the indicated resource descriptor is read into an internal buffer. The fields that are allowed to be modified are copied from the user supplied resource descriptor buffer to the appropriate areas of the internal buffer. Upon successful modification of the resource descriptor in the internal buffer, the resource descriptor is written to the correct location on the volume and the modify lock is released.

The base mode equivalent service is M_REWRIT.

Entry Conditions

Calling Sequence:

```
M.REWRIT   [rdaddr][,cnpaddr]

(or)

LA         R6,rdaddr
LA         R7,cnpaddr (or) ZR  R7
SVC        2,X'2B' (or) M.CALL  H.VOMM,12
```

rdaddr — is the RD buffer address. This must be the same address supplied by the caller for use with the associated Modify Descriptor (H.VOMM,11) call; doubleword bounded, 192W length

cnpaddr — is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.96 M.REWRTU - Rewrite Descriptor User Area

The M.REWRTU service writes a modified resource descriptor back to a volume and releases the modify lock on the descriptor. This is the last step of a two step operation (the first step is M.MODU).

When this service is invoked, the indicated resource descriptor is read into an internal buffer. The data from the buffer supplied by the user is then copied to the appropriate areas of the internal buffer. Upon successful modification of the resource descriptor in the internal buffer, the resource descriptor is written to the correct location on the volume and the modify lock is released.

The base mode equivalent service is M_REWRTU.

Entry Conditions

Calling Sequence:

```
M.REWRTU  [uaaddr] [,cnpaddr]
(or)
LA        R6,uaaddr
LA        R7,cnpaddr  (or)  ZR  R7
SVC       2,X'32'  (or)  M.CALL  H.VOMM,27
```

uaaddr — is the user area buffer address. This must be the same address supplied by the caller for use with the associated Modify Descriptor User Area (H.VOMM,26) call; doubleword bounded and 32W length.

cnpaddr — is a CNP address or zero if CNP not supplied.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.97 M.ROPL - Reset Option Lower

The M.ROPL service allows the calling task to reset the option lower bit. Use the M.SOPL (Set Option Lower) service to set the option lower bit.

The base mode equivalent service is M_ROPL.

Entry Conditions

Calling Sequence:

M.ROPL

(or)

SVC 2,X'78' (or) M.CALL H.TSM,14

Exit Conditions

Return Sequence:

M.RTRN

(or)

M.RTRN (CC1 set)

Registers:

CC1 set Call caused the option lower bit to be reset

Abort cases:

None

Output Message:

None

6.2.98 M.RRES - Release Channel Reservation

If the specified channel has not been reserved by this task, the M.RRES service ignores the request to release the channel and returns to the task. If the channel has been reserved by this task, the channel reserve indication is removed from the CDT entry.

After releasing the reserved channel, if any requests had been queued while the channel was reserved, IOCS resumes I/O to the associated device.

This service is not applicable for extended I/O channels.

The base mode equivalent service is M_RRES.

Entry Conditions

Calling Sequence:

```
M.RRES      channel

(or)

LW          R1,channel
SVC         1,X'3B' (or) M.CALL  H.IOCS,13
```

channel specifies the channel number (hexadecimal). If LW, load bits 24 to 31 of register 1.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.99 M.RSML - Resourcemark Lock

The M.RSML service is called to lock the specified resourcemark. It is used in conjunction with the unlock resourcemark service (M.RSMU) by tasks to synchronize access to a common resource. For further description, see Chapter 2.

The base mode equivalent service is M_RSML.

Entry Conditions

Calling Sequence:

```
M.RSML    lockid , [timev][,P]

(or)

LI        R4,timev
ZR        R5
[SBR      R5,0]
LI        R6,lockid
SVC       1,X'19' (or) M.CALL  H.REXS,62
```

lockid — is a numeric resourcemark index value, 33 to 64 inclusive

timev — is a numeric value which specifies action to be taken if the lock is already set and is owned by another task:

Value	Description
+1	Immediate denial return
0	Wait until this task is the lock owner (default)
-n	Wait until this task is the lock owner, or until n timer units have expired, whichever occurs first

P — indicates that while this task is waiting to become lock owner, the swapping mode is to be set to swap this task only if a higher priority task is requesting memory space. Otherwise, the task is a swap candidate if any task is requesting memory.

Exit Conditions

Return Sequence: M.RTRN R7

Registers:

R7 — Zero if the request was accepted, otherwise contains a request denial code:

Value	Description
1	Lock index exceeds maximum range
2	Lock index is less than minimum range
3	Lock is owned by another task (and timev=+1)
4	Lock is owned by another task, timev=-n and n timer units have elapsed

Abort Cases: None

Output Messages: None

6.2.100 M.RSMU - Resourcemark Unlock

The M.RSMU service unlocks a resourcemark which has previously been locked by a call to the M.RSML service. If any other tasks are waiting to lock the specified resourcemark, the highest priority waiting task becomes the new lock owner.

The base mode equivalent service is M_RSMU.

Entry Conditions

Calling Sequence:

```
M.RSMU    lockid

(or)

LI        R6,lockid
SVC       1,X'1A' (or) M.CALL  H.REXS,63
```

lockid is a numeric resourcemark index value, 33 to 64 inclusive

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 Zero if the request was accepted, otherwise contains a request denial code:

Value	Description
1	Lock index exceeds maximum range
2	Lock index is less than minimum range
3	Lock is not owned by this task

Abort Cases:

None

Output Messages:

None

6.2.101 M.RSRV - Reserve Channel

M.RSRV is a privileged service. If the task is unprivileged or the channel has been reserved previously by another task, this service makes a denial return. If the channel has not previously been reserved, the task number is stored in the CDT or UDT entry to mark the reservation. If any requests are currently queued for this channel, suspend is invoked until completion of any I/O currently in progress is complete. The standard handler is then disconnected from the Service Interrupt (SI) level. After reserving a channel, the task must connect its own handler to the SI dedicated location.

This service is not applicable for extended I/O channels.

The base mode equivalent service is M_RSRV.

Entry Conditions

Calling Sequence:

```
M.RSRV      channel,denial

(or)

LW          R1,channel
LA          R7,denial
SVC         1,X'3A' (or) M.CALL  H.IOCS,12
```

channel — specifies the channel number (hexadecimal) in bits 24 to 32. If using LW, load channel number in register 1.

denial — is the user's denial return address

Exit Conditions

Return Sequence:

```
M.RTRN          Normal return

(or)

M.RTNA      7   Denial return
```

Registers:

None

Abort Cases:

```
IO14        Unprivileged user attempting to reserve channel
```

Output Messages:

None

6.2.102 M.RWND - Rewind File

The M.RWND service performs the following functions:

- issues an end-of-file and purge if the file is a system or blocked file which is output active.
- for system and blocked files, initializes blocking buffer control cells for subsequent access.
- rewinds a file or device.

The base mode equivalent service is M_REWIND.

Entry Conditions

Calling Sequence:

```
M.RWND    fcb

(or)

LA        R1,fcb
SVC       1,X'37' (or) M.CALL  H.IOCS,2
```

fcb is the FCB address

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

IO09	Rewind attempted on SYC file

6.2.103 M.SETS - Set User Status Word

The M.SETS service allows the calling task to modify any task's user status word. Along with the Test User Status Word service, this is one of the means provided by MPX-32 for task-to-task communication. The user status word resides in the CPU Dispatch Queue (DQE.USW) and has a value of zero until modified by this service. The user status word is removed from the queue, modified as specified, and replaced in the queue.

The base mode equivalent service is M_SETS.

Entry Conditions

Calling Sequence:

```
M.SETS    function,statusw [,task]
(or)
LD        R4,task
LI        R6,function
LW        R7,statusw
SVC       1,X'48' (or) M.CALL   H.REXS,7
```

function — STF (1), RSF (2), STC (3), or INC (4) specify the type of modification to be performed and are set flag, reset flag, set counter, and increment counter respectively. If using the macro call, specify the alphabetic code. If loading registers, specify the corresponding numeric.

statusw — contains a function parameter specific to function codes as follows:

Value	Description
1	Bit position in the status word to be set (1 to 31)
2	Bit position in the status word to be reset (1 to 31)
3	Value to set the status word
4	Value to increment the status word

task — is the address of a doubleword containing the name of the task, or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero or omission of the argument specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN    5
```

Registers:

R5 — Bit 0 set if the specified task was not found in the dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by M.KEY file; otherwise, register five is zero.

Abort Cases:

RX05 — Invalid function code specification

Output Messages:

None

6.2.104 M.SETSYNC - Set Synchronous Resource Lock

The M.SETSYNC service is used in conjunction with the Release Synchronous Lock service for resource gating of explicitly shared resources where there is no automatic synchronization performed by the system. The mechanism allows a task to obtain synchronized access to a resource that has been concurrently allocated to multiple tasks. A synchronization lock can be obtained for any resource, provided it has been previously allocated (included for memory partitions) by the calling task. Unlike an exclusive lock, the synchronous lock does not prevent other tasks from allocating the resource in explicit shared mode. It is the sharing tasks' responsibility to synchronize access by cooperative use of the synchronous lock services. The resource is identified by either a logical file code, defined when the resource was assigned, or an allocation index, obtained when the resource was assigned or by a resource inquiry. If the synchronization lock is not available, the calling task can obtain an immediate denial return, or wait for an indefinite or specified period of time.

The base mode equivalent service is M_SETSYNC.

Entry Conditions

Calling Sequence:

```
M.SETSYNC  arga  [,cnpaddr]

(or)

LW         R5,arga
LA         R7,cnpaddr  (or)  ZR   R7
SVC        2,X'46'  (or)  M.CALL   H.REMM,25
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
29	Specified LFC was not assigned by this task
30	Invalid allocation index
38	Time out occurred while waiting to become lock owner
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task

Wait Conditions

The task is placed in a wait state, as appropriate, if specified by the CNP.

6.2.105 M.SETT - Create Timer Entry

The M.SETT service builds an entry in the timer table so that the requested function is performed upon time-out. Timer entries can be created to activate a program, resume a program, set a bit in memory, reset a bit in memory, or request an interrupt. Any task may create a timer to activate or resume a program. Timer entries to set or reset bits can be created by any task, provided the bit is within a static memory partition. Only privileged tasks can set bits in the operating system and request an interrupt.

The base mode equivalent service is M_SETT.

Entry Conditions

Calling Sequence:

```
M.SETT      timer,t1,t2,function,arg4,arg5

(or)

LB          R3,function
SLL         R3,24
ORMW        R3,timer
LW          R4,t1
LW          R5,t2
LW (or LD)  R6,arg4
(LW         R7,arg5)
SVC         1,X'45' (or) M.CALL H.REXS,4
```

timer — is a word containing zeros in bytes 0 and 1, and a two-character timer identification in bytes 2 and 3

t1 — contains the current value the timer will be set to in negative time units

t2 — contains the value the timer will be reset to upon each time out in negative time units. If the reset value is zero, the function is performed upon time out and the timer entry is deleted. This case is called a one-shot timer entry.

function — ACP (1), RSP or RST (2), STB (3), RSB (4), and RQI (5) specify the function to be timed and represent activate program, resume program, set bit, reset bit, and request interrupt, respectively. If using the macro call, specify the alphabetic code. If loading registers, specify the corresponding numeric.

Function, Code and arg4 and arg5 contain values specific to the function being timed as follows:

Function	Code	arg4 and arg5
ACP	1	arg4 is a doubleword containing the one- to eight-character name of the program to be activated (system file), or pathname vector or RID vector in the first half of the doubleword and zero in the second half of the doubleword. If the task named is not currently in execution, it is preactivated to connect the interrupt to the task. This connection remains in effect until the task aborts or the timer is deleted. On normal exit, the timer table is updated to point to the next generation. arg5 is null.
RSP	2	arg4 is a doubleword containing the one- to eight-character name of the task to be resumed or zero in register 6 and the task number in register 7. arg5 is null.
(or)		
RST	2	arg4 is the task number entered into register 7 and register 6 is zeroed. arg5 is null.
STB	3	arg4 contains the address of the word in which the bits are to be set. The address must be within a static memory partition or the operating system. arg5 contains the bit configuration of the mask word to be ORed.
RSB	4	arg4 contains the address of the word in which the bit is to be reset. The address must be within a static memory partition or the operating system. arg5 contains the bit configuration of the mask word to be ANDed.
RQI	5	arg4 contains the priority level of the interrupt to be requested. arg5 is null.

Exit Conditions

Return Sequence:

Normal Return:

```
M.RTRN    R3
```

R3 is nonzero and condition codes are not set

Error Condition:

```
M.RTRN    R3
```

If there are no timer entries available, register three is zero and condition codes are not set.

If there are timer entries available, register three is zero and one of the following condition codes are set:

- CC1 set if requested load module does not exist or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by the M.KEY file.
- CC2 set if requested task is not active
- CC3 set if attempting to create a duplicate timer ID

Abort Cases:

RX02	Invalid function code specified for request to create a timer entry. Valid codes are ACP(1), RSP or RST(2), STB(3), RSB(4) and RQI(5).
RX03	An unprivileged task has attempted to set or reset a bit at an address outside a static memory partition, or privileged task has attempted to set/reset a bit at an address outside a static memory partition or the operating system.
RX04	The requesting task is unprivileged or has attempted to create a timer entry to request an interrrupt with a priority level outside the range of X'12' to X'7F', inclusive.

6.2.106 M.SMSGR - Send Message to Specified Task

The M.SMSGR service allows a task to send up to 768 bytes to the specified destination task. Up to 768 bytes can be accepted as return parameters. For further description, see Chapter 2.

The base mode equivalent service is M_SMSGR.

Entry Conditions

Calling Sequence:

```
M.SMSGR    psbaddr

(or)

LA         R2,psbaddr
SVC        1,X'6C'  (or)  M.CALL  H.REXS,44
```

psbaddr is the logical address of the Parameter Send Block (PSB). See Chapter 2 for PSB description.

Exit Conditions

Return Sequence:

```
M.RTRN     6
```

Registers:

R6 Contains the processing start (initial) error status if any:

Value	Description
0	Normal initial status
1	Task not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by the M.KEY file
2-9	Reserved
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address
14	Memory pool unavailable
15	Destination task queue depth exceeded
16	Invalid PSB address

Abort Cases:

None

Output Messages:

None

6.2.107 M.SOPL - Set Option Lower

The M.SOPL service allows the calling task to set the option lower bit. Use the M.ROPL (Reset Option Lower) service to reset the option lower bit.

The base mode equivalent service is M_SOPL.

Entry Conditions

Calling Sequence:

M.SOPL

(or)

SVC 2,X'77' (or) M.CALL H.TSM,13

Exit Conditions

Return Sequence:

M.RTRN

(or)

M.RTRN (CC1 set)

Registers:

CC1 set Call caused the option lower bit to be set

Abort cases:

None

Output Message:

None

6.2.108 M.SRUNR - Send Run Request to Specified Task

The M.SRUNR service allows a task to activate or reexecute the specified destination task with a parameter pass of up to 768 bytes. Up to 768 bytes can be accepted as return parameters. For further description, see Chapter 2.

When a task name is supplied in words 0 and 1 of the Parameter Send Block (PSB), the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied.

The base mode equivalent service is M_SRUNR.

Entry Conditions

Calling Sequence:

```
M.SRUNR   psbaddr

(or)

LA        R2,psbaddr
SVC       1,X'6D'  (or)  M.CALL  H.REXS,45
```

psbaddr is the logical address of the Parameter Send Block (PSB). See Chapter 2.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6 Contains the processing start (initial) error status if any:

Value	Description
0	Normal initial status
1	Reserved
2	File specified in the PSB not found in directory
3	Reserved
4	File specified in the PSB is not a load module or executable image
5	Dispatch Queue Entry (DQE) unavailable
6	I/O error on directory read
7	I/O error on load module read
8	Memory unavailable
9	Invalid task number for run request to multicopied load module in RUNW state
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address

Value	Description
14	Memory pool unavailable
15	Destination task queue depth exceeded
16	Invalid PSB address
17	Reserved

R7 Contains the task number (taskno) of the destination task, or zero if the request was not processed

Abort Cases:

None

Output Messages:

None

6.2.109 M.SUAR - Set User Abort Receiver Address

The M.SUAR service sets up an address to return control to if an abort condition occurs during task execution.

All files remain open prior to transferring to the user specified address. See Task Termination Sequencing in Chapter 2.

The base mode equivalent service is M_SUAR.

Entry Conditions

Calling Sequence:

```
M.SUAR    address

(or)

LA        R7,address
SVC       1,X'60' (or) M.CALL  H.REXS,26
```

address is the logical address where control is transferred when a task terminates

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 — Bit 0 is zero if the request is honored, or one if the request is denied because the specified address is outside the user's allocated area

Bits 1-31 are unchanged

Abort Cases:

RX89 — An unprivileged task has attempted to re-establish an abort receiver (other than M.IOEX)

Output Messages:

None

6.2.110 M.SUME - Resume Task Execution

The M.SUME service resumes a task that has been suspended. A request to resume a task which is not suspended is ignored.

The base mode equivalent service is M_SUME.

Entry Conditions

Calling Sequence:

```
M.SUME    task

(or)

ZR        R6        } (or) LD R6,taskname
LW        R7,taskno }
SVC       1,X'53' (or) M.CALL  H.REXS,16
```

task — the address of a doubleword containing the name of a task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared.

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, unchanged

Abort Cases:

None

Output Messages:

None

6.2.111 M.SUSP - Suspend Task Execution

The M.SUSP service results in the suspension of the caller or any other specified task 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. Suspension of a task can also be ended upon receipt of a message interrupt. A message sent to a task that is synchronized (M.SYNCH) and suspended is not received, but the task is resumed.

The base mode equivalent service is M_SUSP.

Entry Conditions

Calling Sequence:

```
M.SUSP    task,time1

(or)

LW        R5,time1
LI        R6,0       } (or) LD R6,taskname
LW        R7,taskno  }
SVC       1,X'54' (or) M.CALL  H.REXS,17
```

task — the address of a doubleword containing the name of a task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

time1 — contains zero, if suspension for an indefinite time interval is requested, else the negative number of time units to elapse before the caller is resumed.

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, contains the task number

(or)

Zero and CC1 is set if the specified task name is multicopied

Abort Cases:

None

Output Messages:

None

6.2.112 M.SYNCH - Set Synchronous Task Interrupt

The M.SYNCH service causes message and task interrupts to be deferred until the user makes a call to M.ANYW, M.EAWAIT, M.WAIT, or M.ASYNCH. When this service is used, message interrupts is not interrupted by end-action interrupts. All task interrupt levels cannot be interrupted, except by break, until they voluntarily relinquish control.

If a synchronized task is suspended then a message is sent to the task, the message receiver is not entered and the task resumes.

This service can be executed by the IPU.

The base mode equivalent service is M_SYNCH.

Entry Conditions

Calling Sequence:

```
M.SYNCH

(or)

SVC     1,X'1B' (or)    M.CALL    H.REXS,67
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1 set	Synchronous task interrupt was already set

Abort Cases:

None

Output Messages:

None

6.2.113 M.TDAY - Time-of-Day Inquiry

The M.TDAY service obtains the time-of-day as computed from the real-time clock interrupt counter. The counter is initialized with a SYSGEN parameter.

This service can be executed by the IPU.

The base mode equivalent service is M_TDAY.

Entry Conditions

Calling Sequence:

```
M.TDAY

(or)

SVC     1,X'4E' (or) M.CALL   H.REXS,11
```

Exit Conditions

Return Sequence:

```
M.IPURTN  7
```

Registers:

R7	Byte	Contents
	0	Hours (0 to 23)
	1	Minutes (0 to 59)
	2	Seconds (0 to 59)
	3	Interrupts (less than one second)

Abort Cases:

None

Output Messages:

None

6.2.114 M.TEMP - Create Temporary File

The M.TEMP service creates a temporary file. Temporary files are not given names in directories and remain known to the operating system only for as long as the task that created them is in execution. Typically, when the task that created a temporary file terminates execution (normally or abnormally), associated temporary files are automatically deleted by the operating system.

Temporary files can remain defined to the operating system after the task that created them terminates execution when the temporary file has been made permanent or the temporary file is allocated (assigned) to another task when the creator terminates execution.

This service allocates a resource descriptor for the file and acquires the initial space requirements for the file. As a final step, the attributes of the file are recorded in the resource descriptor.

When a temporary file is created, the typical file attributes are:

- the file's resource identifier (RID)
- the file's protection attributes
- the file's management attributes
- the file's initial space requirements

The file's RID is returned only if a RCB address is specified and an ID location address for the file is also specified within the RCB.

The base mode equivalent service is M_CREATET.

Entry Conditions

Calling Sequence:

```
M.TEMP    [cnpaddr],[arga][,rcbaddr]
(or)
LW        R1,arga (or) ZR  R1
LA        R2,rcbaddr (or) ZR  R2
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'21' (or) M.CALL  H.VOMM,2
```

cnpaddr — is a CNP address or blank if CNP not supplied

arga — contains a PN (volume name only) vector or blank if file is to be created on the current working volume

rcbaddr — is a RCB address or blank if default attributes are desired

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

6.2.115 M.TEMPER - Change Temporary File to Permanent File

The M.TEMPER service makes a temporary file permanent. The temporary file is given a name in the specified directory and the file's resource type is changed from temporary to permanent. The file is made permanent with the attributes that were defined when it was created and with any new attributes that may have been acquired while the file's data was being established, such as additional extensions, end-of-file position, or explicit resource descriptor modifications incurred prior to invocation of this service. The temporary file can only be made permanent on the volume where the temporary file resides, i.e., cross volume definitions are not allowed.

The most common use of this service is to ensure exclusive use of a file while its initial data is being established. This method of operation is desirable because it allows the user to guarantee the integrity of the file before it is defined in a directory where others can conveniently gain access to the file.

When the directory entry is established, it is linked to the resource descriptor of the file. This link makes the relationship of the name of the file to the other attributes of the file. These attributes are the same as the attributes for a permanent file.

The base mode equivalent service is M_TEMPFILETOPERM.

Entry Conditions

Calling Sequence:

```
M.TEMPER        [arga],[argb][,cnpaddr]

(or)

LW              R1,arga
LW              R2,argb
LA              R7,cnpaddr (or) ZR  R7
SVC             2,X'28' (or) M.CALL  H.VOMM,9
```

arga contains an LFC or an FCB address

argb is a PN vector or PNB vector

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.116 M.TRNC - Truncate File

The M.TRNC service allows the caller to truncate the unused space of a file. This service is the complement of the extend service (M.EXTD). A file that has not been manually extended should never have to be truncated.

This service only truncates temporary or permanent files. Directories and memory partitions cannot be truncated. The caller must have write, update or append access in order to truncate the file. A file cannot be truncated to less than the minimum space requirement of the file as defined when the file was created.

The caller can truncate a file regardless of whether the file is currently allocated. Additionally, the caller can supply any allowable resource specification; for example, pathname (PN), pathname block (PNB), resource ID (RID), logical file code (LFC) or address of a File Control Block (FCB).

The base mode equivalent service is M_TRUNCATE.

Entry Conditions

Calling Sequence:

```
M.TRNC    [arga][,cnpaddr]

(or)

LW        R1,arga
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'26' (or) M.CALL  H.VOMM,7
```

arga contains a PN vector, PNB vector, RID vector, LFC, or FCB address

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

6.2.117 M.TSTE - Arithmetic Exception Inquiry

The M.TSTE service resets the arithmetic exception status bit in the user's TSA and returns CC1 set or reset according to the status value. The status bit is set whenever the user is in execution and an arithmetic exception trap occurs. The bit remains set until this service is requested, or the task terminates.

This service can be executed by the IPU.

The base mode equivalent service is M_TSTE.

Entry Conditions

Calling Sequence:

```
M.TSTE

(or)

SVC     1,X'4D' (or) M.CALL  H.REXS,23
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

PSD CC1 contains the value of the arithmetic exception status bit

Abort Cases:

None

Output Messages:

None

6.2.118 M.TSTS - Test User Status Word

The M.TSTS service returns the 32-bit user status word of any specified task in execution. The user status word resides in the CPU Dispatch Queue (DQE.USW) and is modified by the Set User Status Word system service. These two services treat the user status word as either a set of 32 flags or as a 32-bit counter. Bit 0 is used as a status flag.

The base mode equivalent service is M_TSTS.

Entry Conditions

Calling Sequence:

```
M.TSTS      task

(or)

ZR          R6          (or)  LD R6,taskname
LW          R7,taskno
SVC         1,X'49'     (or)  M.CALL   H.REXS,8
```

task — the address of a doubleword containing the name of a task or zero in word 1 and the task number in word 2. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 — Bit 0 is set if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, register seven returns the user-status word

Abort Cases:

None

Output Messages:

None

6.2.119 M.TSTT - Test Timer Entry

The M.TSTT service returns to the caller the negative number of time units remaining until the specified timer entry time out. If the timer has expired, the result returned is zero.

This service can be executed by the IPU.

The base mode equivalent service is M_TSTT.

Entry Conditions

Calling Sequence:

```
M.TSTT      timer

(or)

LW          R6,timer
SVC         1,X'46' (or) M.CALL  H.REXS,5
```

timer two-character ASCII name of a timer, right-justified

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 Negative number of time units remaining until time out or zero if the timer has expired or does not exist

Abort Cases:

None

Output Messages:

None

6.2.120 M.TURNON - Activate Program at Given Time-of-Day

The M.TURNON service activates or resumes a specified task at a specified time and reactivates (resumes) it at specified intervals by creating a timer table entry using a specified timer ID. When a load module or executable image name is supplied as input, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied as input.

The base mode equivalent service is M_TURNON.

Entry Conditions

Calling Sequence:

```
M.TURNON filename,time,[reset],timerid

(or)

LD      R6,filename
LW      R4,time
LW      R5,reset
LW      R3,timerid
SVC     1,X'1E'   (or)   M.CALL   H.REXS,66
```

filename — is either a left-justified blank-filled doubleword containing the one- to eight-character ASCII name of the load module or executable image file (must be a system file), or register six contains the pathname vector or RID vector which points to the task to be activated and register seven is zero.

time — is the time-of-day on the 24-hour clock when the task is activated. It is a word value with the following format:

Byte	Contents in Binary
0	Hours
1	Minutes
2	Seconds
3	Zero

reset — is the time interval on the 24-hour clock to elapse before resetting the clock upon each time out. It has the same format as the time argument above. The task is reactivated at each time out. If a reset value is not specified, the comma denoting the field must still be specified and the task is activated only once.

timerid — is a word variable containing the right-justified, zero-filled, two-character ASCII name of the timer that will be created

Exit Conditions

Return Sequence:

M.RTRN	R3	Nonzero

Error Condition:

M.RTRN	R3	Zero if there are no timer entries available, the requested load module or executable image does not exist, attempting to create a duplicate timer ID, or invalid timer ID

Abort Cases:

None

Output Messages:

None

6.2.121 M.TYPE - System Console Type

The M.TYPE service types a user specified message and performs an optional read on the system console. Input message address validation is performed for the unprivileged task. Operation is wait I/O.

The maximum input or output is 80 characters. If no characters are specified, the maximum is used.

M.TYPE builds a Type Control Parameter Block (TCPB) which defines input and output buffer addresses for console messages and reads as described in Section 5.10.

The base mode equivalent service is M_TYPE.

Entry Conditions

Calling Sequence:

```
M.TYPE    outmess,outcount [,inmess,incount]
(or)
LA        R1,tcpb
SVC       1,X'3F'
```

outmess — specifies address of output message buffer

outcount — specifies transfer count for output; number of bytes, 80 maximum

inmess — specifies address of input message buffer. If not specified, TCPB word 2 is zeroed. The first byte of this field contains the actual input quantity.

incount — transfer count for input; number of bytes, 80 maximum

tcpb — specifies address of Type Control Parameter Block (TCPB)

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03	Nonprivileged user attempting transfer to a logical address outside legal boundaries
IO15	Type request while operation in progress

Output Messages:

None

6.2.122 M.UNLOCK - Release Exclusive Resource Lock

The M.UNLOCK service is used with the Set Exclusive Resource Lock service. When called, the exclusive lock is released if the task has the resource allocated in a shareable mode; otherwise, the lock cannot be released until deallocation of the resource. Once the lock is released, other tasks can allocate the resource in a compatible access mode for the particular shared usage. However, another task is not able to exclusively lock the resource until this task, and all other sharing tasks, deallocate the resource.

The base mode equivalent service is M_UNLOCK.

Entry Conditions

Calling Sequence:

```
M.UNLOCK  arga [,cnpaddr]

(or)

LW        R5,arga
LA        R7,cnpaddr  (or)  ZR   R7
SVC       2,X'45'  (or)  M.CALL   H.REMM,24
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Exit Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in register seven or the status field of the CNP:

Value	Description
29	Specified LFC not assigned by this task
30	Invalid allocation index
32	An exclusive resource lock was not owned by this task
33	Resource is not allocated in a shareable mode by this task
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

Notes:

1. An exclusive resource lock can not be released by a task other than the owning task.

2. Any outstanding exclusive resource locks are released on task termination or on resource deallocation.

6.2.123 M.UNSYNC - Release Synchronous Resource Lock

The M.UNSYNC service is used with the Set Synchronous Resource Lock service to perform gating on resources that have been allocated for explicit shared usage. When called, the synchronization lock is released, and all tasks waiting to own the lock is polled.

The base mode equivalent service is M_UNSYNC.

Entry Conditions

Calling Sequence:

```
M.UNSYNC  arga [,cnpaddr]

(or)

LW        R5,arga
LA        R7,cnpaddr  (or)  ZR   R7
SVC       2,X'47'  (or)  M.CALL   H.REMM,26
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in register seven or the status field of the CNP

Value	Description
29	Specified LFC was not assigned by this task
30	Invalid allocation index
32	Synchronization lock was not set
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

Notes:

1. A synchronization lock may not be cleared by any task other than the one that set the lock.

2. A synchronization lock is automatically released when a task terminates or deallocates the resource.

6.2.124 M.UPRIV - Change Task to Unprivileged Mode

The M.UPRIV service allows a task that was cataloged as privileged to operate in an unprivileged state. This causes the calling task's protection image to be loaded at every context switch. See M.PRIV service to reinstate privilege status.

The base mode equivalent service is M_UNPRIVMODE.

Entry Conditions

Calling Sequence:

```
M.UPRIV

 (or)

SVC   2,X'58' (or)  M.CALL   H.REXS,79
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

6.2.125 M.UPSP - Upspace

The M.UPSP service is not applicable to blocked or SYC files. If BOT is present on multivolume magnetic tape, volume record (header) is written. If EOT is present on multivolume magnetic tape an erase and write EOF is performed.

The base mode equivalent service is M_UPSP.

Entry Conditions

Calling Sequence:

```
M.UPSP      fcb

(or)

LA          R1, Fcb
SVC         1,X'10' (or) M.CALL  H.IOCS,20
```

fcb is the FCB address

Registers:

R1 FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cell for a system or blocked file.
IO09	Illegal operation on the SYC file

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

6.2.126 M.VADDR - Validate Address Range

The M.VADDR service verifies a logical address provided by the user.

The base mode equivalent service is M_VADDR.

Entry Conditions

Calling Sequence:

```
M.VADDR         addr,bytes
(or)
LW              R6,addr
LI              R7,bytes
SVC             2,X'59'   (or)   M.CALL  H.REXS,33
```

addr is the logical starting address

bytes is the number of bytes to validate

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

CC2 set	Address range crosses map block boundary
CC3 set	Locations specified are protected
CC4 set	Invalid address (not in caller's address space)
R0-R7	Unchanged

6.2.127 M.WAIT - Wait I/O

The M.WAIT service provides return to the user when the I/O request associated with the specified FCB is complete. The task is suspended until I/O completes.

The base mode equivalent service is M_WAIT.

Entry Conditions

Calling Sequence:

```
M.WAIT          fcb

(or)

LA              R1,fcb
SVC             1,X'3C' (or) M.CALL  H.IOCS,25
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

MS31	User attempted to go to an any wait state from an end action routine.

Output Messages:

None

6.2.128 M.WEOF - Write EOF

The M.WEOF service performs the following functions:

- prevents a write to a read-only file.
- issues an end-of-file and purge if the file is a system or blocked file which is output active. If EOF is requested for an unblocked disc file, the handler writes an EOF immediately following the last record. The EOF record contains X'0FE0FE0F' in the first word.
- writes volume record if BOT on multivolume magnetic tape.
- performs an erase and write EOF if EOT on multivolume magnetic tape.
- writes one EOF.

The base mode equivalent service is M_WRITEEOF.

Entry Conditions

Calling sequence:

```
M.WEOF          fcb

(or)

LA              R1,fcb
SVC             1,X'38' (or) M.CALL  H.IOCS,5
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO10	FAT entry marked read-only (unless SGO).
IO11	File is SYC.
IO30	Illegal volume record. Either volume number or reel ID from volume record do not match FAT information.

Output Messages:

Dismount/mount messages if EOT on multivolume magnetic tape

6.2.129 M.WRIT - Write Record

The M.WRIT service performs the following functions:

- prevents a write to a read-only file.
- provides special random access handling for disc files.
- blocks records for system and blocked files.
- writes volume record if BOT on multivolume magnetic tape.
- performs an erase and write EOF if EOT on multivolume magnetic tape.
- writes one record from the buffer pointed to by the TCW in the FCB.

The base mode equivalent service is M_WRITE.

Entry Conditions

Calling Sequence:

```
M.WRIT          fcb

(or)

LA              R1,fcb
SVC             1,X'32' (or) M.CALL  H.IOCS,4
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cell for a system or a blocked file
IO09	Illegal operation on the SYC file
IO27	Write attempted while reading from a system or a blocked file
IO38	Write attempted on a file opened in read only mode
RM02	Attempt to write SLO/SBO in update mode

Output Messages:

Dismount/mount messages if EOT on multivolume magnetic tape

6.2.130 M.XBRKR - Exit from Task Interrupt Level

The M.XBRKR service must be called at the conclusion of executing a task interrupt routine. It transfers control back to the point of interruption and resets the interrupt to the level established before the break or M.INT.

The base mode equivalent service is M_XBRKR.

Entry Conditions

Calling Sequence:

```
M.XBRKR

(or)

SVC     1,X'70'  (or)  M.CALL  H.REXS,48
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

RX34	Task has made a break receiver exit call while no break is active

Output Messages:

None

6.2.131 M.XIEA - No-Wait I/O End-action Return

The M.XIEA service is required for exiting any no-wait I/O end-action routine. Both normal and error end-action routines use this exit.

The base mode equivalent service is M_XIEA.

Entry Conditions

Calling Sequence:

```
M.XIEA

(or)

SVC     1,X'2C'  or  M.CALL  H.IOCS,34
```

Exit Conditions

Return Sequence:

```
BL      S.EXEC6  no-wait I/O postprocessing complete
```

Registers:

None

Abort Cases:

None

Output Messages:

None

6.2.132 M.XMEA - Exit from Message End-action Routine

The M.XMEA service is called to exit the end-action routine associated with a no-wait message send request. For further description, see Chapter 2.

The base mode equivalent service is M_XMEA.

Entry Conditions

Calling Sequence:

```
M.XMEA

(or)

SVC     1,X'7E'  (or)  M.CALL  H.REXS,50
```

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at message interrupt or task base level

Registers:

None

Abort Cases:

RX99	End-action interrupt was inactive when end-action exit issued

Output Messages:

None

6.2.133 M.XMSGR - Exit from Message Receiver

The M.XMSGR service must be called to exit the message receiver code of the calling task after the task has received a message from another task. For further description, see Chapter 2.

The base mode equivalent service is M_XMSGR.

Entry Conditions

Calling Sequence:

```
M.XMSGR         [rxbaddr]

(or)

LA              R2,rxbaddr
SVC             1,X'5E' (or) M.CALL  H.REXS,39
```

rxbaddr is the logical address of the Receiver Exit Block (RXB)

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at task base level

Registers:

None

Abort Cases:

RX93	Invalid Receiver Exit Block (RXB) address was encountered during message exit
RX94	Invalid Receiver Exit Block (RXB) return buffer address or size was encountered during message exit
RX95	Task has made a message exit while the message interrupt was not active

Output Messages:

None

6.2.134 M.XREA - Exit from Run Request End-action Routine

The M.XREA service is called to exit the end-action routine associated with having sent a no-wait run request.

The base mode equivalent service is M_XREA.

Entry Conditions

Calling Sequence:

```
M.XREA

(or)

SVC     1,X'7F'  (or)  M.CALL  H.REXS,51
```

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at message interrupt or task base level

Registers:

None

Abort Cases:

RX90	End-action interrupt was inactive when end-action exit issued

Output Messages:

None

6.2.135 M.XRUNR - Exit Run Receiver

The M.XRUNR service is called to exit a task which was executing for a run request issued from another task.

The base mode equivalent service is M_XRUNR.

Entry Conditions

Calling Sequence:

```
M.XRUNR         rxbaddr

(or)

LA              R2,rxbaddr
SVC             1,X'7D'  (or)  M.CALL  H.REXS,49
```

rxbaddr is the logical address of the Receiver Exit Block (RXB). For further description, see Chapter 2.

Exit Conditions

Return Sequence:

The run-receiver queue is examined and if not empty, the task is executed again on behalf of the next request. If the queue is empty, the exit options in the RXB are examined. If option byte is zero, the task is placed in a wait state, waiting for the next run request to be received. If option byte is nonzero, the task exits the system.

Note: If the task is re-executed, control is transferred to the instruction following the M.XRUNR call.

Abort Cases:

RX96	Invalid Receiver Exit Block (RXB) address
RX97	Invalid return parameter buffer address or size
RX98	Run receiver mode not active when run receiver exit issued

Output Messages:

None

6.2.136 M.XTIME - Task CPU Execution Time

The M.XTIME service returns to the caller the total elapsed CPU execution time, in microseconds since the initiation of the task. The time returned does not include any IPU execution time whether or not IPU accounting is enabled.

The base mode equivalent service is M_XTIME.

Entry Conditions

Calling Sequence:

```
M.XTIME

(or)

SVC     1,X'2D'  (or)  M.CALL H.REXS,65
```

Exit Conditions

Return Sequence:

```
M.RTRN  6,7
```

Registers:

R6,R7	CPU execution time in microseconds

Abort Cases:

None

Output Messages:

None

6.3 Nonmacro-Callable System Services

6.3.1 Allocate File Space

The Allocate File Space service is available to privileged users for examining the space allocation map (SMAP) on the specified volume in order to locate a specified number of available contiguous allocation units. When a sufficiently long string of zero bits is located in the SMAP, the bit string is set to all ones and the SMAP rewritten to the volume. The segment definition of the corresponding file space is returned to the caller.

Clean-up Mode (used for volume mounting):

This is indicated by bit one of register one being set. In this mode, I/O is not performed because the SMAP is assumed in memory. The FCB should have valid values for FCB.ERWA, FCB.RECL, FCB.EQTY and FCB.ERAA. FCB.ERAA should contain the EOM block number for SMAP, for example, FCB.RECL/192W.

Normal Mode:

Bit zero of the flags should be reset, the other flag bits are then ignored. Input FCB is also ignored. If a start block is specified, it is honored if possible; otherwise, an error is produced.

Entry Conditions

Calling Sequence:

```
M.CALL          H.VOMM,19
```

Registers:

R1	FCB address and flags
R4	Starting block address or zero if anywhere
R5	Requested space size in blocks
R6	MVTE address

Exit Conditions

Return Sequence:

```
M.RTRN          R4,R5

(or)

M.RTRN          R7 (CC1 set)
```

Registers:

R4,R5	File segment definition
R7	Return status

6.3.2 Allocate Resource Descriptor

The Allocate Resource Descriptor service is available to privileged users for examining the resource descriptor allocation map (DMAP) on the specified volume in order to locate an available resource descriptor (RD). When a zero bit is located in the DMAP, the bit is set to 1 and the DMAP rewritten to the volume. The disc address of the corresponding RD is returned to the caller.

If a RD is not available, the DMAP is automatically extended by allocation of available space on the volume. The DMAP is then reexamined as described above.

Entry Conditions

Calling Sequence:

M.CALL	H.VOMM,17

Registers:

R1	FCB address with valid FCB.ERWA and FCB.EQTY
R6	MVTE address

Exit Conditions

Return Sequence:

M.RTRN	R4
(or)	
M.RTNA	R7 (CC1 set)

Registers:

CC1 reset	Successful operation
R4	RD disc address
(or)	
CC1 set	Error condition
R7	Status

6.3.3 Create Temporary File

The Create Temporary File service is available to privileged users for creating a temporary file. The MVTE and starting block can be specified. This service functions as follows:

VOMM Internal Call

The file space is obtained and the RD is built in the system buffer. System FCB is held open, assigned to the RD, but the RD is not written to disc.

External Call

The file space is obtained and the RD is built in the system buffer. The RD is written to disc, and the system FCB is deassigned.

Default File Attributes

The temporary file is created with owner SYSTEM and no project group. All access is allowed, EOF management is inhibited, and the file is nonsegmented and nonextendible.

Volume Selection

If MVTE is not specified, a mounted volume is selected as follows:

Value	Description
a	The current working volume is tried first
b	The system volume is tried next
c	Public mounted volumes are tried next in MVT order

Entry Conditions

Calling Sequence:

```
M.CALL    H.VOMM,24
```

Registers:

R4	Starting block address or zero if anywhere
R5	Number of blocks required
R6	MVTE or zero if anywhere

Exit Conditions

Registers:

CC1 set	Error
R4	Starting block
R5	Size in blocks
R6	MVTE

The RD for the created file is in the system buffer

R7 Contains the error code as follows:

Value	Description
09	Resource descriptor not available
11	File space not available
12	Unrecoverable I/O error reading DMAP
13	Unrecoverable I/O error writing DMAP
15	Unrecoverable I/O error writing RD
16	Unrecoverable I/O error reading SMAP
17	Unrecoverable I/O error writing SMAP

6.3.4 Deallocate File Space

The Deallocate File Space service is available to privileged users for updating the space allocation map (SMAP) on the specified volume in order to mark the specified file space as available.

Entry Conditions

Calling Sequence:

```
M.CALL      H.VOMM,20
```

Registers:

R4,R5	File segment definition
R6	MVTE address

Exit Conditions

Return Sequence:

```
M.RTRN

(or)

M.RTRN      R7 (CC1 set)
```

Registers:

R7	Return status

6.3.5 Deallocate Resource Descriptor

The Deallocate Resource Descriptor service is available to privileged users for updating the resource descriptor allocation map (DMAP) on the specified volume in order to mark the specified resource descriptor (RD) as available.

Entry Conditions

Calling Sequence:

```
M.CALL      H.VOMM,18
```

Registers:

R1	FCB address with valid FCB.ERWA and FCB.EQTY
R4	RD disc address
R6	MVTE address

Exit Conditions

Return Sequence:

```
M.RTRN

(or)

M.RTRN      R7 (CC1 set)
```

Registers:

R7	Return status

6.3.6 Debug Link Service

The Debug Link service is intended for use only by the interactive debugger (MPXDB, SYMDB, or DEBUGX32) for the purpose of transferring control to the debugger. The debugger plants this SVC trap in the user's task at the desired location.

Entry Conditions

Calling Sequence:

```
SVC     1,X'66' (or) M.CALL  H.REXS,42
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.3.7 Eject/Purge Routine

The Eject/Purge Routine service performs the following functions:

- If a file is blocked and output active, a purge is issued. Return is made to the user.
- Writes volume record if BOT on multivolume magnetic tape.
- Performs an erase and write EOF if EOT on multivolume magnetic tape.
- Eject is not applicable to SYC files.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'0D' (or) M.CALL  H.IOCS,22
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

6.3.8 Erase or Punch Trailer

The Erase or Punch Trailer service writes the volume record if BOT on multivolume magnetic tape or performs an erase and write EOF if EOT on multivolume magnetic tape.

Erase or punch trailer is not applicable to blocked or SYC files.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'3E' (or) M.CALL  H.IOCS,21
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO12	File not opened for write mode

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

6.3.9 Execute Channel Program

The Execute Channel Program service is available to privileged users and allows command and data chaining to General Purpose Multiplexor Controller (GPMC) and extended I/O devices only. Logical execute channel is available to both privileged and nonprivileged users. Physical execute channel is available only to privileged users.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'25'  or  M.CALL  H.IOCS,10
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03	Nonprivileged user attempting transfer to a logical address outside legal boundaries
IO43	IOCL list or data address not in contiguous E-memory, GPMC devices only
IO50	An unprivileged user attempted to execute a physical channel program
IO51	A TESTSTAR command was used in a logical channel program
IO52	A logical channel was too large to be moved to memory pool
IO53	A TIC command follows a TIC command in a logical channel program
IO54	A TIC command attempted to transfer to an address which is not word bounded
IO55	Illegal address in logical IOCL. Address is not in user's logical address space.
IO56	A read-backward command was used in a logical channel program
IO57	Illegal IOCL address. IOCL must be located in the first 128K words of memory.

6.3.10 Get Extended Memory Array

The Get Extended Memory Array service requests an array of extended memory. If the request cannot be met, then all free memory, except 1/8 of the amount of physical memory, is allocated to the task and a count of maps allocated is returned. This service is intended for use by tasks that require the largest possible buffers without being placed on the MRQ for an extended period.

Entry Conditions

Calling Sequence:

```
LW      R1,maps
SVC     2,X'7F'   (or)   M.CALL H.MEMM,14
```

maps is the number of map blocks required

Exit Conditions

Return Sequence:

M.RTRN

Registers:

R2 Number of map blocks allocated

R3 Starting logical address of memory allocated or zero if an error occurred

R4 Ending logical address of memory allocated or error code as follows:

Value	Description
1	CSECT overrun
2	Request for more memory than physically exists
3	M.MEMB service in use
4	Unable to allocate logically contiguous memory

R5 Number of map blocks, all classes, that are now free

6.3.11 Read/Write Authorization File

The Read/Write Authorization File service is available to privileged users for performing the following functions:

- Validating owner name and key and returning default information, even if key is invalid. Callable as M.CALL H.VOMM,25 only.
- Validating project name and key. SVC callable by M.DEFT (H.VOMM,8).

Entry Conditions

Calling Sequence:

```
M.CALL      H.VOMM,25
```

Registers:

R2	Zero validates owner name; one validates project name and key
R4,R5	Contain the left-justified, blank-filled key, or register 4 is zero and register 5 contains the compressed key
R6,R7	Contain the left-justified, blank-filled name

Exit Conditions

Return Sequence:

Registers:

R2	Contains the address of an area in T.BBUFA which contains the authorization entry, or zero if M.KEY file does not exist. The entry is two words for project name and one word for compressed key.
R6,R7	Unchanged if the name is valid. Both register 6 and register 7 are zero if the name contains invalid characters, is not in the M.KEY file, or an incorrect key is supplied.
CC1	Equals one if there is an unrecoverable I/O error

6.3.12 Release FHD Port

The Release FHD Port service is available only to privileged users and is currently only supported by the four megabyte fixed head disc.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'27'   (or)   M.CALL   H.IOCS,27
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.3.13 Reserve FHD Port

The Reserve FHD Port service is available only to privileged tasks and is currently only supported by the four megabyte fixed head disc.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'26'   (or)   M.CALL   H.IOCS,24
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.4 Compatible System Services

This section contains H.ALOC, H.MONS, H.IOCS and H.FISE services that are included for compatibility purposes only, and using them is not recommended. Using compatible services with noncompatible features, e.g., Caller Notification Packet (CNP), is not allowed.

The MPX-32 code for the support of the H.ALOC, H.MONS, and H.FISE services can be optionally removed from the MPX-32 image by the SYSGEN NOCMS directive. Attempted execution of an H.ALOC, H.MONS, or H.FISE service with the support removed causes the calling task to abort with an SV02 abort code.

6.4.1 M.ALOC - Allocate File or Peripheral Device

The M.ALOC service 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 service may also be used to equate a new logical file code with an existing logical file code.

Entry Conditions

Calling Sequence:

```
M.ALOC    retad,lfc,function,arga,argb, [MOUNT] , [UNBLOCKED] [,WAIT]
(or)
LA        R1,retad
LI        R5,function
SLL       R5,24
ORMW      R5,lfc
SBR       R5,0 for MOUNT option inhibited
SBR       R5,1 for UNBLOCKED option
SBR       R5,2 for WAIT for resource requested
LA        R6,arga
LA        R7,argb
SVC       1,X'40' (or) M.CALL  H.MONS,21
```

retad is the denial return logical address

lfc contains the one to three ASCII character logical file code to be assigned. The first byte contains zero to accommodate the function code. The LFC is then left-justified and blank-filled in the three remaining bytes.

function is the function code as follows:

Value	Description
1	Assign logical file code to a user or system permanent file
2	Assign logical file code to a system file code (must be a blocked file)
3	Assign logical file code to a peripheral device (If the device is a disc drive, a formatted volume must already be mounted on it)
4	Assign logical file code to a defined logical file code (must be a blocked file)
5	Assign logical file code to a system permanent file only

arga and argb are addresses of memory locations with contents unique to each function as follows:

1	arga	contains the one to eight ASCII character permanent file name. If a user name is not associated with the calling task, the system file of the name specified is allocated. If a user name is associated with the calling task, an attempt is made to allocate a user file of the name specified. If unsuccessful, a system file is allocated.
	argb	is ignored if specified
2	arga	contains the character string SLO or SBO in bytes 1, 2, and 3
	argb	contains the number of 192-word blocks required for allocation to the file
3	arga	contains the device type code (see Appendix A, Table A-1) in byte 0 and optionally the channel number in byte 2 and the device subaddress in byte 3. If the device subaddress is present, the most significant bit of byte 2 must be set. If the channel number is present, bit 0 of byte 0 must be set. For magnetic tape devices, byte 1 equals the volume number or zero if single volume.
	argb	if arga defines a disc file, this equals the size of file (number of 192-word blocks required). If arga defines a magnetic tape, it equals a four-character reel identifier. For all other devices it equals zero.
4	arga	contains the previously defined logical file code
	argb	equals zero
5	arga	contains the one- to eight-character permanent file name of a system file
	argb	is ignored if specified
MOUNT		is a character string indicating the mount message should not be sent
UNBLOCKED		is a character string indicating the file being allocated is to be unblocked. If not specified, the file is blocked automatically.
WAIT		is a character string indicating the caller wishes to be queued for the resource and relinquishes the CPU until the resource becomes available

Exit Conditions

Return Sequence:

M.RTRN — CC1 is set in the program status doubleword if the calling task has read but has not written access rights to the specified permanent file

Registers: None

(or)

Return Sequence:

M.RTNA 1,6 — Denial returns if the requested file or device cannot be allocated

Registers:

R6 — Equals zero if file or device busy. Condition codes indicate why:

Code	Meaning if Set
CC1	Permanent file is exclusively locked
CC2	File Lock Table (FLT) is full
CC3	Nonshared device is busy (already allocated)
CC4	Disc space is not available

Equals n if an error condition exists as described next. When register six equals n (CC1 through CC4 are not applicable):

Value	Description
1	Permanent file nonexistent
2	Reserved
3	No FAT/FPT space available
4	No blocking buffer space available
5	Shared memory table entry not found
6	Reserved
7	Dynamic common specified in ASSIGN 1
8	Unrecoverable I/O error to directory
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
18	Call was incompatible

Abort Case:

MS16 — Task has requested dynamic allocation with an invalid function code.

Output Messages: See mount message description in Section 5.5.

6.4.2 M.CDJS - Submit Job from Disc File

The M.CDJS service submits a job contained on a blocked permanent or temporary disc file. Prior to calling this service, the specified file should be rewound to purge the contents of the blocking buffer if it has been dynamically built.

Entry Conditions

Calling Sequence:

```
M.CDJS      filename [ ,password]

(or)

LD          R2, password
LD          R6,filename
SVC         1,X'61' (or) M.CALL   H.MONS,27
```

filename contains the one- to eight-character name of the blocked permanent disc file which contains the job. If a user name is associated with the calling task, an attempt is made to allocate a user file of the name specified. If unsuccessful, a system file is allocated.

(or)

contains zero in the first word and the second word contains the address of a file control block which is associated with a blocked temporary file in the calling task

Once submitted, the logical file code associated with the permanent or temporary file is deallocated and may be reassigned by the user task.

password is ignored if specified

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 Zero if successful

(or)

R7 Bit 0 is set if specified file does not exist, invalid password is specified, or FCB is not associated with a temporary file. Bits 1-31 are zero.

(or)

R7 Bit 1 is one if unable to activate system input task. Bits 0, 2-31 are zero.

Abort Cases:

None

Output Messages:

None

6.4.3 M.CREATE - Create Permanent File

The M.CREATE service allocates disc space for the specified permanent file and writes a corresponding entry into the specified directory. The allocated space can be zeroed.

Entry Conditions

Calling Sequence:

```
M.CREATE filename,blocks,,,,

[{R}], password , [S] , [N] , [F] , [type] [,Z]
 [{P}]

(or)

 LD      R6,filename
 LW      R2,blocks
[SBR     R2,2                 if N  -  not SAVE DEVICE file]
[SBR     R2,3                 if F  -  FAST file           ]
 ZR      R3
 ZR      R4 (or) LD  R4,password
 ZR      R1
[LI      R1,X'type' if type present                        ]
[SBR     R1,0                 if S  -  system file         ]
[SBR     R1,1                 if Z  -  prezero             ]
 SVC     1,X'75' (or) M.CALL  H.FISE,12
```

filename — is a doubleword containing the one to eight ASCII character, left-justified, blank-filled name of the file. The operating system automatically encloses the file name in single quotes; therefore, the single quote character is the only character which cannot be used in a file name.

blocks — a variable word containing the size of the file specified as a multiple of 192-word blocks

R,P — is ignored if specified

password — is ignored if specified

S — is a character indicating the file is to be a system file. If not specified, the file is created as a user file in the current user directory.

N — is a character indicating the file is not to be saved in response to the SAVE DEVICE File Manager directive

F — is a character indicating the file is a fast file. If not entered, the file is created as a slow file.

type is a one- or two-digit hexadecimal value specified by the user to identify the origin of the file as follows:

00-9F are available for customer usage. A0 - FF are reserved for Gould CSD development's usage. Known file type codes are:

42	Toolkit blocked mode
43	Toolkit card mode
46	Toolkit formatted mode
50	Toolkit packed mode
55	Toolkit unblocked mode
B0	Base mode object file
BA	Base mode shared image (or BASIC Save file)
BB	Base mode object library file
BC	Base mode macro library file
BE	Base mode load module file
C0	Spooled output file
CA	Cataloged load module
D0	Memory disc save task (J.MDSAVE) file
DB	Symbolic debugger command file
ED	Saved text editor file
EE	Stored text editor file
FE	Text editor work file
FF	SYSGEN generated file

Z indicates the space allocated to the file is to be zeroed

Exit Conditions

Return Sequence:

M.RTRN 6,7

Registers:

R7 Zero if the file was not created. R6 contains the reason.

R6 Contains the reason the file was not created as follows:

Value	Description
1	File of the name specified already exists
2	Fast file was specified and collision mapping occurred with an existing directory entry
3	Reserved
4	Disc space is unavailable
5	Specified device (channel and/or subaddress) is not configured, or no device of the type specified is available
6	Specified device is off-line
7	Directory is full
8	Specified device type (byte 1 of register 3) is not configured
9	File name or password contains invalid characters or imbedded blanks
10	Access mode is invalid

Abort Cases:

FS01	Unrecoverable I/O error to the directory
FS02	Unrecoverable I/O error to file space allocation map

Output Messages:

None

6.4.4 M.DALC - Deallocate File or Peripheral Device

The M.DALC service deallocates a peripheral device or disc file to which the specified logical file code is assigned. Dynamic deallocation of a peripheral or permanent disc file makes that resource available to other tasks. Deallocation of SLO and SBO files result in their definitions being passed to System Output for output to their terminal devices. If the specified logical file code has been equated to other logical file codes in the system, this service deallocates only the specified code.

Entry Conditions

Calling Sequence:

```
M.DALC    lfc

(or)

LW        R5,lfc
SVC       1,X'41' (or) M.CALL  H.MONS,22
```

lfc contains the one to three ASCII character, left-justified, blank-filled logical file code for which deallocation is to be performed (bytes 1 to 3). Byte 0 is zero unless the device is magnetic tape. If tape, bit 0 of byte 0 equals one indicating that the dismount message is to be output, but that the device is not deallocated. If bit 0 of byte 0 equals zero, output dismount message and deallocate.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

*task DISMOUNT reel FROM UNIT xx
*task DISMOUNT reel, VOL volno FROM UNIT xx

task — is the load module name of the task requesting that the magnetic tape unit be deallocated

reel — is the reel identifier of the magnetic tape to be dismounted

volno — is the volume number of the magnetic tape to be dismounted - if single volume equals blank

xx — is the device number of the nonshared magnetic tape unit from which the tape is to be dismounted

6.4.5 M.DELETE - Delete Permanent File or Non-SYSGEN Memory Partition

The M.DELETE service deletes a specified permanent file or non-SYSGEN created memory partition.

Entry Conditions

Calling Sequence:

```
M.DELETE  filename, [S] [,password]

(or)

LD        R6,filename
ZR        R3 (or) LI  R3,G'S'
ZR        R4 (or) LD  R4,password
SVC       1,X'77' (or) M.CALL  H.FISE,14
```

filename is a doubleword containing the one- to eight-character left justified, blank filled name of an existing file to be deleted.

S indicates a system file is to be deleted. If not specified, the file is assumed to be a user file whose user name is that which is associated with the calling task.

password is ignored if specified

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6,R7 Unchanged if the deletion was successful

(or)

R6 Contains the reason that the specified file was not deleted:

Value	Description
1	File of the name specified does not exist or is a SYSGEN created memory partition
2	Reserved
3	User does not have delete access rights to the file

R7 Zero if the specified file was not deleted

Abort Cases:

FS01	Unrecoverable I/O error to the directory
FS02	Unrecoverable I/O error to file space allocation map

Output Messages:

None

6.4.6 M.EXCL - Free Shared Memory

The M.EXCL service allows a task to dynamically deallocate any common areas it has previously shared (M.SHARE) or included (M.INCL).

Entry Conditions

Calling Sequence:

```
M.EXCL partition    {,ownername    }
                    {,tasknumber,TNUM}
```

(or)

```
LD      R6,partition
LD      R2,ownername  }     (or)  LW      R2,tasknumber
                      }           ZR      R3
SVC     1,X'79'             (or)  M.CALL  H.ALOC,14
```

partition — contains a doubleword bounded, left-justified blank-filled memory partition name, such as GLOBAL01

ownername — contains the owner name of the original owner of the partition

tasknumber — contains the left-justified task number of the original owner of the partition

TNUM — indicates a task number is being used instead of an owner name

Exit Conditions

Return Sequence:

M.RTRN (or) abort user with AL39

Registers:

None

Abort Cases:

AL39	Shared memory table entry not found

6.4.7 M.FADD - Permanent File Address Inquiry

The M.FADD service issues I/O directly. It provides the word address of the beginning of a memory partition or the track, head, and sector address of the beginning of a disc file. The device address for disc files is also included in the result. Access restrictions are returned for disc files.

Entry Conditions

Calling Sequence:

```
M.FADD name

(or)

LD      R6,name
SVC     1,X'43' (or) M.CALL  H.MONS,2
```

name — is a doubleword containing the one- to eight-character ASCII, left-justified, blank-filled permanent file name or partition name. If a file name is specified, an attempt is made to locate the file in the current working directory associated with the calling task. If the file is not found, an attempt is made to locate the file in the system directory.

Exit Conditions for Case I, Denial Return

Return Sequence:

```
M.RTRN      7
```

Registers:

R7	Bit 0 is set to one indicating that the specified file name cannot be located. Bits 1-31 are zero.

Exit Conditions for Case II, Memory Partition

Return Sequence:

```
M.RTRN      6,7
```

Registers:

R6	Bit 0 is set to one indicating that the specified name is a memory partition. Bits 1-31 are the number of 512-word pages allocated to partition.
R7	Logical address of the first word of the specified partition

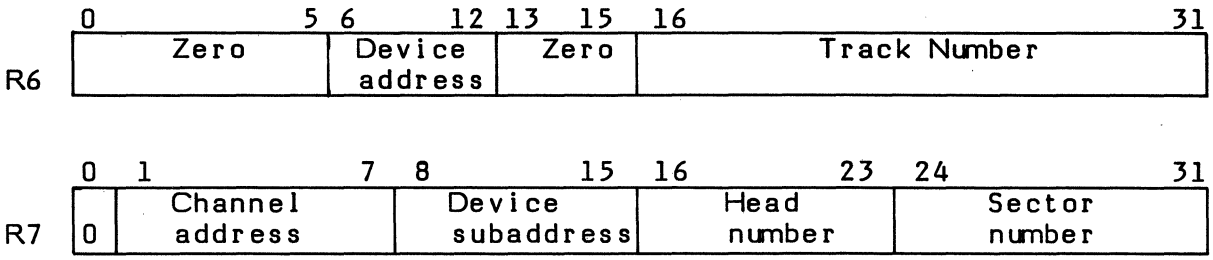
Exit Conditions for Case III, Disc File

Return Sequence:

M.RTRN 6,7

Registers:

R6 and R7 are returned with the address of the beginning of the disc file as follows:



The device address is the sum of the channel address and device subaddress. It is positioned so that it may be ORed into a CD instruction (for nonmultiplexed and nonextended devices).

The following additional parameters are returned:

- CC1 is set in the program status word if a password is required to write the file (read-only file)
- CC2 is set if a password is required to read or write the file (password-only file)
- CC3 is set if the file is a system file or core partition

Abort Cases:

None

Output Messages:

None

6.4.8 M.FILE - Open File

The M.FILE service performs the following functions:

- Establishes appropriate linkages between a user FCB and an assigned file or device.
- Marks a file open for either update access or read-only operations. Increments internal counts of tasks having the file open at this time.
- For SYC or SGO files, completes building the FAT based on job control information.
- For system and blocked files, initializes blocking buffer for subsequent accesss.
- Requests the initial mount message for statically allocated, nonshared magnetic tape devices.

Open requests to a file which is already opened are ignored.

Entry Conditions

Calling Sequence:

```
M.FILE fcb [,RW]

(or)

LA      R1,fcb
[SBR    R1,1 if RW]
SVC     1,X'30' (or) M.CALL H.IOCS,1
```

fcb is the FCB address

RW specifies update access to the file. If RW is not specified or if update access is not compatible with existing access rights to the file, the file is opened read-only.

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

None

Abort Cases:

IO20	An error occurred in the REMM open procedure
RM29	Invalid FCB address or unassigned LFC in FCB

Note: This system service is not excluded by the SYSGEN NOCMS directive.

6.4.9 M.FSLR - Release Synchronization File Lock

The M.FSLR service is used for disc file gating. It is implemented with the set synchronization File Lock service (M.FSLS) to control a synchronization lock indicator. When M.FSLR is called, the synchronization lock is released, and the queue of tasks waiting to own the lock is polled.

Entry Conditions

Calling Sequence:

```
M.FSLR    lfca

(or)

LW        R5,lfca
SVC       1,X'24' (or) M.CALL   H.FISE,25
```

lfca is the address of a word that contains an unused byte in byte 0, and a one- to three-character ASCII, left-justified, blank-filled logical file code in bytes 1, 2, and 3

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 Value	Description
0	Request accepted, synchronization lock released
1	Request denied, synchronization lock was not set
5	Request denied, specified LFC not allocated
6	Request denied, specified LFC is not assigned to a permanent disc file

Notes:

1. A synchronization lock can only be cleared by the task that set the lock.
2. If a task owns a synchronization lock when the task terminates, the lock is automatically released.
3. A synchronization lock is automatically released when the file is deallocated.

6.4.10 M.FSLS - Set Synchronization File Lock

The M.FSLS service is used with the Release Synchronization File Lock service (M.FSLR) for disc file gating. The M.FSLS and M.FSLR services control a synchronization lock indicator which allows synchronized access to a disc file that is concurrently allocated to multiple tasks. To use the M.FSLS service, the file must have been previously allocated to the calling task. The file is identified by logical file code (LFC).

Entry Conditions

Calling Sequence:

```
M.FSLS    lfca [,timev]

(or)

LW        R5,lfca
LI        R4,timev (or) ZR     R4
SVC       1,X'23'  (or) M.CALL H.FISE,24
```

lfca is the address of a word that contains an unused byte in byte 0, and a one- to three-character ASCII, left-justified, blank-filled logical file code in bytes 1, 2, and 3

timev is a numeric value interpreted as follows:

Value	Description
+1	Return immediately with a denial code if the file already has a synchronization lock set
0	Place the requesting task in a wait state until it has become the owner of the synchronization lock
-n	Place the requesting task in a wait state until it owns the synchronization lock, or until the expiration of n timer units, whichever occurs first

Exit Conditions

Return Sequence:

```
M.RTRN R7
```

Registers:

R7

Value	Description
0	Request accepted, synchronization lock set
1	Request denied, synchronization lock is 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
4	Reserved
5	Request denied, LFC not assigned to permanent disc file

6.4.11 M.FXLR - Release Exclusive File Lock

The M.FXLR service is used in conjunction with the set exclusive file lock service (M.FXLS) for disc file gating. When M.FXLR is called, the exclusive lock is released and other tasks can allocate the associated disc file.

Note: Another task is not able to exclusively lock the file until it is deallocated by this task.

Entry Conditions

Calling Sequence:

```
M.FXLR    lfca

(or)

LW        R5,lfca
SVC       1,X'22' (or) M.CALL   H.FISE,23
```

lfca is the address of a word that contains an unused byte in byte 0, and a one- to three-character ASCII, left-justified, blank-filled logical file code in bytes 1, 2, and 3

Exit Conditions

Return Sequence:

```
M.RTRN R7
```

Registers:

R7 Value	Description
0	Request accepted, exclusive file lock released
29	Request denied, specified LFC is not assigned by this task
30	Request denied, invalid allocation index
32	Request denied, an exclusive resource lock was not owned by this task
33	Request denied, resource is not allocated in a shareable mode by this task

Notes:

1. An exclusive file lock can not be released by a task other than the owning task.
2. Any outstanding exclusive file locks are released on task termination or on file deallocation.

6.4.12 M.FXLS - Set Exclusive File Lock

The M.FXLS service is used for disc file gating. It allows the calling task to gain exclusive allocation of a file, as though it were an unshared resource. The file must have been previously allocated, and is identified by the address of logical file code (LFC).

Entry Conditions

Calling Sequence:

```
M.FXLS    lfca [,timev]

(or)

LW        R5,lfca
LI        R4,timev  (or)  ZR     R4
SVC       1,X'21'   (or)  M.CALL  H.FISE,22
```

lfca is the address of a word that contains an unused byte in byte 0, and a one- to three-character ASCII, left-justified, blank-filled logical file code in bytes 1, 2, and 3

timev is a numeric value interpreted as follows:

Value	Description
+1	Return immediately with a denial code if the file is already allocated to another task
0	Place the requesting task in a wait state until the designated file can be exclusively locked
-n	Place the requesting task in a wait state until the designated file can be exclusively locked, or until the expiration of n timer units, whichever occurs first

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 Value	Description
0	Request accepted, file is exclusively locked
1	Request denied, file is allocated to another task, or is already exclusively locked
2	Reserved
3	Reserved
4	Request denied, time out occurred while waiting to become lock owner
6	Request denied, LFC not assigned to permanent file

6.4.13 M.INCL - Get Shared Memory

The M.INCL service allows a task to dynamically include a memory partition into its address space, e.g., GLOBAL01 or DATAPOOL common. The task is suspended until the inclusion is complete. The task performing an M.INCL must know if the shared memory table entry was built with an owner name or task number, and know what they are.

Entry Conditions

Calling Sequence:

```
M.INCL partition  {,ownername}  ,[RW] , [password], [denial][,TNUM]
                  {,tasknumber}

   (or)

LD      R6,partition
LD      R2,ownername  }   (or)   LW      R2,tasknumber
                                 ZR      R3
ZR      R0
ZR      R4                (or)   LD      R4,password
ZR      R5
LA      R0,denial
[SBR    R0,0 if R/W]
SVC     1,X'72'           (or)   M.CALL  H.ALOC,13
```

partition — contains the doubleword-bounded, left-justified, blank-filled memory partition name, such as GLOBAL01

ownername — contains the owner name of the original owner of the partition

tasknumber — contains the left-justified task number of the original owner of the partition

RW — specifies read and write control desired

password — is ignored if specified

denial — specifies a denial return address

TNUM — indicates a task number is being used instead of an owner name

Exit Conditions

Return Sequence:

M.RTRN R3

Registers:

R3 starting address of shared memory partition. This is a 20-bit address.

(or)

M.RTNA R0,R3 for denial returns

R3 Value	Description
1	Entry not found in shared memory table
2	Reserved
3	Memory requirements conflict with task's address space
4	Entry not found in shared memory table after returning from SWGQ state chain

R0 Address to return to within user task body

6.4.14 M.LOG - Permanent File Log

The M.LOG service provides a log of currently existing permanent files.

Entry Conditions

Calling Sequence:

```
M.LOG     type,address [,filename]

(or)

LI        R4,type
LA        R5,address
LD        R6,filename (if TYPE =N or O)
SVC       1,X'73' (or) M.CALL  H.MONS,33
```

type is a byte-scaled value which specifies the type of log to be performed as follows:

Value	Description
=N=0	Specifies a single named file in the current working directory or system directory
=A=1	Specifies all permanent files in the current working directory
=S=2	Specifies all permanent files in the system directory
=U=3	Specifies all permanent files in the current working directory
=O=4	Specifies a single named file in the system directory

If type equals N, an attempt is made to locate the file in the current working directory associated with the calling task. If the file is not found, an attempt is made to locate the file in the system directory.

address is the address of an eight-word area within the calling task where a copy of a SMD entry is to be stored

filename contains a one- to eight-character file name if type equals N or O

Exit Conditions

Return Sequence:

M.RTRN 4,5

An eight-word SMD entry, if any, is stored at the address specified as address. The password field contains zero to indicate the absence of a password.

Registers:

R4 If type equals N or O (register 4 equals zero or four), register 4 is destroyed. If type equals A, S, or U (register 4 equals one, two, or three), this service is called repeatedly to obtain all the pertinent file definitions. The type parameter in register 4 is specified in the first call only. Register 4 is returned containing the address of the next directory entry to be returned. The value returned in register 4 must be unchanged upon the subsequent call to this service.

R5 Contains zero if type equals N or O (register 4 equals zero or four) and the specified file could not be located or type equals A, S, or U (register four equals one, two, or three) and all pertinent files have been logged. Otherwise, register 5 is unchanged.

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.

Output Messages:

None

Note: The M.LOG system service searches the Memory Resident Descriptor Table (MDT) for resource descriptors before it searches the disc-resident resource descriptors. For MDT information, refer to the Rapid File Allocation Utility (J.MDTI) Chapter in Volume II.

6.4.15 M.PDEV - Physical Device Inquiry

The M.PDEV service returns to the caller physical device information describing the unit to which a specified logical file code is assigned.

Entry Conditions

Calling Sequence:

```
M.PDEV   lfc

(or)

LW       R5,lfc
SVC      1,X'42' (or) M.CALL  H.MONS,1
```

lfc — is the address of a word that contains an unused byte in byte 0, and a one- to three-character ASCII, left-justified, blank-filled logical file code in bytes 1, 2, and 3 for which physical device information is requested

Exit Conditions

Return Sequence:

```
M.RTRN   7
```

Registers:

R7 — Zero if the specified logical file code is unassigned

(or)

Return Sequence:

```
M.RTRN   4,5,6,7
```

Registers:

R4 — Bit 0 is one for extended I/O (Class F) device; zero for all other device classes
Bits 1-7 are zero
Byte 1 is zero
Bytes 2 and 3 are device (two ASCII characters, such as MT, DC, etc). See Appendix A.

R5 — Disc equals number of 192-word blocks in file
Magnetic tape equals reel identifier (four ASCII characters)
TSM terminal:

Byte	Contents
2	Number of hexadecimal characters in line
3	Number of hexadecimal lines on screen

All other devices equal 0

R6	Bytes 0 and 1	Maximum number of bytes transferrable to device
	Byte 2	Device channel number
	Byte 3	Device subaddress
R7	Byte 0	Device type code (two hexadecimal digits). See Appendix A.
	Byte 1	
	Bit 8	Zero if file is unblocked, one if file is blocked
	Bits 9-12	Zero
	Bits 13-15	System file code, as follows:

Code	Description
0	Not a system file
1	SYC file
2	SGO file
3	SLO file
4	SBO file

Bytes 2 and 3	Disc equals the number of 192-word sectors per allocation unit Magnetic tape equals the volume number (zero equals single volume) All other devices equal zero

Note: If the specified logical file code is assigned to SYC or SGO, and that file is not open, bits 13 through 15 of register 7 are returned equal to one or two. All other returned parameters are not applicable.

When doing a physical device inquiry while running from the console terminal, register 7 can be returned equal to zero even though the LFC is assigned. When this occurs, the device type code 00 (console terminal) is in register 7.

Abort Cases:

None

Output Messages:

None

6.4.16 M.PERM - Change Temporary File to Permanent

The M.PERM service changes the status of a temporary file allocated to the calling task to permanent.

Entry Conditions

Calling Sequence:

```
M.PERM    filename,lfc [, [ {R} , password ] , [S] , [N] , [F] , [type] [,Z] ]
                         {P}
```

(or)

```
LD        R6,filename
LW        R2,=G'lfc'
ZR        R3
[SBR      R3,6        if R - read only
 SBR      R3,7        if P - password only
 SBR      R3,2        if N - not SAVE DEVICE file
 SBR      R3,3        if F - fast file ]
ZR        R1
[LI       R1,X'type' if type present
 SBR      R1,0        if S - system file
 SBR      R1,1        if Z - pre zero ]
ZR        R4
[LD       R4,password if file is to have a password]
SVC       1,X'76' (or) M.CALL   H.FISE,13
```

filename — is a doubleword containing the one- to eight-character ASCII, left-justified, blank-filled name of the file. The operating system automatically encloses the filename in single quotes; therefore, the single quote character is the only character which cannot be used in a filename.

lfc — is the one- to three-character ASCII, right-justified, zero-filled logical file code assigned to an open, temporary, SLO, or SBO file. The file is marked as permanent in the calling task by this service if successful.

R,P — is ignored if specified

password — is ignored if specified

S — indicates the file is to be a system file. If not specified, the file is created as a user file if a user name is associated with the calling task, or as a system file if no user name is associated with the calling task.

N — is ignored if specified

F — indicates the file is a fast file. If not specified, the file is created as a slow file.

type — is a one- or two-digit hexadecimal value specified by the user to identify the origin of the file as follows:

Value	Description
00-39	Available for customer use
40-5F	Reserved for system use
60-9F	Available for customer use
A0-AF	Reserved for system use
B0	Base mode object file
BA	Base mode shared image (or BASIC file)
BB	Base mode object library file
BC	Base mode macro library file
BE	Base mode load module file
C0	Spooled output file
CA	Cataloged load module
D0	Memory disc save task (J.MDSAVE) file
DB	Symbolic debugger command file
ED	Saved text editor file
EE	Stored text editor file
FE	Text editor work file
FF	SYSGEN generated file

Z — is ignored if specified

Exit Conditions

Return Sequence: M.RTRN 6,7

Registers:

R7 — zero if the file was not created
R6 — Contains the reason that the file was not created:

Value	Description
1	File of the name specified already exists
2	Fast file was specified and collision mapping occurred with an existing directory entry
3	Reserved
4	File associated with the specified logical file code is not a temporary file
5	Directory and temporary file are not on the same volume
7	Directory is full
9	File name contains invalid characters

Abort Cases:

FS01 — Unrecoverable I/O error to the directory
FS02 — Unrecoverable I/O error to file space allocation map

Output Messages:

None

6.4.17 M.SHARE - Share Memory with Another Task

The M.SHARE service dynamically creates a shared memory partition from the partition definition found in the system directory. This definition must have been previously defined by the File Manager utility.

The call results in the creation of a new common area, which are uniquely identified by the owner name or task number of the caller, and by the memory partition name. The memory type is specified by the directory definition. Prezeroing is not performed by this service. The partition is swappable with the task if the use count equals zero. The partition is deallocated when the allocation count equals zero. The task is suspended until the Shared Memory Table entry is built and the memory allocation is complete. The shared partition can be gated from the use of other tasks to allow the initial loading of data. This is called a data lock. Any tasks attempting to include this partition while the lock is set are queued to the SWGQ state (general queue) and remain there until the lock is reset by the M.SMULK service.

Options are:

- Request read and write access, set bit 0 in register 0.
- Request task number instead of owner name, set bit 1 in register 0.
- Request data lock and includers to be enqueued, set bit 2 in register 0.

Entry Conditions

Calling Sequence:

```
M.SHARE partition, [RW], [password] , [TNUM] [,LOCK]
(or)
LD      R6,partition
ZR      R0
SBR     R0,0 if read/write
SBR     R0,1 if task number requested
SBR     R0,2 if data lock requested
LD      R4,password
ZR      R4 if no password
ZR      R5 if no password
SVC     1,X'71'     (or)  M.CALL     H.ALOC,12
```

partition	contains the doubleword-bounded, left-justified memory partition name
RW	is ignored if specified
password	is ignored if specified
TNUM	indicates a task number is being used instead of an owner name
LOCK	specifies a data lock

Exit Conditions

Return Sequence:

M.RTRN R3 (or) abort user with AL40 or AL41

Registers:

R0	Bit 4 is reset if share becomes an include
R3	Starting address of memory partition if share is successful

Abort Cases:

AL40	Partition definition not found on directory
AL41	Directory definition not a dynamic partition

6.4.18 M.SMULK - Unlock and Dequeue Shared Memory

The M.SMULK service allows a user to unlock the data lock associated with a particular shared memory partition. See M.SHARE (ALOC,12) for use of data lock.

Upon executing M.SMULK, the lock on the shared area is reset and all users that are queued to the shared area are relinked from the SWGQ (general queue wait state) to their appropriate run state. At this time they have full access to the shared partition.

Entry Conditions

Calling Sequence:

```
M.SMULK partition, ownername [,TNUM]

(or)

LD      R2,ownername     (or)  { LW    R2,tasknumber
                               { ZR    R3
SVC     1,X'1F'          (or)    M.CALL  H.ALOC,19
```

partition contains a doubleword bounded that is a left-justified and blank-filled memory partition name

ownername contains the owner name of the original owner of the partition

tasknumber contains the left-justified task number of the original owner of the partition

TNUM indicates a task number is being used instead of an owner name

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

6.4.19 M.USER - User Name Specification

The M.USER service associates a user name with the calling task. This service can nullify any user name associated with the calling task. The user name associated with the task is utilized in file create, delete, log, and allocate services called subsequently.

Entry Conditions

Calling Sequence:

```
M.USER    [username] [,key]

(or)

ZR        R6 null username
ZR        R7
SVC       1,X'74' (or) M.CALL  H.MONS,34

(or)

LD        R6,username
LD        R4,key
SVC       1,X'74' (or) M.CALL  H.MONS,34
```

username contains the one- to eight-character user name that is left-justified and blank-filled. Each character must have an ASCII equivalent in the range 01 through 7F.

key is ignored if specified

If both parameters are omitted, user name defaults to system on the current working volume.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6,R7 Zero if the service was not performed because the specified user name contains invalid characters or is not in the user name file or the required key was not furnished; otherwise, unchanged

Abort Cases:

None

Output Messages:

None

CHAPTER 7

BASE MODE SYSTEM SERVICES

7.1 General Description

MPX-32 offers a set of resident base mode system service routines designed to perform frequently required operations with maximum efficiency. Using the Supervisor Call instruction, tasks running in any environment can call these routines.

All system service routines are reentrant. Thus, each service routine is always available to the task which is currently active.

System service routines are provided as standard modular components of the Mapped Programming Executive. The open-ended design of the system, however, gives each user freedom to add whatever service routines are required to tailor MPX-32 to a specific application.

System services enable tasks to:

- Activate, suspend, resume, abort, terminate and hold task execution
- Change a task's priority level
- Create, test, and delete timers
- Interrogate system clocks
- Allocate and deallocate devices and files
- Obtain the characteristics of a device or file
- Communicate with other tasks with messages and status words
- Load and execute overlays
- Obtain information on the memory assigned to a task
- Connect tasks to interrupts
- Interrogate the arithmetic exception and option word status for a task

MPX-32 services are implemented as SVC traps. There are several ways of accessing services:

- By macro calls, with parameter passing as indicated. The expansion code in the system macro library is then accessed automatically during assembly to provide Assembly language setup of appropriate registers and instructions including SVCs, in the user's code.

- By setting up appropriate registers and instructions directly and using appropriate SVCs.

- By following the course above but issuing an M_CALL request to the entry point of the system module that provides the service.

The first two access paths are described for each system service. The third access path is privileged, and is indicated primarily to provide the appropriate system module names and entry point numbers for cross-reference to other documentation when needed.

Special operations performed for a task are:

- Open - If not issued by the task, IOCS opens the file or device for the default access in effect at that time.
- Close - If not issued by the task, the file is closed automatically and a device is deallocated automatically during task termination.

All system services are arranged alphabetically by their macro name. System services that are not macro callable, but yet available to the user, are described in Section 7.3.

Appendix B provides cross-reference charts.

System Services - Syntax Rules and Descriptions

All system services can be called by their macro name, their SVC number, or their module entry point number. It is recommended that whenever possible the macro name be used. When a macro name is used, any optional parameter not specified in the call is handled as follows:

- the appropriate register is assumed to have been previously loaded

  (or)

- the appropriate register will be zeroed

See the Calling Sequence description of each service to determine applicability of missing parameter handling.

Defaults for optional parameters are documented in the description of the individual services.

When a required parameter is not specified or an invalid parameter is specified, an error message is displayed in the listing regardless of the listing controls in effect.

Base mode system services can only be used with the Macro Assembler/X32. When using base mode system services, expanded parameter specification rules apply.

Parameter Specification

Parameters can be specified to a base mode service by using one of two methods: by keyword or by position.

When parameters are specified by keyword, the parameters are not order dependent. Each parameter is denoted by using the parameter keyword followed by an equal sign followed by the value. Multiple parameter groups are separated by a comma. See Example 1.

When parameters are specified by position (i.e., only the value of a parameter is specified), the parameters are order dependent and must be entered in the order shown in the syntax of each service. Multiple parameter groups are separated by a comma. A

comma must also be inserted for an optional parameter not specified if a following optional parameter is specified in the syntax. See Example 2.

Keyword and positional parameters can be mixed within a syntax statement. However, it is not recommended since the position is not advanced when keyword parameters are processed by the assembler.

Example 1 - Specifying Parameters by Keyword

To create a permanent file using the M_CREATEP service, any one of the following keyword syntax variations are valid:

```
M_CREATEP PNADDR=addr1,RCBADDR=addr2,CNPADDR=addr3

M_CREATEP PNADDR=addr1,RCBADDR=addr2

M_CREATEP PNADDR=addr1,CNPADDR=addr3

M_CREATEP RCBADDR=addr2,CNPADDR=addr3,PNADDR=addr1
```

Example 2 - Specifying Parameters by Position

To create a permanent file using the M_CREATEP service, any one of the following positional syntax variations are valid:

```
M_CREATEP addr1,addr2,addr3

M_CREATEP addr1,addr2

M_CREATEP addr1,,addr3

M_CREATEP addr1
```

Parameter Value Specification

Parameter values can be specified in one of six forms: by using a register name, by using a base register name, by using a label which refers to a required data structure, by using a label which refers to a memory location which contains the address of a required data structure, by using a literal, or by using defaults.

When using a register name R0-R7, the value to be used is in the specified register. For example,

```
M_GETMEMBYTES R6

generates:      TRR     R6,R4       !load correct register
                SVC     2,X'4B'     !enter service
```

When using a base register name B0-B7, the value to be used is in the specified base register. Caution should be exercised when using base registers B0-B3 because the base mode operating system uses them for call and return instructions. For example,

```
M_GETMEMBYTES B6

generates:      TBRR    B6,R4           !load correct register
                SVC     2,X'4B'         !enter service
```

When using a label which refers to a memory location, the associated register is automatically selected. For example,

```
M_OPENR FCBOUT

generates:      LW      R1,#A'FCBOUT'   !get FCB address
                SVC     2,X'42'         !enter service
```

When using a label which refers to an absolute memory location and an explicitly defined base register, the sum of the address of the label (relative to the start of the program segment) and the content of the register equal to the destination address. For example,

```
M_OPENR @FCBLOC(B3)

generates:      LW      R1,FCBLOC(B3)   !get FCB address
                SVC     2,X'42'         !enter service
```

When using a literal, the literal should be preceded by a # symbol. For example,

```
M_GETMEMBYTES #8192

generates:      LW      R4,#8192        !get byte count
                SVC     2,X'4B'         !enter service
```

When using defaults, the value passed to the service is the default value defined for the missing parameter(s). For example,

```
M_AWAITACTION

generates:      LW      R6,?0           !ZERO is a location containing zero
                SVC     1,X'1D'         !enter service
```

7.1.1 IPU Executable Base Mode System Services

Once a task has gained entry into the IPU, there is a limited set of system services that the IPU can execute. These are memory reference only system services, since the IPU can not execute any I/O instructions. The following base mode system services are executable in the IPU:

SVC	Description
M_ADRS	Memory Address Inquiry
M_BBTIM	Acquire Current Date/Time in Byte Binary Format
M_BTIM	Acquire Current Date/Time in Binary Format
M_CONABB	Convert ASCII Date/Time to Byte Binary Format
M_CONADB	Convert ASCII Decimal to Binary
M_CONAHB	Convert ASCII Hexadecimal to Binary
M_CONASB	Convert ASCII Date/Time to Standard Binary
M_CONBAD	Convert Binary to ASCII Decimal
M_CONBAF	Convert Binary Date/Time to ASCII Format

SVC	Description
M_CONBAH	Convert Binary to ASCII Hexadecimal
M_CONBBA	Convert Byte Binary Date/Time to ASCII
M_CONBBY	Convert Binary Date/Time to Byte Binary
M_CONBYB	Convert Byte Binary Date/Time to Binary
M_CTIM	Convert System Date/Time Format
M_CMD	Get Command Line
M_CONVERTTIME	Convert Time
M_DATE	Date and Time Inquiry
M_DEVID	Get Device Mnemonic or Type
M_DSMI	Disable Message Task Interrupt
M_DSUB	Disable User Break Interrupt
M_ENUB	Enable User Break Interrupt
M_ENVRMT	Get Task Environment
M_GETTIME	Get Current Date and Time
M_GTIM	Acquire System Date and Time in any Format
M_OPTIONWORD	Task Option Word Inquiry
M_QATIM	Acquire Current Date/Time in ASCII Format
M_SYNCH	Set Synchronous Task Interrupts
M_TDAY	Time-of-Day Inquiry
M_TSTE	Arithmetic Exception Inquiry
M_TSTT	Test Timer Entry

7.2 Macro-Callable System Services

All base mode system services are described in detail in the pages which follow, arranged alphabetically by their macro name. System services which are supported for base mode tasks are prefaced by the characters 'M_'.

7.2.1 M_ACTV - Activate Task

The M_ACTV service activates a task. The task assumes the owner name of the caller. When a load module is supplied as input, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied as input.

The nonbase mode equivalent service is M.ACTV.

Entry Conditions

Calling Sequence:

```
M_ACTV    [LOADMOD=] loadmod
(or)
LD        R6,loadmod
SVC       1,X'52'  (or)  M_CALL   H.REXS,15
```

loadmod is either a left-justified blank-filled doubleword containing the one- to eight-ASCII character name of the load module for which an activation request is queued (must be a system file), or R6 is zero and R7 contains the pathname vector or RID vector which points to the load module to be activated.

Exit Conditions

Return Sequence:

M.RTRN 6,7

Registers:

R6	Equals zero if the service could be performed
R7	Contains the task number of task activated by this service

(or)

R6	Equals one if invalid attempt to multicopy a unique load module
R7	Task number of existing task with same name

(or)

	Value	Description
R6	2	If load module file not in directory
	3	Unable to allocate load module
	4	If file is not a valid load module
	5	If DQE is not available
	6	If read error on resource descriptor
	7	If read error on load module
	8	Insufficient logical/physical address space for task activation

Abort Cases:

None

Output Messages:

None

7.2.2 M_ADRS - Memory Address Inquiry

The M_ADRS service provides the beginning and ending logical addresses of the memory allocated to a task. The beginning address is the location into which the first word was loaded and is a word address. The ending address is also a word address and defines the last word allocated to the task.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.ADRS.

Entry Conditions

Calling Sequence:

M_ADRS

(or)

SVC 1,X'44' (or) M_CALL H.REXS,3

Exit Conditions

Return Sequence:

M.IPURTN 6,7

Registers:

R6	Logical word address of the first location of the task's DSECT. This address is always on a page boundary.
R7	Logical word address of the last location available for loading or expansion of the task's DSECT. This address is always on a map block boundary minus one word.

Abort Cases:

None

Output Messages:

None

7.2.3 M_ADVANCE - Advance Record or File

The M_ADVANCE service is not applicable for SYC files or unblocked disc files. This service performs the following functions for advancing records:

- Verifies volume record if BOT on multivolume magnetic tape.
- Advances specified number of records.

M_ADVANCE performs the following functions for advance files:

- If the file is blocked, logical records are advanced until an end-of-file is found. The read/write control word will point to the first record after the end-of-file.
- Verifies volume record if BOT on multivolume magnetic tape.
- Advances specified number of files.

The nonbase mode equivalent service is M.FWRD.

Entry Conditions

Calling Sequence:

M_ADVANCE [FCBADDR=] addr , [[MODE=] {R / F}], [NUMBER=] num

(or)

```
SVC     1,X'33'  (or)  M_CALL  H.IOCS,7
SVC     1,X'34'  (or)  M_CALL  H.IOCS,8
BIB     R4, $-W
```

addr is the FCB address

R	advance by record (SVC 1,X'33')
F	advance by file (SVC 1,X'34'); default
num	address of word containing the number of records or files to be advanced, or a value of one, if not specified
$-W	branches back to SVC the number of times specified by num

Registers:

R1	Contains addr
R4	Contains num

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cells for blocked file
IO07	A task has attempted to perform an operation which is not valid for the device to which the user's file is assigned
IO09	Illegal operation on the SYC file
IO29	Advance file issued for a blocked file while writing
IO30	Illegal or unexpected volume number or reel ID encountered on magnetic tape

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

7.2.4 M_ANYWAIT - Wait for Any No-Wait Operation Complete, Message Interrupt, or Break Interrupt

The M_ANYWAIT service places the currently executing task in a state waiting for the completion of any no-wait request, for the receipt of a message, or for a break interrupt. The task is removed from the associated ready-to-run list, and placed in the any-wait list. A return is made to the program location following the SVC instruction only when one of the wait conditions has been satisfied or when the optional time-out value has expired.

The nonbase mode equivalent service is M.ANYW.

Entry Conditions

Calling Sequence:

```
M_ANYWAIT [[WAITTIME=] time1]

(or)

LW        R6, time1
SVC       1,X'7C'  (or)  M_CALL  H.REXS,37
```

time1 contains zero if wait for an indefinite period is requested; otherwise, time1 contains the negative number of time units to elapse before the wait is terminated.

Registers:

R6 Contains time1; otherwise, zero

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

MS31 User attempted to go to an any-wait state from an end-action routine

Output Messages:

None

7.2.5 M_ASSIGN - Assign and Allocate Resource

The M_ASSIGN service associates a resource with a Logical File Code (LFC) used by a process and to allocate the resource. This function creates a FAT/FPT pair within the user's TSA and associates an Allocated Resource Table (ART) entry for system administration and control of the resource while allocated. When implicit sharing is indicated by the absence of a specified usage mode, the appropriate linkage is established to coordinate concurrent access. The option is provided to allocate and open a resource via a single call to this function.

The nonbase mode equivalent service is M.ASSN.

Entry Conditions

Calling Sequence:

```
M_ASSIGN [RRSADDR=] addr1 [ , [CNPADDR=] addr2]

(or)

LA    R1, addr1
LA    R7, addr2     (or)    ZR        R7
SVC   2,X'52'       (or)    M_CALL    H.REXS,21
```

addr1 is the address of an RRS entry (Type 1 - 6).

addr2 is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, options field, status field, and parameter link.

The option field contains an access and usage specification for opening of this resource. This field will be used only if the automatic open flag is set in the option word of the RRS. See M_OPENR service.

If automatic open is indicated in the Resource Requirement Summary, word 5 of the CNP must contain the address of a valid File Control Block (FCB) for this assignment. See M_OPENR service.

Registers:

R1 Contains addr1
R7 Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R5	M.RTRN R5
(or)	(or)
M.RTRN R5 (CC1 set)	M.RTRN R5,R7 (CC1 set)

Registers:

R5 Allocation index; a unique 32-bit integer number associated with the allocated resource. This index may be used to set and release resource locks for exclusive or synchronous access.

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
1	Unable to locate resource (invalid pathname)
2	Specified access mode not allowed
3	FAT/FPT space not available
4	Blocking buffer space not available
7	Static assignment to dynamic common
8	Unrecoverable I/O error to volume
9	Invalid usage specification
11	Invalid RRS entry
12	LFC logically equated to unassigned LFC
13	Assigned device not in system
14	Resource already allocated by requesting task
15	SGO or SYC assignment by real-time task
17	Duplicate LFC assignment attempted
19	Invalid resource ID
20	Specified volume not assigned
22	Resource is marked for deletion
23	Assigned device is marked off-line
24	Segment definition allocation by unprivileged task
25	Random access not allowed for this access mode
27	Resource already opened in a different access mode
28	Invalid access specification at open
29	Specified LFC is not assigned to a resource for this task
38	Time out occurred while waiting for resource to become available
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task
51	Shareable resource is allocated in an incompatible access mode
54	Unable to allocate resource for specified usage
55	Allocated Resource Table (ART) space not available

Note: Status values 25-29 are returned only when auto-open is indicated.

Wait Conditions

When the resource is not available (status values 50-63), the task is placed in a wait state, as appropriate, if specified by a CNP.

7.2.6 M_ASYNCH - Set Asynchronous Task Interrupt

The M_ASYNCH service resets the asynchronous task interrupt mode back to the default environment.

The nonbase mode equivalent service is M.ASYNCH.

Entry Conditions

Calling Sequence:

```
M_ASYNCH

(or)

SVC     1,X'1C'    (or)    M_CALL    H.REXS,68
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

CC1	Asynchronous task interrupt already set

Abort Cases:

None

Output Messages:

None

7.2.7 M_AWAITACTION - End Action Wait

The M_AWAITACTION service waits for the completion of any no-wait request or I/O end action if any are requested. If there are not any outstanding, the service returns immediately to the user. This service is similar to the M_ANYWAIT service.

The nonbase mode equivalent service is M.EAWAIT.

Entry Conditions

Calling Sequence:

```
M_AWAITACTION [WAITTIME=] addr

(or)

LW      R6, addr    (or)    ZR      R6
SVC     1,X'1D'     (or)    M_CALL  H.EXEC,40
```

addr — is the address containing the negative number of time units to elapse before the wait is terminated. If not specified, the task waits for an indefinite period.

Registers:

R6 — Contains addr; otherwise, zero

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

EX03 — User attempted to go to an any-wait state from an end-action routine

Output Messages:

None

7.2.8 M_BACKSPACE - Backspace Record or File

The M_BACKSPACE service is not applicable for SYC files or unblocked files. This service performs the following functions for backspacing records:

- If the file is output active, a purge is issued prior to the backspace function. After the specified number of records are backspaced, control returns to the user.
- Backspaces specified number of records.

M_BACKSPACE performs the following functions for blocked files:

- If the file is output active, an end-of-file and purge are issued prior to performing the backspace file function. Records are backspaced until an end-of-file record is found.
- The specified number of files are backspaced.
- The read/write control word then points to the end-of-file just encountered.

The nonbase mode equivalent service is M.BACK.

Entry Conditions

Calling Sequence:

```
M_BACKSPACE [FCBADDR=] addr ,[[MODE=] {R}, [NUMBER=] num
                               {F}]

(or)
LA      R1, addr
LNW     R4, num

(SVC    1,X'35'  (or)  M_CALL  H.IOCS,9)      (or)
(SVC    1,X'36'  (or)  M_CALL  H.IOCS,19)
BIW     R4, $-W
```

addr — is the FCB address

R — backspaces by record (SVC 1,X'35')

F — backspaces by file (SVC 1,X'36'); default

num — address of word containing four times the number of records or files to backspace, or negative four if not supplied

$-W — branches back to SVC the number of times specified by num

Registers:

R1	Contains addr
R4	Contains num, negated, or -4 if not supplied

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cells for blocked file
IO09	Illegal operation on the SYC file

Output Messages:

None

7.2.9 M_BATCH - Batch Job Entry

The M_BATCH service submits a batch job stream located in a disc file. The disc file is described by the calling parameter in register one. Prior to calling this service, the specified disc file should be rewound to purge the contents of the blocking buffer if it has been dynamically built.

The nonbase mode equivalent service is M.BATCH.

Entry Conditions

Calling Sequence:

```
M_BATCH [ARGADR=] arga [ , [CNP=] cnpaddr]

(or)

LW      R1,arga
LA      R7,cnpaddr (or) ZR R7
SVC     2,X'55' (or) M_CALL H.REXS,27
```

arga is a PN vector, PNB vector, or RID vector for a permanent file; or an LFC or FCB address for a temporary file

cnpaddr is a CNP address or zero if CNP is not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN R7 (CC1 set)

(or)

M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, denial address

Status:

CC1 set

Posted in R7 or in the status field of the CNP

Value	Description
0	Operation successful
1	Pathname invalid
2	Pathname consists of volume only
3	Volume not mounted
4	Directory does not exist

5	Disc file has not been previously opened
6	Unable to activate J.SSIN2, batch job not submitted
7	Resource does not exist
14	Unrecoverable I/O error while reading resource descriptor
18	Unrecoverable I/O error while reading directory

Abort Cases:

None

Output Messages:

None

7.2.10 M_BBTIM - Acquire Current Date/Time in Byte Binary Format

The M_BBTIM service acquires the system date and time in byte binary format. The date and time are returned in a two word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.BBTIM.

Entry Conditions

Calling Sequence:

```
M_BBTIM [TIMBUF=] addr

(or)

LA          R1,addr
ORMW        R1,=X'02000000'
SVC         2,X'50'  (or)  M_CALL   H.REXS,74
```

addr is the address of a two-word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.11 M_BORT - Abort Specified Task, Abort Self, or Abort with Extended Message

M_BORT - Specified Task

This service allows the caller to abort another task. If the named task has been swapped out, it is not aborted until it regains CPU control. See Chapter 2, Table 2-2. If the specified task is not in execution, the request is ignored.

The nonbase mode equivalent service is M.BORT.

Entry Conditions

Calling Sequence:

```
M_BORT    [ABCODE=] abcode, [TASK=] task
(or)

LW        R5,abcode
ZR        R6          } (or) LD R6,taskname
LW        R7,taskno   }
SVC       1,X'56'     (or) M_CALL  H.REXS,19
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared.

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — zero if any of the following conditions exist:

- the specified task was not found
- the specified task name was not single copied
- the owner name of the task requesting the abort is not privileged and is restricted from access to tasks with a different owner name (by the M.KEY file)
- the task is in the process of exiting the system

Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

M_BORT - Self

This service aborts the calling task by issuing an abort message, optionally performing a post-abort dump, and performing the functions common to the normal termination service as described in Chapter 2.

Entry Conditions

```
M_BORT    [ABCODE=] abcode
(or)
LW        R5,abcode
SVC       1,X'57' (or) M_CALL  H.REXS,20
```

abcode contains the four-character ASCII abort code

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

task name task number ABORTED AT: xxxxxxxx-xxxxx mm/dd/yy hh:mm:ss zzzz

task is the one-to eight-character name of the task being aborted

number is the task number of the task being aborted

xxxxxxxx is the location the abort occurred

yyyyy is the beginning of the DSECT

zzzz is the four-character abort code

M_BORT - With Extended Message

A call to this service results in an abort of the specified task.

Entry Conditions

Calling Sequence:

```
M_BORT    [ABCODE=] abcode, [TASK=] task, [EXTCODE=] extcode

(or)

LD        R2,extcode
LW        R5,abcode
LI        R6,0        }          (or) LD R6,taskname
LW        R7,taskno   }
SVC       1,X'62' (or) M_CALL   H.REXS,28
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

extcode — contains the extended abort code message consisting of one to eight ASCII characters, left-justified, and blank-filled

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if any of the following conditions exist:

- the specified task was not found
- the specified task name was not single copied
- the owner name of the task requesting the abort is not privileged and is restricted from access to tasks with a different owner name (by the M.KEY file)
- the task is in the process of exiting the system

Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

7.2.12 M_BRK - Break/Task Interrupt Link/Unlink

The M_BRK service allows the caller to clear the user's break receiver or to establish the address of a routine to be entered whenever another task or the operator activates his task interrupt by an M_INT service.

The nonbase mode equivalent service is M.BRK.

Entry Conditions

Calling Sequence:

```
M_BRK       [BREAKADR=] brkaddr

(or)

LA          R7,brkaddr
SVC         1,X'6E' (or) M_CALL  H.REXS,46
```

brkaddr is the logical word address of the entry point of the task's break/task interrupt routine or zero to clear the break receiver

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.13 M_BRKXIT - Exit from Task Interrupt Level

The M_BRKXIT service must be called at the conclusion of executing a task interrupt routine. It transfers control back to the point of interruption.

The nonbase mode equivalent service is M.BRKXIT.

Entry Conditions

Calling Sequence:

M_BRKXIT

(or)

RETURN

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.14 M_BTIM - Acquire Current Date/Time in Binary Format

The M_BTIM service acquires the system date and time in binary format. The date and time are returned in a two word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.BTIM.

Entry Conditions

Calling Sequence:

```
M_BTIM  [TIMBUF=] addr

(or)

LA          R1,addr
ORMW        R1,=X'01000000'
SVC         2,X'50'  (or)  M_CALL   H.REXS,74
```

addr is the address of a two-word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.15 M_CHANPROGFCB - Execute Channel Program File Control Block

The M_CHANPROGFCB service is used to define a File Control Block for use with an execute channel program request.

Entry Conditions

Calling Sequence:

M_CHANPROGFCB [LABEL=] label , [LFC=] lfc , [CPADDR=] addr1 ,

[[TOUT=]sec], [[PCP=] {Y / N}] , [[NWI=] {Y / N}], [[NST=] {Y / N}],

[[SENSESIZE=] size1] , [[SENSEBUFFER=] addr2] , [[NOWAIT=] addr3] ,

[[NOWAITERROR=] addr4] , [[WAITERROR=] addr5] , [[PPCISIZE=] size2],

[[PPCIADDR=] addr6]

label	ASCII string to use as symbolic label for address of this FCB
lfc	logical file code; word 0, bits 8-31 of the FCB
addr1	logical address of the channel program to be executed
sec	time-out value specified in seconds
PCP	specifies physical channel program
NWI	specifies no-wait I/O request
NST	specifies status checking not requested
size1	size of the user specified sense buffer
addr2	address of the user specified sense buffer
addr3	normal no-wait end-action return address
addr4	no-wait end-action error return address
addr5	wait end-action error return address
size2	size of PPCI status buffer to use
addr6	PPCI end action address

7.2.16 M_CLOSER - Close Resource

The M_CLOSER service is used to terminate operations in the current access mode on a resource. The resource is marked closed in the FPT. The user count in the appropriate Allocated Resource Table (ART) entry is decremented if implicit shared use is in effect. For access modes other than READ, the resource descriptor is updated. When the closing of a file implies a change of use or access mode for that resource, any tasks waiting for access to the resource in a compatible access mode are dequeued. If any logically equivalent resources are open, no further action is taken. For blocked files, any active output blocking buffer is purged. A close request to a resource that is already closed will result in an immediate return with the appropriate status posted.

The nonbase mode equivalent service is M.CLOSER.

Entry Conditions

Calling Sequence:

```
M_CLOSER [FCBADDR=] addr1 [ , [CNPADDR=] addr2]

(or)

LA      R1, addr1
LA      R7, addr2
SVC     2,X'43'  (or)  M_CALL  H.REMM,22
```

addr1 is the address of a File Control Block (FCB)

addr2 is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Registers:

R1 Contains addr1
R7 Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
8	Unrecoverable I/O error to volume
29	Logical file code associated with FCB does not exist
31	Resource was not open
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

7.2.17 M_CLSE - Close File

The M_CLSE service marks a file closed in the File Pointer Table (FPT) and the count of open files (DFT.OPCT) is decremented. If any logically equivalent files (ASSIGN4) are open no further action is taken, for example, if count after decrementing is not equal to zero.

If the file is a system file or blocked file, purges any active output blocking buffer. The file is marked closed (open bit cleared in FAT).

For files assigned to SYC or SGO, the current disc address updates the Job Table for Job Control.

This service issues an EOF prior to purging system files SLO and SBO which were opened for read/write. Also issues an EOF prior to purging for blocked files which are output active.

Close requests to a file that is already closed are ignored.

The nonbase mode equivalent service is M.CLSE.

Entry Conditions

Calling Sequence:

```
M_CLSE    [FCB=] fcb [, [EOFF=] EOF] [ , [REWF=] REW]
(or)
 LA       1,fcb
[SVC      1,X'38'  or  M_CALL  H.IOCS,5]
[SVC      1,X'37'  or  M_CALL  H.IOCS,2]
 SVC      1,X'39'  or  M_CALL  H.IOCS,23
```

fcb	is the FCB address
EOF	writes EOF (SVC 1,X'38'). See M.WEOF description.
REW	rewinds file or device (SVC 1,X'37'). See M.RWND description.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO38	Task attempted to write to a file opened in read-only mode

Output Messages:

None

7.2.18 M_CMD - Get Command Line

The M_CMD service returns the portion of the command line between the program name and the end of the line if the program name is specified on the command line. If data does not exist or the command line has already been issued, a null string is returned.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CMD.

Entry Conditions

Calling Sequence:

```
M_CMD

(or)

SVC     2,X'61'   (or)   M_CALL   H.REXS,88
```

Registers:

None

Exit Conditions

Return Sequence:

```
M.RTRN  R6,R7
```

Registers:

R6	Contains the length of the string in bytes, if found; otherwise, zero
R7	Contains the first byte address of the string, if found; otherwise, zero

Abort Cases:

None

Output Messages:

None

7.2.19 M_CONABB - Convert ASCII Date/Time to Byte Binary Format

The M_CONABB service converts the system date and time from ASCII format to byte binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONABB.

Entry Conditions

Calling Sequence:

```
M_CONABB [ASCBUF=] ascbuf, [BBBUF=] bbbuf

(or)

LA          R1,ascbuf
ORMW        R1,=X'06000000'
LA          R2,bbbuf
SVC         2,X'51'  (or)  M_CALL   H.REXS,75
```

ascbuf — is the address of a four-word buffer containing the ASCII-formatted date and time

bbbuf — is the address of a two-word buffer where the byte binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.20 M_CONADB - Convert ASCII Decimal to Binary

The M_CONADB service converts ASCII decimal doublewords to their binary equivalent.

An all blank doubleword converts to zero.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONADB.

Entry Conditions

Calling Sequence:

```
M_CONADB  [VALUEADDR]=n]

(or)

LD   R6,n
SVC  1,X'28' (or) M_CALL H.TSM,7
```

n — is the address of a left-justified, doubleword-bounded, ASCII-coded decimal number, blank-filled. If not specified, contents of registers six and seven are converted.

Exit Conditions

Return Sequence:

```
M.IPURTN  6,7
```

Registers:

R6	Zero if a character is nonnumeric
R7	Binary equivalent of input

7.2.21 M_CONAHB - Convert ASCII Hexadecimal to Binary

The M_CONAHB service converts ASCII hexadecimal doublewords to their binary equivalent.

An all blank doubleword converts to zero.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONAHB.

Entry Conditions

Calling Sequence:

```
M_CONAHB [VALUEADDR=]n]

(or)

LD   R6,n
SVC  1,X'29' (or) M_CALL  H.TSM,8
```

n is the address of a left-justified, doubleword-bounded, ASCII-coded decimal number, blank-filled. If not specified, contents of registers six and seven are converted.

Exit Conditions

Return Sequence:

```
M.IPURTN  6,7
```

Registers:

R6	Zero if a character is not hexadecimal
R7	Binary equivalent of input

7.2.22 M_CONASB - Convert ASCII Date/Time to Standard Binary

The M_CONASB service converts the system date and time from ASCII format to binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONASB.

Entry Conditions

Calling Sequence:

```
M_CONASB  [ASCBUF=] ascbuf, [BINBUF=] binbuf

(or)

LA        R1,ascbuf
ORMW      R1,=X'05000000'
LA        R2,binbuf
SVC       2,X'51'  (or)  M_CALL   H.REXS,75
```

ascbuf — is the address of a four-word buffer containing the ASCII formatted date and time

binbuf — is the address of a two-word buffer where the binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.23 M_CONBAD - Convert Binary to ASCII Decimal

The M_CONBAD service converts binary words to their ASCII decimal equivalent.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBAD.

Entry Conditions

Calling Sequence:

```
M_CONBAD [VALUEADDR=]n]

(or)

LW   R5,n
SVC  1,X'2A' (or) M_CALL  H.TSM,9
```

n the address of a positive binary number

Exit Conditions

Return Sequence:

M.IPURTN	6,7

Registers:

R6,R7	ASCII result, right-justified with leading ASCII zeros

7.2.24 M_CONBAF - Convert Binary Date/Time to ASCII Format

The M_CONBAF service converts the system date and time from binary format to ASCII format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBAF.

Entry Conditions

Calling Sequence:

```
M_CONBAF  [BINBUF=] binbuf, [ASCBUF=] ascbuf

(or)

LA        R1,binbuf
ORMW      R1,=X'2000000'
LA        R2,ascbuf
SVC       2,X'51'  (or)  M_CALL   H.REXS,75
```

binbuf — is the address of a two-word buffer containing the binary formatted date and time

ascbuf — is the address of a four-word buffer where the ASCII formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.25 M_CONBAH - Convert Binary to ASCII Hexadecimal

The M_CONBAH service is used to convert binary words to their ASCII hexadecimal equivalent.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBAH.

Entry Conditions

Calling Sequence:

```
M_CONBAH [VALUEADDR=]n]

(or)

LW   R5,n
SVC  1,X'2B' (or) M_CALL  H.TSM,10
```

n is the address of a binary number

Exit Conditions

Return Sequence:

M.IPURTN	6,7

Registers:

R6,R7	ASCII result, right-justified with leading ASCII zeros

7.2.26 M_CONBBA - Convert Byte Binary Date/Time to ASCII

The M_CONBBA service converts the system date and time from byte binary format to ASCII format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBBA.

Entry Conditions

Calling Sequence:

```
M_CONBBA  [BBBUF=] bbbuf, [ASCBUF=] ascbuf

(or)

LA        R1,bbbuf
ORMW      R1,=X'04000000'
LA        R2,ascbuf
SVC       2,X'51'  (or)  M_CALL   H.REXS,75
```

bbbuf — is the address of a two-word buffer containing the byte binary formatted date and time

ascbuf — is the address of a four-word buffer where the ASCII formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.27 M_CONBBY - Convert Binary Date/Time to Byte Binary

The M_CONBBY service converts the system date and time from binary format to byte binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBBY.

Entry Conditions

Calling Sequence:

```
M_CONBBY  [BINBUF=] binbuf, [BBBUF=] bbbuf

(or)

LA        R1,binbuf
ORMW      R1,=X'01000000'
LA        R2,bbbuf
SVC       2,X'51'  (or)  M_CALL   H.REXS,75
```

binbuf — is the address of a two-word buffer containing the binary formatted date and time

bbbuf — is the address of a two-word buffer where the byte binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.28 M_CONBYB - Convert Byte Binary Date/Time to Binary

The M_CONBYB service converts the system date and time from the byte binary format to binary format. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CONBYB.

Entry Conditions

Calling Sequence:

```
M_CONBYB  [BBBUF=] bbbuf, [BINBUF=] binbuf

(or)

LA        R1,bbbuf
ORMW      R1,=X'03000000'
LA        R2,binbuf
SVC       2,X'51'  (or)  M_CALL   H.REXS,75
```

bbbuf — is the address of a two-word buffer containing the byte binary formatted date and time

binbuf — is the address of a two-word buffer where the binary formatted date and time is returned

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.29 M_CONN - Connect Task to Interrupt

The M_CONN service indirectly connects a task to an interrupt level so that when the interrupt occurs, the specified task will be scheduled for execution (resumed). If the specified task is not active, M_CONN will preactivate it. If preactivation is required, but the actual interrupt connection is denied, M_CONN deletes the residual task because the task would continue in the suspended state indefinitely.

The nonbase mode equivalent service is M.CONN.

Entry Conditions

Calling Sequence:

```
M_CONN      [TASK=] task, [INTLEVEL=] intlevel
(or)
LW          R5,intlevel
LI          R6,0      } (or) LD R6,taskname (or)  LW R6, PNV
LW          R7,taskno }                           LI R7, 0
SVC         1,X'4B'  (or) M_CALL   H.REXS,10
```

task — the address of a doubleword containing the left-justifed blank-filled one-to eight-character ASCII name of the task (system file only); or zero in word zero and the task number in word one; or pathname or RID vector in word zero and zero in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

intlevel — is the hardware priority level where the task is to be connected

Exit Conditions

Return Sequence:

```
M.RTRN      6,7
```

Registers:

R6 — Denial Code:

Value	Description
1	Task already connected to an interrupt
2	Another task connected to the specified interrupt
3	Interrupt not SYSGEN specified indirectly connectable
4	Specified task not found in dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (by the M.KEY file).

R7 — Zero if task not connected to interrupt; otherwise, contains the task number

Abort Cases:

None

Output Messages:

None

7.2.30 M_CONSTRUCTPATH - Reconstruct Pathname

The M_CONSTRUCTPATH service constructs and returns the pathname string that was used to assign a file. In most cases, this service acquires the complete name of a file that was statically assigned to a task. If a pathname component contains special characters, the component is returned enclosed within single quotes.

The nonbase mode equivalent service is M.PNAM.

Entry Conditions

Calling Sequence:

```
M_CONSTRUCTPATH [RESOURCE=] addr1 , [PATH=] addr2 [ , [CNPADDR=] addr3]

(or)

LW      R1, addr1
LW      R4, addr2
LA      R7, addr3     (or)    ZR        R7
SVC     2,X'2F'       (or)    M_CALL    H.VOMM,16
```

addr1 is a FCB address or LFC for the assigned volume resource

addr2 is the PN address and maximum length

addr3 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R4	Contains addr2
R7	Contains addr3; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R4	M.RTRN R4
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R4	Actual PN length and PN address
R7	Return status if a CNP is not supplied; otherwise, unchanged

7.2.31 M_CONVERTTIME - Convert Time

The M_CONVERTTIME service returns the current date and time to the caller in the requested format. The caller must specify both the input and output format. This service can be executed by the IPU.

Entry Conditions

Calling Sequence:

```
M_CONVERTTIME [INBUFFER=] addr1 , [INFORMAT=] value1 ,
              [OUTBUFFER=] addr2 , [OUTFORMAT=] value2
```

(or)

```
SVC     2,X'51'   (or)   M_CALL   H.REXS,75
```

addr1 specifies the address of the input buffer

value1 is the keyword for the input format. The three valid keywords are as follows:

BIN Two-word internal binary value
word 1 is the number of days since January 1, 1960
word 2 is the number of clockticks since midnight

BYTE Eight-byte binary value

Byte	Contents in Binary
0	Century
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	Number of interrupts

QUAD Four-word ASCII string formatted the same as BYTE

addr2 specifies the address of the output buffer

value2 is the keyword for the output format. The three valid keywords are the same as for value1.

Registers:

R1	Contains addr1
R2	Contains addr2

Exit Conditions

Return Sequence:

M.RTRN

Registers:

R1,R2	Used by the call

Abort Cases:

RX13	Function code supplied is out of range

Output Messages:

None

7.2.32 M_CREATEFCB - Create File Control Block

The M_CREATEFCB service is used to define an expanded File Control Block.

The nonbase mode equivalent service is M.DFCB.

Entry Conditions

Calling Sequence:

```
M_CREATEFCB [LABEL=] label , [LFC=] lfc , [COUNT=] count ,
        [DATABUFFER=] addr1 , [ [WAITERROR=] addr2] , [ [RANDOM=] addr3] ,
        [[NWT=]{Y|N}] , [[NER=]{Y|N}] , [[DFI=]{Y|N}] , [[NST=]{Y|N}] ,
        [[RAN=]{Y|N}] , [[BIN=]{Y|N}] , [[SKIPLDR=]{Y|N}] , [[PACKED=]{Y|N}] ,
        [[PARITY=]{O|E}] , [[BPI=]{556|800}] , [ [NOWAIT=] addr4 ] ,
        [ [NOWAITERROR=] addr5 ] , [ [BLOCKBUFFER=] addr6 ]
```

label	ASCII string to used as symbolic label for address of this FCB
lfc	logical file code. See Section 5.9.1, word 0, bits 8-31.
count	transfer count specified in number of bytes. See Section 5.9.1, word 9, bits 0-31.
addr1	start address of data buffer (reserve on word boundary). See Section 5.9.1, word 8, bits 8-31.
addr2	error return address for wait I/O. See Section 5.9.1, word 6, bits 8-31.
addr3	random access address. See Section 5.9.1, word 10, bits 0-31.
NWT	control flag for no-wait I/O. See Section 5.9.1, word 2, bit 0.
NER	control flag for error return processing. See Section 5.9.1, word 2, bit 1.
DFI	control flag for data format. See Section 5.9.1, word 2, bit 2.

NST	control flag for status checking by the handler. See Section 5.9.1, word 2, bit 3.
RAN	control flag for random access. See Section 5.9.1, word 2, bit 4.
BIN	for use with a card reader. See Table 5-8.
SKIPLDR	for use with a paper tape reader. See Table 5-8.
PACKED	for use with 5-track magnetic tape. See Table 5-8.
PARITY	indicates even or odd parity. See Table 5-8.
BPI	designates bits per inch density. See Table 5-8.
addr4	specifies the address for normal no-wait I/O end action return
addr5	specifies the address for no-wait I/O error end action return
addr6	specifies the address of a 192-word user-supplied blocking buffer to be used for blocked I/O

7.2.33 M_CREATEP - Create Permanent File

The M_CREATEP service creates a permanent file. Permanent files are given names in directories and remain known to the operating system until explicitly deleted.

This service allocates a resource descriptor and the initial file space requirements for the file. Next, the specified attributes of the file are recorded in the resource descriptor. As a final step, the name of the file is established in the indicated directory.

When a directory entry is established, the directory entry is linked to the resource descriptor of the file. This link makes the relationship of the name of the file to the other attributes of the file. Typical file attributes are:

- The file's name
- The file's resource identifier (RID)
- The file's protection attributes
- The file's management attributes
- The file's initial space requirements

The nonbase mode equivalent service is M.CPERM.

Entry Conditions

Calling Sequence:

```
M_CREATEP [PNADDR=] addr1 [ , [RCBADDR=] addr2] [ , [CNPADDR=] addr3]

(or)

LW      R1, addr1
LA      R2, addr2    (or)   ZR       R2
LA      R7, addr3    (or)   ZR       R7
SVC     2,X'20'      (or)   M_CALL   H.VOMM,1
```

addr1 is an address containing a PN vector or PNB vector

addr2 is a RCB address or zero if default attributes are desired

addr3 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R2	Contains addr2; otherwise, zero
R7	Contains addr3; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.34 M_CREATET - Create Temporary File

The M_CREATET service creates a temporary file. Temporary files are not given names in directories and remain known to the operating system only for as long as the task that created them is in execution. Typically, when the task that created a temporary file terminates execution (normally or abnormally), associated temporary files are automatically deleted by the operating system.

Temporary files can remain defined to the operating system after the task that created them terminates execution when the temporary file has been made permanent or the temporary file is allocated (assigned) to another task when the creator terminates execution.

This service allocates a resource descriptor for the file and acquires the initial space requirements for the file. As a final step, the attributes of the file are recorded in the resource descriptor.

When a temporary file is created, the typical file attributes are:

- The file's resource identifier (RID)
- The file's protection attributes
- The file's management attributes
- The file's initial space requirements

The file's RID is returned only if a RCB address is specified and an ID location address for the file is also specified within the RCB.

The nonbase mode equivalent service is M.TEMP.

Entry Conditions

Calling Sequence:

```
M_CREATET [[PATH=] vector][ ,[RCBADDR=] addr1][ ,[CNPADDR=] addr2]

(or)

LW      R1 vector    (or)  ZR       R1
LA      R2, addr1    (or)  ZR       R2
LA      R7, addr2    (or)  ZR       R7
SVC     2,X'21'      (or)  M_CALL   H.VOMM,2
```

vector — is an address containing a PN (volume name only) vector or zero if the file to be created is on a working volume

addr1 — is a RCB address or zero if default attributes are desired

addr2 — is a CNP address or zero if CNP not supplied

Registers:

R1	Contains vector; otherwise, zero
R2	Contains addr1; otherwise, zero
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.35 M_CTIM - Convert System Date/Time Format

The M_CTIM service converts the system date and time from one of three standard formats (see Appendix H for time formats) to either of the other two formats. This service is callable from specific case macros that provide the function code in the macro call itself.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.CTIM.

Entry Conditions

Calling Sequence:

```
M_CTIM  [FUNCT=] funct, [STIME=] addr1 , [DTIME=] addr2
(or)
LA          R1,addr1
ORMW        R1,funct
LA          R2,addr2
SVC         2,X'51'  (or)  M_CALL   H.REXS,75
```

funct — is the address of a word that contains the function code (see chart below) in byte zero (most significant) and zeros in bytes one, two, and three

addr1 — is the address of a two- or four-word buffer where the user provides the date and time, in any of the three standard formats, for the system to convert

addr2 — is the address of a two- or four-word buffer where the system returns the converted date and time values in the format requested by the user

Funct code	Input format	Return format	Buffer in	Length out
1	Binary	Byte binary	2W	2W
2	Binary	Quad ASCII	2W	4W
3	Byte binary	Binary	2W	2W
4	Byte binary	Quad ASCII	2W	4W
5	Quad ASCII	Binary	4W	2W
6	Quad ASCII	Byte binary	4W	2W

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1,R2 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.36 M_CWAT - System Console Wait

The M_CWAT service suspends operation of the calling program until the specified I/O transfer is complete.

The nonbase mode equivalent service is M.CWAT.

Entry Conditions

Calling Sequences:

```
M_CWAT  [TCP=] tcpb

(or)

LA          R1,tcpb
SVC         1,X'3D'  (or)  M_CALL  H.IOCS,26
```

tcpb is the address of a Type Control Parameter Block (TCPB). See Section 5.10.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.37 M_DATE - Date and Time Inquiry

The M_DATE service returns to the caller the date (in ASCII), calendar information (century, year, month and day), and a count of the number of real-time clock interrupts since midnight. To aid in converting the interrupt count to time-of-day, counts of the number of interrupts per second and the number of interrupts per time unit are also returned.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.DATE.

Entry Conditions

Calling Sequence:

```
M_DATE  [PBADR=] pbaddr

(or)

LA          R7,pbaddr
SVC         1,X'15' (or) M_CALL  H.REXS,70
```

pbaddr is the logical word address of the first location of a parameter block formatted as follows:

- Words 0 and 1 contain the current date in the format entered at IPL time.
- Word 2

Bytes	Contents in Binary
0	Century
1	Year
2	Month
3	Day

- Word 3 contains the number of clock interrupts since midnight
- Word 4 contains the number of clock interrupts per second (initialized by SYSGEN)
- Word 5 contains the number of clock interrupts per time unit (initialized by SYSGEN)

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.38 M_DEASSIGN - Deassign and Deallocate Resource

The M_DEASSIGN service deallocates a resource and disassociates it from a logical file code. When a device associated with any unformatted media is detached, a message is issued to inform the operator to dismount the medium, unless inhibited by user request or system constraints. Deallocation of a nonshared resource makes it available to other tasks. Deallocation of a shared resource makes the resource available, if the caller is the last task to deallocate it or the access mode changes as a result of the deallocation to allow other compatible tasks to attach to the resource. Deallocation of SLO and SBO files result in their definitions being passed to the system output task for processing. If the specified logical file code has been equated to other logical file codes in the system, only the specified LFC is deallocated. If close has not been issued, the resource is also closed. This function can also issue a dismount message for an unformatted medium with no resource deallocation.

The nonbase mode equivalent service is M.DASN.

Entry Conditions

Calling Sequence:

```
M_DEASSIGN [RESOURCE=] addr1 [ , [CNPADDR=] addr2]

(or)

LW    R1, addr1
LA    R7, addr2    (or)    ZR        R7
SVC   2,X'53'      (or)    M_CALL    H.REXS,22
```

addr1 is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

addr2 is the address of a Caller Notification Packet (CNP) if notification is desired

Applicable portions of the CNP for this function are abnormal return address, options field, and status field.

The options field of the CNP has the following bit significance when set:

0 - issue dismount message with no resource deallocation

Registers:

R1 Contains addr1
R7 Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or in the status field of the CNP:

Value	Description
8	Unrecoverable I/O error to volume
29	Logical file code associated with FCB not assigned
30	Invalid allocation index
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

7.2.39 M_DEBUG - Load and Execute Interactive Debugger

The M_DEBUG service causes one of the following events to occur:

- If the interactive debugger is currently loaded at the time the service is called, control is transferred to the debugger.
- If the interactive debugger is not currently loaded at the time the service is called, the debugger is loaded as an overlay segment, then control is transferred to the debugger.

DEBUGX32 is loaded with base mode tasks.

The nonbase mode equivalent service is M.DEBUG.

Entry Conditions

Calling Sequence:

```
M_DEBUG

(or)

SVC     1,X'63' (or) M_CALL   H.REXS,29
```

Exit Conditions

Normal Return Sequence to Debugger:

M.RTRN	R7	Contains the transfer address of the debugger if the debugger was loaded by this service call. Register seven contains zero if the debugger was already loaded at the time this service was called.

Abnormal Return Sequence to Caller:

	Value	Description
R7	2	Debugger load module not found
	4	Invalid preamble
	5	Insufficient task space for loading
	6	I/O error on resource descriptor
	7	I/O error on resource
	8	Loading error

Abort Cases:

None

Output Messages:

None

7.2.40 M_DEFT - Change Defaults

The M_DEFT service changes the caller's working directory or project group protection or any combination of the two attributes of the caller. The caller can change current working directory only or project group protection only or both.

Typically, the caller should invoke this service with two separate calls as though changing project group protection and changing the current working directory were two distinct services.

In cases where it is desirable to change both attributes with a single call, the caller should be aware of the effects. When both project group and working directory are specified, the project group is changed prior to attempting to change the current working directory. After changing the project group, if the attempt to establish the current working directory fails, the new project group protection will remain in effect and the caller will be notified via an error status code that the current working directory request failed. It is the responsibility of the caller to determine whether or not to continue with the new project group or to reestablish another project group.

The nonbase mode equivalent service is M.DEFT.

Entry Conditions

Calling Sequence:

```
M_DEFT  [ARGA=] arga [,[CNP=] cnpaddr] [,[PRJADR=] prjaddr] [,[KEYADR=] keyaddr]

(or)

LW          R1,arga
LA          R4,prjaddr  (or)  ZR  R4
LA          R5,keyaddr  (or)  ZR  R5
LA          R7,cnpaddr  (or)  ZR  R7
SVC         2,X'27'  (or)  M_CALL  H.VOMM,8
```

arga — contains a PN vector, PNB vector, or RID vector

cnpaddr — is a CNP address or zero if CNP not supplied

prjaddr — is the address of the new project group name or zero if no change to project group name

keyaddr — is the address of the new project group key or zero if not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

7.2.41 M_DELETER - Delete Resource

The M_DELETER service explicitly deletes volume resources. This service must be used to delete directories, files, and memory partitions. The caller cannot delete a resource to which the caller does not have delete access.

As the first step, this service deletes the directory entry for the specified resource. Next, the volume space requirements are released. As a final step, the resource descriptor is released.

If the resource is allocated at the time of the delete request, only the directory entry is deleted. The volume space requirements and the resource descriptor for the resource will be released when the last assignment to the resource is removed.

To delete a permanent file or memory partition, the pathname or pathname block must be supplied. To delete a directory, the pathname or pathname block must be supplied and all files which were defined in the directory must have been previously deleted.

To delete a temporary file, the caller can provide the resource identifier (RID), or specify the logical file code (LFC), or the address of a File Control Block (FCB).

The nonbase mode equivalent service is M.DELR.

Entry Conditions

Calling Sequence:

```
M_DELETER [RESOURCE=] addr1 [ , [CNPADDR=] addr2]

(or)

LW      R1, addr1
LA      R7, addr2 (or) ZR R7
SVC     2,X'24'  (or)  M_CALL  H.VOMM,5
```

addr1 is an address containing a PN vector, PNB vector, LFC, or FCB

addr2 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.42 M_DELTSK - Delete Task

The M_DELTSK service forces I/O completion and immediately aborts the specified task. See Task Termination Sequencing in Chapter 2. This service should only be used when abort fails to remove a task or when a task is queued for a resource. File integrity can be affected because operations are not allowed to complete normally. To preserve system integrity, the kill directive is processed as an abort for ten seconds. If this does not remove the task, it is killed.

The nonbase mode equivalent service is M.DELTSK.

Entry Conditions

Calling Sequence:

```
M_DELTSK  [ABCODE=] abcode, [TASK=] task, [EXTCODE=] extcode

(or)

LD        R2,extcode
LW        R5,abcode
LI        R6,0        }    (or) LD R6,taskname
LW        R7,taskno   }
SVC       1,X'5A'  (or)  M_CALL  H.REXS,31
```

abcode — contains the abort code consisting of four ASCII characters

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

extcode — contains the extended abort code message consisting of one- to eight-ASCII characters, left-justified, and blank-filled

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (by the M.KEY file); otherwise, contains the task number.

Abort Cases:

None

Output Messages:

Modifies abort message to:

ABORT task REASON: xxxx zzzzzzzz AT: yyyyyyyy

where:

zzzzzzzz is the extended message code supplied with the call to this service

7.2.43 M_DEVID - Get Device Mnemonic or Type Code

The M_DEVID service allows the user to pass a device mnemonic or a generic device type code and receive the corresponding type code or mnemonic. See Appendix A for device mnemonic and device type codes.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.DEVID.

Entry Conditions

Calling Sequence:

```
M_DEVID  [DEVID=] id

(or)

LW          R2,id
SVC         1,X'14' (or)  M_CALL  H.REXS,71
```

id contains either a device mnemonic in the right halfword with the left halfword zero or a device type code in byte 3 with zero in bytes 0-2

Exit Conditions

Return Sequence:

```
M.IPURTN                2
```

Registers if input was a device mnemonic:

R2 Bytes 0-2 contain zeros. Byte 3 contains corresponding device type code

Registers if input was a device type code:

R2 Left halfword contains zero. Right halfword contains corresponding device mnemonic

Registers if input was a mnemonic or device type code not in the system Device Type Table (DTT):

R2 Bit 0 is set. Bits 1-31 are unchanged

Abort Cases:

None

Output Messages:

None

7.2.44 M_DIR - Create Directory

The M_DIR service creates a permanent directory. Permanent directories are given names in the root directory and remain known to the operating system until explicitly deleted.

Directories are used to contain the names of permanent files and memory partitions that are created in the directories. In this way, directories are used to classify permanent files as to whatever classification category is determined by the creator of the directories.

This service allocates a resource descriptor and the volume space requirements for the directory. Next, the indicated attributes of the directory are recorded in the resource descriptor. As a final step, the name of the directory is established in the indicated previous level (parent) directory.

When the directory is established, the directory entry is linked to the resource descriptor of the new directory. This link makes the relationship of the name of the new directory to the other attributes of the new directory. Typical directory attributes are:

- The directory's name
- The directory's resource identifier (RID)
- The directory's protection attributes
- The directory's management attributes
- The directory's volume space requirements

The nonbase mode equivalent service is M.DIR.

Entry Conditions

Calling Sequence:

```
M_DIR  [PNADR=] pnaddr[,[CNP=] cnpaddr] [, [RCB=] rcbaddr]
(or)
LW          R1,pnaddr
LA          R2,rcbaddr  (or)  ZR  R2
LA          R7,cnpaddr  (or)  ZR  R7
SVC         2,X'23'  (or)  M_CALL  H.VOMM,4
```

pnaddr	contains a PN vector or PNB vector
cnpaddr	is a CNP address or zero if CNP not supplied
rcbaddr	is a RCB address or zero if default attributes are desired

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

7.2.45 M_DISCON - Disconnect Task from Interrupt

The M_DISCON service disconnects a task that has previously been centrally connected to an interrupt level.

The nonbase mode equivalent service is M.DISCON.

Entry Conditions

Calling Sequence:

```
M_DISCON [TASK=] task

(or)

LI      R6,0        }   (or)    LD R6,taskname
LW      R7,taskno   }
SVC     1,X'5D'     (or)    M_CALL  H.REXS,38
```

task — the address of a doubleword containing the name of the task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Normal Return Sequence:

```
M.RTRN          7
```

Registers:

R7 Contains the task number

Abnormal Return Sequence:

```
M.RTRN          6,7
```

Registers:

R6 Contains denial code as follows:

Value	Description
1	Task not found in dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name (via the M.KEY file).
2	Task not indirectly connected
3	Task connected to invalid interrupt

R7 Zero if the task was not previously connected to an interrupt level

Abort Cases:

None

Output Messages:

None

7.2.46 M_DISMOUNT - Dismount Volume

The M_DISMOUNT service decrements the number of established users of a volume. If there are no remaining users of the volume, the Mounted Volume Table Entry (MVTE) associated with the volume is deallocated, and the mount device made available. If appropriate, a dismount message is displayed on the console to inform the operator that the volume should be dismounted.

The nonbase mode equivalent service is M.DMOUNT.

Entry Conditions

Calling Sequence:

```
M_DISMOUNT [VOLADDR=] addr1][ , [CNPADDR=] addr2]

(or)

LA      R1, addr1
LA      R7, addr2    (or)  ZR       R7
SVC     2,X'4A'      (or)  M_CALL   H.REMM,19
```

addr1 is the address (doubleword bounded) of the volume name

addr2 is the address of a Caller Notification Packet if notification is desired

Applicable portions of the CNP for this function are abnormal return address and status field.

Registers:

R1	Contains addr1
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or in the status field of the CNP:

Value	Description
14	Caller has outstanding resource assignments on this volume
20	Volume not assigned to this task or volume is public

Wait Conditions

None

7.2.47 M_DLTT - Delete Timer Entry

The M_DLTT service resets the timer for the specified task so that its specified function is no longer performed upon time-out. Deletion of the timer entry does not delete the associated task. One-shot timers are deleted on expiration.

The nonbase mode equivalent service is M.DLTT.

Entry Conditions

Calling Sequence:

```
M_DLTT      [TIMER=] timer

(or)

LW          R7,timer
SVC         1,X'47' (or) M_CALL   H.REXS,6
```

timer right-justified two-character ASCII name of a timer

Exit Conditions

Return Sequence:

M.RTRN

CC1 Timer entry not found

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.48 M_DSMI - Disable Message Task Interrupt

The M_DSMI service disables the task interrupts for messages sent to the calling task. M_DSMI is useful for synchronization gating of the task message interrupts.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.DSMI.

Entry Conditions

Calling Sequence:

```
M_DSMI

(or)

SVC     1,X'2E' (or) M_CALL  H.REXS,57
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1	Task interrupts were already disabled

Abort Cases:

None

Output Messages:

None

7.2.49 M_DSUB - Disable User Break Interrupt

The M_DSUB service deactivates the user break interrupt (see M_ENUB) and allows user breaks by the terminal BREAK key to be acknowledged.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.DSUB.

Entry Conditions

Calling Sequence:

```
M_DSUB

(or)

SVC     1,X'12' (or) M_CALL  H.REXS,73
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1	User break already disabled

Abort Cases:

None

Output Messages:

None

7.2.50 M_DUMP - Memory Dump Request

The M_DUMP service provides a dump of the caller's Program Status Doubleword (PSD), General Purpose Registers, and specified memory limits. The output is to a SLO file in side-by-side hexadecimal with ASCII format, with the PSD and registers preceding the specified memory limits. The PSD and registers are extracted from the first level of push-down of the calling task. Optionally, register 5 can specify the address of a ten word block containing registers 0 through 7 and the PSD to be dumped, respectively. Any task can request a memory dump.

The nonbase mode equivalent service is M.DUMP.

Entry Conditions

Calling Sequence:

```
M_DUMP    [START=] start, [ENDADR=] end [, [CTXBUF=] ctxbuf]
(or)
ZR        R5 (or) LA  R5,ctxbuf
LW        R6,start
LW        R7,end
SVC       1,X'4F' (or) M_CALL  H.REXS,12
```

start — contains the low logical word address requested in dump

end — contains the high logical word address requested in dump

ctxbuf — is the optional address of ten consecutive words containing R0 through R7 and a PSD, respectively. If R5=0, the registers and PSD dumped are taken from the first level of push-down.

NOTE: Start and end are truncated to the nearest eight-word boundaries and memory is dumped between the truncated limits.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6

Value	Description
1	High dump limit less than low limit
4	No FAT or FPT space available
5	Request made with insufficient levels of push-down available
6	Cannot allocate SLO file
7	Unrecoverable I/O error

R7 — Zero if dump could not be performed

Abort Cases:

None

Output Messages:

None

7.2.51 M_ENMI - Enable Message Task Interrupt

The M_ENMI service enables task interrupts for messages sent to the calling task. It removes an inhibit condition previously established by invoking the M_DSMI service.

The nonbase mode equivalent service is M.ENMI.

Entry Conditions

Calling Sequence:

```
M_ENMI

(or)

SVC     1,X'2F' (or) M_CALL  H.REXS,58
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

CC1 set Task interrupts were already enabled

Abort Cases:

None

Output Messages:

None

7.2.52 M_ENUB - Enable User Break Interrupt

The M_ENUB service activates the user break interrupt and causes further user breaks from the user terminal break key to be ignored.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.ENUB.

Entry Conditions

Calling Sequence:

```
M_ENUB

(or)

SVC     1,X'13' (or) M_CALL  H.REXS,72
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1 set	User break already enabled

Abort Cases:

None

Output Messages:

None

7.2.53 M_ENVRMT - Get Task Environment

The M_ENVRMT service obtains more information on the task environment than what is provided in the Task Option Word.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.ENVRMT.

Entry Conditions

Calling Sequence:

```
M_ENVRMT

(or)

SVC     2,X'5E'   (or)   M_CALL   H.REXS,85
```

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 Contains the task environment word as follows:

Bit	Definition
0	0 if batch task 1 if interactive or real-time task
1	0 if option NOCOMMAND is set 1 if option COMMAND is set
2	0 if option NOERR is set 1 if option ERROR is set
3	0 if cataloged or linked unprivileged 1 if cataloged or linked privileged
4	0 if currently unprivileged 1 if currently privileged
5-31	Reserved

Abort Cases:

None

Output Messages:

None

7.2.54 M_EXCLUDE - Exclude Shared Image

The M_EXCLUDE service allows a base mode task to dynamically exclude a shared image previously included. This service causes the assign count and user count to be decremented. The shared image is deleted and its resources returned to the free list when the assign count goes to zero. This service is also called by the exit processor (H.REMM,3) whenever a task aborts or comes to an unnatural end while associated with a shared image. The shared image is identified by the allocation index obtained when the shared image was included, or the resource identifier (RID) used to create it along with the owner name used to include it.

The nonbase mode equivalent service is M.EXCLUDE.

Entry Conditions

Calling Sequence:

```
M_EXCLUDE [RESOURCE=] addr1 [ , [CNPADDR=] [addr2]

(or)

LW      R1, addr1
LA      R7, addr2 (or) ZR  R7
SVC     2,X'41' (or) M_CALL H.REMM,14
```

addr1 contains a PN vector, PNB vector, RID vector, or the allocation index obtained from the M_INCLUDE service

addr2 is a CNP address or zero if CNP not supplied. Applicable portions of the CNP are abnormal return address and status field.

Registers:

R1	Contains addr1
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R3, R5	M.RTRN R3, R5
(or)	(or)
M.RTRNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
5	Unable to locate Shared Memory Table entry
30	Invalid allocation index
35	Attempt to exclude shared image that is not mapped into the calling task's address space
39	Unable to write back data section
58	Shared memory table space is not available

7.2.55 M_EXIT - Terminate Task Execution

The M_EXIT service performs all normal termination functions required of exiting tasks. See Chapter 2. All devices and memory are deallocated, related table space is erased, and the task's Dispatch Queue entry is cleared.

The nonbase mode equivalent service is M.EXIT.

Entry Conditions

Calling Sequence:

M_EXIT

(or)

SVC 1,X'55' (or) M_CALL H.REXS,18

Register 0 must be preloaded with an ASCII abort code or zero.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

RX92 Task has attempted normal exit with messages in its receiver queue

Output Messages:

None

7.2.56 M_EXTENDFILE - Extend File

The M_EXTENDFILE service allows the space of a file to be manually extended. The caller may specify the size of the requested extension or may choose to use the default file extension parameters defined when the file was created. If the file was created with the zero option specified, the extension will be zeroed.

This service will only extend temporary or permanent files that are manually extendable. Directories and memory partitions cannot be extended. The caller must have write, update, or append access in order to extend the file.

The caller can extend a file regardless of whether the file is currently allocated. Additionally, the caller can supply any allowable resource specification, such as pathname (PN), pathname block (PNB), resource ID (RID), logical file code (LFC) or address of a File Control Block (FCB).

The nonbase mode equivalent service is M.EXTD.

Entry Conditions

Calling Sequence:

```
M_EXTENDFILE [RESOURCE=] addr1 [ , [BLOCKS=] num] [ , [CNPADDR=] addr2]

(or)

LW      R1, addr1
LN      R6, num      (or)   ZR R6
LA      R7, addr2    (or)   ZR R7
SVC     2,X'25'      (or)   M_CALL H.VOMM,6
```

addr1 contains a PN vector, PNB vector, RID vector, LFC, or FCB

num is an address containing the number of blocks to extend the file by or zero if RCB extension parameters specified during file creation are to be used

addr2 is a CNP address or zero if not supplied

Registers:

R1	Contains addr1
R6	Contains num; otherwise, zero
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R6	M.RTRN R6
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R6	Number of contiguous blocks file actually extended by
R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.57 M_EXTSTS - Exit With Status

The M_EXTSTS service provides exit status of a task, for example, was the task successful or not. Register 0 contains either a zero or a valid four-character ASCII abort code. If register 0 is not a zero or a valid abort code, the task is terminated with an RX36 abort condition.

Entry Conditions

Calling Sequence:

```
M_EXTSTS  [STATADDR=] addr

(or)

LW        R0, addr
SVC       2,X'5F'  (or)  M_CALL  H.REXS,86
```

addr is the address where the final status code is to be stored

Registers:

R0 Contains addr

Exit Conditions

Return Sequence:

None

Registers:

None

Abort Cases:

RX36 Status in register 0 is not a zero or a valid abort code

Output Messages:

task #number ABORT AT: xxxxxxxx - yyyyy mm/dd/yy hh:mn:ss zzzz

task is the one- to eight-character name of the task being aborted

number is the task number of the task being aborted

xxxxxxxx is the location the abort occurred

yyyyy is the beginning of the DSECT

mm is the two-character decimal number of the month between 01 and 12

dd is the two-character decimal number of the day between 01 and 31

yy is the two-character decimal number of the year between 00 and 99

hh is the two-character decimal number of the hour between 00 and 23

mn is the two-character decimal number of the minutes between 00 and 59

ss is the two-character decimal number of the seconds between 00 and 59

zzzz is the four-character abort code

7.2.58 M_FREEMEMBYTES - Free Memory in Byte Increments

The M_FREEMEMBYTES service allows the task to dynamically deallocate acquired memory. Deallocation can be random. The space address must have been previously obtained from the M_GETMEMBYTES service. All of the space obtained from a given call is deallocated.

The nonbase mode equivalent service is M.MEMFRE.

Entry Conditions

Calling Sequence:

```
M_FREEMEMBYTES [MEMADDR=] addr

(or)

LW      R3, addr
SVC     2,X'4C'  (or)  M_CALL  H.REMM,29
```

addr — is the starting address of dynamic space previously acquired from the M_GETMEMBYTES service

Registers:

R3 — Contains addr

Exit Conditions

Return Sequence:

M.RTRN — R3 (or) abort user with RM77

Registers:

R3 — Equals zero if deallocation could not be performed. Deallocation address was not found in allocation table.

Abort Cases:

RM77 — A task has destroyed the allocation linkages in this dynamic expansion space

7.2.59 M_GETCTX - Get User Context

The M_GETCTX service is used to store the current values for user context into a specified buffer. The buffer must be on a word boundary.

Entry Conditions

Calling Sequence:

```
M_GETCTX  [BUFFER=] addr , [NUMBER=] num

(or)

LA        R1, addr
LI        R4, num
SVC       2,X'70'  (or)  M_CALL  H.EXEC,41
```

addr is a word bounded buffer address where the context is to be stored

num is the number of bytes allocated for the buffer

Registers:

R1	Contains addr
R4	Contains num

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

CC1	Equals zero for Debugger Status Word
CC1	Equals one if an error condition occurs
R7	Contains error condition (hexadecimal equivalent) as follows:

Value	Description
257	Destination buffer is in the operating system
258	Destination buffer is in the TSA
259	Invalid destination buffer address
260	Buffer length not a word multiple

7.2.60 M_GETMEMBYTES - Get Memory in Byte Increments

The M_GETMEMBYTES service allows the task to dynamically expand its memory allocation in doubleword increments starting at the end of its DSECT up to the end of its logical address space. The additional memory will be of the same type specified when the task was linked. The task is mapped in a logically contiguous manner up to the end of its address space. The task is suspended until the allocation is successful. Repeated calls to this service are allowed. Allocation is not contiguous with previously allocated space.

The nonbase mode equivalent service is M.MEMB.

Entry Conditions

Calling Sequence:

```
M_GETMEMBYTES [NUMBER=] num

(or)

LI      R4, num
SVC     2,X'4B'   (or)   M_CALL   H.REMM,28
```

num is the number of bytes to allocate

Registers:

R4	Contains num

Exit Conditions

Return Sequence:

```
M.RTRN      R3,R4
```

Registers:

CC1	Equals zero
CC2	Equals zero
R3	Contains the 24-bit starting logical doubleword address of allocated space
R4	Contains the number of bytes actually allocated (modulo 2W)

(or)

CC1	Equals zero
CC2	Equals one

R3	Contains the 24-bit starting logical doubleword address of allocated space
R4	Contains the number of bytes actually allocated (modulo 2W). However, the number is less than requested.

Error Conditions

Allocation Denied:

CC1	Equals one
CC2	Equals one
R3	Equals zero
R4	Equals zero

7.2.61 M_GETTIME - Get Current Date and Time

The M_GETTIME service returns the current date and time to the caller in any one of the three standard formats described in Appendix H.

This service can be executed by the IPU.

Entry Conditions

Calling Sequence:

```
M_GETTIME [OUTBUFFER=] addr , [OUTFORMAT=] value

(or)

SVC       2,X'50'   (or)   M_CALL   H.REXS,74
```

addr specifies the address of the output buffer

value is the keyword for the format in which the date and time will be returned. The three valid keywords are as follows:

Function Code	Keyword	Format
1	BIN	Two-word internal binary value as follows:

Word	Contents
1	Number of days since January 1, 1960
2	Number of clockticks since midnight

Function Code	Keyword	Format
2	BYTE	Eight-byte binary value as follows:

Byte	Contents in Binary
0	Century
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	Number of interrupts

Function Code	Keyword	Format
3	QUAD	Four-word ASCII string formatted as follows:

Halfword	Contents in ASCII
0	Century
1	Year
2	Month
3	Day
4	Hour
5	Minute
6	Second
7	Number of interrupts

Registers:

R1	Byte 0 contains the function code. Bytes 1-3 contain the addr.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

R1	Used by the call

Abort Cases:

RX13	Function code supplied is out of range

Output Messages:

None

7.2.62 M_GMSGP - Get Message Parameters

The M_GMSGP service is called from the message receiver routine of a task that has received a message interrupt. It transfers the message parameters into the designated receiver buffer, and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB). For description of the PRB, see Chapter 2.

The nonbase mode equivalent service is M.GMSGP.

Entry Conditions

Calling Sequence:

```
M_GMSGP    [PRB=] prbaddr

(or)

LA         R2,prbaddr
SVC        1,X'7A'  (or)  M_CALL  H.REXS,35
```

prbaddr is the logical address of the Parameter Receive Block (PRB)

Exit Conditions

Return Sequence:

```
M.RTRN     6
```

Registers:

R6 Contains the processing status error code:

Value	Description
0	Normal status
1	Invalid PRB address
2	Invalid receiver buffer address or size detected during parameter validation
3	No active send request
4	Receiver buffer length exceeded during transfer

Abort Cases:

None

Output Messages:

None

7.2.63 M_GRUNP - Get Run Parameters

The M_GRUNP service is called by a task that is executing for a run request. It transfers the run parameters into the designated receiver buffer, and posts the owner name and task number of the sending task into the Parameter Receive Block (PRB). See Chapter 2.

The nonbase mode equivalent service is M.GRUNP.

Entry Conditions

Calling Sequence:

```
M_GRUNP    [PRB=] prbaddr

(or)

LA         R2,prbaddr
SVC        1,X'7B'  (or)  M_CALL  H.REXS,36
```

prbaddr is the logical address of the Parameter Receive Block (PRB)

Exit Conditions

Return Sequence:

```
M.RTRN     6
```

Registers:

R6 Contains the processing status error code:

Value	Description
0	Normal status
1	Invalid PRB address
2	Invalid receiver buffer address or size detected during parameter validation
3	No active send request
4	Receiver buffer length exceeded during transfer

Abort Cases:

None

Output Messages:

None

7.2.64 M_GTIM - Acquire System Date/Time in Any Format

The M_GTIM service acquires the system date and time in any one of the three standard formats described in Appendix H. The user can also get the system date/time by using any of the three specific case macros. These macros generate the same SVC call but the function code is provided by the macro.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.GTIM.

Entry Conditions

Calling Sequence:

```
M_GTIM  [FUNCT=] funct, [TIMBUF=] timbuf

(or)

LA      R1,timbuf
ORMW    R1,funct
SVC     2,X'50'  (or)  M_CALL   H.REXS,74
```

funct — is the address of a word containing the function code (see chart below) in byte zero (most significant) and zeros in bytes one, two, and three

timbuf — is the address of a buffer where the service places the date and time in the format requested by the user. This buffer is two or four words in length depending on the format desired.

Funct	Return format	Buffer length
1	Binary	2W
2	Byte binary	2W
3	Quad ASCII	4W

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1 — Used by call - all others returned intact

Abort Cases:

RX13 — Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.65 M_HOLD - Program Hold Request

The M_HOLD service makes the specified task ineligible for CPU control by setting the hold bit in the CPU Dispatch Queue. The specified task remains in the hold state until the operator issues the OPCOM CONTINUE directive. If the specified task is not in the CPU Dispatch Queue, the request is ignored.

The nonbase mode equivalent service is M.HOLD.

Entry Conditions

Calling Sequence:

```
M_HOLD    [TASKID=] task

(or)

LI        R6,0      } (or) LD R6,taskname
LW        R7,taskno }
SVC       1,X'58        (or) M_CALL  H.REXS,25
```

task — the address of a doubleword containing the name of the task or zero in word zero and the task number in word one. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with the M.KEY file; otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

7.2.66 M_ID - Get Task Number

The M_ID service allows the user to pass the address of a parameter block containing any of the following: task number, task name, owner name, or task pseudonym, and the service will provide the missing items if a matching entry is found. Initially, the caller passes zero as the index value following the parameter block address. If more than one task in the Dispatch Queue satisfies the given parameters, the service returns to the caller with an index value in register five for retrieval of further entries. The caller is responsible for updating the index with the contents of register five and reissuing M_ID until all tasks that meet specifications have been identified or register five equals zero.

The nonbase mode equivalent service is M.ID.

Entry Conditions

Calling Sequence:

```
M_ID        [PBADRR=]pbaddr,[INDEX=]index

(or)

LW          R5,a variable equal to 0 or an index
LA          R7,pbaddr
SVC         1,X'64' (or) M_CALL  H.REXS,32
```

pbaddr — is the logical word address of the first location of a parameter block formatted as follows:

Word	Contents
0	Task activation sequence number
1-2	Task name
3-4	Owner name
5-6	Pseudonym

The user supplies those items that are known and zeros the other words.

index — a variable equal to zero for initial call, then previous DQE address for each subsequent call

Exit Conditions

Return Sequence:

M.RTRN 5

Registers:

Normal Return

R5	Bit 0 is set if more than one task satisfies the given parameters. Bits 1-31 contain the DQE address of the first matching task found. If no entry satisfies the given parameters, register five equals zero. Register five may be used as input for subsequent calls.

Abnormal Return

CC1 set	Invalid parameter block address. Register five remains unchanged.

Abort Cases:

None

Output Messages:

None

7.2.67 M_INCLUDE - Include Shared Image

The M_INCLUDE service allows a base mode task to include a shared image or a memory partition into its address space. The task is suspended until the inclusion is complete. If the resource has not been included by another task, an Allocated Resource Table (ART) and a Shared Memory Table (SMT) entry is established for the resource. The resource is automatically allocated for explicit shared use. If inclusion is successful, the assign and user counts are incremented for the resource. The shared image is identified by resource pathname or a resource identifier (RID) that was defined when the partition was created. A partition is identified by an eight-character partition name, or a resource identifier (RID) that was defined when the partition was created. If the partition is dynamic, it is identified by an eight-character owner name that is used to associate this copy of the partition with a particular set of users. Prezeroing of partitions is not performed by this service. The resource is swappable with the task, if the user count goes to zero, and remains allocated until the assign count is zero. The option is provided to lock the resource for exclusive use. However, the resource remains locked until: the owner of the lock terminates, the Release Exclusive Lock service is explicitly called, or the resource is excluded by the task.

Once a shared image has been included by a task, subsequent includes by that task are ignored.

The nonbase mode equivalent service is M.INCLUDE.

WARNING: To ensure proper inclusion, the first eight characters of a shared image file name or partition name should be unique within the system.

Entry Conditions

Calling Sequence:

```
M_INCLUDE [RESOURCE=] addr1 [ , [CNPADDR=] addr2]

(or)

LW      R1, addr1
LA      R7, addr2 (or)  ZR  R7
SVC     2,X'40' (or) M_CALL H.REMM,12
```

addr1 contains a PN vector, PNB vector, or RID vector

addr2 is a CNP address or zero if CNP not supplied. Applicable portions of the CNP for this function are time-out value, abnormal return address, options field, and status field.

The options field has the following bit significance when set:

Bit	Meaning if Set
0	Read/write access
1	Use written back data section
2	Set exclusive resource lock
3	Reserved for MPX-32
4-15	Reserved

Registers:

R1 contains addr1

R7 contains addr2; otherwise zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R3, R5	M.RTRN R3,R5
(or)	(or)
M.RTRNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R3 Starting logical address of the shared image
R5 Allocation index which can be used to identify the shared image for the resource lock and image exclusion services. It contains the nonzero bias SMT index in the first byte and the address of the associated ART entry in the next three.
R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
1	Unable to locate shared image file
2	Requested access mode not allowed
8	Unrecoverable I/O error to volume
16	Memory requirements conflict with task's address space
35	Resource is not a shared image
36	Requested physical memory already allocated
37	Nonpresent physical memory requested
38	Time out occurred waiting for shared memory to become available
40	Invalid load module
41	Invalid requested physical memory
44	Write back requested and shared image has no write back section
45	Loading error during inclusion of read only section of shared image
47	Loading error during inclusion of read and write section of shared image
48	Incompatible load addresses for shared image
49	Excessive multicopied shared images with no read-only section (only three are allowed)
55	Allocated Resource Table (ART) is full
58	Shared Memory Table (SMT) space unavailable
80	Shared image version level is not compatible with executable image

7.2.68 M_INQUIRER - Resource Inquiry

The M_INQUIRER service is used to obtain information specific to a resource allocated by a base mode task. The information is returned in the form of a series of pointers to the various data structures within the system that describe the resource. The resource must have been previously allocated (included for memory partitions) by the caller. Resources are identified by: a logical file code obtained when the resource was allocated, a memory partition name defined when the partition was created, or an allocation index obtained when the resource was allocated/included. If not supplied as an argument, the caller is provided with the unique allocation index which can be used to set and release exclusive or synchronous locks on the resource while it remains allocated. It is the caller's responsibility to interpret the information in the identified structures as the application dictates. It is recommended that this be performed by a user-supplied subroutine which acts as a common interface between application programs and this service. In this way, resource inquiries will be less sensitive to changes in system structures.

The nonbase mode equivalent service is M.INQUIRY.

Entry Conditions

Calling Sequence:

```
M_INQUIRER [BUFFER=] addr1 , [RESOURCE=] addr2 [ , [CNPADDR=] addr3]

(or)

LA      R1, addr1
LD      R4, addr2
LA      R7, addr3 or ZR R7
SVC     2,X'48'  (or)  M_CALL  H.REMM,27
```

addr1 is the address of an eight-word parameter description area where the pointers to the appropriate system structure entries corresponding to this resource are to be returned

addr2 is a doubleword address containing byte 0 cleared and a one- to three-character (left-justified, blank-filled) LFC in bytes 1, 2, and 3 of word 0 with word 1 zero

(or)

a doubleword address where word 0 contains the address of a one- to eight-character (left-justified, blank-filled) memory partition name and word 1 contains the left-justified task number or address of a one- to eight-character (left-justified, blank-filled) owner name

(or)

a doubleword address in which word 0 is zero and the allocation index obtained when the resource was assigned is in word 1

addr3 is the address of a Caller Notification Packet (CNP) if notification is desired. Applicable portions of the CNP for this function are abnormal return address and status field.

Registers:

R1	Contains addr1
R4	Contains addr2
R7	Contains addr3; otherwise, zero

Exit Conditions

Calling Sequence:

(with CNP)	(without CNP)
M.RTRN R5	M.RTRN R5
(or)	(or)
M.RTNA R5 (CC1 set)	M.RTRN R5,R7 (CC1 set)

Registers:

R5	Allocation index if not supplied as an argument, or zero if resource is undefined
R7	Return status if a CNP is not supplied; otherwise, unchanged

The interpretation of each word in the parameter description area and some of the more pertinent information that can be extracted from each structure is as follows:

Word 0 - Allocated Resource Table (ART) address:
- Number of tasks assigned to this resource
- Number of tasks currently using resource
- Exclusive lock owner (DQE index)
- Synchronous lock owner (DQE index)
- Current allocation usage mode
- Current allocation access mode (implicit shared)
- Shared relative EOF block number (implicit shared)
- Shared relative EOM block number (implicit shared)

Word 1 - File Assignment Table (FAT) address:
- Relative EOF block
- Relative EOM block
- Number of segments in file
- Current segment number
- Current access mode
- Relative file block position
- Volume number (unformatted media only)
- Unformatted ID (unformatted media only)
- Assigned access restrictions
- File attribute and status flags

Word 2 - Unit Definition Table (UDT) address:

- Device type code
- Logical channel number
- Logical subchannel number
- Physical channel number (if different)
- Physical subchannel number (if different)
- Sectors per block (disc/floppy)
- Sectors per allocation unit (disc/floppy)
- Sectors per track (disc/floppy)
- Number of heads (disc/floppy)
- Total number of allocation units (disc/floppy)
- Sector size (disc/floppy)
- Characters per line (TTY/terminal)
- Lines per screen (TTY/terminal)
- Tab size (TTY/terminal)
- Tab settings (TTY/terminal)

Word 3 - Device Type Table (DTT) address:

- Number of controller entries for device
- ASCII device mnemonic
- Device type code

Word 4 - Controller Definition Table (CDT) address:

- Controller I/O class
- Number of devices on controller
- Device type code
- Interrupt priority level
- Logical channel number
- Logical subchannel number of first device
- Address of interrupt handler
- Interrupt vector location
- Controller definition flags

Word 5 - Shared Memory Table (SMT) address (Applies to memory partitions and shared images):

Extractable information:

- Starting map register number
- Memory type
- Starting page number
- Total number of pages
- Number of map image descriptors
- Address of map image descriptor list

Word 6 - File Pointer Table (FPT) address:

- Logical file code associated with resource

Word 7 - Mounted Volume Table (MVT) address (Applies only to volume resources):
Volume name
Current number of users of volume
Volume definition flags
Root directory resource ID
Number of descriptors available on volume
Number of allocation units available
Volume access restrictions

Notes:

1. A value of zero returned in any word of the parameter description area implies the corresponding structure does not apply to the resource for which the inquiry was made. For example, only words zero and five apply for memory partitions or shared images.

2. For volume resources, words two through four pertain to the device upon which the volume is mounted.

3. The various system structure contents are described in the MPX-32 Technical Manual.

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
5	Shared memory table entry not found for partition
10	Illegal address range
29	Logical file code not assigned
30	Invalid allocation index

Wait Conditions

None

7.2.69 M_INT - Activate Task Interrupt

The M_INT service allows the calling task to cause the previously declared break/task interrupt receiver routine of the specified task to be entered.

The nonbase mode equivalent service is M.INT.

Entry Conditions

Calling Sequence:

```
M_INT     [TASK=] task

(or)

ZR    R6          }     (or)  LD R6,taskname
LW    R7,taskno   }
SVC   1,X'6F'  (or)  M_CALL  H.REXS,47
```

task — the address of a doubleword containing the name of the task or 0 in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of 0 specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN    6,7
```

Registers:

R6 — Unchanged if R7 is zero. If R7 is not zero, bit 0 of R6 is one if the specified task was not set up to receive a pseudointerrupt; otherwise bit 0 is zero. Bits 1-31 of R6 are zero in all cases.

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file. Otherwise, contains the task number.

Abort Cases:

None

Output Messages:

None

7.2.70 M_IPUBS - Set IPU Bias

The M_IPUBS service allows the user to dynamically change the IPU bias state for the current task.

The nonbase mode equivalent service is M.IPUBS.

Entry Conditions

Calling Sequence:

```
M_IPUBS [BIAS=] bias
```

or

```
LW      R7,bias
SVC     2,X'5B'   (or)   M_CALL  H.REXS,82
```

bias is the IPU bias state requested as follows:

Value	Description
0	Nonbiased task, for example, can be executed by either the CPU or IPU
1	CPU only, for example, can be executed only by the CPU
2	IPU bias, for example, can be executed by either the CPU or IPU but is given priority status by the IPU

Exit Conditions

Return Sequence:

```
M.RTRN    R6,R7
```

Registers:

R6 Contains execution status as follows:

Value	Description
0	Normal return
1	IPU is not configured in the system
2	IPU is currently marked off-line

R7 contains the IPU bias state of the task before this service was issued as follows:

Value	Description
0	Nonbiased task
1	CPU only
2	IPU bias

Abort Cases:

None

Output Messages:

None

7.2.71 M_LIMITS - Get Base Mode Task Address Limits

The M_LIMITS service is used to return the current task limits of a specified base mode task into sequential word locations of a specified buffer until all the limits are provided or the buffer is exhausted. The values returned are the TSA base address, the stack lower bound, the stack upper bound, the read-only section low address, and the read/write section low address.

Entry Conditions

Calling Sequence:

```
M_LIMITS  [BUFFER=] addr , [NUMBER=] num

(or)

LA        R1, addr
LI        R4, num
SVC       2,X'5D'   (or)   M_CALL   H.REXS,84
```

addr — is the word bounded address of a buffer where the task address limits are to be stored

num — is the number of bytes allocated for the buffer

Registers:

R1	Contains addr
R4	Contains num

Exit Conditions

Return Sequence:

Normal Return

```
M.RTRN
```

Abnormal Return

```
M.RTRN    R7
```

Registers:

CC1	Error condition
R7	Contains error condition (hexadecimal equivalent) as follows:

Value	Description
257	Destination buffer is in the operating system
258	Destination buffer is in the TSA
259	Invalid destination buffer address
260	Buffer length not a word multiple

7.2.72 M_LOCK - Set Exclusive Resource Lock

The M_LOCK service allows a task to obtain exclusive allocation of a resource, as though it were nonshareable, for as long as the lock is owned. The resource must have been previously allocated (included for memory partitions), and is identified by either a logical file code (defined when the resource was assigned) or an allocation index (obtained when the resource was assigned or by a resource inquiry). The task may request immediate denial if the lock is not available, or wait for an indefinite or specified period of time. An exclusive resource lock may be obtained for any allocated resource that is not being shared by multiple tasks at the time of the call to this service.

The nonbase mode equivalent service is M.LOCK.

Entry Conditions

Calling Sequence:

```
M_LOCK  [[ARGA=] arga][ ,[CNP=] cnpaddr]

(or)

LW          R5,arga
LA          R7,cnpaddr  (or)  ZR   R7
SVC         2,X'44'  (or)  M_CALL   H.REMM,23
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains a LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
29	Specified LFC is not assigned
30	Invalid allocation index
38	Timeout occurred while waiting to become lock owner
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task
51	Resource is allocated to another task

Wait Conditions

The task is placed in a wait state, as appropriate, if specified with the CNP.

7.2.73 M_LOGR - Log Resource or Directory

The M_LOGR service provides the base mode task with a convenient interface to locate the directory entry and resource descriptor for a single resource or for all the resources defined in a specified directory.

The caller must specify a resource specification in the Resource Logging Block (RLB). The log service will evaluate the resource specification and determine whether to log a single resource or all the resources defined in a directory. Some resource specifications are ambiguous and require the caller to specify additional information so the type of log function requested can be determined. Each resource specification is described in more detail below.

In order to log all the resources defined in a specified directory, the M_LOGR service must be called repeatedly until the last resource in the directory has been logged. The user must set bit 0 to zero in RLB.TYPE to indicate the first call. The operating system automatically changes the contents of bit 0 to 1 to indicate recall. Once all resources on the directory are logged, the operating system automatically changes bit 0 back to zero to indicate all resources have been logged.

Note: The M_LOGR system service does not search the Memory Resource Descriptor Table (MDT) for resource descriptors.

Resource Specifications for Pathnames

The caller can specify any valid pathname that is recognized by the Volume Management Module. The log service recognizes all valid pathname variations. However, some pathnames are ambiguous within the context of this service and require special considerations in order for the service to function with the expected results.

Specifically, pathnames that end with a directory specification are interpreted to mean log the contents of the directory. Directories can be logged as resources utilizing one of two methods. Method 1 furnishes a pathname that specifies the directory as a resource. This specification is not ambiguous. Method 2 furnishes a pathname that ends with a directory specification. This type of pathname is ambiguous and requires special handling.

Example of Method 1

```
@VOLUME(DIRECTORY)RESOURCE
```

This type of pathname always logs the directory entry and resource descriptor for the specified resource.

Examples of Method 2

```
@VOLUME(DIRECTORY)
```

This type of pathname normally means log the contents of the specified directory. The meaning of this pathname can be changed by setting the log single flag (RLB.LS) bit in the RLB flag word (RLB.INT). When the RLB.LS flag is set, the directory entry and resource descriptor for the specified directory are returned.

@VOLUME DIRECTORY

This type of pathname means log the specified directory. The directory entry and resource descriptor for the specified directory are returned.

Resource Specifications for Pathname Blocks

Pathname blocks are processed in the same manner as pathnames.

Resource Specifications for Resource Identifier

When a resource identifier (RID) is furnished, the log service assumes the indicated resource is a directory and attempts to log the indicated resource as a directory.

Resource Specifications for Logical File Code (LFC), FCB Address, or Allocation Index

When this type of resource specification is provided the log service makes the following assumptions.

- The implied File Control Block (FCB) is assigned to a directory.
- The implied File Control Block (FCB) is opened.
- The buffer address contained in the FCB is the buffer to be used by the log service for locating directory entries.
- The transfer quantity contained in the FCB is the maximum size of the directory entry buffer.
- The FCB must be an extended FCB and must be opened in random access mode.
- The log service assumes the buffer is empty on the initial call and will position to the beginning of the directory and prime the supplied buffer. The directory will not be read again until it is exhausted.

The caller should assign the directory in read mode so the directory can be searched by other users as it is being logged.

The nonbase mode equivalent service is M.LOGR.

Entry Conditions

Calling Sequence:

```
M_LOGR  [RLBADDR=] addr1 [ , [CNPADDR=] addr2]

(or)

LA      R2, addr1
LA      R7, addr2 (or) ZR  R7
SVC     2,X'29'   (or)  M_CALL  H.VOMM,10
```

addr1 is the address of the Resource Logging Block (RLB)

addr2 is a CNP address or zero if not supplied

Registers:

R2	contains addr1
R7	contains addr2; otherwise, register seven is zero

RLB Structure on Initial Call

Word		
0	PN vector or RID address or zero (RLB.TGT). See Note 1.	
1	192-word buffer address or zero (RLB.BUFA). See Note 2.	
2	Reserved for system use	
3	Reserved for system use	
4	Type (RLB.TYPE). See Note 3.	Zero (RLB.BOFF). See Note 3.
5	Length. See Note 4.	Directory return buffer address (RLB.DIRA). See Note 4.
6	User FCB address or zero (RLB.FCB). See Note 5.	
7	Flags (RLB.INT). See Note 6.	

Notes:

1. If the PN vector (length and address) specifies a resource, only one item will be logged. If it does not end with a resource, but with a directory, the entire directory may be logged by repeated calls. A call by RID will imply the RID is for a directory and all entries may be logged. A value of zero implies the entire contents of the current working directory.

2. If this field is zero, the RD will not be returned.

3. The type value should be zero if the call is by PN vector (length and address) or zero to indicate working directory. Type should be one to indicate a call by RID. If all resources in a directory are to be logged, bit 0 of RLB.TYPE must be set to zero to indicate the first call.

4. This word contains the address of a buffer and its length in words. The buffer may be up to 16 words long. The log service will place the first n words of the logged directory entry into this buffer. This provides the user access to the file name and other attributes that exist only in the directory entry.

5. This service uses the system FCB by default. There is a potential for phasing problems as the directory to be logged must be deassigned between calls if multiple entries are desired. In many cases, the impact of having an entry deleted just after it has been logged, or having an entry appear after that spot in the directory has been scanned, will be small or nonexistent. In other cases, such as saving files in a directory, it may be major. To prevent these problems, the user may provide the address of a FCB that will be used to hold the directory while logging occurs.

6. Bits in this word are assigned as follows:

Bit	Description
0-1	Reserved
2	If set, directory entry and resource descriptor for specified directory are returned (RLB.LS)
3	Reserved
4	Used on return to indicate if resource was located (see RLB Structure on Return)
5-31	Reserved

Exit Conditions

(with CNP)	(without CNP)
M.RTNA (CC1 set)	M.RTRN R7

Registers:

R7 Return status if a CNP is not supplied; otherwise CNP address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

RLB Structure on Return

Word			
	0	PN vector or RID address or zero (RLB.TGT)	
	1	192-word buffer address or zero (RLB.BUFA)	
	2	MVTE address (RLB.MVTE). See Note 1.	
	3	Disc address of directory RD (RLB.RDAD) See Note 1.	
	4	Type (RLB.TYPE) See Note 2.	Byte offset of entry (RLB.BOFF).
	5	Length	Directory return buffer address (RLB.DIRA)
	6	User FCB address or zero (RLB.FCB)	
	7	Flags (RLB.INT). See Note 3.	

Notes:

1. When all resources in a directory are to be logged, RLB.MVTE and RLB.RDAD are used by the operating system as input after the first call.

2. The operating system automatically changes the contents of bit 0 in RLB.TYPE as follows:

Value	Description
0	All resources in the directory have been logged; do not recall this service
1	Recall this service and log the next resource in the directory

3. Bits in this word are assigned as follows:

Bit	Contents
0-3	Reserved
4	Contains one of the following values: 0 = resource was not located 1 = resource was located
5-31	Reserved

7.2.74 M_MEM - Create Memory Partition

The M_MEM service creates permanent memory partition definitions. Permanent memory partition definitions are given names in directories and remain known to the operating system until explicitly deleted.

Memory partition definitions are the mechanism used by the operating system to establish the relationship of named globally accessible areas of memory to the tasks that require them.

This service allocates a resource descriptor and defines the memory requirements for the partition. Next, the attributes of the partition are recorded in the resource descriptor. As a final step, the name of the partition is established in the indicated directory.

When a directory entry is established, the directory entry is linked to the resource descriptor for the partition. This link makes the relationship of the name of the partition to the other attributes of the partition. Typical partition attributes are:

- The partition's name
- The partition's resource identifier (RID)
- The partition's protection attributes
- The partition's management attributes
- The partition's memory requirements

The nonbase mode equivalent service is M.MEM.

Entry Conditions

Calling Sequence:

```
M_MEM   [ [PNADDR=] pnaddr] [ , [RCB=] rcbaddr] [ , [CNP=] cnpaddr]
(or)
LW      R1,pnaddr
LA      R2,rcbaddr
LA      R7,cnpaddr  (or)  ZR   R7
SVC     2,X'22'  (or)  M_CALL   H.VOMM,3
```

pnaddr contains a PN vector or PNB vector

rcbaddr is the RCB address (required)

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.75 M_MOD - Modify Descriptor

The M.MOD service allows the owner of a resource to change or alter the protection or other resource management attributes of a resource. The owner can restrict or allow attributes with this mechanism.

Because of complications or confusion that can arise when certain attributes are changed, this service limits the information in a descriptor that can be changed. Information such as the volume space occupied by the resource cannot be changed as this would allow the caller to violate the integrity of the volume on which the resource resides.

The caller is allowed the following modifications:

- The protection fields of the descriptor
- The accounting fields of the descriptor
- The extension attribute fields of the descriptor
- User data field of the descriptor (words 160 through 175)
- The shared image field of the descriptor

This service is the first part of a two step operation. The caller is required to read the resource descriptor into memory in order to modify it. Once read into memory, the resource descriptor is locked (for example, protected from access) until the caller writes the modified descriptor back to the volume with the Rewrite Descriptor (M_REWRIT) service. The caller must issue the rewrite before modifying another descriptor.

Only the resource owner or the System Administrator are allowed to modify a resource descriptor. The format of the descriptor and the type of data to be modified must be known by the modifier.

The nonbase mode equivalent service is M.MOD.

Entry Conditions

Calling Sequence:

```
M_MOD     [ [PNADDR=] pnaddr] [ , [RD=] rdaddr] [ , [CNP=] cnpaddr]

(or)

LW        R1,pnaddr
LA        R6,rdaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2A' (or) M_CALL  H.VOMM,11
```

pnaddr contains a PN vector, PNB vector, or RID vector

rdaddr is a RD buffer address (doubleword bounded, 192W length)

cnpaddr is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.76 M_MODU - Modify Descriptor User Area

The M_MODU service allows users with write, update, append, or modify access to a resource to change or alter the user area of the resource descriptor of that resource.

Although the user is allowed to change all fields in the user area of the resource descriptor, words 176 through 190 are used by some utilities. Word 191 of the resource descriptor is a reserved location, and any changes to this word are ignored. For these reasons, the user is advised to use only words 160 through 175 of the resource descriptor.

This service is the first part of a two step operation. The caller is required to read the user area of the resource descriptor into memory in order to modify it. Once read into memory, the resource descriptor is locked (for example, protected from access) until the caller writes the modified user area back to the volume with the Rewrite Descriptor User Area (M_REWRTU) service. The caller must issue the rewrite before modifying another descriptor or descriptor user area.

The nonbase mode equivalent service is M.MODU.

Entry Conditions

Calling Sequence:

```
M_MODU [PNADDR=] pnaddr [ , [UAADDR=] uaaddr] [ , [CNP=] cnpaddr]

(or)

LW    R1,pnaddr
LA    R6,uaaddr
LA    R7,cnpaddr
SVC   2,X'31'   (or)   M_CALL   H.VOMM,26
```

pnaddr — contains a PN vector, PNB vector, or RID vector

uaaddr — is a user area buffer address (doubleword bounded, 32W length)

cnpaddr — is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.77 M_MOUNT - Mount Volume

The M_MOUNT service establishes a process as a user of a volume. If the volume has not been previously mounted, the operator is notified to mount the volume, and a Mounted Volume Table Entry (MVTE) is created for the volume. This entry remains memory resident as long as there are established users of the volume. A Volume Assignment Table (VAT) entry is allocated within the user's TSA for nonpublic volumes, provided the requested usage classification is compatible. Mount requests for an already mounted volume are ignored.

The nonbase mode equivalent service is M.MOUNT.

Entry Conditions

Calling Sequence:

```
M_MOUNT  [RRSADDR=] addr1 [ , [CNPADDR=] addr2]

(or)

LA       R1, addr1
LA       R7, addr2     (or)    ZR       R7
SVC      2,X'49'       (or)    M_CALL   H.REMM,17
```

addr1 is the address of a RRS entry (Type 9). See Section 5.2.4.1 for a description of RRS entries.

addr2 is the address of a Caller Notification Packet (CNP) if notification is desired. Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Registers:

R1	Contains addr1
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
1	Invalid volume name
2	Volume use not allowed to this user
6	Volume Assignment Table (VAT) space not available
8	Unrecoverable I/O error to volume
11	Invalid RRS entry
13	Mount device not in system
18	Invalid mount device specified
20	Unable to initialize volume (volume unsafe)
21	J.MOUNT run request failed
34	System Administrator attribute required to mount public volume
38	Time out occurred while waiting for resource
53	Mount device unavailable
55	Allocated Resource Table (ART) space not available for mount device
57	Unable to mount volume for requested usage
59	Mounted Volume Table (MVT) space not available
73	File overlap occurred. Check system console.

Wait Conditions

When the volume is unavailable (status values 50-63), the task is placed in a wait state, as appropriate.

7.2.78 M_MOVE - Move Data to User Address

The M_MOVE service is used to move an arbitrary number of bytes from a location in the user's logical address space to a location in the user's read only memory section or the user's read/write memory section. This service can be used to set traps or modify data in the user's memory sections. This service cannot be used in shared read-only sections.

The nonbase mode equivalent service is M.MOVE.

Entry Conditions

Calling Sequence:

```
M_MOVE [BUFFER=] addr1 , [DESTADDR=] addr2 , [NUMBER=] num

(or)

LA    R1, addr1
LA    R2, addr2
LI    R4, num
SVC   2,X'62' (or) M_CALL H.REXS,89
```

addr1 is the byte address of the buffer to be moved

addr2 is the destination byte address

num is the number of bytes to be moved

Registers:

R1	Contains addr1
R2	Contains addr2
R4	Contains num

Exit Conditions

Return Sequence:

Normal Return

```
M.RTRN
```

Abnormal Return

```
M.RTRN    R7
```

Registers:

CC1 set — Error condition

R7 — Contains error condition, hexadecimal equivalent as follows:

Value	Description
256	Invalid source buffer address
257	Destination buffer is in the operating system
258	Destination buffer is in the TSA
259	Invalid destination buffer address

7.2.79 M_MYID - Get Task Number

The M_MYID service allows the user to obtain status on the currently executing task.

The nonbase mode equivalent service is M.MYID.

Entry Conditions

Calling Sequence:

```
M_MYID      [PBADDR=] pbaddr
(or)
ZR          R5
SBR         R5,0
LA          R7,pbaddr
SVC         1,X'64' (or) M_CALL  H.REXS,32
```

pbaddr is the logical word address of the first location of a parameter block formatted as follows:

Word	Contents
0	Task activation sequence number
1-2	Task load module name
3-4	Owner name
5-6	Pseudonym
7-8	Current working directory, truncated to the first eight characters
9	Reserved
10	Scheduling flags (DQE.USHF)

Exit Conditions

Return Sequence:

```
M.RTRN      5
```

Registers:

CC1 set	Invalid parameter block address	
	R5,R7	Unchanged

Abort Cases:

RX32	Invalid DQE address

Output Message:

None

7.2.80 M_OPENR - Open Resource

The M_OPENR service prepares a resource for logical I/O and defines the intended access mode for subsequent operations on the resource. Protection is provided for both the requestor and the resource against indiscriminate access. If appropriate, additional FAT information is posted at this time. A blocking buffer may be allocated if not previously specified, explicitly or implicitly, during allocation of the resource. However, if a user-supplied buffer is specified in the FCB, it will be used and any previously allocated blocking buffer will be released. A mount message is issued as a result of this function when the I/O is to be performed to a device associated with unformatted media, and the message has not been inhibited by user request or previous open on the resource by another user. An open request to a resource that is already opened in the same access mode will be ignored.

The nonbase mode equivalent service is M.OPENR.

Entry Conditions

Calling Sequence:

```
M_OPENR  [FCBADDR=] addr1 [ , [CNPADDR=] addr2]

(or)

LA       R1, addr1
LA       R7, addr2      (or)    ZR       R7
SVC      2,X'42'        (or)    M_CALL   H.REMM,21
```

addr1 is the address of a File Control Block (FCB)

addr2 is the address of a Caller Notification Packet (CNP) if notification is desired. The applicable items of the CNP for this function are time-out value, abnormal return address, options field, and status field.

The options field describes the access and usage with the following interpretation:

byte 0 contains an integer value representing the specific access for which the resource is being opened. The following values are valid:

Value	Description
0	Open for default access
1	Open for read
2	Open for write (resource redefined)
3	Open for modify
4	Open for update
5	Open for append
6-255	Reserved

byte 1 indicates the usage under which the resource is to be opened with the following bit significance when set:

Bit	Meaning if Set
0	Open for explicit shared use
1	Open for exclusive use
2	Open in unblocked mode (overrides any specification made at resource assignment)
3	Open in blocked mode (overrides any specification made at resource assignment)
4	Resource data blocked. If the file is actually written to in any access mode (append, modify, update, write), the data will be recorded as blocked in the resource descriptor at the time the file is closed, regardless of whether or not the I/O was actually performed in blocked mode.
5-7	Reserved

Registers:

R1 Contains addr1
R7 Contains addr2; otherwise, zero

Restrictions:

1. An error condition is returned if an invalid integer value is present in byte 0.

2. Only one of bits 0 and 1 in byte 1 may be set. If set, any usage specified at the time the resource was assigned is overridden. This may result in a denial condition if the usage specified at open differs from that specified at assignment.

Defaults:

If a CNP is not supplied or the specification in the options field is zero:

1. the resource is opened for read access (volume resource) or update access (device) unless only a specific access mode was allowed at assignment, in which case the resource will be opened for that access

2. the usage will be implicit shared or that specified at resource allocation, whichever is appropriate.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
2	Specified access mode not allowed
4	Blocking buffer not available
9	Invalid usage specification
25	Random access not allowed for this access mode
27	Resource already opened in a different access mode
28	Invalid access specification
29	No LFC which matches FCB file code
38	Time out occurred waiting for resource
46	Unable to obtain resource descriptor lock (multiprocessor only)
54	Unable to allocate resource for specified usage

Wait Conditions

If an access/usage specification is made at open which changes the nature of the resource allocation, the task may be placed in a wait state, as appropriate, if specified in the CNP.

7.2.81 M_OPTIONWORD - Task Option Word Inquiry

The M_OPTIONWORD service provides the caller with the 32-bit task option word (same as program option word).

This service may be executed by the IPU.

The nonbase mode equivalent service is M.PGOW.

Entry Conditions

Calling Sequence:

```
M_OPTIONWORD

(or)

SVC     1,X'4C'   (or)   M_CALL   H.REXS,24
```

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

```
R7          Contains the 32-bit task option word
```

Abort Cases:

None

Output Messages:

None

7.2.82 M_PNAMB - Convert Pathname to Pathname Block

The M_PNAMB service converts a pathname to a form that can be more easily analyzed by software. In most cases, utility programs use this to syntax check a pathname that has been input on a directive line.

When called, this service parses the input pathname. If any errors are detected in the pathname syntax, this service is terminated and register 1 is updated to indicate the point where the error was detected.

The nonbase mode equivalent service is M.PNAMB.

Entry Conditions

Calling Sequence:

```
M_PNAMB   [ [PNADDR=] pnaddr] , [PNB=] pnbaddr] [ , [CNP=] cnpaddr]
(or)
LW        R1,pnaddr
LW        R4,pnbaddr
LA        R7,cnpaddr (or) ZR  R7
SVC       2,X'2E' (or) M_CALL  H.VOMM,15
```

pnaddr contains a PN vector

pnbaddr contains a PNB vector

cnpaddr is a CNP address or zero if CNP not supplied.

Applicable portions of the CNP for this function are time-out value, abnormal return address, option field, and status field. The options field of the CNP has the following interpretation:

Byte 0 is reserved

Byte 1 has the following significance when set:

Bit	Meaning if Set
0-6	Reserved
7	Parsing of the pathname includes only the volume and directory portions of the supplied pathname. This bit is usually set by J.TSM.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R1,R4	M.RTRN R1,R4
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R1	Address of first PN character not processed and remaining length
R4	PNB address and actual PNB length
R7	Return status if a CNP is not supplied; otherwise unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.83 M_PRIL - Change Priority Level

The M_PRIL service is provided to the privileged caller who wishes to dynamically alter the priority level of the specified task. Valid priority levels for real-time tasks are 1-54 inclusive. Valid priority levels for time distribution tasks are 55-64 inclusive. A real-time task cannot be changed to a time distribution priority level and a time distribution task cannot be changed to a real-time priority level. I/O continues to operate at base priority level of the cataloged task.

The nonbase mode equivalent service is M.PRIL.

Entry Conditions

Calling Sequence:

```
M_PRIL    [TASK=] task , [PRTY=] priority

(or)

LW        R5,priority
ZR        R6,=0      }      (or)  LD R6,taskname
LW        R7,taskno  }
SVC       1,X'4A'  (or)  M_CALL   H.REXS,9
```

task — the address of a doubleword containing the name of the task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

priority — is the priority level to be assigned to the task (1-54 for a real-time task; 55-64 for a time-distribution task)

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found; otherwise, contains the task number

Abort Cases:

RX06 — Specified priority not in 1-64 range

Output Messages:

None

7.2.84 M_PRIVMODE - Reinstate Privilege Mode to Privilege Task

The M_PRIVMODE service allows a task that was linked as privileged to return to a privileged status. See M_UNPRIVMODE service.

The nonbase mode equivalent service is M.PRIV.

Entry Conditions

Calling Sequence:

```
M_PRIVMODE

(or)

SVC     2,X'57'   (or)   M_CALL   H.REXS,78
```

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

```
None
CC1 set     Successful operation
```

Abort Cases:

None

Output Messages:

None

7.2.85 M_PTSK - Parameter Task Activation

The M_PTSK service is provided to the privileged caller who wishes to override specific task parameters (contained in the load module or executable image preamble) during activation the task name, optional resource requirements, and optional pseudonym are specified to the service call. When a task name is supplied in Words 2 and 3 of the Parameter Task Activation Block, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied.

The nonbase mode equivalent service is M.PTSK.

Entry Conditions

Calling Sequence:

```
M_PTSK      [ACTADR=] actaddr [ , [PSB=] psbaddr]

(or)

LA          R1,actaddr
LA          R2,psbaddr (or) ZR   R2
SVC         1,X'5F' (or) M_CALL  H.REXS,40
```

actaddr is the logical word address of the first location of an activation parameter block formatted as shown below:

Parameter Block

(Words 0-12 are required. The RRS entries (words 13-n) are optional as indicated by byte 1.)

Word 0	byte 0	Bit	Contents
		1	Reserved
		2	Terminal task
		3	Batch task
		4	Debug overlay required
		5	RTM resident (ESTABLISH)
		6	Command file active
		7	SLO assigned to SYC
	byte 1		Number of resource requirements or zero if same as summary entries in the load module or executable image preamble
	byte 2		Memory requirement = number of 512 word pages exclusive of TSA or zero if memory requirements are to be taken from the preamble

byte 3 — Memory class (ASCII E, H or S) or zero if memory class to be taken from the preamble

(or)

If the memory class is to be taken from the preamble, the caller has the option of specifying the task's logical address space in this field as follows:

Bits	Contents
0-3	Hexadecimal value 0 through F representing the task's logical address space in megabytes where zero is 1MB and F is 16MB
4-7	Zero

Word 1 — byte 0 is the number of blocking buffers required or zero if same as the preamble

byte 1 is the number of FAT/FPT pairs (files) to be reserved or zero if same as the preamble

byte 2 is the priority level of task being activated or zero if same as the preamble

byte 3 is the number of segment definition areas to be reserved for extendible files or zero if same as the preamble

Word 2 — Contains the first 4 bytes of the ASCII character left-justified, blank-filled load module or executable image name, if word 3 contains the last 4 bytes; else zero

Word 3 — Contains the last 4 bytes of the ASCII character left-justified, blank-filled load module or executable image name, if word 2 contains the first 4 bytes; else contains a pathname vector or RID vector if word 2 is zero

Word 4,5 — Pseudonym - one- to eight-character ASCII left-justified, blank-filled pseudonym to be associated with task or zero if no pseudonym is desired. Pseudonym's are intended to be unique - the responsibility for uniqueness rests with the caller.

Word 6,7 — Owner Name - one- to eight-character ASCII left-justified, blank-filled owner name to be associated with task or zero if task to default to current owner name. Valid only when task has System Administrator attribute.

Word 8,9 — Project Name - one-to eight-character ASCII left-justified, blank-filled project name to be associated with files referenced by this task or zero if same as LMIT

Word 10 — byte 0 is the number of Volume Assignment Table (VAT) entries to be reserved for dynamic mount requests or zero if same as the preamble

bytes 1,2 and 3 are reserved

Word 11 Task Option Word - contains the initial value of the task option word or zero

Word 12 Task Status Word - contains the initial value of the task status word or zero

Words 13-n (optional) Resource Requirement Summary List - each entry contains a variable length RRS. The RRS list is limited to a maximum of 384 words. Each entry is compared with the RRS entries in the LMIT. If the logical file code currently exists, the specified LFC assignment will override the cataloged assignment, otherwise it will be treated as an additional requirement (merged). If MPX-32 Release 1.x format of the RRS is specified, it is converted to the format acceptable for assignment processing by the Resource Management Module (H.REMM). See MPX-32 Release 1.x Technical Manual for format of the RRS.

psbaddr is the logical address of the Parameter Send Block (PSB) or zero if no parameters are to be passed. See Chapter 2 for PSB description.

Exit Conditions

Return Sequence:

M.RTRN 6,7

Registers:

R0 Destroyed
R6 Equals zero if the service could be performed
R7 Contains the task number of task activated by this service

(or)

R0 Destroyed
R6 Equals one if invalid attempt to multicopy a unique task
R7 Task number of existing task with same name

(or)

R0 Destroyed

R6 Value	Description
2	If file specified in words two and three of the PSB not in directory
3	Unable to allocate file specified in words two and three of the PSB
4	If file is not a valid load module or executable image
5	If DQE is not available
6	If read error on resource descriptor
7	If read error on load module
8	Insufficient logical/physical address space for task activation
9	Calling task is unprivileged
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address

14	Memory pool unavailable
15	Destination task receiver queue full
16	Invalid PSB address
17	RRS list exceeds 384 words
18	Invalid RRS entry in parameter block

Abort Cases:

None

Output Messages:

None

7.2.86 M_PUTCTX - Put User Context

The M_PUTCTX service overwrites the most recent user context previously saved in the TSA stack area. If control is transferred to the user before the context is stored again, this will be the context used. The values are loaded as words from the specified address up to the last even word allocated in the block or until all context is provided.

If an attempt is made to modify the most significant byte of Program Status Doubleword (PSD) one, only the condition code values are changed.

Entry Conditions

Calling Sequence:

```
M_PUTCTX  [BUFFER=] addr , [NUMBER=] num

(or)

LA        R1, addr
LI        R4, num
SVC       2,X'71'   (or)   M_CALL   H.EXEC,42
```

addr is a word bounded logical address in memory to return the context from

num is the number of bytes allocated for the context block

Registers:

R1	Contains addr
R4	Contains num

Exit Conditions

Normal Return Sequence:

```
M.RTRN
```

Abnormal Return Sequence:

```
M.RTRN    R7
```

Registers:

CC1 set	Error condition
R7	Contains error condition (hexadecimal equivalent) as follows:

Value	Description
256	Invalid source buffer address
260	Buffer length not a word multiple

7.2.87 M_QATIM - Acquire Current Date/Time in ASCII Format

The M_QATIM service acquires the system date and time in ASCII format. The date and time are returned in a four word buffer, the address of which is contained in the call. See Appendix H for time formats.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.QATIM.

Entry Conditions

Calling Sequence:

```
M_QATIM [TIMBUF=] addr

(or)

LA          R1,addr
ORMW        R1,=X'03000000'
SVC         2,X'50'  (or)  M_CALL   H.REXS,74
```

addr is the address of a four word buffer to contain the date and time

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

R1	Used by call - all others returned intact

Abort Cases:

RX13	Function code supplied to a date/time service is out of range

Output Messages:

None

7.2.88 M_RADDR - Get Real Physical Address

The M_RADDR service allows unprivileged tasks to determine the physical memory address associated with a given logical address.

The nonbase mode equivalent service is M.RADDR.

Entry Conditions

Calling Sequence:

```
M_RADDR [ [LOGADDR=] logicaladdr]

(or)

LA      R1, logicaladdr
SVC     1,X'0E'     or    M_CALL   H.REXS,90
```

logicaladdr is the logical address to be translated

Exit Conditions

Return Sequence:

Registers:

R7 Contains the physical address

Abort Cases:

None

Output Messages:

None

7.2.89 M_RCVR - Receive Message Link Address

The M_RCVR service allows the caller to establish the address of a routine to be entered for the purpose of receiving messages sent by other tasks.

The nonbase mode equivalent service is M.RCVR.

Entry Conditions

Calling Sequence:

```
M_RCVR    [RCVRADR=] recvaddr

(or)

LA        R7,recvaddr
SVC       1,X'6B' (or) M_CALL  H.REXS,43
```

recvaddr is the logical word address of the entry point of the receive message routine in the user's task

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 Contains zero if the receiver address was invalid, otherwise contains the receiver address

Abort Cases:

None

Output Messages:

None

7.2.90 M_READ - Read Record

The M_READ service performs the following functions:

- Provides special random access handling for disc files.
- Deblocks system files and blocked files.
- Reads one record into the buffer indicated by the Transfer Control Word (TCW) in the FCB.

The nonbase mode equivalent service is M.READ.

Entry Conditions

Calling Sequence:

```
M_READ  [FCBADDR=] addr

(or)

LA          R1, addr
SVC         1,X'31'   (or)   M_CALL H.IOCS,3
```

addr — is the FCB address. Appropriate transfer control parameters are defined in the TCW. See Section 5.9.1.2.

Registers:

R1 — Contains addr

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03	Nonprivileged user attempting transfer to a logical address outside legal boundaries
IO06	Invalid blocking buffer control cell for a system or blocked file
IO26	Read attempted for a system or blocked file while write in process
IO30	Illegal volume record. Either volume number or reel ID from volume record do not match FAT information
IO32	Second attempt to read a $ statement in an SYC file

Output Messages:

Dismount/mount messages if EOT and multivolume magnetic tape

7.2.91 M_READD - Read Descriptor

The M_READD service reads a resource descriptor for a specified resource. This service can examine the attributes of any volume resource. It is the responsibility of the caller to be familiar with the fields of the resource descriptor in order to determine the recorded information. It is recommended that this service is called by a user-supplied subroutine which acts as a common interface between application programs and this service. In this way, application programs are less sensitive to changes in organization and content of these data structures.

The nonbase mode equivalent service is M.LOC.

Entry Conditions

Calling Sequence:

```
M_READD [RESOURCE=] addr1 , [RDADDR=] addr2 [ , [CNPADDR=] addr3]

(or)

LW      R1, addr1
LA      R6, addr2
LA      R7, addr3     (or)    ZR      R7
SVC     2,X'2C'       (or)    M_CALL  H.VOMM,13
```

addr1 contains a PN vector, PNB vector, RID vector, LFC, or FCB address

addr2 is a resource descriptor buffer address; doubleword bounded, 192W length

addr3 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R6	Contains addr2
R7	Contains addr3; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN R4	M.RTRN R4
(or)	(or)
M.RTNA R2 (CC1 set)	M.RTRN R7,R2 (CC1 set)

Registers:

R2	Address of last PN item processed (abnormal return)
R4	MVTE address for specified volume
R7	Return status if a CNP is not supplied. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.92 M_RELP - Release Dual-ported Disc/Set Dual-channel ACM Mode

The M_RELP service applies to dual-port extended I/O discs and allows the privileged user to release a device from its reserved state. When issued to an ACM that has been SYSGENed as full-duplex, this service can be used to set the ACM from single-channel to dual-channel mode (applies to ACMs using the H.F8XIO handler only).

The nonbase mode equivalent service is M.RELP.

Entry Conditions

Calling Sequence:

```
M_RELP      [FCB=] fcb

(or)

LA          R1,fcb
SVC         1,X'27' (or) M_CALL  H.IOCS,27
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.93 M_RENAME - Rename File

The M_RENAME service changes the name of an existing permanent file. This service can move a file from one directory to another directory on the same volume.

When called, this service creates the new name of the file in the specified directory and then deletes the old name of the file from the specified directory.

The nonbase mode equivalent service is M.RENAM.

Entry Conditions

Calling Sequence:

```
M_RENAME  [FILEADDR=] addr1 , [PNADDR=] addr2  [ , [CNPADDR=] addr3]

(or)

LW        R1, addr1
LW        R2, adddr2
LA        R7, addr3     (or)     ZR        R7
SVC       2,X'2D'       (or)     M_CALL    H.VOMM,14
```

addr1 contains the old PN or PNB vector

addr2 contains the new PN or PNB vector

addr3 is a CNP address or zero if CNP not supplied

Registers:

R1	contains addr1
R2	contains addr2
R7	contains addr3; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA R7 (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.94 M_REPLACE - Replace Permanent File

The M_REPLACE service replaces the data contents of an existing permanent file with the data contents of an existing temporary file. At the completion of this service, the permanent file retains its original directory entry and resource descriptor.

This service is provided so utility programs can change the data contents of a file without changing any of the files other attributes. In other words, this service maintains the integrity of a file's resource identifier and therefore provides the fast file mechanism.

This service can be used on any permanent file. At the completion of this service, the temporary file is deallocated and deleted. An error conditon is returned if the permanent file is allocated to another at the time of the service call, and bit 0 of the CNP option field is not set.

This service should only be used on files with the fast access attribute. For files which do not have the fast access attribute, this same functionality can be accomplished by using the Delete Resource (M_DELETER) service followed by the Change Temporary File to Permanent File (M_TEMPFILETOPERM) service.

The nonbase mode equivalent service is M.REPLAC.

Entry Conditions

Calling Sequence:

```
M_REPLACE [RESOURCE=] addr1 , [PATH=] vector [ , [CNPADDR=] addr2]

(or)

LA      R1, addr1
LW      R2, vector
LA      R7, addr2     (or)    ZR        R7
SVC     2,X'30'       (or)    M_CALL    H.VOMM,23
```

addr1 is the FCB or LFC address of the temporary file

vector is the pathname vector of the permanent file

addr2 is a CNP address or zero if CNP is not supplied

Registers:

R1	Contains addr1
R2	Contains vector
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)		(without CNP)
M.RTRN		M.RTRN
(or)		(or)
M.RTNA	(CC1 set)	M.RTRN R7 (CC1 set)

Registers:

CC1 set	Error condition
R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.95 M_RESP - Reserve Dual-ported Disc/Set Single-channel ACM Mode

The M_RESP service applies to dual-port extended I/O discs and allows the privileged user to reserve a device to the requesting CPU until such time as a release (M_RELP) is issued. When issued to an ACM that has been SYSGENed as full-duplex, this service can reset the ACM from dual-channel to single-channel mode (applies to ACMs using the H.F8XIO handler only).

The nonbase mode equivalent service is M.RESP.

Entry Conditions

Calling Sequence:

```
M_RESP      [FCB=] fcb

(or)

LA          R1,fcb
SVC         1,X'26' (or) M_CALL  H.IOCS,24
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.96 M_REWIND - Rewind File

The M_REWIND service performs the following functions:

- Issues an end-of-file and purge if the file is a system or blocked file which is output active.
- For system and blocked files, initializes blocking buffer control cells for subsequent access.
- Rewinds file or device.

The nonbase mode equivalent service is M.RWND.

Entry Conditions

Calling Sequence:

```
M_REWIND [FCBADDR=] addr

(or)

LA          R1, addr
SVC         1,X'37'   (or)   M_CALL  H.IOCS,2
```

addr is the FCB address

Registers:

R1	Contains addr

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Rewind attempted on SYC file

7.2.97 M_REWRIT - Rewrite Descriptor

The M_REWRIT service writes a modified resource descriptor back to a volume and releases the modify lock on the descriptor. This is the last step of a two step operation (the first step is M_MOD).

When this service is invoked, the indicated resource descriptor is read into an internal buffer. The fields that are allowed to be modified are copied from the user supplied resource descriptor buffer to the appropriate areas of the internal buffer. Upon successful modification of the resource descriptor in the internal buffer, the resource descriptor is written to the correct location on the volume and the modify lock is released.

The nonbase mode equivalent service is M.REWRIT.

Entry Conditions

Calling Sequence:

```
M_REWRIT   [ [RD=] rdaddr] [ , [CNP=] cnpaddr]

(or)

LA         R6,rdaddr
LA         R7,cnpaddr (or) ZR  R7
SVC        2,X'2B' (or) M_CALL  H.VOMM,12
```

rdaddr — is the RD buffer address. This must be the same address supplied by the caller for use with the associated Modify Descriptor (H.VOMM,11) call; doubleword bounded, 192W length

cnpaddr — is a CNP address or zero if CNP not supplied

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.98 M_REWRTU - Rewrite Descriptor User Area

The M_REWRTU service writes a modified resource descriptor back to a volume and releases the modify lock on the descriptor. This is the last step of a two step operation, the first step is M_MODU.

When this service is invoked, the indicated resource descriptor is read into an internal buffer. The data from the buffer supplied by the user is then copied to the appropriate areas of the internal buffer. Upon successful modification of the resource descriptor in the internal buffer, the resource descriptor is written to the correct location on the volume and the modify lock is released.

The nonbase mode equivalent service is M.REWRTU.

Entry Conditions

Calling Sequence:

```
M_REWRTU [[UA=] uaaddr][ ,[CNP=] cnpaddr]
(or)

LA      R6,uaaddr
LA      R7,cnpaddr  (or)  ZR  R7
SVC     2,X'32'  (or)  M_CALL  H.VOMM,27
```

uaaddr — is the user area buffer address. This must be the same address supplied by the caller for use with the associated Modify Descriptor User Area (H.VOMM,26) call; doubleword bounded and 32W length.

cnpaddr — is a CNP address or zero if CNP not supplied.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise denial address. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.99 M_ROPL - Reset Option Lower

The M_ROPL service allows the calling task to reset the option lower bit. Use the M_SOPL (Set Option Lower) service to set the option lower bit.

The nonbase mode equivalent service is M.ROPL.

Entry Conditions

Calling Sequence:

```
M_ROPL

(or)

SVC     2,X'78' (or) M_CALL H.TSM,14
```

Exit Conditions

Return Sequence:

```
M.RTRN

(or)

M.RTRN  (CC1 set)
```

Registers:

CC1 set	Call caused the option lower bit to be reset

Abort cases:

None

Output Message:

None

7.2.100 M_RRES - Release Channel Reservation

If the specified channel has not been reserved by this task, the M_RRES service ignores the request to release the channel and returns to the task. If the channel has been reserved by this task, the channel reserve indication is removed from the CDT entry.

After releasing the reserved channel, if any requests had been queued while the channel was reserved, IOCS resumes I/O to the associated device.

This service is not applicable for extended I/O channels.

The nonbase mode equivalent service is M.RRES.

Entry Conditions

Calling Sequence:

```
M_RRES      [CHAN=] channel

(or)

LW          R1,channel
SVC         1,X'3B' (or) M_CALL  H.IOCS,13
```

channel specifies the channel number (hexadecimal). If LW, load bits 24 to 31 of register 1.

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.101 M_RSML - Resourcemark Lock

The M_RSML service is called to lock the specified resourcemark. It is used in conjunction with the unlock resourcemark service (M_RSMU) by tasks to synchronize access to a common resource. For further description, see Chapter 2.

The nonbase mode equivalent service is M.RSML.

Entry Conditions

Calling Sequence:

```
M_RSML    [LCKID=] lockid , [ [TOPT=] timev] [ , [POPT=] P]
(or)
LI        R4,timev
ZR        R5
[SBR      R5,0]
LI        R6,lockid
SVC       1,X'19' (or) M_CALL  H.REXS,62
```

lockid is a numeric resourcemark index value, 33 to 64 inclusive

timev is a numeric value which specifies action to be taken if the lock is already set and is owned by another task:

Value	Description
+1	Immediate denial return
0	Wait until this task is the lock owner (default)
-n	Wait until this task is the lock owner, or until n timer units have expired, whichever occurs first

P indicates that while this task is waiting to become lock owner, the swapping mode is to be set to swap this task only if a higher priority task is requesting memory space. Otherwise, the task is a swap candidate if any task is requesting memory.

Exit Conditions

Return Sequence: M.RTRN R7

Registers:

R7 zero if the request was accepted, otherwise contains a request denial code:

Value	Description
1	Lock index exceeds maximum range
2	Lock index is less than minimum range
3	Lock is owned by another task (and timev=+1)
4	Lock is owned by another task, timev=-n and n timer units have elapsed

Abort Cases: None

Output Messages: None

7.2.102 M_RSMU - Resourcemark Unlock

The M_RSMU service unlocks a resourcemark which has previously been locked by a call to the M_RSML service. If any other tasks are waiting to lock the specified resourcemark, the highest priority waiting task becomes the new lock owner.

The nonbase mode equivalent service is M.RSMU.

Entry Conditions

Calling Sequence:

```
M_RSMU    [LCKID=] lockid

(or)

LI        R6,lockid
SVC       1,X'1A' (or) M_CALL  H.REXS,63
```

lockid is a numeric resourcemark index value, 33 to 64 inclusive

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 zero if the request was accepted, otherwise contains a request denial code:

Value	Description
1	Lock index exceeds maximum range
2	Lock index is less than minimum range
3	Lock is not owned by this task

Abort Cases:

None

Output Messages:

None

7.2.103 M_RSRV - Reserve Channel

M_RSRV is a privileged service. If the task is unprivileged or the channel has been reserved previously by another task, this service makes a denial return. If the channel has not previously been reserved, the task number is stored in the CDT or UDT entry to mark the reservation. If any requests are currently queued for this channel, suspend is invoked until completion of any I/O currently in progress is complete. The standard handler is then disconnected from the Service Interrupt (SI) level. After reserving a channel, the task must connect its own handler to the SI dedicated location.

This service is not applicable for extended I/O channels.

The nonbase mode equivalent service is M.RSRV.

Entry Conditions

Calling Sequence:

```
M_RSRV      [CHAN=] channel , [DENADR=] denial

(or)

LW          R1,channel
LA          R7,denial
SVC         1,X'3A'  (or)  M_CALL  H.IOCS,12
```

channel — specifies the channel number (hexadecimal) in bits 24 to 32. If using LW, load channel number in register 1.

denial — is the user's denial return address

Exit Conditions

Return Sequence:

```
M.RTRN          Normal return

(or)

M.RTNA      7   Denial return
```

Registers:

None

Abort Cases:

```
IO14        Unprivileged user attempting to reserve channel
```

Output Messages:

None

7.2.104 M_SETERA - Set Exception Return Address

The M_SETERA service changes the destination address upon exit from an established exception handler. This service can only be called from within an exception handler established by the M_SETEXA system service.

Entry Conditions

Calling Sequence:

```
M_SETERA [TASK=] addr

         (or)

LA       R7, addr

SVC 2,X'79'  (or)  M_CALL H.REXS,81
```

addr is the address where execution continues upon exit from the handler

Register:

R7	Contains addr or zero if the execution is to continue from point of trap

Exit Conditions

Return Sequence:

```
M.RTRN R7
```

Registers:

Normal Return

R7	Contains previous value

Abnormal Return

CC1 set	Error condition
R7	Contains error condition (hexadecimal equivalent) as follows:

Value	Description
259	Invalid destination address

Abort Cases:

RX15	Attempt to set exception return address when arithmetic exception is not in progress

7.2.105 M_SETEXA - Set Exception Handler

The M_SETEXA service establishes a task exception handler, change the location of the current task exception handler, or delete the current task exception handler. See Chapter 2 for a description of the handler format for arithmetic exception handling, plus the information passed when an exception occurs, and the types of exceptions recognized.

Entry Conditions

Calling Sequence:

```
M_SETEXA  [TASKID=] addr

(or)

LA        R7, addr
SVC       2,X'5C'  (or)  M_CALL  H.REXS,83
```

addr is the task address of the exception handler

Registers:

R7 Contains addr

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

Normal Return

R7 Contains previous value

Abnormal Return

CC1 set Error condition

R7 Contains error condition (hexadecimal equivalent) as follows:

Value	Description
259	Invalid destination buffer address

7.2.106 M_SETS - Set User Status Word

The M_SETS service allows the calling task to modify any task's user status word. Along with the Test User Status Word service, this is one of the means provided by MPX-32 for task-to-task communication. The user status word resides in the CPU Dispatch Queue (DQE.USW) and has a value of zero until modified by this service. The user status word is removed from the queue, modified as specified, and replaced in the queue.

The nonbase mode equivalent service is M.SETS.

Entry Conditions

Calling Sequence:

```
M_SETS    [FUNCT=] function , [STATW=] statusw [ , [TASK=] task]
(or)

LD        R4,task
LI        R6,function
LW        R7,statusw
SVC       1,X'48' (or) M_CALL   H.REXS,7
```

function — STF (1), RSF (2), STC (3), or INC (4) specify the type of modification to be performed and are set flag, reset flag, set counter, and increment counter respectively. If using the macro call, specify the alphabetic code. If loading registers, specify the corresponding numeric.

statusw — contains a function parameter specific to function codes as follows:

Value	Description
1	Bit position in the status word to be set (1 to 31)
2	Bit position in the status word to be reset (1 to 31)
3	Value to which the status word is to be set
4	Value by which the status word is to be incremented

task — is the address of a doubleword containing the name of the task, or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero or omission of the argument specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN    5
```

Registers:

R5 — Bit 0 set if the specified task was not found in the dispatch queue or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by M.KEY file; otherwise, register 5 is zero.

Abort Cases:

RX05 — Invalid function code specification

Output Messages:

None

7.2.107 M_SETSYNC - Set Synchronous Resource Lock

The M_SETSYNC service is used in conjunction with the Release Synchronous Lock service for resource gating of explicitly shared resources where there is no automatic synchronization performed by the system. The mechanism allows a task to obtain synchronized access to a resource that has been concurrently allocated to multiple tasks. A synchronization lock can be obtained for any resource, provided it has been previously allocated (included for memory partitions) by the calling task. Unlike an exclusive lock, the synchronous lock does not prevent other tasks from allocating the resource in explicit shared mode. It is the sharing tasks' responsibility to synchronize access by cooperative use of the synchronous lock services. The resource is identified by either a logical file code, defined when the resource was assigned, or an allocation index, obtained when the resource was assigned or by a resource inquiry. If the synchronization lock is not available, the calling task can obtain an immediate denial return, or wait for an indefinite or specified period of time.

The nonbase mode equivalent service is M.SETSYNC.

Entry Conditions

Calling Sequence:

```
M_SETSYNC [ARGA=] arga [ , [CNP=] cnpaddr]

(or)

LW      R5,arga
LA      R7,cnpaddr  (or)  ZR   R7
SVC     2,X'46'  (or)  M_CALL   H.REMM,25
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are time-out value, abnormal return address, and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in R7 or the status field of the CNP:

Value	Description
29	Specified LFC was not assigned by this task
30	Invalid allocation index
38	Timeout occurred while waiting to become lock owner
46	Unable to obtain resource descriptor lock (multiprocessor only)
50	Resource is locked by another task

Wait Conditions

The task is placed in a wait state, as appropriate, if specified by the CNP.

7.2.108 M_SETT - Create Timer Entry

The M_SETT service builds an entry in the timer table so that the requested function is performed upon time-out. Timer entries can be created to activate a program, resume a program, set a bit in memory, reset a bit in memory, or request an interrupt. Any task may create a timer to activate or resume a program. Timer entries to set or reset bits can be created by any task, provided the bit is within a static memory partition. Only privileged tasks can set bits in the operating system and request an interrupt.

The nonbase mode equivalent service is M.SETT.

Entry Conditions

Calling Sequence:

```
M_SETT      [TIMER=] timer, [SETVAL=] t1, [RSTVAL=] t2, [FUNCT=] function,
            [ARG4=] arg4, [ARG5=] arg5

(or)

LB          R3,function
SLL         R3,24
ORMW        R3,timer
LW          R4,t1
LW          R5,t2
LW (or LD)  R6,arg4
(LW         R7,arg5)
SVC         1,X'45'  (or)  M_CALL H.REXS,4
```

timer — is a word containing zeros in bytes 0 and 1, and a two-character timer identification in bytes 2 and 3

t1 — contains the current value to which the timer will be set in negative time units

t2 — contains the value to which the timer will be reset upon each time-out in negative time units. If the reset value is zero, the function is performed upon time-out and the timer entry is deleted. This case is called a "one-shot" timer entry.

function — ACP (1), RSP or RST (2), STB (3), RSB (4), and RQI (5) specify the function to be timed and represent activate program, resume program, set bit, reset bit, and request interrupt, respectively. If using the macro call, specify the alphabetic code. If loading registers, specify the corresponding numeric.

Function, Code and arg4 and arg5 contain values specific to the function being timed as follows:

Function	Code	arg4 and arg5
ACP	1	arg4 is a doubleword containing the one- to eight-character name of the program to be activated (system file), or pathname vector or RID vector in the first half of the doubleword and zero in the second half of the doubleword. If the task named is not currently in execution, it is preactivated to connect the interrupt to the task. This connection remains in effect until the task aborts or the timer is deleted. On normal exit, the timer table is updated to point to the next generation. arg5 is null.
RSP	2	arg4 is a doubleword containing the one- to eight-character name of the task to be resumed or zero in register 6 and the task number in register 7. arg5 is null.
(or)		
RST	2	arg4 is the task number entered into register 7 and register 6 is zeroed. arg5 is null.
STB	3	arg4 contains the address of the word in which the bits are to be set. The address must be within a static memory partition or the operating system. arg5 contains the bit configuration of the mask word to be ORed.
RSB	4	arg4 contains the address of the word in which the bit is to be reset. The address must be within a static memory partition or the operating system. arg5 contains the bit configuration of the mask word to be ANDed.
RQI	5	arg4 contains the priority level of the interrupt to be requested. arg5 is null.

Exit Conditions

Return Sequence:

Normal Return:

M.RTRN R3

R3 is nonzero and condition codes are not set

Error Condition:

M.RTRN R3

If there are no timer entries available, register 3 is zero and condition codes are not set.

If there are timer entries available, register 3 is zero and one of the following condition codes are set:

- CC1 set if requested load module does not exist or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by the M.KEY file.
- CC2 set if requested task is not active
- CC3 set if attempting to create a duplicate timer ID

Abort Cases:

RX02	Invalid function code specified for request to create a timer entry. Valid codes are ACP(1), RSP or RST(2), STB(3), RSB(4) and RQI(5).
RX03	An unprivileged task has attempted to set or reset a bit at an address outside a static memory partition, or privileged task has attempted to set/reset a bit at an address outside a static memory partition or the operating system.
RX04	The requesting task is unprivileged or has attempted to create a timer entry to request an interrrupt with a priority level outside the range of X'12' to X'7F', inclusive.

7.2.109 M_SMSGR - Send Message to Specified Task

The M_SMSGR service allows a task to send up to 768 bytes to the specified destination task. Up to 768 bytes can be accepted as return parameters. For further description, see Chapter 2.

The nonbase mode equivalent service is M.SMSGR.

Entry Conditions

Calling Sequence:

```
M_SMSGR    [PSB=] psbaddr

(or)

LA         R2,psbaddr
SVC        1,X'6C'  (or)  M_CALL  H.REXS,44
```

psbaddr is the logical address of the Parameter Send Block (PSB). See Chapter 2 for PSB description.

Exit Conditions

Return Sequence:

```
M.RTRN     6
```

Registers:

R6 Contains the processing start (initial) error status if any:

Value	Description
0	Normal initial status
1	Task not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name by the M.KEY file
2-9	Reserved
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address
14	Memory pool unavailable
15	Destination task queue depth exceeded
16	Invalid PSB address

Abort Cases:

None

Output Messages:

None

7.2.110 M_SOPL - Set Option Lower

The M_SOPL service allows the calling task to set the option lower bit. Use the M_ROPL (Reset Option Lower) service to reset the option lower bit.

The nonbase mode equivalent service is M.SOPL.

Entry Conditions

Calling Sequence:

M_SOPL

(or)

SVC 2,X'77' (or) M_CALL H.TSM,13

Exit Conditions

Return Sequence:

M.RTRN

(or)

M.RTRN (CC1 set)

Registers:

CC1 set Call caused the option lower bit to be set

Abort cases:

None

Output Message:

None

7.2.111 M_SRUNR - Send Run Request to Specified Task

The M_SRUNR service allows a task to activate or reexecute the specified destination task with a parameter pass of up to 768 bytes. Up to 768 bytes can be accepted as return parameters. For further description, see Chapter 2.

When a task name is supplied in words 0 and 1 of the Parameter Send Block (PSB), the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied.

The nonbase mode equivalent service is M.SRUNR.

Entry Conditions

Calling Sequence:

```
M_SRUNR    [PSB=] psbaddr

(or)

LA         R2,psbaddr
SVC        1,X'6D'  (or)  M_CALL  H.REXS,45
```

psbaddr is the logical address of the Parameter Send Block (PSB). See Chapter 2.

Exit Conditions

Return Sequence:

```
M.RTRN     6,7
```

Registers:

R6 Contains the processing start (initial) error status if any:

Value	Description
0	Normal initial status
1	Reserved
2	File specified in the PSB not found in directory
3	Reserved
4	File specified in the PSB is not a load module or executable image
5	Dispatch Queue Entry (DQE) unavailable
6	I/O error on directory read
7	I/O error on load module read
8	Memory unavailable
9	Invalid task number for run request to multicopied load module in RUNW state.
10	Invalid priority
11	Invalid send buffer address or size
12	Invalid return buffer address or size
13	Invalid no-wait mode end-action routine address

14	Memory pool unavailable
15	Destination task queue depth exceeded
16	Invalid PSB address
17	Reserved

R7 Contains the task number (taskno) of the destination task, or zero if the request was not processed

Abort Cases:

None

Output Messages:

None

7.2.112 M_SUAR - Set User Abort Receiver Address

The M_SUAR service sets up an address to return control to if an abort condition occurs during task execution.

All files remain open prior to transferring to the user specified address. See Task Termination Sequencing in Chapter 2.

The nonbase mode equivalent service is M.SUAR.

Entry Conditions

Calling Sequence:

```
M_SUAR      [RCVADR=] address

(or)

LA          R7,address
SVC         1,X'60' (or) M_CALL  H.REXS,26
```

address is the logical address to which control will be transferred on task termination

Exit Conditions

Return Sequence:

```
M.RTRN      R7
```

Registers:

R7 Bit 0 is zero if the request is honored, or one if the request is denied because the specified address is outside the user's allocated area

Bits 1-31 are unchanged

Abort Cases:

RX89 An unprivileged task has attempted to re-establish an abort receiver (other than M.IOEX)

Output Messages:

None

7.2.113 M_SUME - Resume Task Execution

The M_SUME service resumes a task that has been suspended. A request to resume a task which is not suspended is ignored.

The nonbase mode equivalent service is M.SUME.

Entry Conditions

Calling Sequence:

```
M_SUME    [TASK=] task

(or)

ZR        R6          } (or) LD R6,taskname
LW        R7,taskno   }
SVC       1,X'53' (or) M_CALL   H.REXS,16
```

task — the address of a doubleword containing the name of a task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared.

Exit Conditions

Return Sequence:

```
M.RTRN    R7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, unchanged

Abort Cases:

None

Output Messages:

None

7.2.114 M_SUSP - Suspend Task Execution

The M_SUSP service results in the suspension of the caller or any other specified task 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. Suspension of a task can also be ended upon receipt of a message interrupt. A message sent to a task that is synchronized (M_SYNCH) and suspended is not received, but the task is resumed.

The nonbase mode equivalent service is M.SUSP.

Entry Conditions

Calling Sequence:

```
M_SUSP    [TASK=] task, [TIM=] time1

(or)

LW        R5,time1
LI        R6,0      } (or) LD R6,taskname
LW        R7,taskno }
SVC       1,X'54' (or) M_CALL  H.REXS,17
```

task — the address of a doubleword containing the name of a task or zero in word 0 and the task number in word 1. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

time1 — contains zero, if suspension for an indefinite time interval is requested, else the negative number of time units to elapse before the caller is resumed.

Exit Conditions

Return Sequence:

```
M.RTRN    7
```

Registers:

R7 — Zero if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, contains the task number

(or)

Zero and CC1 is set if the specified task name is multicopied

Abort Cases:

None

Output Messages:

None

7.2.115 M_SYNCH - Set Synchronous Task Interrupt

The M_SYNCH service causes message and task interrupts to be deferred until the user makes a call to M_ANYWAIT, M_AWAITACTION, M_WAIT, or M_ASYNCH. When this service is used, message interrupts is not interrupted by end-action interrupts. All task interrupt levels cannot be interrupted, except by break, until they voluntarily relinquish control.

If a synchronized task is suspended then a message is sent to the task, the message receiver is not entered and the task resumes.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.SYNCH.

Entry Conditions

Calling Sequence:

```
M_SYNCH

(or)

SVC     1,X'1B' (or)    M_CALL    H.REXS,67
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

CC1 set	Synchronous task interrupt was already set

Abort Cases:

None

Output Messages:

None

7.2.116 M_TDAY - Time-of-Day Inquiry

The M_TDAY service obtains the time-of-day as computed from the real-time clock interrupt counter. The counter is initialized with a SYSGEN parameter.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.TDAY.

Entry Conditions

Calling Sequence:

```
M_TDAY

(or)

SVC     1,X'4E' (or) M_CALL   H.REXS,11
```

Exit Conditions

Return Sequence:

```
M.IPURTN  7
```

Registers:

R7

Byte	Contents
0	Hours (0 to 23)
1	Minutes (0 to 59)
2	Seconds (0 to 59)
3	Interrupts (less than one second)

Abort Cases:

None

Output Messages:

None

7.2.117 M_TEMPFILETOPERM - Change Temporary File to Permanent File

The M_TEMPFILETOPERM service makes a temporary file permanent. The temporary file is given a name in the specified directory and the file's resource type is changed from temporary to permanent. The file is made permanent with the attributes that were defined when it was created and with any new attributes that may have been acquired while the file's data was being established, such as additional extensions, end-of-file position, or explicit resource descriptor modifications incurred prior to invocation of this service. The temporary file can only be made permanent on the volume where the temporary file resides, for example, cross volume definitions are not allowed.

The most common use of this service is to ensure exclusive use of a file while its initial data is being established. This method of operation is desirable because it allows the user to guarantee the integrity of the file before it is defined in a directory where others can conveniently gain access to the file.

When the directory entry is established, it is linked to the resource descriptor of the file. This link makes the relationship of the name of the file to the other attributes of the file. These attributes are the same as the attributes for a permanent file.

The nonbase mode equivalent service is M.TEMPER.

Entry Conditions

Calling Sequence:

```
M_TEMPFILETOPERM [RESOURCE=] addr1 , [PNADDR=] addr2 [ , [CNPADDR=] addr3]
```

(or)

```
LW      R1, addr1
LW      R2, addr2
LA      R7, addr3     (or)    ZR       R7
SVC     2,X'28'       (or)    M_CALL   H.VOMM,9
```

addr1 is an LFC or FCB address

addr2 contains a PN vector or PNB vector

addr3 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R2	Contains addr2
R7	Contains addr3; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.118 M_TRUNCATE - Truncate File

The M_TRUNCATE service allows the caller to truncate the unused space of a file. This service is the complement of the extend service (M_EXTENDFILE). A file that has not been manually extended should never have to be truncated.

This service only truncates temporary or permanent files. Directories and memory partitions cannot be truncated. The caller must have write, update or append access in order to truncate the file. A file cannot be truncated to less than the minimum space requirement of the file as defined when the file was created.

The caller can truncate a file regardless of whether the file is currently allocated. Additionally, the caller can supply any allowable resource specification, such as pathname (PN), pathname block (PNB), resource ID (RID), logical file code (LFC) or address of a File Control Block (FCB).

The nonbase mode equivalent service is M.TRNC.

Entry Conditions

Calling Sequence:

```
M_TRUNCATE [RESOURCE=] addr1 [ , [CNPADDR=] addr2]

(or)

LW      R1, addr1
LA      R7, addr2     (or)    ZR       R7
SVC     2,X'26'       (or)    M_CALL   H.VOMM,7
```

addr1 contains a PN vector, PNB vector, RID vector, LFC, or FCB

addr2 is a CNP address or zero if CNP not supplied

Registers:

R1	Contains addr1
R7	Contains addr2; otherwise, zero

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7	Return status if a CNP is not supplied; otherwise, unchanged. For return status codes, refer to the H.VOMM status codes in the Resource Assignment/Allocation and I/O chapter.

7.2.119 M_TSTE - Arithmetic Exception Inquiry

The M_TSTE service resets the arithmetic exception status bit in the user's TSA and returns CC1 set or reset according to the status value. The status bit is set whenever the user is in execution and an arithmetic exception trap occurs. The bit remains set until this service is requested, or the task terminates.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.TSTE.

Entry Conditions

Calling Sequence:

```
M_TSTE

(or)

SVC     1,X'4D' (or) M_CALL  H.REXS,23
```

Exit Conditions

Return Sequence:

M.IPURTN

Registers:

PSD — CC1 contains the value of the arithmetic exception status bit

Abort Cases:

None

Output Messages:

None

7.2.120 M_TSTS - Test User Status Word

The M_TSTS service returns the 32 bit user status word of any specified task in execution. The user status word resides in the CPU Dispatch Queue (DQE.USW) and is modified by the Set User Status Word system service. These two services treat the user status word as either a set of 32 flags or as a 32 bit counter. Bit 0 is used as a status flag.

The nonbase mode equivalent service is M.TSTS.

Entry Conditions

Calling Sequence:

```
M_TSTS      [TASK=] task

(or)

ZR          R6          (or)  LD R6,taskname
LW          R7,taskno
SVC         1,X'49'     (or)  M_CALL   H.REXS,8
```

task — the address of a doubleword containing the name of a task or zero in word 1 and the task number in word 2. Task number must be used if the task is multicopied or shared. A task number of zero specifies the calling task.

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 — Bit 0 is set if the specified task was not found or the requesting task is not privileged and the owner name is restricted from access to tasks with a different owner name with M.KEY file; otherwise, register seven returns the user-status word.

Abort Cases:

None

Output Messages:

None

7.2.121 M_TSTT - Test Timer Entry

The M_TSTT service returns to the caller the negative number of time units remaining until the specified timer entry time-out. If the timer has expired, the result returned is zero.

This service can be executed by the IPU.

The nonbase mode equivalent service is M.TSTT.

Entry Conditions

Calling Sequence:

```
M_TSTT      [TIMER=] timer

(or)

LW          R6,timer
SVC         1,X'46' (or) M_CALL  H.REXS,5
```

timer two-character ASCII name of a timer, right-justified

Exit Conditions

Return Sequence:

```
M.RTRN      7
```

Registers:

R7 Negative number of time units remaining until time-out or zero if the timer has expired or does not exist

Abort Cases:

None

Output Messages:

None

7.2.122 M_TURNON - Activate Program at Given Time-of-Day

The M_TURNON service activates or resumes a specified task at a specified time and reactivates (resumes) it at specified intervals by creating a timer table entry using a specified timer ID. When a load module or executable image name is supplied as input, the operating system defaults to a search in the system directory only. For activations in other than the system directory, a pathname or RID vector must be supplied as input.

The nonbase mode equivalent service is M.TURNON.

Entry Conditions

Calling Sequence:

```
M_TURNON [FNAME=] filename, [TIME=] time [ , [RST=] reset] , [TIMID=] timerid

(or)

LD      R6,filename
LW      R4,time
LW      R5,reset
LW      R3,timerid
SVC     1,X'1E'   (or)   M_CALL   H.REXS,66
```

filename — is either a left-justified blank-filled doubleword containing the one- to eight-character ASCII name of the load module or executable image file (must be a system file), or register six contains the pathname vector or RID vector which points to the task to be activated and register seven is zero.

time — is the time-of-day on the 24-hour clock when the task is activated. It is a word value with the following format:

Byte	Contents
0	Binary hours
1	Binary minutes
2	Binary seconds
3	Zero

reset — is the time interval on the 24-hour clock to elapse before resetting the clock upon each time out. It has the same format as the time argument above. The task is reactivated at each time out. If a reset value is not specified, the comma denoting the field must still be specified and the task is activated only once.

timerid — is a word variable containing the right-justified, zero-filled, two-character ASCII name of the timer that will be created

Exit Conditions

Return Sequence:

M.RTRN	R3	Nonzero

Error Condition:

M.RTRN	R3	Zero if there are no timer entries available, the requested load module or executable image does not exist, attempting to create a duplicate timer ID, or invalid timer ID

Abort Cases:

None

Output Messages:

None

7.2.123 M_TYPE - System Console Type

The M_TYPE service types a user specified message and performs an optional read on the system console. Input message address validation is performed for the unprivileged task. Operation is wait I/O.

The maximum input or output is 80 characters. If no characters are specified, the maximum is used.

M_TYPE builds a Type Control Parameter Block (TCPB) which defines input and output buffer addresses for console messages and reads as described in Section 5.10.

The nonbase mode equivalent service is M.TYPE.

Entry Conditions

Calling Sequence:

```
M_TYPE  [OUTMES=] addr1 , [OUTCNT=] num1  [ , [INMES=] addr2 , [INCNT=] num2]

(or)

SVC          1,X'3F'   (or)   M_CALL H.IOCS,14
```

addr1 specifies address of output message buffer

num1 specifies transfer count for output (number of bytes, 80 maximum)

addr2 specifies address of input message buffer. If not specified, TCPB word 2 is zeroed. The first byte of this field contains the actual input quantity.

num2 specifies transfer count for input; number of bytes, 80 maximum

Register:

R1 Contains address of Type Control Parameter Block (TCPB)

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03 Nonprivileged user attempting transfer to a logical address outside legal boundaries

IO15 Type request while operation in progress

Output Messages:

None

7.2.124 M_UNLOCK - Release Exclusive Resource Lock

The M_UNLOCK service is used with the Set Exclusive Resource Lock service. When called, the exclusive lock is released if the task has the resource allocated in a shareable mode; otherwise, the lock cannot be released until deallocation of the resource. Once the lock is released, other tasks can allocate the resource in a compatible access mode for the particular shared usage. However, another task is not able to exclusively lock the resource until this task, and all other sharing tasks, deallocate the resource.

The nonbase mode equivalent service is M.UNLOCK.

Entry Conditions

Calling Sequence:

```
M_UNLOCK  [ARGA=] arga [ , [CNP=] cnpaddr]

(or)

LW        R5,arga
LA        R7,cnpaddr  (or)  ZR   R7
SVC       2,X'45'  (or)  M_CALL   H.REMM,24
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Exit Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in register seven or the status field of the CNP:

Value	Description
29	Specified LFC not assigned by this task
30	Invalid allocation index
32	An exclusive resource lock was not owned by this task
33	Resource is not allocated in a shareable mode by this task
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

Notes:

1. An exclusive resource lock can not be released by a task other than the owning task.

2. Any outstanding exclusive resource locks are released on task termination or on resource deallocation.

7.2.125 M_UNPRIVMODE - Change Task to Unprivileged Mode

The M_UNPRIVMODE service allows a task that was linked as privileged to operate in an unprivileged state. This causes the calling task's protection image to be loaded at every context switch. See M_PRIVMODE service to reinstate privilege status.

The nonbase mode equivalent service is M.UPRIV.

Entry Conditions

Calling Sequence:

```
M_UNPRIVMODE

(or)

SVC     2,X'58'   (or)   M_CALL   H.REXS,79
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.126 M_UNSYNC - Release Synchronous Resource Lock

The M_UNSYNC service is used with the Set Synchronous Resource Lock service to perform gating on resources that have been allocated for explicit shared usage. When called, the synchronization lock is released, and all tasks waiting to own the lock is polled.

The nonbase mode equivalent service is M.UNSYNC.

Entry Conditions

Calling Sequence:

```
M_UNSYNC  [ARGA=] arga  [ , [CNP=] cnpaddr]

(or)

LW        R5,arga
LA        R7,cnpaddr  (or)  ZR   R7
SVC       2,X'47'  (or)  M_CALL   H.REMM,26
```

arga — is an address containing the allocation index obtained when the resource was assigned
(or)
an address containing the address of a File Control Block (FCB) which contains an LFC in word 0

cnpaddr — is the address of a Caller Notification Packet (CNP) if notification is desired.

Applicable portions of the CNP for this function are abnormal return address and status field.

Exit Conditions

Return Sequence:

(with CNP)	(without CNP)
M.RTRN	M.RTRN
(or)	(or)
M.RTNA (CC1 set)	M.RTRN R7 (CC1 set)

Registers:

R7 — Return status if a CNP is not supplied; otherwise, unchanged

Status:

CC1 set

Posted in register 7 or the status field of the CNP

Value	Description
29	Specified LFC was not assigned by this task
30	Invalid allocation index
32	Synchronization lock was not set
46	Unable to obtain resource descriptor lock (multiprocessor only)

Wait Conditions

None

Notes:

1. A synchronization lock may not be cleared by any task other than the one that set the lock.

2. A synchronization lock is automatically released when a task terminates or deallocates the resource.

7.2.127 M_UPSP - Upspace

The M_UPSP service is not applicable to blocked or SYC files. If BOT is present on multivolume magnetic tape, volume record (header) is written. If EOT is present on multivolume magnetic tape an erase and write EOF is performed.

The nonbase mode equivalent service is M.UPSP.

Entry Conditions

Calling Sequence:

```
M_UPSP      [FCB=] fcb

(or)

LA          R1, fcb
SVC         1,X'10' (or) M_CALL  H.IOCS,20
```

fcb is the FCB address

Registers:

R1 FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cell for a system or blocked file.
IO09	Illegal operation on the SYC file

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

7.2.128 M_VADDR - Validate Address Range

The M_VADDR service verifies a logical address provided by the user.

The nonbase mode equivalent service is M.VADDR.

Entry Conditions

Calling Sequence:

```
M_VADDR  [STARTADDR=] addr , [RANGE=] num

(or)

LW       R6, addr
LI       R7, num
SVC      2,X'59'   (or)   M_CALL   H.REXS,33
```

addr is the logical starting address

num is the number of bytes to validate

Registers:

R6	Contains addr
R7	Contains num

Exit Conditions

Return Sequence:

```
M.RTRN
```

Registers:

CC2 set	Address range crosses map block boundary
CC3 set	Locations specified are protected
CC4 set	Invalid address (not in caller's address space)
R0-R7	Unchanged

7.2.129 M_WAIT - Wait I/O

The M_WAIT service provides return to the user when the I/O request associated with the specified FCB is complete. The task is suspended until I/O completes.

The nonbase mode equivalent service is M.WAIT.

Entry Conditions

Calling Sequence:

```
M_WAIT      [FCB=] fcb

(or)

LA          R1,fcb
SVC         1,X'3C' (or) M_CALL  H.IOCS,25
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

MS31	User attempted to go to an any wait state from an end action routine.

Output Messages:

None

7.2.130 M_WRITE - Write Record

The M_WRITE service performs the following functions:

- Prevents a write to a read-only file.
- Provides special random access handling for disc files.
- Blocks records for system and blocked files.
- Writes volume record if BOT of multivolume magnetic tape.
- Performs an erase and write EOF if EOT of multivolume magnetic tape.
- Writes one record from the buffer pointed to by the TCW in the FCB.

The nonbase mode equivalent service is M.WRIT.

Entry Conditions

Calling Sequence:

```
M_WRITE  [FCBADDR=] addr

(or)

LA          R1, addr
SVC         1,X'32'  (or)  M_CALL  H.IOCS,4
```

addr is the FCB address

Registers:

R1 Contains addr

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO06	Invalid blocking buffer control cell for a system or a blocked file
IO09	Illegal operation on the SYC file
IO27	Write attempted while reading from a system or a blocked file
IO38	Write attempted on a file opened in read-only mode
RM02	Write attempted to SLO/SBO in update mode

Output Messages:

Dismount/mount messages if EOT on multivolume magnetic tape

7.2.131 M_WRITEEOF - Write EOF

The M_WRITEEOF service performs the following functions:

- Prevents a write to a read-only file.
- Issues an end-of-file and purge if the file is a system or blocked file which is output active. If EOF is requested for an unblocked disc file, the handler ignores the request.
- Writes volume record if BOT on multivolume magnetic tape.
- Performs an erase and write EOF if EOT on multivolume magnetic tape.
- Writes one EOF.

The nonbase mode equivalent service is M.WEOF.

Entry Conditions

Calling Sequence:

```
M_WRITEEOF [FCBADDR=] addr

(or)

LA          R1, addr
SVC         1,X'38'  (or)  M_CALL H.IOCS,5
```

addr is the FCB address

Registers:

R1 Contains addr

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO10	FAT entry marked read-only (unless SGO)
IO11	File is SYC
IO30	Illegal volume record. Either volume number or reel ID from volume record do not match FAT information

Output Messages:

Dismount/mount messages if EOT on multivolume magnetic tape

7.2.132 M_XBRKR - Exit from Task Interrupt Level

The M_XBRKR service must be called at the conclusion of executing a task interrupt routine. It transfers control back to the point of interruption and resets the interrupt to the level established before the break or M_INT.

The nonbase mode equivalent service is M.XBRKR.

Entry Conditions

Calling Sequence:

M_XBRKR

(or)

RETURN

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

RX34	Task has made a break receiver exit call while no break is active

Output Messages:

None

7.2.133 M_XIEA - No-Wait I/O End-Action Return

The M_XIEA service is required for exiting any no-wait I/O end-action routine. Both normal and error end-action routines use this exit.

The nonbase mode equivalent service is M.XIEA.

Entry Conditions

Calling Sequence:

M_XIEA

(or)

RETURN

Exit Conditions

Return Sequence:

BL	S.EXEC6 no-wait I/O postprocessing complete

Registers:

None

Abort Cases:

None

Output Messages:

None

7.2.134 M_XMEA - Exit from Message End-Action Routine

The M_XMEA service is called to exit the end-action routine associated with a no-wait message send request. For further description, see Chapter 2.

The nonbase mode equivalent service is M.XMEA.

Entry Conditions

Calling Sequence:

M_XMEA

(or)

RETURN

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at message interrupt or task base level

Registers:

None

Abort Cases:

RX99	End action interrupt was inactive when end action exit issued

Output Messages:

None

7.2.135 M_XMSGR - Exit from Message Receiver

The M_XMSGR service must be called to exit the message receiver code of the calling task after the task has received a message from another task. For further description, see Chapter 2.

The nonbase mode equivalent service is M.XMSGR.

Entry Conditions

Calling Sequence:

```
M_XMSGR    [ [RXB=] rxbaddr ]

(or)

LA                 R1,rxbaddr
RETURN
```

rxbaddr is the logical address of the Receiver Exit Block (RXB)

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at task base level

Registers:

None

Abort Cases:

RX93	Invalid Receiver Exit Block (RXB) address was encountered during message exit
RX94	Invalid Receiver Exit Block (RXB) return buffer address or size was encountered during message exit
RX95	Task has made a message exit while the message interrupt was not active

Output Messages:

None

7.2.136 M_XREA - Exit from Run Request End Action Routine

The M_XREA service is called to exit the end-action routine associated with having sent a no-wait run request.

The nonbase mode equivalent service is M.XREA.

Entry Conditions

Calling Sequence:

M_XREA

(or)

RETURN

Exit Conditions

Return Sequence:

M.RTRN	Interrupts context at message interrupt or task base level

Registers:

None

Abort Cases:

RX90	End-action interrupt was inactive when end action exit issued

Output Messages:

None

7.2.137 M_XRUNR - Exit Run Receiver

The M_XRUNR service is called to exit a task which was executing for a run request issued from another task.

The nonbase mode equivalent service is M.XRUNR.

Entry Conditions

Calling Sequence:

```
M_XRUNR      [[RXB=] rxbaddr]

(or)

LA           R1,rxbaddr
RETURN
```

rxbaddr is the logical address of the Receiver Exit Block (RXB). For further description, see Chapter 2.

Exit Conditions

Return Sequence:

The run-receiver queue is examined and if not empty, the task is executed again on behalf of the next request. If the queue is empty, the exit options in the RXB are examined. If option byte is zero, the task is placed in a wait state, waiting for the next run request to be received. If option byte is nonzero, the task exits the system.

Note: If the task is reexecuted, control is transferred to the instruction following the M_XRUNR call.

Abort Cases:

RX96	Invalid Receiver Exit Block (RXB) address
RX97	Invalid return parameter buffer address or size
RX98	Run receiver mode not active when run receiver exit issued

Output Messages:

None

7.2.138 M_XTIME - Task CPU Execution Time

The M_XTIME service returns to the caller the total elapsed CPU execution time, in microseconds since the initiation of the task. The time returned does not include any IPU execution time whether or not IPU accounting is enabled.

The nonbase mode equivalent service is M.XTIME.

Entry Conditions

Calling Sequence:

```
M_XTIME

(or)

SVC        1,X'2D'  (or)  M_CALL H.REXS,65
```

Exit Conditions

Return Sequence:

```
M.RTRN  6,7
```

Registers:

R6,R7	CPU execution time in microseconds

Abort Cases:

None

Output Messages:

None

7.3 Nonmacro-Callable System Services

7.3.1 Debug Link Service

The Debug Link service is to be used only by the interactive debugger (MPXDB, SYMDB, or DEBUGX32) for the purpose of transferring control to the debugger. The debugger plants this SVC trap in the user's task at the desired location.

Entry Conditions

Calling Sequence:

```
SVC     1,X'66'
```

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.3.2 Eject/Purge Routine

The Eject/Purge Routine service performs the following functions:

- If a file is blocked and output active, a purge is issued. Return is made to the user.
- Writes volume record if BOT on multivolume magnetic tape.
- Performs an erase and write EOF if EOT on multivolume magnetic tape.
- Eject is not applicable to SYC files.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'0D'
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

7.3.3 Erase or Punch Trailer

The Erase or Punch Trailer service writes the volume record if BOT on multivolume magnetic tape or performs an erase and write EOF if EOT on multivolume magnetic tape.

Erase or punch trailer is not applicable to blocked or SYC files.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'3E'
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO09	Illegal operation on the SYC file
IO12	File not opened for write mode

Output Messages:

Mount/dismount messages if EOT on multivolume magnetic tape

7.3.4 Execute Channel Program

The Execute Channel Program service is available to privileged users and allows command and data chaining to General Purpose Multiplexor Controller (GPMC) and extended I/O devices only. Logical execute channel is available to both privileged and nonprivileged users. Physical execute channel is available only to privileged users.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'25'
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

IO03	Nonprivileged user attempting transfer to a logical address outside legal boundaries
IO43	IOCL list or data address not in contiguous E memory, GPMC devices only
IO50	An unprivileged user attempted to execute a physical channel program
IO51	A TESTSTAR command was used in a logical channel program
IO52	A logical channel was too large to be moved to memory pool
IO53	A TIC command follows a TIC command in a logical channel program
IO54	A TIC command attempted to transfer to an address which is not word bounded
IO55	Illegal address in logical IOCL. Address is not in user's logical address space
IO56	A read-backward command was used in a logical channel program
IO57	Illegal IOCL address. IOCL must be located in the first 128K words of memory

7.3.5 Get Extended Memory Array

The Get Extended Memory Array service requests an array of extended memory. If the request cannot be met, then all free memory, except one-eighth of the amount of physical memory, is allocated to the task and a count of maps allocated is returned. This service is intended for use by tasks that require the largest possible buffers without being placed on the MRQ for an extended period.

Entry Conditions

Calling Sequence:

```
LW      R1,maps
SVC     2,X'7F'
```

maps is the number of map blocks required

Exit Conditions

Return Sequence:

M.RTRN

Registers:

R2 Number of map blocks allocated

R3 Starting logical address of memory allocated or zero if an error occurred

R4 Ending logical address of memory allocated or error code as follows:

Value	Description
1	CSECT overrun
2	Request for more memory than physically exists
3	M_GETMEMBYTES service in use
4	Unable to allocate logically contiguous memory

R5 Number of map blocks, all classes, that are now free

7.3.6 Release FHD Port

The Release FHD Port service is available only to privileged users and is only supported by the four megabyte fixed head disc.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'27'
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

7.3.7 Reserve FHD Port

The Reserve FHD Port service is available only to privileged tasks and is only supported by the four megabyte fixed head disc.

Entry Conditions

Calling Sequence:

```
LA      R1,fcb
SVC     1,X'26'
```

fcb is the FCB address

Exit Conditions

Return Sequence:

M.RTRN

Registers:

None

Abort Cases:

None

Output Messages:

None

Gould Inc., Computer Systems Division
6901 W. Sunrise Blvd.
P. O. Box 409148
Fort Lauderdale, FL 33340-9148
Telephone (305) 587-2900

GOULD
Electronics

Users Group Membership Application

USER ORGANIZATION: _____

REPRESENTATIVE(S): _____

_____

_____

ADDRESS: _____

_____

TELEX NUMBER: _____ PHONE NUMBER: _____

NUMBER AND TYPE OF GOULD CSD COMPUTERS: _____

_____

OPERATING SYSTEM AND REV. LEVEL: _____

_____

APPLICATIONS (Please Indicate)

1. EDP
   - A. Inventory Control
   - B. Engineering & Production Data Control
   - C. Large Machine Off-Load
   - D. Remote Batch Terminal
   - E. Other

2. Communications
   - A. Telephone System Monitoring
   - B. Front End Processors
   - C. Message Switching
   - D. Other

3. Design & Drafting
   - A. Electrical
   - B. Mechanical
   - C. Architectural
   - D. Cartography
   - E. Image Processing
   - F. Other

4. Industrial Automation
   - A. Continuous Process Control Op.
   - B. Production Scheduling & Control
   - C. Process Planning
   - D. Numerical Control
   - E. Other

5. Laboratory and Computational
   - A. Seismic
   - B. Scientific Calculation
   - C. Experiment Monitoring
   - D. Mathematical Modeling
   - E. Signal Processing
   - F. Other

6. Energy Monitoring & Control
   - A. Power Generation
   - B. Power Distribution
   - C. Environmental Control
   - D. Meter Monitoring
   - E. Other

7. Simulation
   - A. Flight Simulators
   - B. Power Plant Simulators
   - C. Electronic Warfare
   - D. Other

8. Other

Please return to:

Users Group Representative

Date: _____

Gould Inc., Computer Systems Division Users Group. . .

The purpose of the Gould CSD Users Group is to help create better User/User and User/Gould CSD communications.

There is no fee to join the Users Group. Simply complete the Membership Application on the reverse side and mail to the Users Group Representative. You will automatically receive Users Group Newsletters, Referral Guide and other pertinent Users Group activity information.

Fold and Staple for Mailing

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST-CLASS MAIL PERMIT NO. 947 FT. LAUDERDALE, FL

POSTAGE WILL BE PAID BY ADDRESSEE

GOULD INC., COMPUTER SYSTEMS DIVISION
ATTENTION: USERS GROUP REPRESENTATIVE
6901 W. SUNRISE BLVD.
P. O. BOX 409148
FT. LAUDERDALE FL 33340-9970

Fold and Staple for Mailing

(Detach Here)

GOULD
Electronics