17329125

CONTROL DATA CORPORATION

MPX/OS VERSION 3
REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

REVISION RECORD	
REVISION	DESCRIPTION
A	Original release.
(02/01/83)	
Document No. 17329125	

Revision letters I, O, Q, and X are not used.

Printed in the United States of America

Please address comments concerning this manual to:

CONTROL DATA CORPORATION
SYSTEMS TECHNOLOGY DIVISION
215 Moffett Park Drive
Sunnyvale, California 94086

Or use Comment Sheet in the back of this manual.

PREFACE

This document describes the CONTROL DATA MP-32 Computer Systems in general and the CONTROL DATA MPX Operating System (MPX/OS) Version 3 specifically.

Although this document contains sufficient information to be studied independently, the following documents are also available to meet specific user needs:

Control Data Publication Number	Title
17329120	MP-60 Emulation Reference Manual
14061300	MP-60 COMPASS Reference Manual
17329115	MP-60 MPX/OS Version 3 Installation Handbook
14391700	MP-60 Program Command Console Reference Manual
14063800	MP-60 Utility Reference Manual
14062200	MP-60 PRELIB Reference Manual
14061100	MP-60 FORTRAN Reference Manual
14351100	MP-60 UPDATE Reference Manual
17328900	MASS/MPSIM Reference Manual
17329145	MP-32 MPX/OS Version 3 Operator's Guide
17329140	ITS Version 2 Reference Manual

TABLE OF CONTENTS

APPENDIXES

FIGURES

FIGURES (Contd)

TABLES

INTRODUCTION 1

The Control Data MP-60 computer system, using advanced concepts in microprogramming architecture, can be configured to:

- o Utilize one to eight central processing units (CPUs).
- o Provide a multiprocessing environment.
- o Provide service to one or more independent work requests (jobs) per CPU.
- o Provide service to one or more job subdivisions (tasks) per job.

The microprogrammable processor (MP-32) is microprogrammed to provide the MP-60 software environment described in the MP-60 Emulation Reference Manual. Additional instructions can be added to enhance the performance of the MP-60 for specific applications.

The MP-60 Operating System (MPX/OS) was developed in modular building blocks and establishes basic system functions. The modular structure of the system software facilitates the incorporation of software modifications to enhance the performance of MPX/OS for specific applications.

MP-60 HARDWARE OVERVIEW

Complete details of the MP-32 hardware are contained in the MP-60 Emulation Reference Manual. The details presented in this section are provided in support of the operating system definition.

Table 1-1 presents the minimum and maximum configurations supported by MPX/OS with the Interactive Terminal Subsystem. Figure 1-1 illustrates a multiprocessor configuration as an example.

TABLE 1-1. MINIMUM, MAXIMUM MPX/OS SYSTEM CONFIGURATIONS

	Minimum	Target	Maximum
Program states	6	6	48
Main memory	262K Bytes 64K Words	524K Bytes 131K Words	167,772K Bytes 4,194K Words
Card reader	1	1	1
Line printer	1	1	1
Display console	1	1	1
Display station	1	1	1
Interactive terminals	0	48	256
Mass Storage (megabytes)	10	400	1,500
Magnetic tape	0	2	8
Number of CPUs	1	1	8

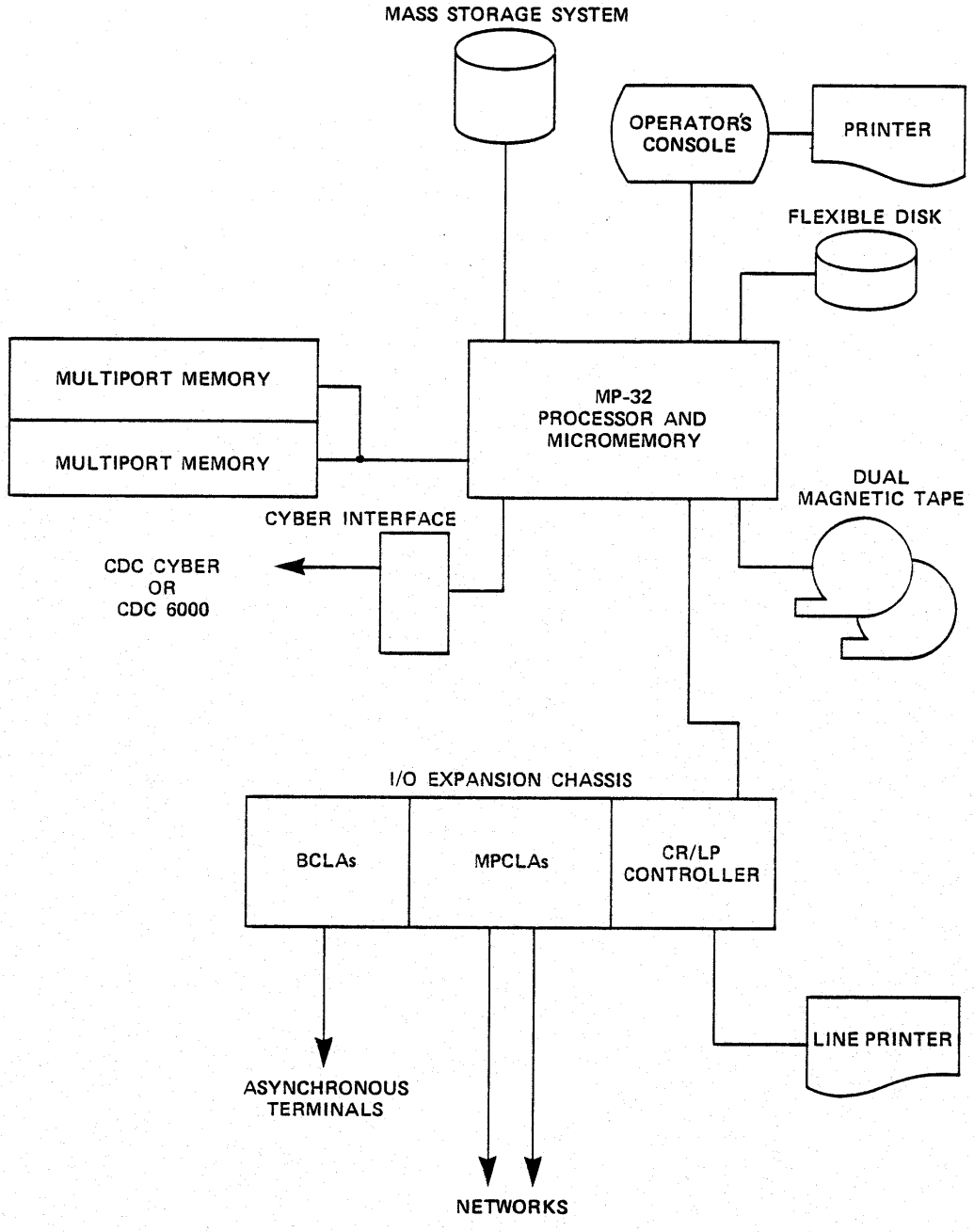


Figure 1-1. Hardware Configuration

MAIN MEMORY

The main memory of the MP-32 is modular and in increments from a minimum of 65,536 32-bit words to a maximum of 4,194,304 32-bit words.

CENTRAL PROCESSING UNITS

System configurations containing more than one MP-32 CPU provide direct connection between the CPUs, as well as an indirect connection through the main memory. The direct connection provides one signal path in each direction, an associate CPU interrupt signal. The associate CPU interrupt signal is used during normal operation to direct the attention of a CPU to a message area maintained in the main memory. The interprocessor communication facility is accessible only to the monitor state environment. All CPUs in a multiprocessing environment provide identical capabilities to the program state tasks. One CPU is designated the master CPU; the remaining CPUs are designated slave CPUs. The master CPU conducts system startup and has connections to all slave CPUs. Slave CPUs have a connection to the master CPU only.

INTERRUPTS

The MP-60 utilizes interrupts to signal event occurrences in a processing environment in which many activities may be occurring concurrently and asynchronously. At the start of each MP-60 instruction, a test is made for interrupt conditions. If an interrupt condition exists, execution of the current code sequence halts and execution of an interrupt routine is initiated. Upon regaining control of the CPU, the interrupted code resumes without notice of the interrupt processing.

The MP-60 recognizes two categories of interrupts: external and internal. External interrupts consist of input/output (I/O), real-time, and interprocessor interrupts. Internal interrupts consist of monitor call, clock, arithmetic (arithmetic overflow, divided, exponent, function) faults, and environmental (page, memory parity error, illegal instruction, memory reject, power failure) faults.

Under MPX/OS, the master CPU recognizes and services all interrupts which occur on the master CPU. Slave CPUs recognize monitor call interrupts, interprocessor interrupts, clock interrupts, arithmetic fault interrupts, and environmental fault interrupts. Of those interrupts recognized by slave CPUs, only the interprocessor, power failure, and clock interrupts are serviced by the slave CPUs. The remaining interrupts are routed to the master CPU for servicing.

Interrupts are used by MPX/OS to facilitate task switching, and to continue I/O processing. During task mode execution, master and slave CPUs operate with all recognizable interrupts enabled (except possibly the arithmetic fault interrupts). During executive mode execution of the master CPU, only the environmental fault interrupts are unconditionally enabled. Real-time interrupts are disabled only during list processing, and/or real-time executive execution. All other interrupts are always disabled. During executive mode execution of a slave CPU, only the environmental fault interrupts are unconditionally enabled. All other interrupts are always disabled.

MPX/OS gains control of the CPU when any interrupt is recognized. Tasks may elect to regain control if they cause an arithmetic fault, page fault, or an illegal instruction fault. An executive service request (ESR) is provided to enable the task (see section 4, ENABLE, Enable and Select Interrupt Control) to recognize these interrupts. If the interrupt condition is recognized, but the task has not elected to regain control, MPX/OS terminates the task and its job. The MP-60 recognizes the interrupt conditions according to the priority order defined in table 1-2.

MACHINE STATES

The MP-60 provides nearly identical resources for eight execution environments called states. Each environment includes 32 full-word (32-bit) registers, one 1-bit register, a 65,536 32-bit word address space, and a status flag for each of the four arithmetic fault conditions. Machine state 0 differs from the remaining seven states through its ability to execute privileged instructions and by its obligation to process interrupts. MPX/OS uses state 0 to execute the system executive code. The terms monitor state and executive state are synonyms for state 0.

TABLE 1-2. INTERRUPT PRIORITY

LEVEL	INTERRUPT GROUPS
1	Power Failure
2	CPU Memory Errors
3	DMA Memory Errors
4	Illegal Instruction
5-12	Micro I/O 0-8
13-28	Macro I/O 0-15
29-44	Real-Time 0-15
45-50	Open
51	Open
52	Clock Interval
53-55	Open
56	Inter-Processor
57-60	Faults Divide-Arithmetic
1 - highest	60 - lowest

PAGING

All references to main memory are routed through the MP-60 paging hardware for potential relocation. Addresses that originate from the eight states are relocated by the paging hardware.

The paging hardware contains 16 16-bit registers for each paged state. Each page register has the format shown in figure 1-2.

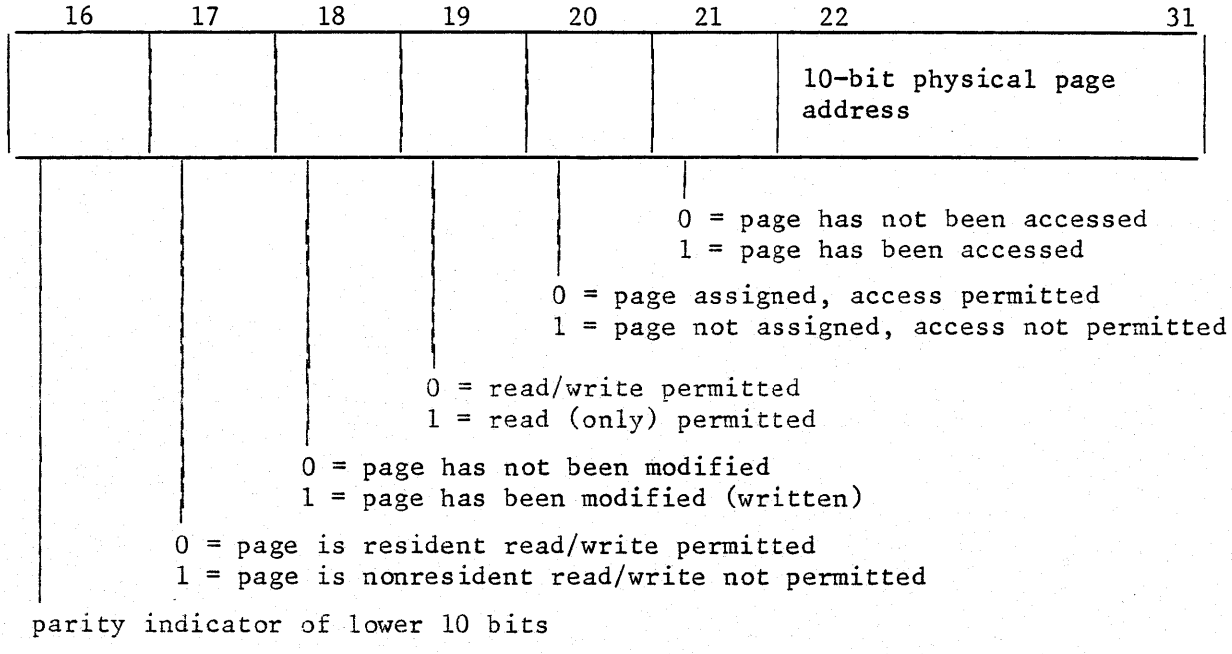


Figure 1-2. Paging Address Layout

The upper 4 bits of an address originating from a state are used to select one of the 16 page registers assigned to that state. Bits 17, 19 and 20, maintained by the operating system software, are used by the hardware to detect unauthorized use of a memory address. Bit 18, maintained by the hardware, is available to the operating system and is forced to 1 on memory write operations. Bit 18 is also forced to 1 by the operating system when read functions are processed. Bit 23 maintained by the hardware, is available to the operating system and is forced to 1 on memory reference. Address relocation is accomplished by substituting the 8-bit physical page address for the 4-bit page register selector.

The relocated address began as a 16-bit value. Removing four bits to select a page register yields a page size of 4096 32-bit words. The main memory of an MP-60 computer system, therefore, consists of a minimum of 16 4096-word pages, and can be expanded in increments of 16 pages to a maximum of 1024 pages.

The following sections describe the operating system from two views. The first view is a conceptual view. The second view is a physical/functional view.

JOBS

A job is a request from a user to have the computational facility perform work. The work request is submitted to the operating system in the form of a card deck or a file. The work request is processed by an operating system task, the job manager. The job manager interprets the control statements and initiates any additional system activity required to satisfy the work request.

A job is established in the operating system environment by building a table entry (a job control table entry) and by initiating execution of the job manager system task. In response to appropriate job control statements, the job manager system task causes additional tasks to execute. The additional tasks may be system tasks, library tasks, or user tasks. System tasks are part of MPX/OS, and may receive special treatment. Library tasks operate under the same rules as user tasks, but are maintained in the system library mass storage file. User tasks receive no special treatment by the executive.

Each job in the system has its own job control table (JCT) entry. The contents of a JCT include the following:

- Job identification (from *JOB or *RJOB control card)
- Job submitter identification (from]BATCH or]JOB)
- Job accounting information (account number and CPU charges)
- Job resource parameters (see *SCHED control card)
- I/O assignments (see *OPEN and *EQUIP control cards)
- List of tasks established for the job

TASKS

A task is an independent unit of work that competes for the resources of the system. The work requested by a job is accomplished as the summation of task efforts. The CPU always executes either executive (i.e., MPX/OS Resident Programs) or task code. The executive executes on demand, either

from user-issued Executive Service Requests (ESRs) or from event occurrences signaled by interrupts. Tasks compete for use of the CPU and other resources on the basis of their priority.

TASK ORIGIN

Tasks can reside in memory, on the library, or in a file accessible to the user's job. Memory resident tasks are system tasks or user tasks that have previously executed within a job, but on their return did not request release from memory (see section 4, RETURN, Terminate Task Execution).

Tasks that reside in the system library are referred to as library tasks. Library tasks are brought into execution with a job control statement that uses the name of the library task as the control card name (section 3, Library Task Statement).

Tasks that reside in files maintained by the user are referred to as user tasks. User tasks are brought into execution by *LOAD and *RUN control cards (see section 3), or by a CALL ESR from within an executing task (see section 4).

Library tasks and user tasks can be maintained in two forms: relocatable and absolute binary. The relocatable binary form normally originates from the assembler (COMPASS) or compiler (FORTRAN). Absolute binary form is obtained using the *ABS job control statement to record an image of the task after it has been prepared for execution.

TASK IDENTIFIERS

A task identifier is established through the *TASK job control statement and the CALL ESR. The task identifier is used to request status of tasks, label diagnostic messages, and construct operator displays reflecting system activity.

A task identifier is valid for the entire existence of the task (see section 4, RETURN, Terminate Task Execution).

Library tasks are assigned their library names as task identifiers when placed into execution. Library task identifiers cease to exist after task termination.

TASK LOADING

Each task executed under MPX/OS control is assigned a program state and up to 65,536 words of main memory. The tasks reference memory with logical addresses as assigned by the MPX/OS loader; the logical addresses are transformed into physical addresses through the MP-32 paging hardware.

Figure 1-3 illustrates the important features of loaded tasks. Logical page 15 of each task of the job is the same physical page of memory. It is used for buffers and data storage in support of the job manager task and the Blocker/Deblocker library routines.

The communication area (PARM) is 50 words in length and resides in page 14. Its format and content are described by the ESRs which utilize the area.

Logical pages 0 through 14 are prepared with an executable image of the task by the loader. The memory is loaded with program code and data common blocks from logical page 14 downward. If some pages are left unused after loading is complete, the corresponding page registers are set to reflect the unassigned pages. As a result, task address references that are out of range are detected by the hardware and generate a fault interrupt.

During the load process, the loader code occupies logical page 0 and the loader symbol table occupies logical page 1. The blank and numbered common blocks of the task are allocated over the loader code and tables. The task can only achieve full use of the 15 logical pages by allocating 8192 or more words of space to the blank and numbered common blocks. Any pages occupied by the loader, and not used for blank or numbered common allocation, are returned to the system for reuse.

TASK RELATIONSHIPS

The multitasking feature of MPX/OS allows one task to establish and initiate execution of another task. Two terms, caller and callee, are used to identify the relationship between two such tasks. A caller is the task that issues the CALL ESR. A callee is the task that the CALL establishes and initiates. A task can be both a callee and a caller at the same time.

For example, the job manager interprets the *TASK, *LOAD, and *RUN job control statements and thereby establishes and initiates execution of a user task, TASKA. TASKA can issue a CALL ESR to establish and initiate execution of a second user task, TASKB. The job manager is the caller of TASKA, and TASKA is the callee of the job manager. At the same time, TASKA is the caller of TASKB.

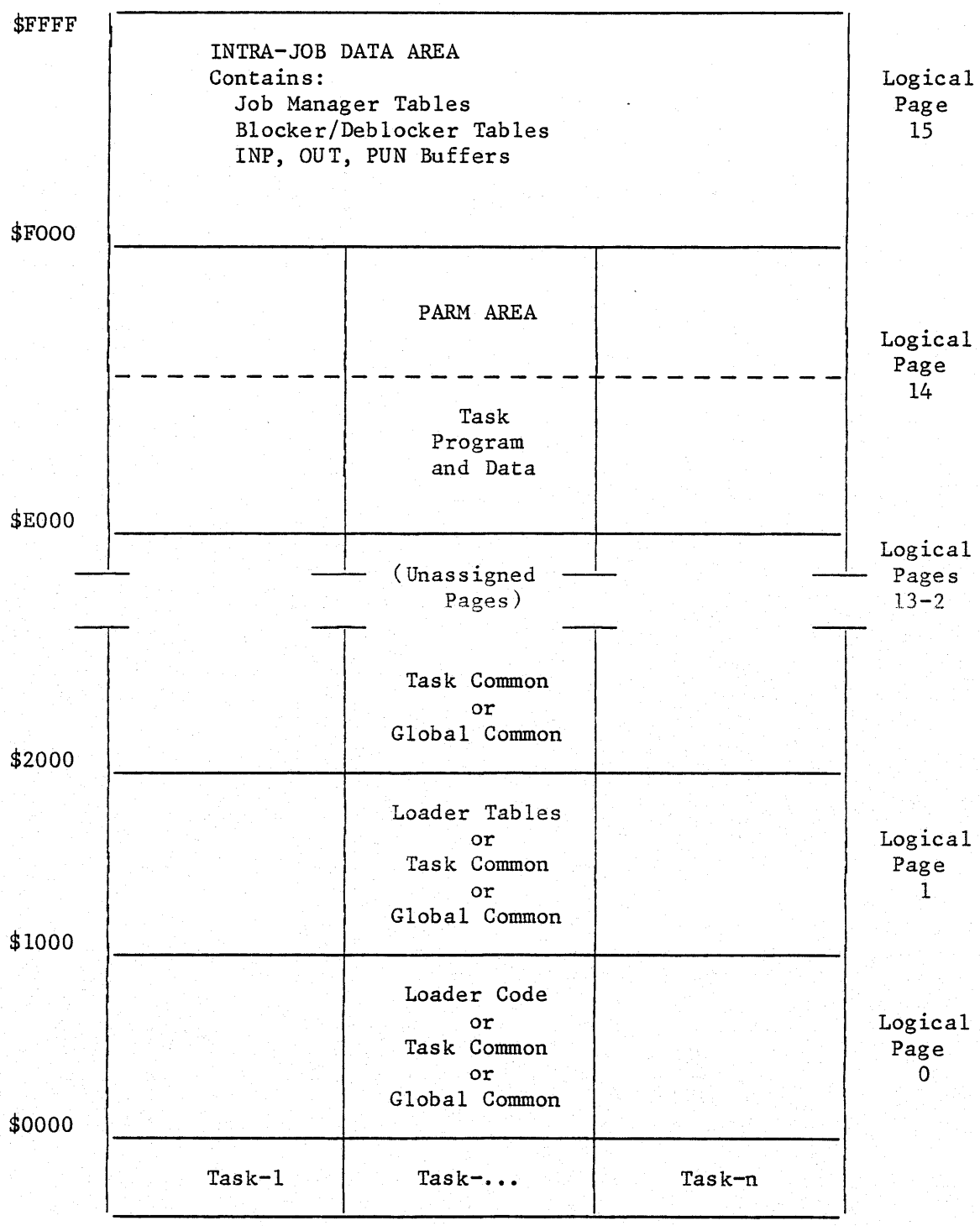


Figure 1-3. Task Memory Layout

TASK STAGING

A task running under MPX/OS exists in several stages. These stages are defined by table 1-3. A task in the ready stage can only go to the running stage. A task in a wait stage must go to the ready stage before running. The running task can voluntarily go to wait or terminated stages, or it may involuntarily go to the ready stage if a higher-priority task becomes ready in the same CPU. A terminated task may be released or simply be allowed to go dormant.

TASK CONTROL

Each task in the system has its own task control table (TCT) entry. The contents of each TCT include the following:

- o Task identifier
- o Task accounting information (accumulated task CPU time)
- o Task priority
- o Task status definition
- o Caller wait list thread
- o Current caller definition

The TCT is used by the executive for implementing prioritized delivery of the CPU and resources to the task. (See List Processing and Priorities of this section.)

MULTITASKING, MULTIPROGRAMMING, AND MULTIPROCESSING

MPX/OS provides a multitasking capability which enables a task to initiate execution of one or more tasks concurrent (simultaneous, if a multiprocessor system) with its own execution.

MPX/OS provides a multiprogramming capability, which means that the system shares the CPU between two or more tasks over a period of time. Multiprogramming occurs when system tasks share the CPU with user tasks, when tasks from two or more separate jobs share the CPU, when two or more tasks of the same job share the CPU, and when any or all combinations of these occur.

TABLE 1-3. TASK STATUS ASSIGNMENT DEFINITIONS

No.	Status	Description
00	Dormant	The task has completed its work and returned, without release, to its caller. Status remains dormant until the task is called again.
01	Active, Ready	The task is currently executing or ready to resume execution.
02	I/O Wait	The task has requested I/O on a data set that is currently busy. The task is threaded by priority in a wait list for the data set.
04	File Manager Wait	The task has requested a file manager function and the file manager is active. The task is threaded on a priority basis in a call list for the file manger.
05	Call Wait	The task has called another task. Until the call can be connected, the caller remains in call status; it cannot resume execution.
06	Callee Wait	The task called another task and, as a parameter of the call, requested not to be multiprogrammed with its callee. After the callee returns, the caller's status will be set to active.
07	Deferred Wait	The task has called other tasks, multiprogrammed with them, and then requested that it not be permitted to resume execution until one of a set of callees returns.
08	FINIS	The task has returned, but has outstanding callees.
09	TSCHED Wait	The task issued a TSCHED request. After the specified time interval has elapsed, the task status will be set to active.
10	Operator Wait	The task is waiting for the operator to respond to a pending message. Task becomes ready after the operator responds.
12	Multiple Status Wait	The task has issued a MUST ESR and is awaiting one or more possible events.
13	Task Idle	The task has been taken out of potential active status via the IDLE command (ITS-Operator Control Facility).
14	SYSQS	The task has issued a SYSQS ESR, and the job scheduler is busy.
15	Suspend	The task has been temporarily taken out of potential active status.

MPX/OS provides a multiprocessing capability, which means that the system services can be delivered by two or more CPUs.

Multitasking and multiprogramming provide service to the user from a single CPU in the form of nonsimultaneous, interleaved task execution. They can provide service from multiple CPUs in the form of simultaneous execution of two or more jobs and/or tasks.

MASTER-SLAVE ORGANIZATION

MPX/OS utilizes a master-to-slaves architecture to provide multiprocessor capabilities. Under this architecture, one CPU (the master) manages system resources, performs all I/O operations, provides all executive service functions, and distributes program execution assignments to all other CPUs (the slaves) and to itself.

Slave CPUs are computational resources to which the master CPU assigns user tasks for execution. A slave CPU performs no I/O operations or ESR functions. Any such request from an executing user task is routed to the master CPU for servicing.

Under MPX/OS, each CPU operates independently of all others. This allows each CPU to move from task to task with occasional interrupt processing and very little synchronized activity. Executive functions are provided by the slave CPUs where system resource management is not involved. Current examples of such functions are the CPU ready list and CPU state availability list management and task level accounting.

LIST PROCESSING

MPX/OS uses the concept of list processing to reserve and allocate system resources. Tasks making ESRs which cannot be serviced immediately are placed in a list and are serviced as time and resources permit. Each list is ordered according to the priority of the tasks in the list. At each opportunity, the highest-priority task in the list is readily obtained for servicing. Examples of lists maintained by MPX/OS include:

- CPU ready list for tasks awaiting control of the CPU
- I/O wait lists for tasks awaiting access to a data set or device
- Task wait lists for tasks awaiting access to an already active task

Opportunities to service a task in a list occur as a function of the list. CPU ready list members are serviced as higher-priority tasks, leave the system, or are placed on wait lists. I/O wait list members are serviced as I/O completes or tasks release resources. Task wait list members are serviced as tasks complete execution.

PRIORITIES

Control of resources under MPX/OS is on a priority basis, managed through the various lists. Priorities range from 2048 (highest) to 0 (lowest) with ranges 256 through 511 and 1 through 9 reserved for real-time and system tasks. Priorities from 512 to 2048 are reserved for system tasks only. Table 1-4 summarizes the priority scheme and defines the priorities normally assigned to system tasks.

Priorities are maintained on an individual-task basis. Priorities are established with the CALL ESR from executing tasks or with the *TASK control statement from batch jobs. If the priority of the called task (callee) is not made explicit, the callee task inherits the priority of the calling task (caller).

An executing task may cease to execute by issuing an ESR, causing a fault, or by the occurrence of an interrupt beyond the control of the task. In the first instance, the task is entered into lists at the bottom of its priority group, eventually including the ready list. In the second instance (faults), the task reenters the ready list at the bottom of its priority group if control is returned or the job enters the ready list for termination processing at the bottom of the job manager priority group. In the final instance (nonuser interrupts), the task is placed at the top of its priority group in the ready list with one exception: a real-time clock interrupt can result in the task being scheduled at the bottom of its priority group as an installation option effectively creating time slicing.

I/O PROCESSING

MPX/OS provides both logical and physical I/O facilities for data transfer. MPX/OS provides ESRs to perform physical data transfer, device control, and status checking. A system logical I/O routine (blocker/deblocker) can be loaded from the system library with a task to perform automatic blocking/deblocking of data records with single or double buffering and truncating of individual records.

MPX/OS utilizes peripheral devices which are classified as unit record devices or mass storage devices. Unit record devices are serially accessible from only one user job. Mass storage devices are randomly accessible from one or more user jobs. Logical I/O functions are device-type (unit record versus mass storage) independent. Physical I/O functions are definitions for each device type.

The system user accumulates a data set and stores the data set on peripheral equipment. The method of storage differs if the peripheral equipment is a unit record device or a mass storage device, but the method of identifying the data set to the logical I/O routines and the physical I/O executive routines is the same. The data set is identified by a number in the range

TABLE 1-4. TASK PRIORITY ASSIGNMENTS

Priority Use	Priority Use	Tasks
	0	Idle
Low-priority real-time and system tasks	1	Real-time
	9	System Queue Manager (SYSQS)
Real-time and system and non-real-time tasks	10	Job Manager (JMGR,JLDR)
		Non-real-time Real-time
	255	
High-priority real-time and system tasks	256	Real-time
	511	
System tasks	512	All other system tasks
	2048	

of 1 to 63. The number may be called a data set number, a logical unit number, or a logical file number. The logical I/O routine deals with data set numbers (device-independent functions). The executive converts the data set number into a logical unit number or logical file number using tables defined with the user's assistance.

Unit record devices are accessible to the user through the *SCHED , *DEVICE, and *EQUIP control statements in the job control statement deck. The *SCHED control statement reserves the device for the job, and the *DEVICE or *EQUIP control statement connects the data set number to the device and defines the number as a logical unit number.

Mass storage devices are accessible to the user through the file management services ALLOCATE, CLOSE, MODIFY, OPEN, and RELEASE. Job control statements and ESRs by the preceding names are provided. The mass storage capacity of the system is treated as one device from the user's viewpoint, unless explicit action to the contrary is taken. In the default circumstance, the user's data set may reside in "bits and pieces" on several physical mass storage devices, a condition which is transparent to the user. Each of the bits and pieces is called a segment. MPX/OS requires a file to consist of 32 or fewer segments. Each physical mass storage device is assigned a name or device identifier (DID) that can be used in ALLOCATE and MODIFY functions to control the spread of segmented files.

During execution of a job under MPX/OS, unit record devices are secure from access by other jobs because unit record devices are assigned to only one job. Mass storage devices, on the other hand, are normally accessible to all users of the system. Any mass storage file is accessible to any job if four pieces of data are known: the file name code, the file edition code, the file owner code, and the file access privacy code.

In actual use, a unit record device has a defined position and often a variable capacity. For example, the number of physical records a job will be able to place on a magnetic tape is not generally known. Mass storage devices can be accessed in the same sequential fashion as a unit record device, but also provide less rigidly defined positioning (random access) and known capacity. MPX/OS maintains three data values which enable the system to provide the indicated modes of mass storage use: the next block number, the block count, and the number of allocated blocks. The next block number is a position indicator and defines the next block that will be transmitted to/from memory. The block count records the highest block of the file actually written and serves the same function that a magnetic tape file mark serves. The number of allocated blocks is the number of the highest block allocated and serves the same basic function that the magnetic tape end-of-tape (EOT) mark serves.

Use of physical I/O allows the user to format the data in each physical record according to need. Logical I/O provides the same basic format, a system-defined format for all device types (section 5, Blocker/Deblocker).

Data sets can be shared under MPX/OS. For unit record devices, two tasks of the same job could read or write the device. For mass storage devices, this

means that more than one task (not necessarily from the same job) can open the same file for read only at the same time. Each logical file number has its own next-block number. An attempt to share the file for both reads and writes causes tasks to be wait-listed. Only one logical file number from one job can have access to a mass storage file with write permission.

REAL-TIME CAPABILITIES

MPX/OS design emphasizes support of real-time (or time-critical) applications. The primary concept is the ability to respond quickly to time-critical events. MPX/OS provides this fast response by:

- Providing high-priority interrupt recognition
- Allowing tasks to be scheduled with reserved high priority in response to real-time events
- Minimizing CPU time in monitor mode per executive entry
- Servicing I/O requests by priority
- Dispatching tasks for execution by priority

A set of monitor-state registers is reserved for real-time processing use when responding to real-time interrupts. The associate processor interrupts are considered real-time, and, therefore, use reserved machine registers.

Real-time tasks are established in program states and communicate with the executive via the normal MPX/OS ESRs mechanism. All normal MPX/OS services are available to the real-time tasks unless eliminated as a result of real-time environment tailoring of the system.

The real-time environment is established through submission of a real-time job, (section 3, Real-Time Job Statement). Real-Time activity is sustained through task calls, time scheduling and real-time interrupts. Real-time tasks can be executed at high-system reserved priority.

MPX/OS OPERATION

The operating system code is divided into components that execute as part of the executive (from state 0), components that execute as system tasks competing for resources with other tasks, and components that execute from library and user tasks. The division of the system code into dispersed parts serves two purposes: it places the component where the job can be performed with the least overhead and facilitiates prioritized delivery of services.

Figures 1-4 and 1-5 illustrate the normal system flow on the master and slave CPUs. These figures illustrate the system from its functional divisions and do not illustrate the physical divisions to any meaningful extent. The following descriptions of the functional divisions address the physical structure of the system.

The figures show the CPU startup followed by a predominantly counterclockwise loop beginning with the DISPATCHER. The following descriptions proceed according to the same pattern. In addition, the system is maintained as a single copy of core resident code (except for the startup code). The two figures are described in parallel.

SYSTEM AND SLAVE STARTUP

System startup accomplishes CPU initialization (firmware loading), operating system loading, operating system initialization, slave CPU normal activity startup, and master CPU normal activity startup.

Slave startup accomplishes CPU initialization (firmware loading) and slave identity definition, and awaits the signal to start normal activity (section 4, CALL, Establish, and Execute Task).

DISPATCHER

DISPATCH (executive code) selects the highest-priority task ready to execute and gives that task control of the CPU.

Low-priority tasks only obtain service when there are no high-priority tasks or when the higher-priority tasks are unable to execute (awaiting I/O completion, for example).

The initial entry into the normal cycle of execution on a slave CPU causes the IDLE system task to be placed into execution. As the only task, it is the highest-priority task until another task assignment arrives from the master CPU. The initial entry into the normal cycle of execution on the master CPU causes the operator I/O system task to be placed into execution since its priority exceeds that of IDLE system task (see table 1-4).

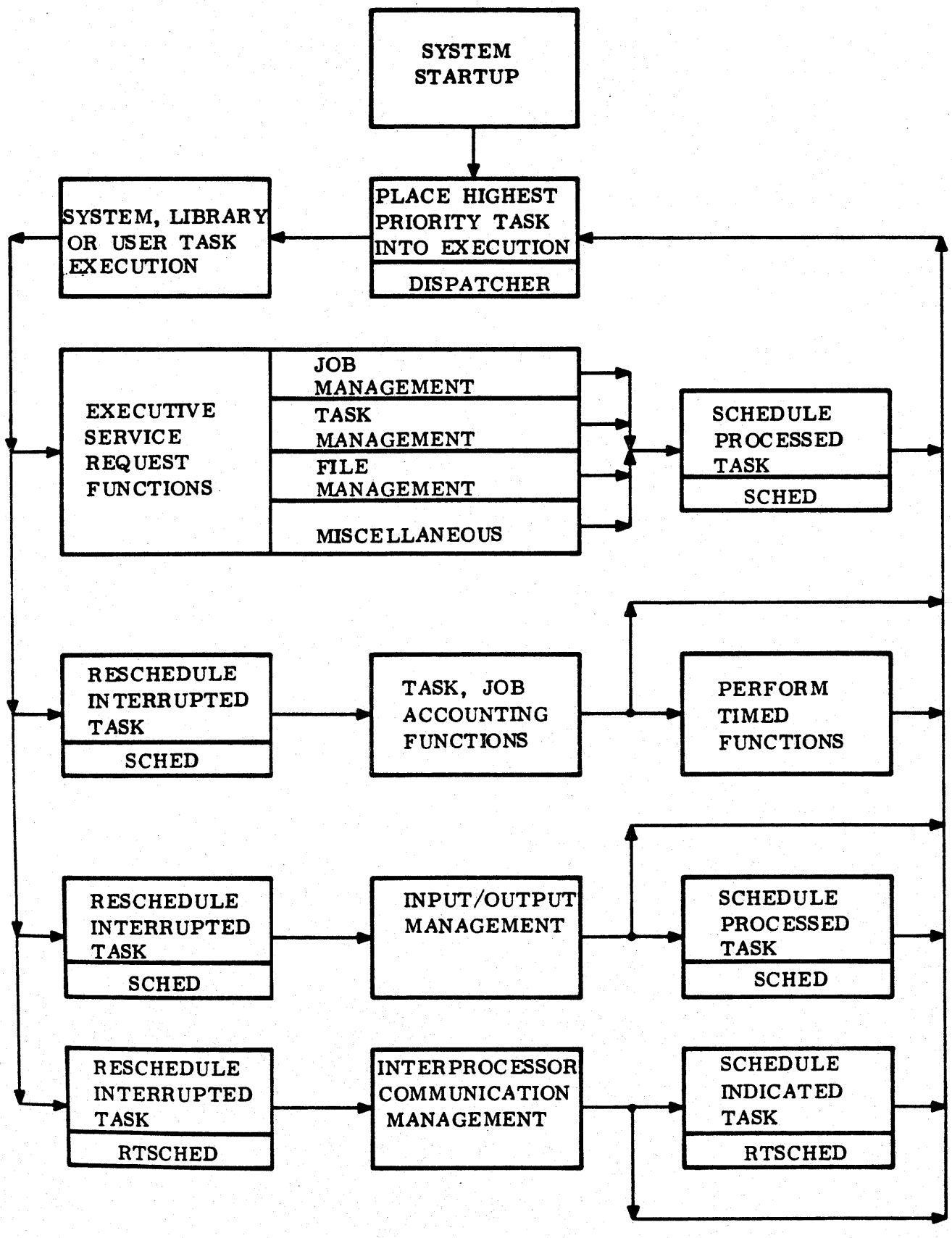


Figure 1-4. Normal System Flow (Master Processor)

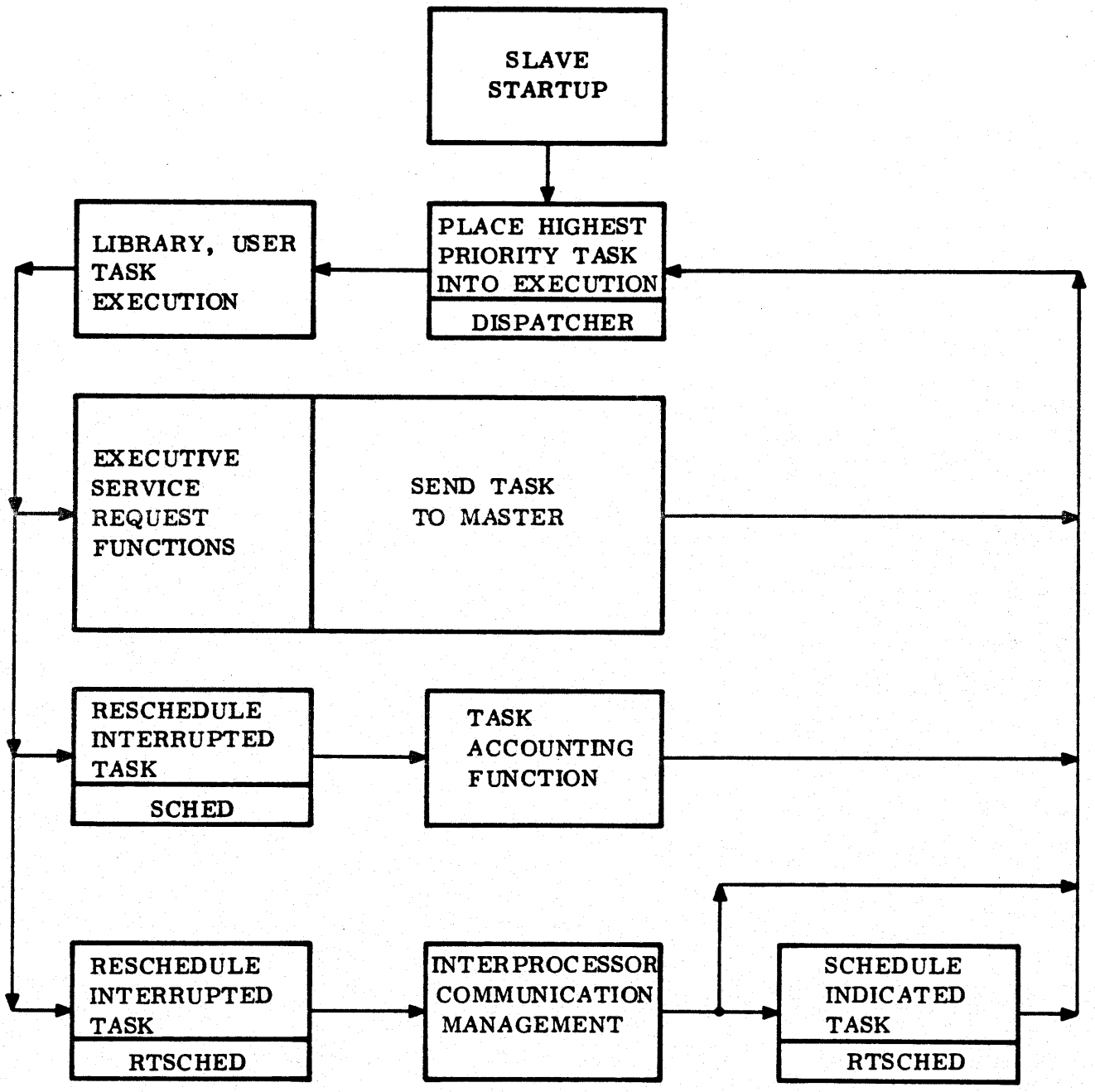


Figure 1-5. Normal System Flow (Slave Processors)

IDLE SYSTEM TASK

The IDLE system task is placed on each CPU ready list so that the CPU always has a task it can execute. IDLE is given control of the CPU when all other tasks are awaiting completion of requested executive services. IDLE frees the executive to await interrupts signalling progress on services underway in other CPUs, or signaling a time interval lapse which may allow a task to be scheduled for execution.

TASK SCHEDULER

Task execution is initiated by the DISPATCHER and continues until the task requests service from the operating system (voluntary interrupt) or until an interrupt condition arrises and the task is (involuntarily) interrupted. A task voluntarily interrupted is serviced and then scheduled (by the SCHEDULER) at the bottom of its priority group. Involuntary interrupts are of two types: faults (task) and nonfaults. If the task fault interrupts out of execution, it is serviced and rescheduled at the bottom of its priority group. If the task elects to regain control (see ENABLE and PFAULT ESRs), it is rescheduled for abort processing at the bottom of the job manager system task priority group (see the ABORT ESR). If the task nonfault interrupts out of execution, it is placed at the top of its priority group and the interrupt is processed (also see Priorities in this section).

Two entries, SCHED and RTSCHED, perform an identical function; they place a task on the CPU ready list. Two copies are required because the executive can be interrupted to service real-time interrupts, including the scheduling of real-time tasks for execution.

Two system functions are normally loaded with user tasks: the task monitor and the blocker/deblocker modules. Both are obtained from the system library file.

The task monitor provides the task entrance, a task exit, and the task-system communication area. The task-monitor entry point is the starting point for task execution. It immediately passes control to the user task main entry point. The task monitor is inserted to allow for main programs which exit with a normal subroutine return sequence instead of with a RETURN ESR. If the task returns to the task monitor, the task monitor issues a RETURN (with release) ESR to bring about a normal task termination.

The standard input (INP), standard output (OUT), and standard punch (PUN) files are required to have a specific format. The format of these files is generally processed by a collection of subroutines, supplied by MPX/OS, called the blocker/deblocker modules (see section 6, Blocker/Deblocker). The standard file buffers in logical page 15 are maintained by the blocker/deblocker modules. Page 15 is the same physical page for all tasks

of the job (see figure 1-2). Blocker/deblocker also maintains tables in logical page 15, which are used to control the blocker/deblocker functions and to ensure that only one task is reading or writing the same file at the same time.

JOB MANAGEMENT

Job management is totally a system function. It is carried out in large measure by system tasks. Job management accomplishes defining a job in the system, identifying and initiating job requested work (job manager system task), and returning the job output.

STANDARD INPUT/OUTPUT

Standard I/O (STDIO) obtains job control decks from one or more card readers. When the card reader is empty, a message is sent to the operator. When the operator responds to the message, Standard I/O again processes card reader data. Job control decks are saved in a file for submission to the System Queue Manager, SYSQS.

The Standard I/O post processor prints the OUT file to one or more printers and is activated by the JOB terminator.

SYSTEM QUEUE MANAGER (SYSQS)

SYSQS is a system task whose primary function is the management of the system queues (that is, INPUT, OUTPUT, and HOLD Queues). Jobs are prepared as standard input files and sent to SYSQS via the ROUTEQ ESR. SYSQS determines a job's eligibility for execution by checking for resource availability. If the job specifies resource requirements in excess of the system maximum, it is rejected. Otherwise, if a job is already waiting on the input queue, subsequent submittals are threaded by priority onto the queue.

The input queue is a disk file containing Job Control Table (JCT) information. This JCT information is derived primarily from the *JOB and *SCHED control cards.

The SYSQS will place the job at the top of the queue into execution after resource requirements have been met. A job number is obtained and placed in the JCT to uniquely identify the job. The time of day is acquired to define the time that job execution begins. Then, the job manager is established and scheduled.

Non-system tasks can communicate with SYSQS through the ROUTEQ ESR. The ROUTEQ ESR provides the capability to ADD or DELETE job files on a system queue. The ROUTEQ ESR is described in more detail in section 4.

JOB MANAGER SYSTEM TASK

The job manager* assumes control of the job until normal or abnormal job termination. Job termination is complete when the job accounting information has been summarized and written on the OUT file and the standard files have been closed or released. The job manager exits by making an executive service call, which releases the job resources to the system and assigns the OUT file to the STDIO system task for post processing.

The job manager contains the necessary routines to process the user's control statements from the standard input unit. The user's job is processed in four phases: PRELOAD, LOAD, POSTLOAD, and EXECUTE.

The PRELOAD phase provides the user with the ability to allocate files, to assign logical unit numbers to files and peripheral equipment (tape units, displays, etc.), to perform tape handling functions, and to communicate with the operator.

The LOAD phase allows the user to create a task from binary modules contained on a logical unit or contained on a system library.

The POSTLOAD phase provides the facilities to perform tape handling functions, to communicate with the operator, to create an absolute copy of the task upon a logical unit, and to modify the task using hexadecimal corrections cards.

The EXECUTE phase is the execution of the user's task. If the task terminates normally, control returns to the preload phase.

MPX JOB FLOW

Figure 1-6 illustrates the job flow of MPX. Following system initialization (DEADSTART), the SYSQS routine receives control (step 1, figure 1-6) and job manager searches for a JOB statement on the standard input device, INP. If the JOB card is valid, the SCHED statement is processed. A job is placed into execution if sufficient system resources are available to meet the requirements of the SCHED control statement. Control is then passed to the job monitor.

---

* The externally observable features of the job manager are the subject of section 3 and are not described here.

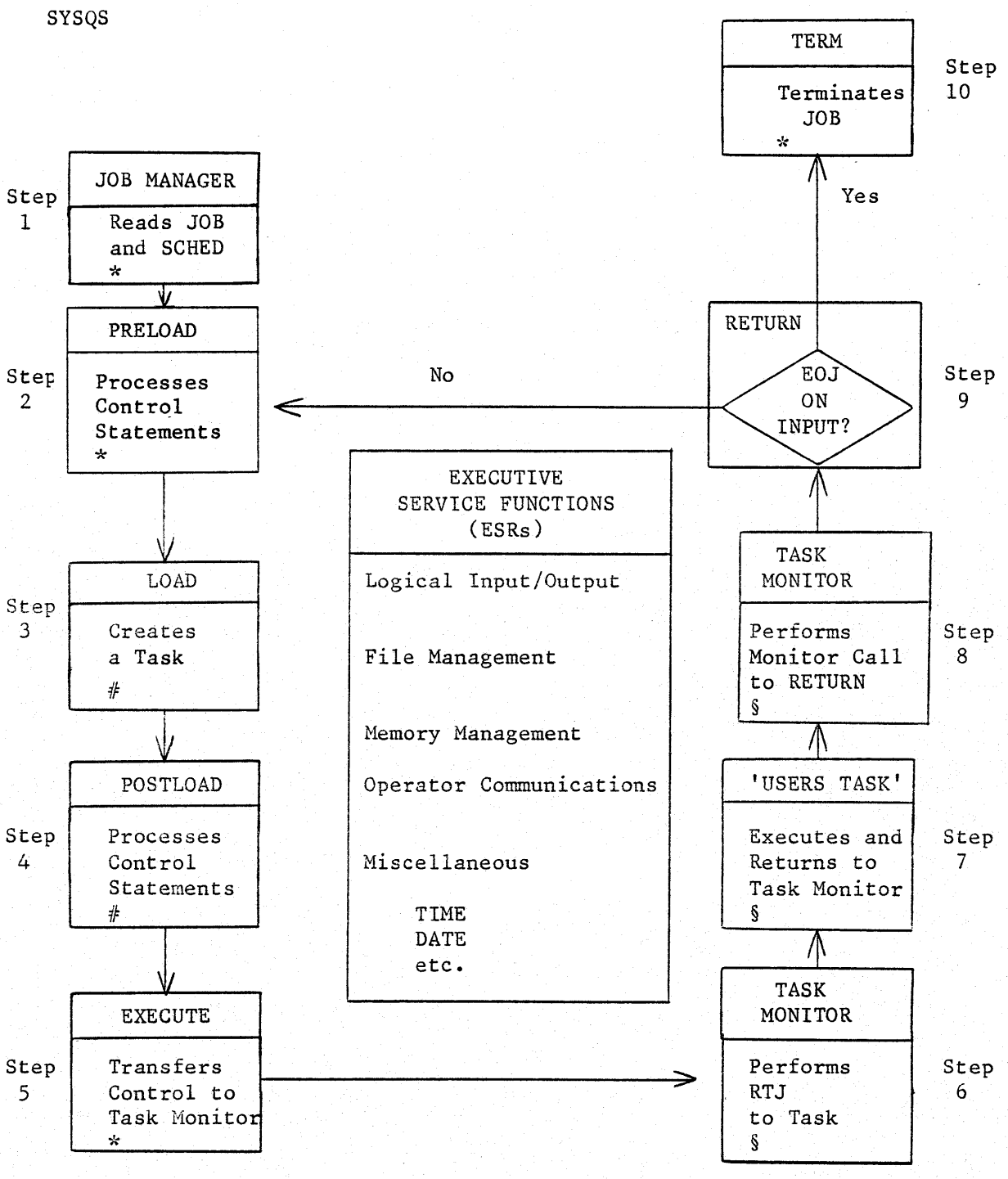


Figure 1-6. Job Flow

The PRELOAD phase (step 2) of the job manager processes control statements until a LOAD card or library name card is encountered. PRELOAD utilizes the executive services functions to process most of the statements.

The LOAD phase (step 3) creates a user task by loading the specified programs. Logical memory is assigned on a demand basis by calling the memory management services functions in the executive.

The POSTLOAD phase (step 4) processes control statements until a RUN card is encountered. The RUN processor passes control to the executive control function, EXECUTE (step 5). The legal control cards during the POSTLOAD phase are REWIND, UNLOAD, CTO, PAUSE, ABS, HCC, and RUN statements.

EXECUTE updates the job control table to indicate that the user has control and passes control, via an exit monitor instruction, to the TASK MONITOR.

The TASK MONITOR (step 6) passes control to the user's task via a return jump instruction to the task's last transfer address. A transfer address is an entry point in a program specified on the END card (see COMPASS assembler).

The USER TASK (step 7) terminates by returning to the TASK MONITOR through the linkage provided by the return jump in the TASK MONITOR.

The TASK MONITOR (step 8) then passes control to the executive control function, RETURN.

The RETURN function (step 9) returns control to the job manager (step 2).

If an end-of-file on the standard input device is encountered by the job monitor, control is passed to the executive control function, TERM (step 10). TERM releases the job's resources to the system and passes control to the operating system.

INTERACTIVE TERMINAL SUBSYSTEM

The Interactive Terminal Subsystem (ITS) extends the full range of MPX/OS features from on-site batch to terminal access. The ITS 2.0 provides operator facilities to the remote terminal user and interactive capabilities to on-site operator. The ITS 2.0 is intended to be used for both software development and applications. Typical applications of the ITS 2.0 for software development are:

Source program entry
Source program maintenance
Compilation and assembly
Listing inspection
User task execution
User task debug
Documentation generation

TASK MANAGEMENT

Task management (executive code) establishes tasks in the system, manages intertask activities, and manages task access to the CPU. Establishing a task involves defining the task in system tables (TCT) and ensuring that the task is loaded into main memory. Managing intertask activities involves the CALL, RETURN, TSTATUS, and DWAIT ESRs. CALL establishes and initiates execution of new tasks. RETURN signals the end of a task execution (for a specific CALL). TSTATUS allows one task to determine the status of another task. DWAIT allows a task to suspend operation until one or more called tasks have completed execution. Managing task access to the CPU involves task scheduling, dispatching, memory limit changes, task termination, task suspension, and task fault control recovery.

FILE MANAGEMENT

The file manager is a system task activated by MPX/OS to service ALLOCATE, CLOSE, EXPAND, MODIFY, OPEN, and RELEASE functions. The servicing of such requests may involve disk reads/writes and task queueing, which result in unpredicatable patterns of service completion. An execution of the file manager services one task. Other queued tasks must await the next entry to the executive. A first task may request service and be queued, allowing a second task to request and receive service while the first task waits. File manager execution time is charged to the job for which the function is provided.

TASK ACCOUNTING

Task accounting (executive code) is simply accumulating CPU execution time on a task basis. File management time is charged directly to the job. Accumulated task-execution time serves as a task clock and can be used for task-performance analysis (section 4, Executive Service Requests).

JOB ACCOUNTING

Job accounting (executive code) consists of accumulating the task CPU times as tasks terminate, and of testing for job time limit being exceeded. Also, resources such as memory, mass storage scratch, print lines, and punch cards reserved and not used are maintained and summarized on the job's OUT listing.

TIMED FUNCTIONS

MPX/OS periodically totals the job accumulated times and the outstanding task accumulated times and compares the sum to the time limit defined on the *SCHED control card. When the sum exceeds the limit, the job is aborted.

MPX/OS schedules tasks for execution when requested time intervals lapse.

I/O MANAGEMENT

I/O management (executive code) controls the access to devices and mass storage files. The I/O manager accepts the data set number from the ESRs, determines the device or file, and administers the delivery of resources to the requesting tasks on a priority basis.

A requested service may involve several distinct entries to the I/O management modules. When all required steps have been completed, the requesting task may need to be rescheduled for execution (section 4, UST, Unit Status Test).

INTERPROCESSOR COMMUNICATION MANAGEMENT

Figure 1-7 illustrates the flow of a service request originating on the master CPU and on a slave CPU. Requests that originate on a slave CPU

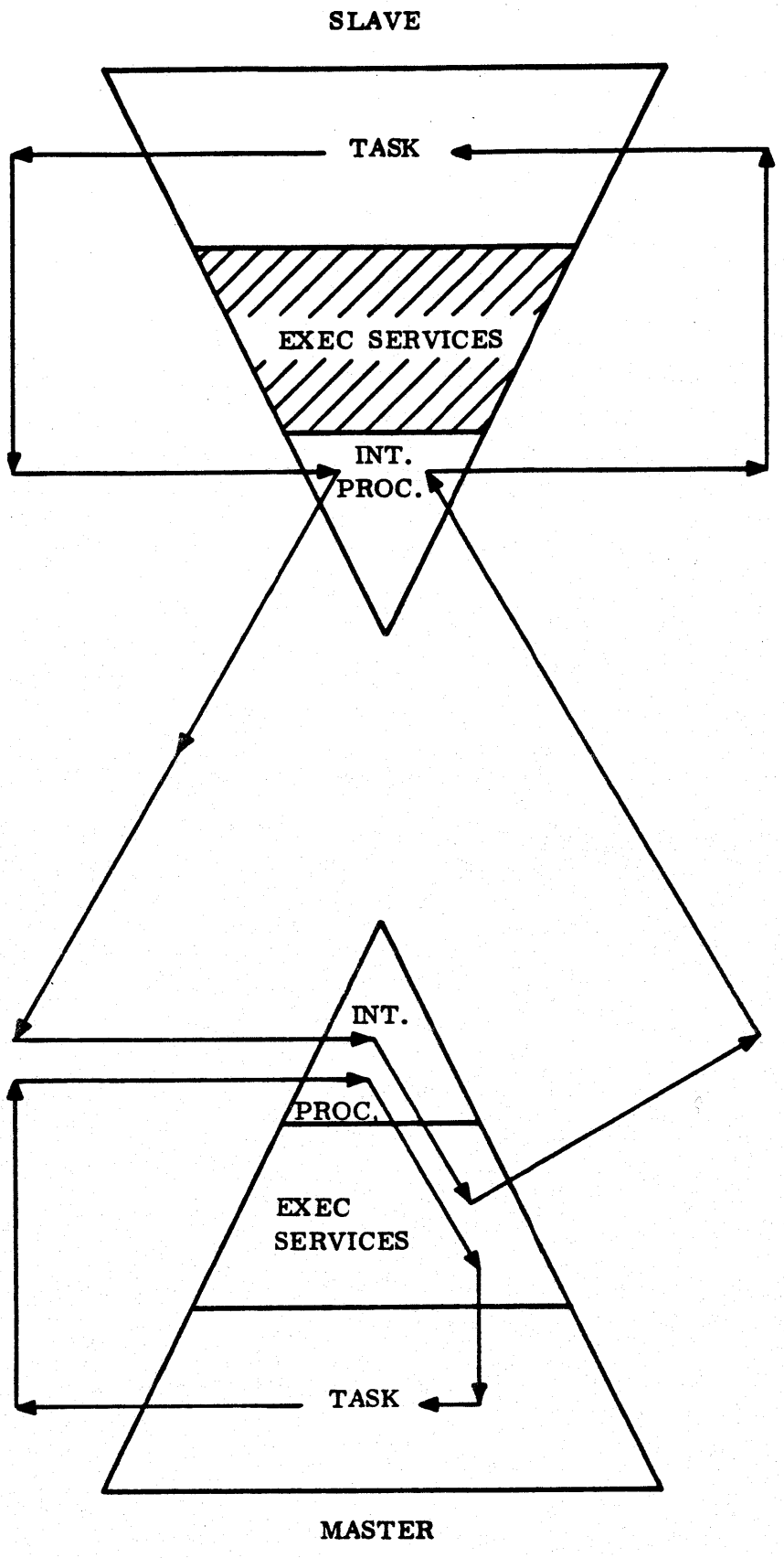


Figure 1-7. ESR Processing Flow

(upper triangle) are recognized by the slave (INT PROC - interrupt processing) but are routed to the master for service. After servicing is complete, the task status (READY for execution) is relayed back to the slave, causing the task to appear on the slave ready task list.

While on the master CPU, the task is placed in the master ready list so that the granting of services can be accomplished according to priority.

Requests that originate on the master CPU more directly enter the executive for servicing. Since the task was executing, it is the highest-priority task at that time. After the service is supplied, the task is placed on the master ready list.

The executive may be interrupted by real-time (master only) or associate CPU interrupts, but only to schedule a task for execution. That is, once the processing of a service request has started, it runs to completion or to a standard point of suspension (for example, waiting for file access).

Note that since every exit from the executive is through the DISPATCHER, every interrupt and every ESR provides an opportunity for a higher-priority task to obtain control of the CPU.

ABORT PROCESSING

MPS/OS enters the executive job abort processor (JABRT) when an abnormal condition occurs.

JABRT effectively idles the job and all its tasks prior to proceeding with the abort processing. The job manager termination processor (JABT) is scheduled by JABRT to complete the abort processing and create a special dump.

A special case may occur when an abnormal condition exists during the job abort processing. A secondary abort entry in job manager (JABT2) is provided to give an abbreviated abort in this event. An example might be an I/O conflict between a task and the abort processor.

OPERATOR COMMUNICATIONS

Operator control of the MPX/OS operating system is handled through extensions of the ITS 2.0 Subsystem. Any ITS terminal can act as an operator console if sufficient security permissions are granted to that terminal/user combination. Only the System Deadstart function is restricted to the Operator Console.

MEMORY MANAGEMENT

MPX/OS manages units of memory termed pages, where each page contains 4096 32-bit words. The paging hardware translates logical addresses into physical addresses allowing each program state to access potentially all of physical memory and providing protection against illegal memory references.

The executive and system task portions of MPX/OS reside in the lowest physical pages. This is because certain addresses representing hardware interfaces (for example, interrupt addresses) must be fixed in low memory.

The remaining memory pages not reserved by MPX/OS are available for task loading.

Each job must reserve its maximum memory requirements on the SCHED statement. Actual memory usage may increase and decrease during a job's life, but can never exceed the scheduled amount.

The use of global common presents some special memory considerations. Pages used for global common do not actually count as memory scheduled. This is because these pages are actually only mapped into a task's address space. MPX/OS does not allow any combination of task-executable code, local common (addressable only by defining task), or global common to exceed 15 pages. Actually, a total of 16 pages is addressable by a program state, but one page is needed for the intrajob data area.

GLOBAL COMMON

Global common satisfies the requirement for a data base shared by several tasks. In addition, global common may be retained when the system is restarted (reloaded) following a failure.

Global common can be defined to the operating system at the time of system build or defined dynamically by user tasks (ASNGC). At time of definition, memory is reserved and tables containing the names, lengths, and addresses of each global common block are initialized. Each global common block must start on a memory page boundary and must be a multiple of pages in length.

ESRs are provided to allow a task to manage its access to global common. The ESRs allow a task to status (STATGC), attach (GETGC), and detach (RETGC) global common blocks. Status allows a user to dynamically manage global common assignments.

ESRs for memory expansion and reduction, OPENMEM and RELMEM, prevent a conflict with global common assignments. Task memory pages can not be assigned as global common memory and dynamic local task memory simultaneously.

A reserved scratch common name (GLOBAL) is used to declare arrays for global common. This does not preclude declaration of local (memory referenced only by declaring task) common. The loader searches for GLOBAL and aligns the first global common array to the next page boundary. When obtaining memory, the loader ensures that memory is not allocated for global common. Therefore, the user need not reserve memory for global common (CM parameter on *SCHED control card).

The reserved scratch common block name GLOBAL is used by the programmer to signal the beginning of global common block declarations. Scratch common blocks encountered by the loader before the occurrence of the common block name GLOBAL constitute local scratch common. All subsequent scratch common blocks including GLOBAL will start on a page boundary and comprise global common. An example of assembly coding is as follows:

```
                SCOM
A               BSS     100
B               BSS     100
BLOCK 1         SCOM
C               BSS     4096
GLOBAL          SCOM
D               BSS     2048
E               BSS     2048
BLOCK 2         SCOM
F               BSS     8192
```

The same example in FORTRAN is as follows:

```
SCRATCH COMMON A(100), B(100)
SCRATCH COMMON BLOCK1/C(4096)
SCRATCH COMMON GLOBAL/D(2048), E(2048)
SCRATCH COMMON BLOCK2/F(8192)
```

The examples generate two pages of local scratch common and three pages of global scratch common. Global common would start at logical address $2000. Global common blocks could be mapped into addresses $2000 through $4FFF and referenced by the arrays D, E, and F.

SCHED AND RTSCHED

Two modules (executive code), SCHED and RTSCHED, perform an identical function - they place a task on the CPU ready list. Two copies are required because the executive can be interrupted to service real-time interrupts, including the scheduling of real-time tasks for execution.

SECURITY CONTROLS

MPX/OS security controls are composed of tables and procedures for the protection of data from unauthorized access.

MPX/OS security controls are intended to prevent access to MP-32 Computer System resources by nonvalidated users. These controls are based on information obtained from three sources: initial system tables, system files, and operator entry.

The MPX Operating System maintains security information in the following tables and system files:

System security control (security control mask)
Peripheral control tables (security control mask)
Job control tables (job security level)
Task control tables (task current security level)
Mass storage directory (file security level)
User validation file (interactive user security levels)
Port setup file (remote circuit security levels)

The system security mask is initialized at system installation but can be changed by the operator during MPX operations. The system security mask is a global value levied on all system functions by the operator. No accesses are allowed which are in conflict with the system security mask. If the operator attempts to alter the value while a job requires resources which are in conflict with this value, the operator is informed and the system security mask is not altered.

The peripheral control tables contain the security mask for each peripheral device within the system but only the user assignable unit record equipment security masks can to be altered by the operator. The others (mass storage, communications line adapters, etc.) cannot be altered and are controlled by other system tables and files. If the operator attempts to alter a non-unit record device, the operator is informed and no changes are made.

The task control tables contain security level information pertaining to an associated task. The MPX system will use this security level information to control resource allocation to a task. Both the task and the operator may change a task's security level but in no case will the level be allowed to

conflict with the system security mask. If an attempt is made to alter the task's security level in a manner which makes it conflict with the system security level or with resources already obtained, then the task/operator is informed and no change is made in the task's security level.

The mass storage directory contains the information necessary to access the data contained within a file. This directory also contains a use parameter which defines the control level of the file. The use field may be one of the following control types:

Read/Write
Read only

The user validation file is used to control the interactive user access to the MPX Operating System. This file contains the user validation parameters described in the ITS feature description. These parameters include facility access, security level, and username/password combinations.

The MPX/OS port setup file contains configuration and validation parameters for each communications line serviced by the MPX system. The port validation mask parameter contains the security control mask, and when the port validation mask is combined with the user validation mask, the resulting mask is used to limit the types of accesses the user/port can perform.

FILE STRUCTURE 2

The MPX/OS system operates in an environment in which all files have an identical basic structure. All mass storage for MPX/OS is subdivided into two levels. The device label is the higher level and represents the on-line units in the form of disk drives and disk packs. The unit of allocatable storage is the lower level and represents a multiple of physical hardware records (sectors).

DEVICES

Mass storage devices are hardware entities with independent schemes of addressing. MPX/OS distinguishes between devices which are logically or physically affixed to drives (system devices) and devices which are removable (user devices).

System devices must be on-line at all times. User devices need be on-line only when the device is referenced by the user (ALLOCATE, OPEN, etc.). System devices are defined by system installation.

DEVICE LABELS

MPX/OS uses device labels to identify all mass storage devices. Each mass storage device has a device label written on its first hardware address. Device labels are written by the utility routine INSTALL prior to using the device.

Device labels contain information pertaining to mass storage devices, including a device identifier (DID), which is used for internal and external identification, and a device allocation map, which identifies the used and unused allocation units.

The content and format of device labels are described in appendix G. The physical characteristics of various devices are described in appendix F.

FILES

All mass storage data operated on by MPX/OS must be in entities of logical block structure. These entities are called files. A logical block size is the number of 32-bit words in each block. Each logical block starts at the beginning of a physical hardware record.

FILE LABELS

File labels are entries in the system LABEL file that identify, describe, and reserve space (files) on mass storage. A mass storage file exists in the system when the user defines a label (allocates a file). The user makes a request to MPX/OS to create a file label via an ALLOCATE call. This call provides the file identification, access code, block size, block count, and so forth. MPX/OS uses the caller-supplied information to create a file label and to update the allocation map of necessary device labels. File labels are described in appendix G.

FILE IDENTIFICATION

File name, edition, and owner make up a file identification. The file label contains the file identification for MPX/OS comparison during label modification calls (RELEASE, MODIFY). If identification in a call does not match identification in a label, an error results.

FILE ACCESS PRIVACY

Each file label has a provision for an access privacy code and an access type code (USE). The access privacy code protects a file from unauthorized use. If the access privacy code in an access call (OPEN) does not match the one in a file label, the call is rejected. The access type code allows the user to specify the file as read-only. If the access type code is read-only in the file label, USE in the OPEN call must be read-only.

FILE SEGMENTATION

When space must be segmented on mass storage to satisfy a file allocation call, MPX/OS allows files to contain up to 32 segments. One or more segments of a file can be on one or more devices. MPX/OS allows a file to be segmented on to a maximum of eight devices.

When allocating space for a file, the user can specify that the space be contiguously allocated. If insufficient contiguous space is available on the device, ALLOCATE rejects the request.

FILE ALLOCATION METHOD

ALLOCATE, EXPAND, and MODIFY (expand file) assign space sequentially on a device basis beginning with the first specified device; however, when allocating space on a specific device, ALLOCATE, EXPAND, and MODIFY check the device label map to find the smallest contiguous area large enough to satisfy the request. If such an area does not exist, the largest available area becomes the first segment, followed by the next largest, and so forth.

Job processing flow is illustrated by figure 3-1. The processing of a job is initiated by submission of the job deck to the input queue. From this point on, MPX/OS assumes control of the job.

The operating system task, Standard I/O (STDIO) reads job files and places them in mass storage files. STDIO or Interactive Communication Facility (ICF) causes mass storage files to be passed to the system queue manager (SYSQS) for inclusion in the input queue. SYSQS then examines the job and schedule statements, determines the resource requirements for the job, and attempts to secure the necessary resources. When all of the resources have been acquired, SYSQS initiates execution of the job manager task (JMGR). SYSQS is now free to process any new jobs that might appear in the input queue.

JMGR reads and interprets the job control statements in the sequence they appear in job deck (the mass storage file). These control statements consist of a statement name and the parameters necessary to define the operation. The specified operations allow for the management of the peripheral environment of a job, for the loading and execution of user and library tasks (TASK1 ... TASKn), and for job termination (JTRM).

Control statements contain an asterisk(*) in column 1, followed by the requested function name. The parameter list extends through the remaining columns of the statement. The parameter list is enclosed in parentheses, with commas separating each parameter. Comments are permitted on the control statements, but must follow the corresponding parentheses that terminate the parameter list. A control statement which contains a pound sign (#) in column 1 is treated entirely as a comment (they can be placed anywhere after the *SCHED card).

The job statement file must be organized in three sections: job definition, job activity, and job termination. Figure 3-2 defines a job statement file and identifies the three sections. The job definition section contains sufficient information to define the job in the system. The job activity section manipulates the peripheral environment and causes task executions.

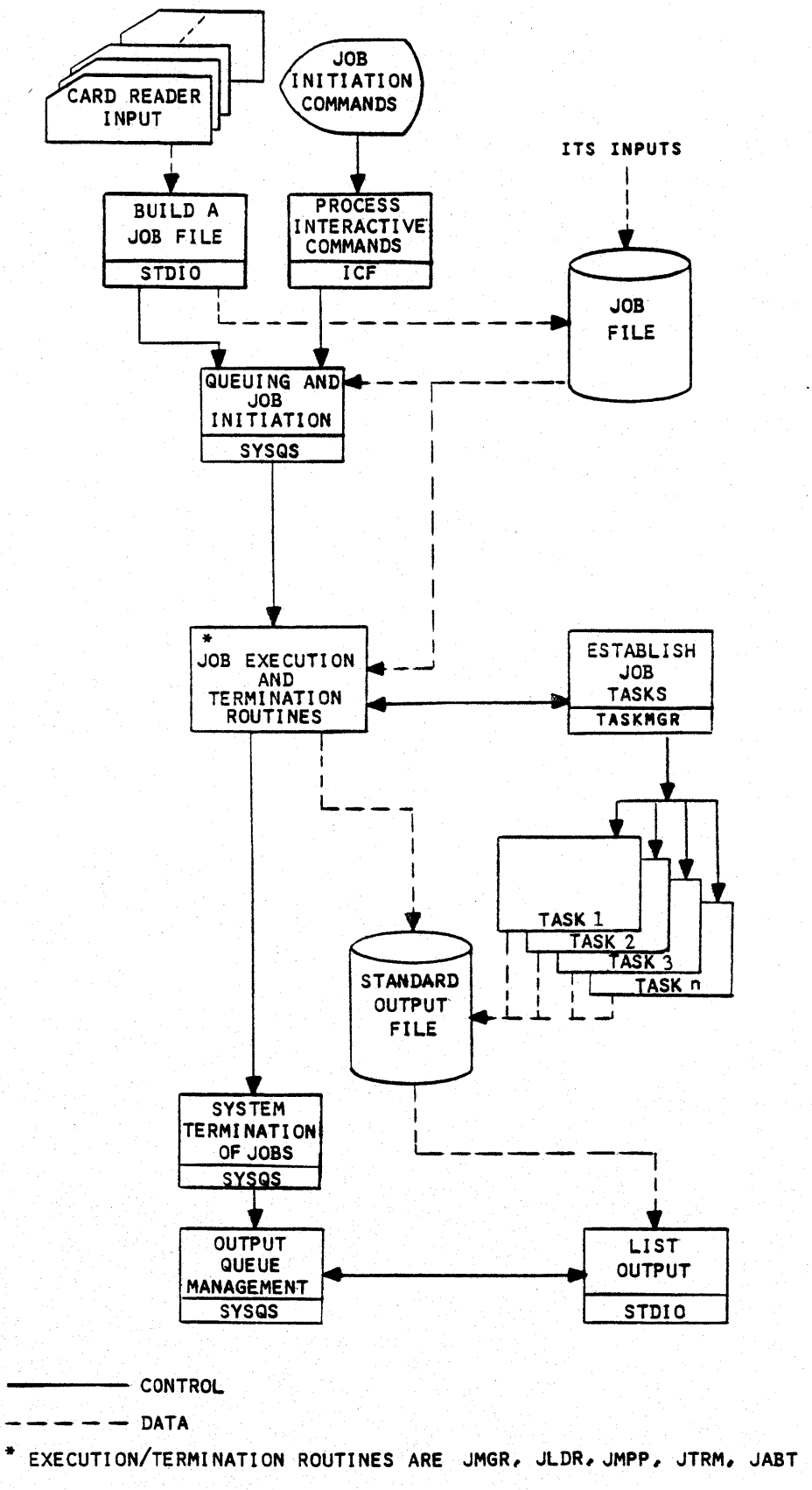
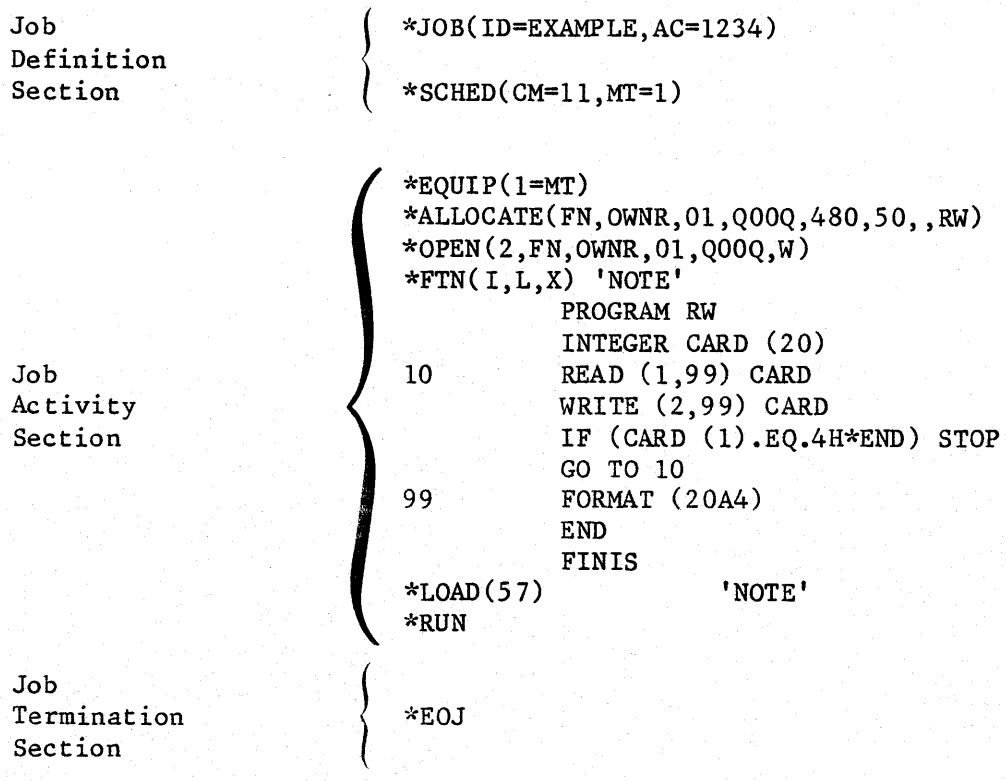


Figure 3-1. Job Processing Flow



'NOTE' X without any parameters assumes standard load and go LUN=57. *LOAD without any parameters also assumes standard load and go.

Figure 3-2. Batch Job File Example

The job termination section releases resources assigned to the job, adds job accounting information to the OUT file, and eliminates the job from the system. The jobs OUT file can be released or retained for user disposition, depending on an option selected by the ROUTEQ ESR.

JOB DEFINITION STATEMENTS

The job definition statements characterize the job as a real-time or non-real-time job and identify the resources required to complete the job successfully. The job is not started until all required resources are available. If insufficient resources are requested, the job is aborted when a request for the undeclared resource is encountered.

*JOB, Non-Real-Time Job Statement

```
*JOB(ID=    ,AC=   ,UN=   ,SC=   ,QP=   ,TP=   ,OU=)
```

A non-real-time job (*JOB) statement serves as identification of a non-real-time job, and of an input file. Any additional job statements are ignored if they are encountered before the end-of-job statement (*EOJ).

Parameter	Definition
ID=	1 to 8 characters indicating the job identification. This parameter is optional. If omitted, ID=.JOB. is supplied by MPX/OS.
AC=	1 to 8 characters indicating the job account number. This is an optional parameter. If omitted, defaults to the account number of the submitter or .ACCT. if source is the card reader.
UN=	1 to 8 characters indicating the Username of the job owner. If omitted, defaults to the Username of the job submitter or .UN. if source is the card reader.
SC=	1 or 8 characters indicating the security level or security mask for this job. An installation option determines the format of this parameter.
QP=	Priority at which job will be placed on the system queues (Input, Output, Hold). Decimal constant in the range from 0 to 32751.
TP=	Default priority for all tasks established by job. Decimal constant in the range from 10 to 255.

Parameter	Definition
OU=	Disposition of job's OUT file. Disposition code is 2 ASCII characters. This parameter overides disposition supplied by submitter.

*RJOB, Real-Time Job Statement

```
*RJOB(ID=  ,AC=  ,UN=  ,SC=  ,QP=  ,TP=  ,OU=)
```

The real-time job (*RJOB) statement replaces the *JOB statement when identifying a real-time job. The parameters are identical to those of the *JOB statement.

A real-time job is expected to cause a real-time task to be loaded. The real-time job differs from the non-real-time job in its ability to use the reserved priorities (1 through 9, 256 through 511) for its tasks. The real-time task is established and control is passed to the task. After the real-time task has initialized itself, it returns to MPX/OS without releasing its resources.

*SCHED, Schedule Statement

```
*SCHED(CM=  ,TL=  ,PL=  ,PC=  ,SCR=  ,hh=)
```

The schedule (*SCHED) statement, if present, follows the job statement (*JOB or *RJOB), and is used to allocate and reserve resources. The value of the last appearance of a parameter is the one used.

Parameter	Definition
CM=	Memory limit (in pages) assigned to the job. The upper limit for this parameter is dependent on the amount of physical core memory available. The parameter can be omitted, in which case an MPX/OS-defined limit is applied to the job.
TL=	Job time limit in CPU seconds, value from 1 to 99999. A value of 99999 is regarded as infinity. Time charged against this limit is CPU usage only. The parameter can be omitted, in which case an MPX/OS-defined time limit is used.
PL=	Print line limit assigned to job, value from 0 to 65535. The parameter can be omitted, in which case an MPX/OS-defined print limit is used.
PC=	Punch card limit assigned to job, value from 0 to 65535. The parameter can be omitted, in which case an MPX/OS-defined punch limit can be used.

Parameter	Definition
SCR=	Maximum total number of mass storage segments to be shared among system scratch 1 (SCR1), system scratch 2 (SCR2), standard Hollerith scratch (SHC), and standard load and go (LGO) for a job. The size of a segment is a system parameter. This parameter can be omitted, in which case an MPX/OS-defined scratch limit is used.
hh=	The number of this type of peripheral equipment to be reserved for the job, where hh has the following definitions.

Mnemonic	Hardware Type
CCC	CYBER Channel Coupler
FDD	Flexible Disk Drive
MT7	7-track magnetic tape
MT	9-track magnetic tape (800bpi)
MT9	Same as MT

CATALOGUED JOBS

A catalogued job is a file in standard blocked form; the file contains valid batch job deck card images, *JOB or *RJOB through *EOJ. The catalogued job's feature is the mechanism used to support task initiation of jobs.

A catalogued job submitted for execution will terminate without destroying the input file. This allows concurrent and/or repetitive submissions of a single, catalogued job without redefinition.

A system build-time parameter controls output file disposition for catalogued jobs initiated through the operator's console. The job output file can be retained for printing or the output file can be released without printing according to the definition of the build-time parameter.

Catalogued jobs can be submitted to the operating system from the following sources:

Operator command
Executing user tasks

Each submittal source must prepare a file containing a complete job deck (*JOB or *RJOB through *EOJ). The ROUTEQ ESR processing will initiate job execution (if all required resources as defined by the *JOB or *RJOB and *SCHED cards in the submitted file can be allocated), will add the job to the input queue, or will reject the request. Immediate execution will occur if resources can be allocated and if the input queue is empty or if

preemption is selected. Queuing can occur if jobs are already waiting in the queue or if all resources are not available. A reject will occur if the input queue is filled or if the ROUTEQ ESR specified a reject in place of queuing:

A first in, first out queue is retained in memory to expedite the resources available check. Established operating system approaches dictate that the submit processing be divided into executive state and task state processing.

The executive state portion manages access to the task state portion and provides the parameter passing function between the caller and the task state services. The task state portion provides for the disk I/O processing and sequencing through a series of requests. The task state portion allows continued system operation while doing the system task of processing submitted jobs.

JOB ACTIVITY CONTROL STATEMENTS

The job activity section of a job consists of four types of control statements: miscellaneous, data set identification, data set modification, and task preparation and use control statements. A job normally has at least one of the control statement types but need not have each type represented.

MISCELLANEOUS STATEMENTS

The miscellaneous statements allow messages and action requests to be sent to/received from the operator through the console display.

*CTO, Comment-to-Operator Statement

```
*CTO message
```

The comment-to-operator (*CTO) statement causes the message appearing on the statement to be output on the console display. The *CTO card can appear anywhere in the control statement deck between the *SCHED card and the end-of-job card, except among the task data areas.

*PAUSE, Pause Statement

```
*PAUSE message
```

The pause (*PAUSE) statement causes job processing to be suspended. The message is copied to the console display. The operator then performs the requested action and continues the job by acknowledging the message. If the message is rejected, the job is aborted.

DATA SET IDENTIFICATION STATEMENTS

The data set identification statements associate a logical unit number with a data set. For a mass storage file, an *ALLOCATE/*OPEN statement sequence, or an *OPEN statement is used. For a unit record device data set, a *DEVICE or an *EQUIP statement is used. A new logical unit number can be defined as being equivalent to an already-defined logical unit number with a *DEVICE or an *EQUIP statement.

*ALLOCATE, Allocate Statement

*ALLOCATE(FN,OWNER,ED,AK,BLKSIZE,NOBLKS,S,USE,SLVL,DT,DID1,...,DIDn)

The allocate (*ALLOCATE) function is used to describe (and thus create) a file in the mass storage system. Once a file has been created, it remains allocated until released.

Parameter	Definition
FN	1 to 14 characters specifying the file name.
OWNER	1 to 4 characters specifying the file owner.
ED	1 or 2 characters specifying the edition number.
AK	1 to 4 characters specifying the access privacy key. This field is not copied on the job's OUT file.
BLKSIZE	Number of words in a logical block. Decimal constant in the range of 1 to 65535.
NOBLKS	Number of logical blocks in the file. Decimal constant in the range of 1 to 65535.
S	Segmentation flag:

Entry	Meaning
Blank	File can be segmented.
S	File can be segmented.
NS	File cannot be segmented.

Parameter	Definition
USE	Protection flag:

Entry	Meaning
Blank	File can be accessed as read/write.
R	File can be accessed as read only. It can not be written until modified.
RW	File can be accessed as read/write.

SLVL — Security level. Decimal constant in the range of 0 to 7.

DT — Device type:

Entry	Meaning
Blank or 0	File allocated on system device.
1	CONTROL DATA 9425 Cartridge Disk Drive.
2	CONTROL DATA 844 Disk Storage Unit.
3	CONTROL DATA 9427 Cartridge Disk Drive.
4	CONTROL DATA 1867-10 Storage Module Drive.
5	CONTROL DATA 1867-20 Storage Module Drive.

DID — 1 to 8 characters identifying the device to be used for the file. Up to eight devices can be specified.

The parameters must appear in the indicated order with omitted parameters specified by adjacent commas.

When *ALLOCATE detects an error, the entire control statement is ignored and a diagnostic is written on the job OUT file.

***DEVICE, Assigning Unit Devices**

*DEVICE(lu=hhhh,ID=name,...lu_n=$hhhh_m$,ID=$name_n$)

***EQUIP, Assigning Devices**

*EQUIP(lu=hhh,...lu_n=hhh_m)

The device assignment statements, *DEVICE and *EQUIP allow a hardware type to be assigned to a logical unit number.

Parameter	Description
lu	The logical unit specified can be a number 1 through 63 with the exception of the fixed assignments presented in figure 3-3.
hhh(h)	The mnemonics used for device identification devices can be found in appendix K, Valid Hardware Types.
name	Device name is an ASCII 8 character identifier for a specific device. Device names which contain blanks must be enclosed in single quotes. Example: ID='MT9 1'

The designated logical unit is assigned to an available equipment of the specified hardware type. If hardware of the designated type is not available, or if the assignment request results in exceeding the number of scheduled equipment of this type, an error message is issued and the job is aborted.

***DEVICE, Logical Unit Equivalencing**

*DEVICE(lu_1=lu_2...lu_n=lu_m)

***EQUIP, Logical Unit Equivalencing**

*EQUIP(lu_1=lu_2,...lu_n=lu_m)

These DEVICE and EQUIP commands equate logical units. The logical unit, (lu_1) is equated to (lu_2). The unit (lu_2) must have previously had a hardware type assigned to it.

The following are fixed assignments which are not available for reassignment via *DEVICE or *EQUIP requests:

Logical Unit Number	Assignment
63	Standard input (INP)
62	Standard output (OUT)
61	Standard punch (PUN)
60	System scratch 1 (SC1)
59	System scratch 2 (SC2)
58	Library (LIB)
57	Standard load and go (LGO)
56	Standard Hollerith scratch (SHC)
55	Label file (LBL)
54	Reserved
53	Reserved
52	PCC change file
51	Reserved
50	Reserved

Figure 3-3. Fixed Logical Unit Assignments

*DEVICE, Data Pipe Assignment Statement

```
*DEVICE (lu,PN=namel,dir)
```

The Data Pipe assignment statement allows the user to specify a Pipe Name and data flow direction to be matched with a similar assignment from another JOB or TASK within the user's JOB.

Parameter	Definition
lu	The logical unit is a number between 1 and 63, excepting the fixed assignments as listed in figure 3-3.
namel	Pipe Name for matching (8 ASCII characters).
dir	The letters IN or OUT indicate data flow direction with respect to the user of the lu.

*DEVICE, Interactive Device Assignment Statement

```
*DEVICE(lu=hh,LN=name,CC=code,SP=port,EM=mask1,IM=mask2,UN=user name,
       CL=class,CM=mask3)
```

The interactive device assignment statement allows the user to assign a linkage name or task name to a specified interactive device. The parameters are as follows:

Parameter	Definition
lu	The logical unit is a number between 1 and 63, exempting the fixed assignments as listed in figure 3-3.
hh	The 2-character mnemonic: IT - Interactive Terminal, CN - Communication Network.
name	Name of the task linkage to be connected to the interactive device (8 ASCII characters).
code	Connect code as follows: I - immediate connect
port	If the CC parameter is omitted, the job will await connection by the terminal. The system port is an integer between 0 and 255. If this parameter is omitted, DEVICE will default to any available port.
mask1	The exclusion mask (EM) limits the port being defined by excluding ports for which specified bits are set in the PORT or USER validation masks. The port and user validation masks are combined to form a single validation mask (VM). If EM .AND. VM $\neq$ 0, a port will be excluded. A description of the PORT/USER validation mask is given in the Interactive Terminal Subsystem (ITS) User's Guide.
mask2	The inclusion mask (IM) limits the port being defined by accepting only those ports for which specified bits are set in the PORT or USER validation masks. The PORT and USER validation masks are combined to form a single validation mask (VM). If IM .AND. .NOT. VM = 0, a port will be accepted. A description of the PORT/USER validation mask is given in the Interactive Terminal Subsystem (ITS) User's Guide.

user name

User name is an ASCII-8 character identifier which specifies the user's desires. Question marks (?) can be used for "Don't Care" characters.

class

Class of interactive device type requested (an integer, 1-32). Current terminal classes are as follows:

Unmanned Terminals

1 - Mode 4 Line (200UT)
2 - X.25 Packet Network line
3 - CPU-CPU Async Protocol (undefined)
4 - Undefined
5 - Undefined
6 - AWN Network Line
7 - NEDN Network Line
8 - NMC Network Line
9 - ID50 Network Line
10 - Undefined
11 - 200UT Card Reader (RBTMGR)
12 - 200UT Line Printer (RBTMGR)
13 - Output Only Port (RO terminal)
14 - Input Only Terminal (Sensor/CR)
15 - Undefined

Manned Terminals

16 - Generic - Glass Teletype
17 - Generic - Printing Terminal
18 - CDC 751
19 - CDC 752
20 - CDC 756
21 - Undefined
22 - Undefined
23 - 200UT Console (RBTMGR)
24 - X.25 Packet Assembly/Disassembly
25 - CYBER Virtual Terminal (OPFMGR)

mask3

The Class Mask limits the port being defined by accepting only those ports of specified classes. If the bit for the class is set, a terminal will be accepted.

***LINK, Limk to Interactive Device**

```
*LINK(lu,...,lu)
```

The LINK statement permits I/O files to be attached to Interactive Devices. Subsequent input or output requests will be input from or routed to the Interactive Device.

***UNLINK, Unlink From Interactive Device**

```
*UNLINK(lu,...,lu)
```

The UNLINK statement permits I/O files to be unlinked from an Interactive Device and reattached to the I/O devices. Subsequent input or output requests will be input from or routed to the I/O device.

DATA SET MODIFICATION STATEMENTS

The data set modification statements change the position, content, access and/or attributes of the data set. Three of the statements apply only to mass storage file data sets. The remaining statements have definitions, summarized by table 4-1, for both mass storage device and unit record device data sets.

*CLOSE, Close Statement

*CLOSE(LUN)

The close (*CLOSE) statement clears the LUN definition from the system tables. The file must be opened following a *CLOSE to be referenced again. The LUN parameter indicates the logical unit number of the file to be closed.

*EOF, Write End-of-File Statement

*EOF(lu_1,lu_2,..,lu_n)

The write end-of-file (*EOF) statement causes an end-of-file mark to be written on the specified logical units that have magnetic tapes or cartridge tapes assigned to them. For mass storage files opened as write or read/write, the highest block written is set to the current block number. Logical units with any other type of assignment are ignored.

*EXPAND, Expand Statement

*EXPAND(LUN,NOBLKS)

The EXPAND statement is used to increase the mass storage space reserved for a file. Before the EXPAND control card can be invoked, the user must have established a linkage to the file with the OPEN control card.

Parameter	Definition
LUN	Logical file number.
NOBLKS	Number of blocks to add to the file.

*MODIFY, Modify Statement

*MODIFY(FN,OWNER,ED,AK,NFN,NOWNER,NED,NAK,NOBLKS,S,USE,SLVL,DID_1,...,DID_n)

The modify (*MODIFY) statement is used to change the attributes of an existing, closed mass storage file. *MODIFY can be used to expand an existing file, or to change the control parameters of the file.

Old control parameters:

Parameter	Definition
FN	1 to 14 characters specifying the file name.
OWNER	1 to 4 characters specifying the file owner.
ED	1 or 2 characters specifying the edition number.
AK	1 to 4 characters specifying the access privacy key for the existing file. This field is not copied on the job OUT file.

New control parameters:

Parameter	Definition
NFN	1 to 14 characters specifying the new file name (blank = no change).
NOWNER	1 to 4 characters specifying the new owner (blank = no change).
NED	1 or 2 characters specifying the new edition number (blank = no change).
NAK	1 to 4 characters specifying the new access privacy key. This field is not copied on the job OUT file (blank = no change).

File expansion parameters:

Parameter	Definition
NOBLKS	The number of logical blocks to be added to the file (a decimal constant in the range of 1 to 65535. The total number of blocks in the expanded file can not exceed 65535. If blank, there is no change.

S — Segmentation Flag:

Entry	Meaning
Blank	Added blocks can be segmented.
S	Added blocks can be segmented.
NS	Added blocks can not be segmented.

USE — New protection flag for the modified file:

Entry	Meaning
Blank	Does not modify existing usage parameter.
R	File can be accessed as read only.
RW	File can be accessed as read/write.

SLVL — Security level. Decimal constant in the range of 0 to 7.

DID — 1 to 8 characters identifying the device to be used for the expanded blocks. If this parameter is omitted, the device of the last segment is used. The total number of devices used by the file can not exceed eight.

***OPEN, Open Statement**

```
*OPEN(LUN,FN,OWNER,ED,AK,USE,BLOCK)
```

The open (*OPEN) statement is used to prepare an existing mass storage file for data transmission by locating the file and requesting the device be put on-line, if necessary.

Parameter	Definition
LUN	Logical file number.
FN	1 to 14 characters specifying the file name.
OWNER	1 to 4 characters specifying the file owner.

ED	1 or 2 characters specifying the edition number of file to be opened.
AK	1 to 4 characters specifying the access privacy key. This field is not copied on the job OUT file.
USE	Protection flag:

Entry	Meaning
Blank	File can be accessed as read/write.
R	File can be accessed as read only.
RW	File can be accessed as read/write.
W	File can be accessed as read/write and the block count (highest block written) is set to 0.

BLOCK	If 0 or blank, the file is completely opened; otherwise, only the device containing the referenced block is opened (partially open).

*RELEASE, Release Statement

```
*RELEASE(FN,OWNER,ED,AK,NOBLKS)
```

The release (*RELEASE) statement is used to release some or all the space allocated to a mass storage file.

Parameter	Definition
FN	1 to 14 characters specifying the file name.
OWNER	1 to 4 characters specifying the file owner.
ED	1 or 2 characters specifying the edition number.
AK	1 to 4 characters specifying the access privacy key for the file. This field is not copied to the OUT file.
NOBLKS	The number of logical blocks to be deleted from the file. The highest-numbered blocks are released.

Entry	Meaning
Blank or 0	Entire file is released.
R	All blocks following the highest block written are released.

*REWIND, Rewind Statement

*REWIND(lu_1,lu_2,...,lu_n)

The rewind (*REWIND) statement positions a magnetic tape, cartridge tape, or file to the initial location. That is, a magnetic tape is rewound to load point, and a file is positioned to the first opened block of the file. The units to be rewound are indicated by the parameters lu_1 through lu_n, which are logical unit numbers. Files specified in figure 3-3, Fixed Logical Unit Assignments, should not be rewound.

*SAVEPF, Save Scratch File Statement

*SAVEPF(LU,FN,OWNER,ED,AK,NR)

The save scratch file (*SAVEPF) statement, allows the user to save a file allocated by the operating system. The occurrence of the control card causes the job manager to CLOSE, RELEASE unused, and MODIFY the file specified. The format of the *SAVEPF control card is as follows:

Parameter	Definition
LU	Logical file number of currently open file whose name is to be changed. Can be a number from 1 to 63.
FN	New file name to replace the previous name; must be specified (up to 14 characters).
OWNER	New file owner to replace previous owner; if absent, old owner is used (up to 2 characters).
ED	New edition to replace previous file's edition; if omitted, old edition is retained (up to 4 characters).
AK	New access key to replace original access key; if absent, old access key is retained. (up to 4 characters)

NR	Optional parameter. If absent, any old file with the new file name is released prior to changing the name of the currently open file (functional replace). If present, this release does not occur. NR=0, Release Old File; NR=1, Do Not Release Old File.

EXAMPLE: A job creates a scratch file and saves it.

```
*JOB(ID=SAVE)
*SCHED(CM=11,PL=9999,TL=9999)
*FTN(X)

      PROGRAM SCRATCH

                .
                .
                .

      FIRST REFERENCE TO LU 10

      WRITE (10,100) A, B, C

                .
                .
                .

      END

         FINIS
*SAVEPF(10,FILENAME,OWNR,ED,AKEY)
*EOJ
```

*SEOF, Search End-of-File Statement

*SEOF(lu_1=B,lu_2=F,lu_n)

The search end-of-file (*SEOF) statement positions a magnetic tape, cartridge tape, or file to an end-of-file. A second parameter of B indicates a search backward (lu = B). The F parameter (lu = F) indicates a search forward. If no second parameter (lu) is included, the search is performed forward.

An *SEOF forward causes a magnetic tape unit to be positioned immediately after an end-of-file mark.

An *SEOF backward causes a magnetic tape unit to be positioned immediately before an end-of-file mark.

When the *SEOF statement is used on a logical unit number that has a file assigned to it, a search backward positions to the first opened block of the file. A search forward positions to the block past the highest block written.

*UNLOAD, Unload Statement

*UNLOAD(lu_1,lu_2 .. ,lu_n)

The unload (*UNLOAD) statement rewinds and unloads the specified logical units that have magnetic tapes assigned to them. Logical units with any other type of assignment are ignored. Note that dismounting the tape does not affect the logical unit to physical device assignment.

TASK PREPARATION AND USE STATEMENTS

The task preparation and use statements define attributes of a task, prepare an executable image of a task in memory, save the prepared task in a data set and/or initiate execution of the prepared task.

LIBRARY TASK STATEMENT

A library task, such as an assembler or compiler, is loaded and executed by a library name control statement.

*name(parameter list)

A program can be called by a library name statement by using PRELIB to place the program on the library (LIB) file. A library program is automatically executed after loading. For example:

*FTN(I,L,X)

*ABS, Build Absolute Task

*ABS(lu)

The build absolute task (*ABS) statement causes an absolute copy of a loaded task to be written on the logical unit specified by lu. The *ABS statement must follow any load command (*LOAD statement or binary decks) and precede the *RUN statement, if used. The absolute file is preceded by a header record describing the contents of the file. The absolute task is in blocker/deblocker format with the system standard block size. An absolute file can be used in a task call sequence to decrease the load time of the called task.

*MAP, Request Load Map

*MAP

The *MAP card is used to request that the loader generate a load map for the next loaded task. If the card is left out, no map is generated. (See section 6, MAP, Memory Allocation Printout.)

*LOAD, Load Statement

*LOAD(lu_1,lu_2,lu_3,lu_4)

The load (*LOAD) statement specifies the sources for a binary load. Up to four defined logical units can be used as parameters of this statement. Programs are loaded from the assigned units in order of appearance prior to the loading of any binary information contained on INP. If the parameter list is omitted, the LGO file is assumed for the load source. When the statement is omitted, the occurrence of a binary deck initiates the load process. Only one logical unit should be specified when an absolute formatted file is to be loaded. (See section 6, MPX/OS Loader.)

*RUN, Run Statement

*RUN(parameter list)

The run (*RUN) statement initiates program execution by transferring control to the object program. This statement is necessary to execute any user-defined task. The *RUN statement follows the binary decks if the program is on INP. It follows the *LOAD statement if the program is on any other unit. Any parameter list is passed to the executing task in the PARM region starting at PARM+5. (See section 4, Initial Task Entry.)

*TASK, Task Statement

*TASK(ID= ,PCC,DMP,PRTY= ,CPU=)

A task (*TASK) statement precedes a load statement. It establishes run time control parameters for the next task loaded.

Parameter	Definition
ID=	1 to 8 characters indicating the task identification; this parameter is optional. If not present, the system default DUMYTASK is assigned. The task identification is used to identify abort messages.

PCC — A copy of program control console (PCC), a debug aid, is requested. If PCC is to be used, the PCC parameter is required.

DMP — Dump control, indicating that all of task memory is to be dumped upon recognition of an abnormal condition. If the parameter is omitted, only the contents of the page and operand registers are dumped.

PRTY= — 1 to 3 numeric characters indicating the task priority. This parameter is optional. It must be number, n, where 1 .LE. n .LE. 511 for real-time jobs and 10 .LE. n .LE. 255 for non-real-time jobs. If n .LT. 10 or .GT. 255 for a non-real-time job, the task priority is set respectively to 10 or 255. If the parameter is not present or zero, the system default of 10 is assigned. This value is passed to the executing task in PARM+4. (See section 4, Initial Task Entry.)

CPU= — Numeric identifier of a CPU in the configuration. Any undefined value is treated as a default. The default placement of a task is an installation option. A nondefault value constrains the execution of the task to the designated CPU, except for ESR processing. There can be multiple occurrences of this parameter to identify a set of CPUs to be used. Values are: 1 = master; 2 = backup (if applicable); 3, 4, 5 = slaves. For example, CPU = 1, CPU = 3, will provide CPU1 and/or CPU3 as valid CPUs to execute in.

In the instance where the task will be multitasking (that is, calling subtasks), the following apply (section 4, Task Manager ESRs):

- If PCC is specified, only one task will be loaded with a copy of PCC. This task is specified in parentheses [for example, PCC=(TASKNAME)]. If no task name is specified, the default is to assign PCC to the next task loaded. For library tasks, ID must be used where ID = library task name.
- If the DMP parameter is specified, the memory of a task and all its subtasks will be dumped.

A *TASK statement pertains to one and only one *LOAD statement. Therefore, for each task to which run-time parameters are assigned, a *TASK statement must be included. Figure 3-4 illustrates a job control deck with three task executions. In the example, tasks 1 and 3 are assigned run-time controls via their corresponding *TASK statements. Task 2 would be assigned default run-time controls, because it does not have a corresponding *TASK statement.

```
*JOB (. . .)
*SCHED (. . .)
.
.
.
*TASK (ID=TASK1,PCC,DMP,PRTY=25)
.
.
.
*LOAD(20) Load Task 1
*RUN
.
.
.
*LOAD(30) Load Task 2
*RUN
.
.
*TASK(ID=TASK3,PCC,DMP,PRTY=26)
.
.
.
*LOAD(40) Load Task 3
*RUN
.
.
.
EOJ
```

Figure 3-4. *TASK Control Statement Example

JOB TERMINATION

The job termination phase is initiated by one of two means. Normal job termination is initiated by the job manager upon encountering the *EOJ control statement in the job control stream. Abnormal job termination is initiated by any task through the ABORT ESR or by the system executive when one of the abort conditions identified in appendix D occurs.

*EOJ, End-of-Job Statement

```
*EOJ
```

The end-of-job (*EOJ) statement causes the normal termination of the job. This statement is the last statement processed for the job.

ABNORMAL JOB TERMINATION (JOB ABORTED)

When a condition causing abnormal termination occurs, MPX/OS responds with a diagnostic and task dumps. The dump is according to the format specified on the *TASK statement. If a *TASK statement has not been included in the job, MPX/OS dumps only the contents of the task registers. The format of the diagnostic is illustrated in figure 3-5.

JOB ACCOUNTING STATISTICS

MPX/OS writes the following job-related information on the standard output file after a job has terminated.

- Job name
- Submitter Username
- Account number
- Sequence number
- Security
- Date (mm/dd/yy)
- Time on (hh:mm:ss.sss)
- Time off (hh:mm:ss.sss)

- o MPX/OS resident edition number
- o Version name
- o Library edition number
- o CPU time used (hh/mm/ss.sss)
- o Resources reserved but not used:
  - Memory
  - Scratch segments
  - Print lines
  - Punch cards

The job accounting statistics are illustrated in figure 3-6.

```
TASK NAME = JMGR     STATUS = 01  CPU = 1  STATE = 2
Page Registers    103F0800    08000800    08000800    08000800    08000800    08000800    08000800    0800000D
                XO  00000573    00000000    20202020    00000000    00000000    00000020    00000001 X7 00000007
                HO  00000000    00000000    00000000    00000000    00000000    0001F998    00000000 H7 00000000
                RO  52415446    4F522020    0000003A    00001201    00000003    00000000    00000000 R7 000006AA
                R8  0000020E    0000006C    43202020    0000003E    0003C258    00000088    00000000 RF F0120769

                573 = Abort Address + 1          XO register is not displayed.
```

This information is repeated for each task in the job.

Figure 3-5. Example of Abort Listing of Registers

```
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************
**********************************************************************************

 JOB =  REFMAN   USER = GBF      ACCT = ACCNUM    SEQ = 2N     SC = 0
 DATE = 05/14/81  TIME ON = 15:49:20.541 TIME OFF = 15:49:21.379
 RESIDENT EDITION = TS    VERSION = MP32  LIBRARY EDITION = X1
 ACCUMULATED TIME = 00/00/00.089
 ACCOUNTING INFORMATION
    RESOURCES NOT USED
    MEM =    1
    SCR =   10
    LINE=  963
    CARD=    0

**********************************************************************************
***JOB ABORTED ABORT TYPE = 15 ABORT CODE = 06  TASK = JMGR       **********
**********************************************************************************
```

Figure 3-6. Abort Message and Accounting Statistics Example

EXECUTIVE SERVICE REQUESTS 4

An executive service request (ESR) is issued by the monitor call instruction, MON,R ESRID. R is the first of four contiguous registers. The contents of all four registers are passed to the executive as ESR parameters. ESRID is the name of the executive service requested. The name appears in the description title. Some requests pass and/or receive data through the PARM area. The PARM area is allocated by the loader, the ESRID values are defined by the loader; they are accessed in the COMPASS code modules by their declaration as externals. The format of data to pass through registers, through the PARM area, and through any additional data area is drawn, and the data fields are labeled and explained for each ESR. PARM status codes are provided in figure 4-1.

ESRs are requested by execution of the MON instruction. This is the only voluntary method for user and system tasks to request action from the executive. Involuntary entry to the executive occurs as a result of interrupt recognition and is transparent to the interrupted task, except for task fault interrupts. Task execution resumes with the instruction following the MON instruction for all but a few system ESRs. Task execution resumes at a specified address for fault execution interruptions if such a return has been requested; otherwise, the job is terminated.

Register names specified in calling sequences using an executive service routine are only examples. (See appendix H for register conventions.)

PARM STATUS CODES

The following status codes can be returned in the user PARM area following execution of an ESR.

0	Successful Processing.
1	File Manager Busy.
2	Unused.
3 and above.	See File Manager Error Codes in appendix D.

Figure 4-1. PARM Status Codes

DEVICE AND FILE MANAGEMENT ESRS

The Device and File Management feature is invoked by control cards and Executive Service Requests (ESRs). The control cards and ESRs are used to manage I/O resource allocation. The DEVICEQ and OPEN ESRs are used to establish the linkage between a task and the device, while the CLOSE ESR removes this linkage. The other Device and File Management ESRs allow the user to create, maintain, and remove files in the mass storage system.

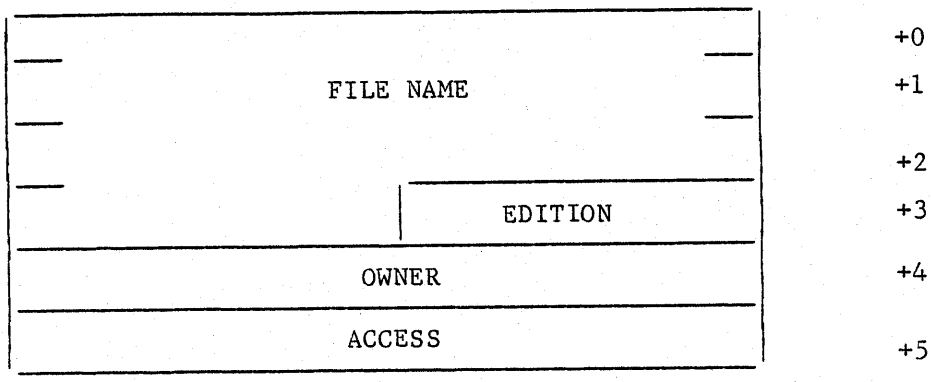
The user invokes Device and File Management features with the following ESRs:

ALLOCATE	Reserve mass storage space.
CLOSE	Clear logical unit assignment.
DEVICEQ	Assign logical unit to device.
EXPAND	Increase mass storage space.
MODIFY	Change mass storage attributes.
OPEN	Assign logical unit to file.
RELEASE	Reduce mass storage space.
SAVEQ	Alter mass storage attributes.

The Device and File Management ESRs, used to manage the files on mass storage, maintain file directory on the System resident pack (SYSTEM01). This file directory contains information about the file necessary to uniquely identify and locate the file in the mass storage system. The user must create a communications area in memory before invoking a file management ESR. The first six words of this area contain the information necessary to locate the file in the file directory. These first six words are referred to as the File Identification area and are described in figure 4-2. The appropriate Device and File Management parameters are left-justified, blank filled within the File Identification Area.

Device and File Management ESRs may require a number of disk references and should not be issued during time critical operations.

The default system device list is created during system initialization and reflects the entries in the mass storage tables which are installation dependent.



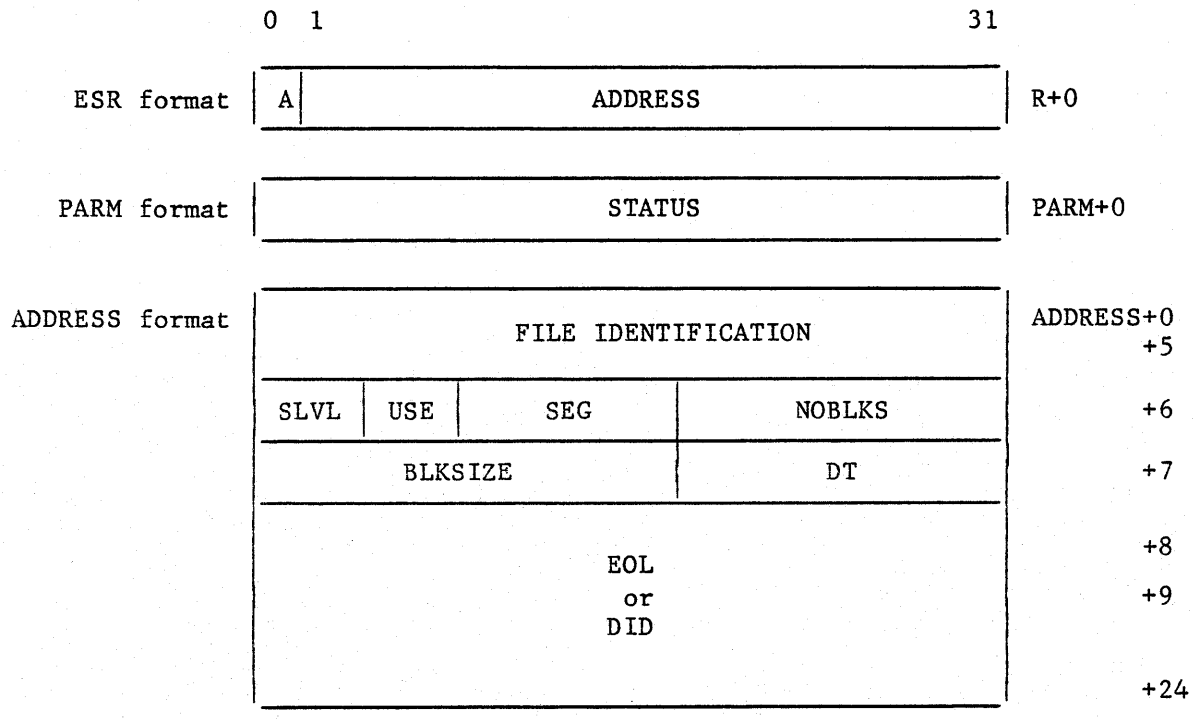
Parameter	Definition
FILE NAME	14-character string that defines the file name.
EDITION	2-character string that defines the file edition.
OWNER	4-character string that defines the file owner.
ACCESS	4-character string that defines that file access privacy key.

The first five words must uniquely identify the file within the file system. All file identification area parameters are left-justified with blank fill.

Figure 4-2. File Identification Area

ALLOCATE, Allocate Mass Storage File Space

This ESR reserves space in the mass storage system and builds a file label entry in the system label directory. Once the file is successfully created, it remains allocated until released. (See RELEASE, Release Mass Storage File Space in this section.)



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
ADDRESS	Address of the first word of the file definition.
STATUS	See PARM Status Codes, figure 4-1.
FILE IDENTIFICATION	See File Identification Area description, figure 4-2.
SLVL	Binary value that defines security level of the file. Value may range from 0 to 7.
USE	Binary value that defines the allowed file usage:

Entry	Meaning
=0	File can be opened for read/write use.
=1	File can be opened for read use only.

SEG — Binary value defining acceptable segmentation mode for file allocation:

Entry	Meaning
=0	File can be allocated in segments.
=1	File can not be segmented when allocated.

NOBLKS — Binary value defining the size of the file in logical blocks. Value can be from 1 to 65,535.

BLKSIZE — Binary value defining the number of words in each logical block. Value can be from 1 to 4096.

DT — Binary value defining a specific device type to be used for file allocation. The defined values and devices represented are as follows:

Entry	Meaning
=0	System device.
=1	Control Data 9425 Cartridge Disk Drive.
=2	Control Data 844 Disk Storage Unit.
=3	Control Data 9427 Cartridge Disk Drive.
=4	Control Data 1867-10 Storage Module Drive.
=5	Control Data 1867-20 Storage Module Drive.

DID — 8-character string defining the device or devices to be used for file allocation. Up to eight devices are allowed.

EOL — -1 End of List of DIDs.

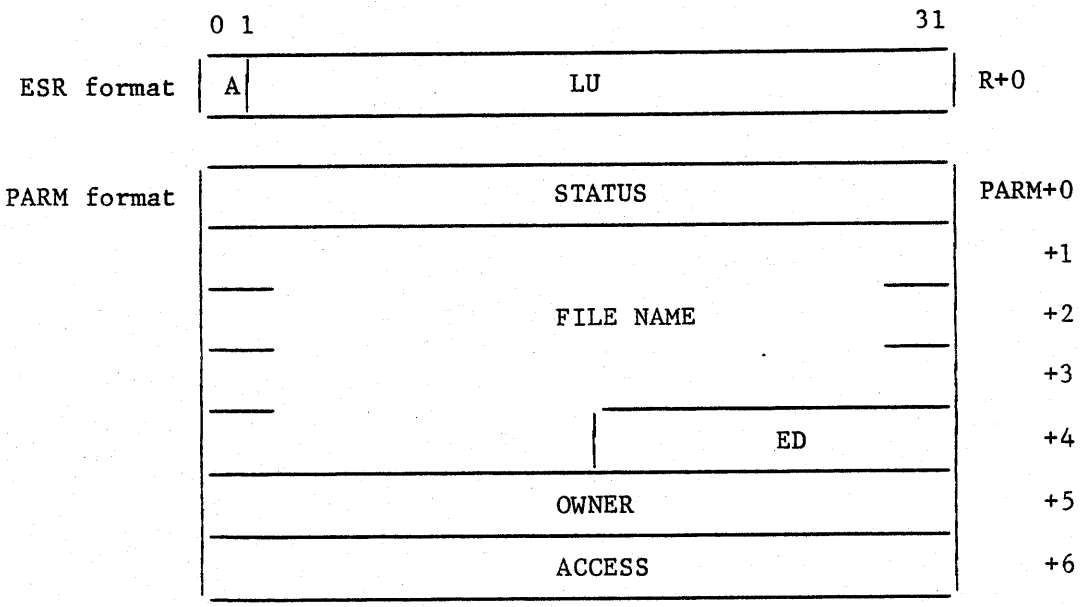
An example of the calling sequence is as follows:

```
EXT       ALLOCATE      Externally defined symbol
LDA,RO    FAA           File Allocation Area address
MON,RC    ALLOCATE      Monitor request
```

```
FAA     BSS         0
        TEXTC       14,SCRATCH          File Name
        TEXTC       2,01                Edition
        TEXTC       4,DDPG              Owner
        TEXTC       4,DNSS              Privacy access key
        VFD         4/0,4/0,8/0,16/100
        VFD         16/480,16/5
        TEXTC       8,SYSTEM01
        GEN         -1
```

CLOSE, Close Mass Storage File Space

The CLOSE ESR clears the file logical unit definition, and the job no longer has access to the file. When the file is open for write access, the CLOSE ESR allows other tasks to obtain access to the file. CLOSE accomplishes three things: it frees a logical unit number, may remove restrictions on file usage, and returns the file definition parameters which may be needed if the file was allocated by BLOCKER/DEBLOCKER. Since all files are closed by the system when a job terminates, the CLOSE ESR is used for overall efficiency. Logical unit numbers 61, 62, 63 cannot be closed unless the closer is a system task. If the logical unit was not explicitely opened (that is with a *OPEN), then the auto-allocated file will also be released.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
LU	Number of the logical unit to be closed.
STATUS	See PARM Status Codes, figure 4-1.
FN	14-character string that defines the file name.
ED	2-character string that defines the file edition.
OWNER	4-character string that defines the file owner.
ACCESS	4-character string that defines the file access key.

An example of the calling sequence is as follows:

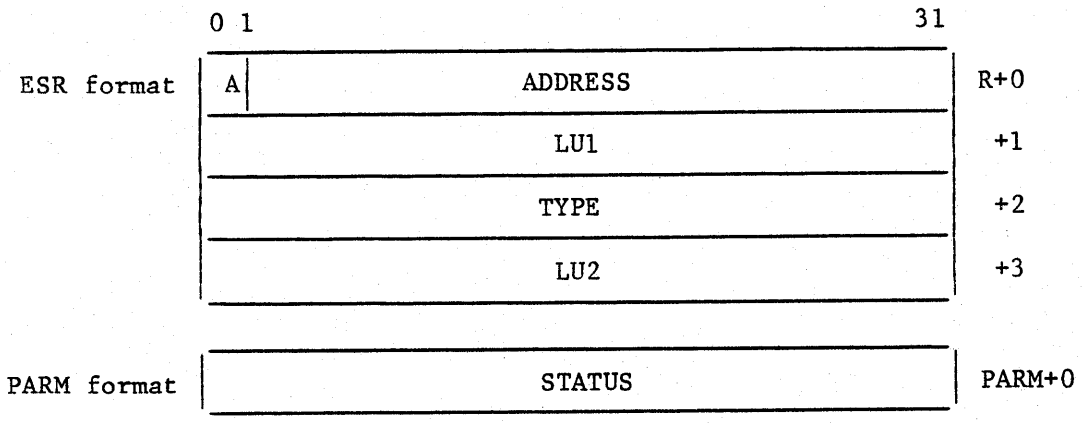
```
EXT             CLOSE           Externally defined symbol
LDI,R0          15              Logical unit number
MON,R0          CLOSE           Monitor request
```

DEVICEQ, Assign Logical Unit to Device

The DEVICEQ ESR operates in two modes:

Assignment
Equivalence

In assignment mode, the DEVICEQ ESR assigns the specified logical unit to a logical or physical device by establishing a linkage between the task and the device. In equivalence mode, the DEVICEQ ESR allows the assignment of a new logical unit to a previously assigned logical unit and that logical unit may have been assigned to a device. The logical units specified must be numbers between 1 and 63.



Parameter	Definition
A	A = 0 thread request if File Manager Busy. = 1 return if File Manager Busy.
ADDRESS	Assignment mode: address of the first word of the DEVICE ASSIGNMENT AREA (DAA). See appendix J, Device Assignment Areas. Equivalence mode: address of the first word of the device assignment area (DAA).
LU1	Assignment mode: logical unit to be assigned. Equivalence mode: logical unit to be assigned.
TYPE	Assignment mode: one of the hardware types from appendix K, Valid Hardware Types. Equivalence mode: set to zero.
LU2	Assignment mode: set to zero. Equivalence mode: previously assigned logical unit.

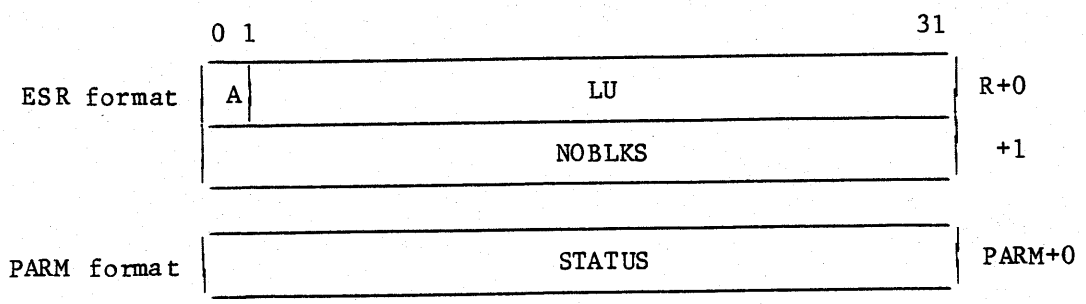
STATUS	See PARM Status Codes, figure 4-1. If no errors: For Unit Record and Data Pipe devices, a copy of the Equipment Status Table (EST) entry for the device is returned in PARM+1 through PARM+8. For Interactive devices, the security of the device is returned in PARM+1.

An example of the calling sequence is as follows:

```
        EXT         DEVICEQ         Externally defined symbol
        LDA,R0      DAA             Device Assignment Area Address
        LDA,R1      15              Logical unit number
        LDA,R2      2               Magnetic Tape
        MON,R0      DEVICEQ         Monitor request

DAA     BSS         0
        GEN         -1              Any Unit
```

EXPANDQ, Increase Mass Storage Space

The EXPANDQ ESR is used to increase the mass storage space reserved for a file. Before the EXPANDQ ESR can be invoked, the user must have established a linkage to the file with the OPEN control card or ESR.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
LU	Logical unit.
NOBLKS	The number of logical blocks the file is to increase.
STATUS	See PARM Status Codes, figure 4-1.

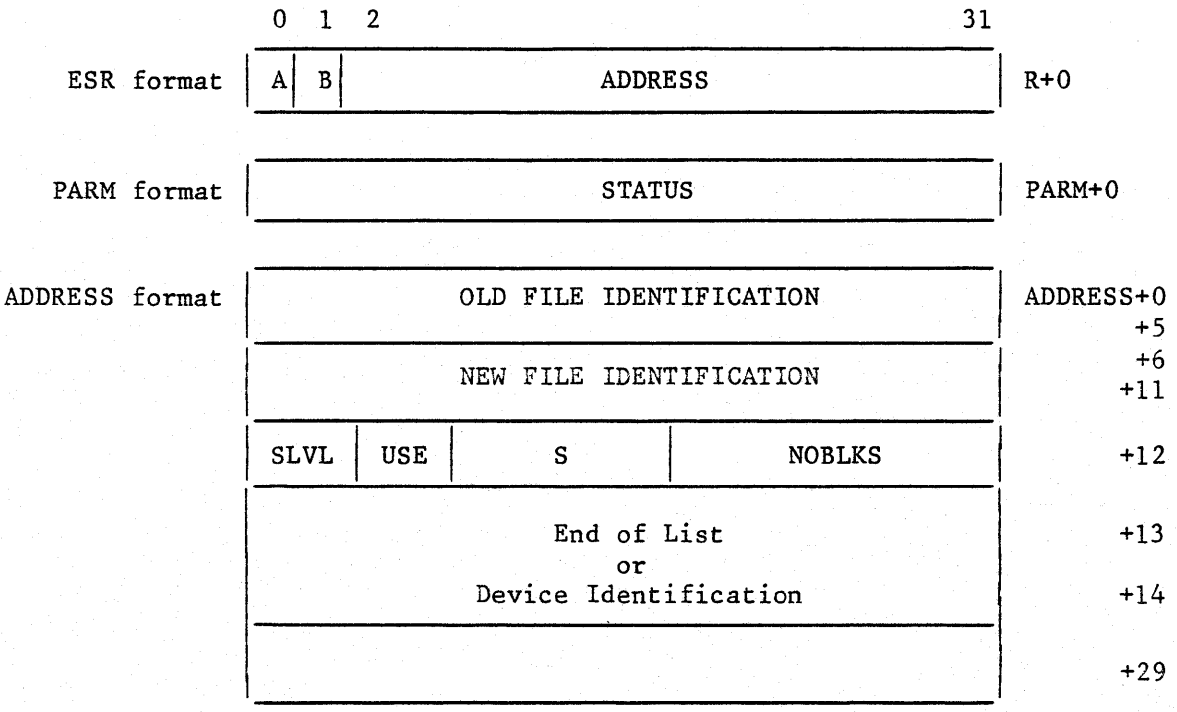
An example of the calling sequence is as follows:

```
EXT       EXPANDQ     Externally defined symbol
LDA,R0    15          Logical unit number
LDA,R1    48          Number of blocks to add
MON,R0    EXPANDQ     Monitor request
```

MODIFY, Modify Mass Storage File Definition

The MODIFY ESR is used to alter the file label definition of an existing, closed file. This ESR can be used to expand an existing file or to change its control parameters.

This ESR cannot be performed on an assigned file. If the file is assigned to the calling job then an error indicator is returned. If the file is assigned to another job, the caller may request a wait until the file is closed by the other job.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
B	B = 0 thread request if file is open. = 1 return if file is open.
ADDRESS	Address of first word of the File Modification Description.
STATUS	See PARM Status Codes, figure 4-1.
OLD/NEW FILE IDENTIFICATION	See File Identification description, figure 4-2.
SLVL	Binary value that defines the new security level of the file. Value may range from 0 to 7.

USE — Binary value that defines the (new) allowed file usages:

Entry	Meaning
= 0	File can be opened for read/write use.
= 1	File can be opened for read-only use.

S — Binary value defining the permitted segmentation mode for the added file space:

Entry	Meaning
= 0	Addition can be allocated in segments.
= 1	Addition can not be segmented.

NOBLKS — Binary value defining the number of blocks to be added to the file. Total file allocation cannot exceed 65,535 blocks.

LE — The list-end flag:

Entry	Meaning
= -1	The list has ended.
≠ -1	Another DID specification begins.

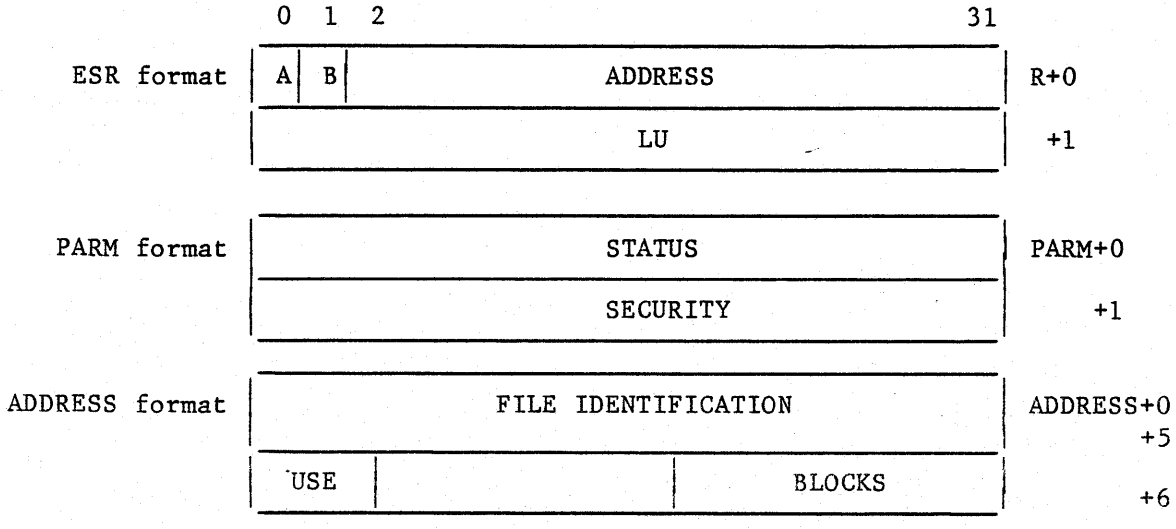
DID — 8-character string defining the device or devices to be used for the expanded blocks.

An example of the calling sequence is as follows:

```
        EXT         MODIFY              Externally defined symbol
        LDA,R0      FMA                 File Modification Area address
        MON,R0      MODIFY              Monitor request

FMA     BSS         0
        TEXTC       14,TAPE10           Old File name
        TEXTC       2,00                Edition
        TEXTC       4,00SC              Owner
        TEXTC       4,$$$$              Privacy access key
        TEXTC       14,SCRATCH          New File Name
        TEXTC       2,01                Edition
        TEXTC       4,DDPG              Owner
        TEXTC       4,DNSS              Privacy access key
        VFD         4/0,4/0,8/0,16/100
        TEXTC       8,SYSTEM01
        GEN         -1
```

OPEN, Establish Access to Mass Storage File

The OPEN ESR assigns the specified logical unit to a file thus establishing a linkage between a task and the device. The logical unit must be a number between 1 and 63. The OPEN ESR also allows the assignment of the file in an exclusive access mode, with which the caller can request a wait until exclusive access can be established or a previous exclusive access is cleared by the CLOSE control card or ESR.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
B	B = 0 thread request if file is already open. = 1 return if file is already open.
ADDRESS	Address of the first word of the file identification specification.
LU	Number of the logical unit to be assigned to the file being opened.
STATUS	See PARM Status Codes, figure 4-1.
SECURITY	Security Level of the file.
FILE IDENTIFICATION	See File Identification Area description, figure 4-2.
USE	Binary value defining the intended use of the file during this access. The file-label definition field for the file defines the allowed access modes. If the file-label definition allows only read usage, the open must specify read-only use. The three values allowed for USE are:

Entry	Meaning
= 0	File to be used for read/write.
= 1	File to be used for read only.
= 2	File to be used for read/write. Set the highest block written count to 0. The next block written will be the first block of the file.

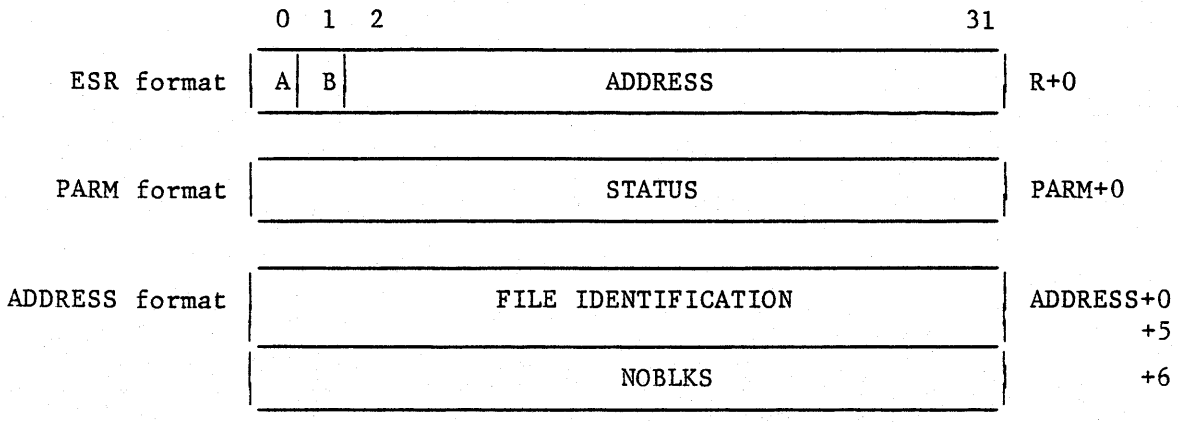
BLOCK The next block to be read/written.

An example of the calling sequence is as follows:

```
        EXT       OPEN                Externally defined symbol
        LDA,R0    FDA                 File Description Area address
        LDA,R1    15                  Logical unit number
        MON,R0    OPEN                Monitor request

FDA     BSS       0
        TEXTC     14,SCRATCH          File name
        TEXTC     2,01                Edition
        TEXTC     4,DDPG              Owner
        TEXTC     4,DNSS              Privacy access key
        VFD       4/0,4/0,8/0,16/0
```

RELEASE, Release Mass Storage File Space

The RELEASE ESR is used to remove some or all of the space reserved for a file. This ESR cannot be performed on a file which is assigned and the caller may request a wait until previous file assignments are cleared.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
B	B = 0 thread request if file is open. = 1 return if file is open.
ADDRESS	Address of first word of the file release description.
STATUS	See PARM Status Codes, figure 4-1.
FILE IDENTIFICATION	See File Identification Area description, figure 4-2.
NOBLKS	A binary number indicating the number of blocks to release. If the value is zero, the entire file is released. If the value is -1, the unused portion of the file is released.

An example of the calling sequence is as follows:

```
        EXT       RELEASE          Externally defined symbol
        LDA,R0    FDA              File Description Area address
        MON,R0    RELEASE          Monitor request

FDA     BSS       0
        TEXTC     14,SCRATCH       File Name
        TEXTC     2,01             Editor
        TEXTC     4,DDPG           Owner
        TEXTC     4,DNSS           Privacy access key
        GEN       -1
```

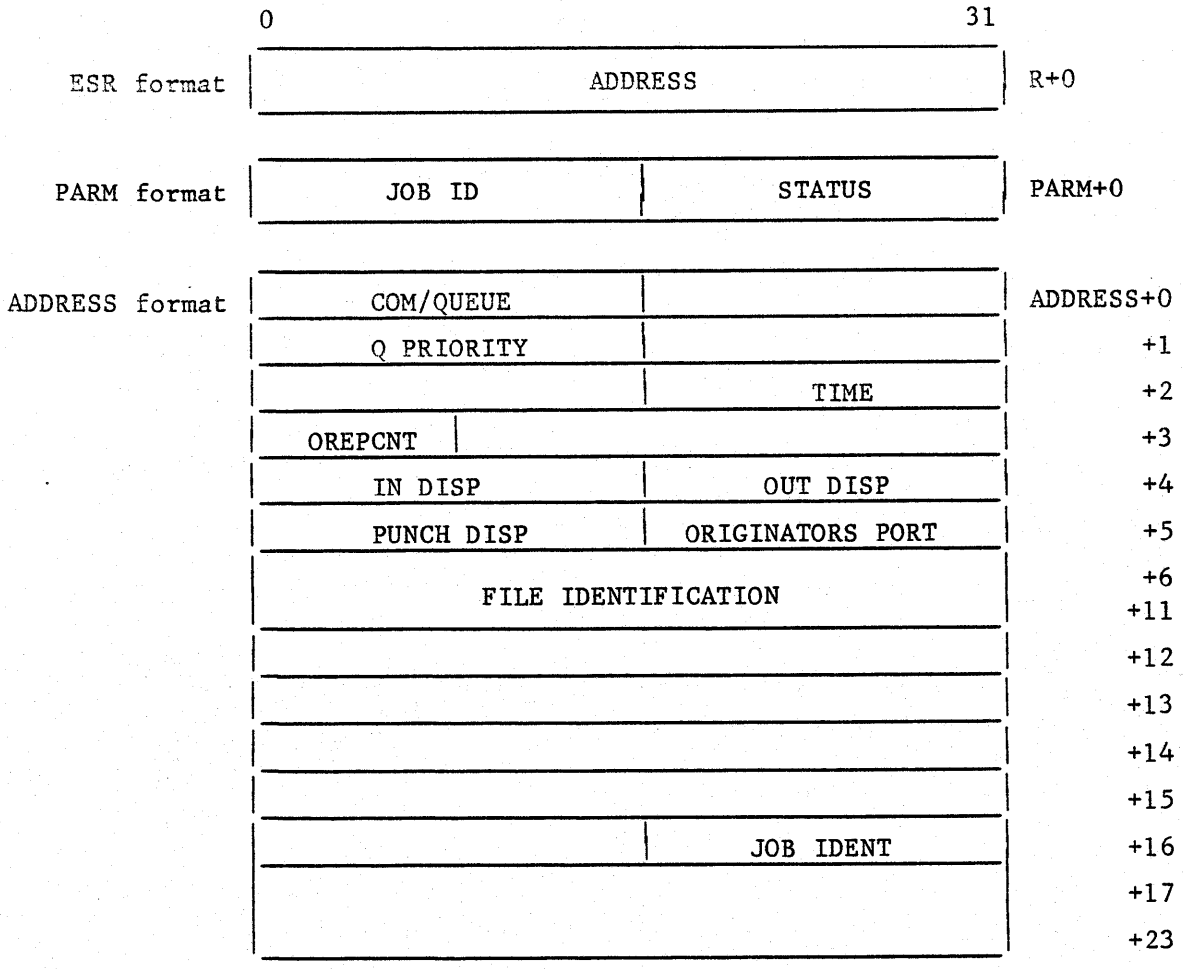
The ROUTEQ ESR provides the capability to add or delete JOB files from a SYSTEM queue.

An ADD issued for the INPUT queue causes a job file to be placed on the INPUT queue for execution. If SYSQS encounters errors in either the JOB or SCHED control cards, status is returned in PARM and the job is not placed on the INPUT queue.

An ADD issued for the OUTPUT queue causes the specified file to be listed.

An ADD issued to the HOLD queue will cause JOB or OUTPUT files to be placed on the HOLD queue for removal to the INPUT or OUTPUT queue at the time specified.

A DELETE request results in the removal of a file from the specified queue. A DELETE request is rejected if the file is not on the specified queue or if it is active when the request is executed.



Parameter	Definition
ADDRESS	Address of the first word of the Queue Header Table.
JOB ID	JOB identification (two ASCII characters). Also referred to as JOB sequence number.
STATUS	Status information as follows: 1 - Request is ACTIVE on input/output 2 - Request SATISFIED 3 - Job queue file is full 4 - No memory pool space 5 - Illegal command code 6 - Illegal JOB ID 7 - Attempt to delete active queue entry 8 - File error 9 - Control card order error 10 - Control card format error 11 - Waiting file deleted from input 12 - Illegal buffer address
COM	Command code as follows: $1000 - ADD $2000 - DELETE
QUEUE	Queue to which the command code applies. Queue codes are as follows: $100 - Input queue $200 - Output queue $401 - Hold queue - Input $402 - Hold queue - Output
Q PRIORITY	Queue Priority (0-$7FFF). The priority cannot be a number greater than the user's task priority. A priority of $7FF0- $7FFF will result in the immediate scheduling of the file. A priority of $0001-$0015 will result in the file being placed on the requested queue, but not executed until the priority is modified by an appropriate operator command.
TIME	Time of day in minutes (binary) that a file is to be removed from the hold queue. The time must be within twenty-four hours of the time the request was issued.
OREPCNT	Output Repeat Count is the number of copies desired of a file sent to the Output Queue (0-$EF).

IN/OUT/PUNCH DISP — Some disposition codes are determined during system installation. Those permanently defined follow:

- SC - Release file following execution
- MS - Mass Storage
- PR - Any printer
- Ln - Output to printer n, where n is a legal printer number.

ORIGINATORS PORT — System port number if job will request connection to a specific system port. Must be $FFFF if no connection or no specific port required.

FILE ID — See File Identification Area description, figure 4-2.

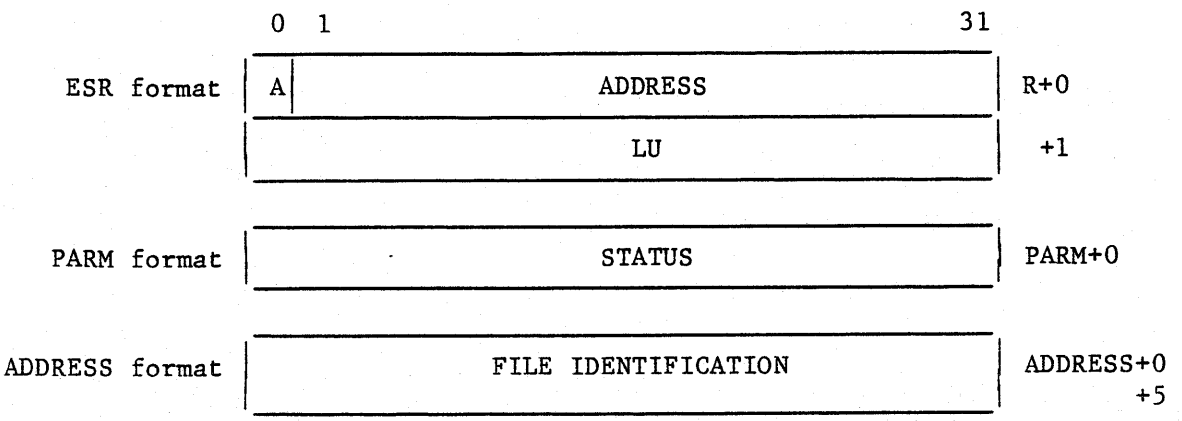
JOB IDENT — Job identification for DELETE request (two ASCII characters, 00-ZZ)

An example of the calling sequence is as follows:

```
        EXT         ROUTEQ              Externally defined symbol
        LDA,R0      QIA                 Queue Interface Area address
        MON,R0      ROUTEQ              Monitor request

QIA     BSS         0
        VFD         4/1,12/100,16/00    Add to Input queue
        VFD         16/0020,16/0        Queue Priority
        GEN         0,0
        TEXTC       2,SC                Input Disposition
        TEXTC       2,PR                Output Disposition
        TEXTC       2,SC                Punch Disposition
        GEN,H       $FFFF               No originating port
        TEXTC       14,GORP             File Name
        TEXTC       2,00                Edition
        TEXTC       4,DDPG              Owner
        TEXTC       4,$$$$              Privacy access key
        GEN         0,0,0,0
        GEN         0,0,0,0
        GEN         0,0,0,0
```

SAVEQ, Alter Mass Storage File Identification

The SAVEQ ESR is used to change the file identification of an assigned file. Before the ESR can be invoked the user must have established a linkage with the OPEN control card or ESR.



Parameter	Definition
A	A = 0 thread request if File Manager busy. = 1 return if File Manager busy.
ADDRESS	Address of first word of the file identification description.
STATUS	See PARM Status Codes, figure 4-1.
FILE IDENTIFICATION	See File Identification Area description, figure 4-2.

An example of the calling sequence is as follows:

```
        EXT         SAVEQ           Externally defined symbol
        LDA,R0      FDA             File Description Area address
        LDA,R1      15              Logical unit number
        MON,R0      SAVEQ           Monitor request

FDA     BSS         0
        TEXTC       14,SCRATCH      File Name
        TEXTC       2,01            Edition
        TEXTC       4,DDPG          Owner
        TEXTC       4,DNSS          Privacy access key
        GEN         -1
```

STANDARD UNIT

Standard units such as INP, OUT, and PUN (see section 3, EQUIP Assignment) can be accessed by the user. The user should access these units through BLOCKER/DEBLOCKER with PICK and PACK. (See section 5.) The block pointer, record headers and trailers, block numbers and so forth are defined for the job by the system (block size is 480 words).

Using direct physical I/O ESRs for Standard unit I/O may be destructive to the job.

DATA TRANSFER ESRs

The ability to transfer data between a task's memory buffer and a device is supplied by the Data Transfer ESRs.

The Data Transfer feature is invoked by the following ESRs:

FORMATQ	Initialize disk track.
READLU	Read from logical unit.
READDS	Alternate read from logical unit.
WRITLU	Write to logical unit.
WRITDS	Alternate write to logical unit.

These ESRs are legal on devices as specified in appendix L.

Since the data transfer features are scheduled for device manager processing in priority order, a lower priority request could wait for a higher priority request to complete.

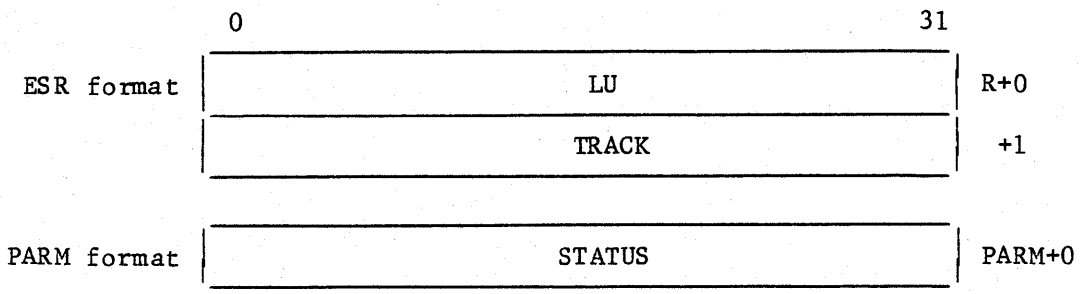
The following abort conditions are possible with these ESRs:

TYPE	ABORT CODE	
1	1	Operator rejected request to ready a unit.
1	2	Buffer size larger than 4096 words.
1	3	Logical unit unassigned.
1	4	Attempt to write on read-only file.
1	5	An input was attempted into a read-only page.
1	6	Hardware reject.
1	7	An input or output was attempted upon a protected page.
1	8	Illegal logical unit number.
1	9	Command is not legal for assigned device.

Hardware and data transmission error recovery procedures are error type and device dependent. The MPX/OS error recovery procedures are described in appendix E.

FORMATQ, Initialize Disk Track

The FORMATQ ESR writes the track addresses and timing marks necessary for subsequent data storage on a disk pack. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



Parameter	Definition
LU	Logical unit.
TRACK	Track number to format.
STATUS	See PARM Status Codes, figure 4-1.

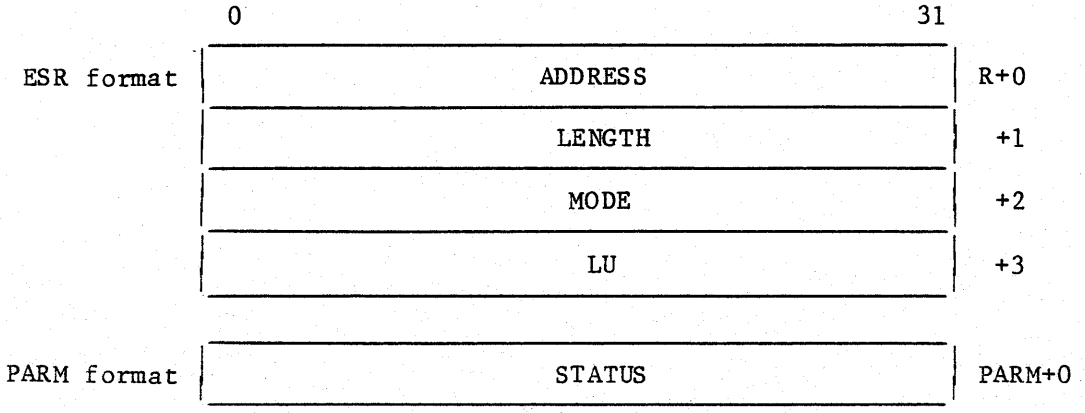
An example of a calling sequence is as follows:

```
        EXT     FORMAT      Externally defined symbol
        LDI,R4  47          Logical unit number
        LDI,R5  500         Track number
        MON,R4  FORMAT      Monitor request
```

READLU, Read From Logical Unit

READDS, Alternate Read From Logical Unit

The READLU and READDS ESRs initiate a data transfer from a specified logical unit to a buffer residing in the requesting task's memory. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



Parameter	Definition
ADDRESS	The address of the first element (word or byte) of the data buffer. The format of the data is determined by the MODE parameter.
LENGTH	The number of words (bytes) to be transferred. LENGTH values can be from 0 to 4096 words (0 to 16,384 bytes). A value of zero (0) is treated as the maximum LENGTH value of 4096 words (16,384 bytes).
MODE	The data transmission mode (format) code: 0 = ASCII record, word format 16 = ASCII record, byte format 32 = Binary record, word format
LU	Logical unit to be read.
STATUS	See PARM Status Codes, figure 4-1.

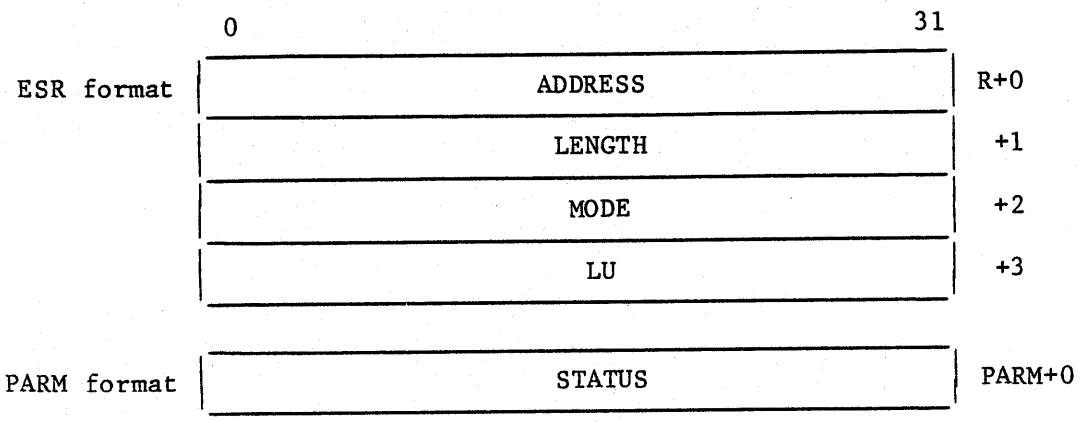
An example of the calling sequence is as follows:

```
        EXT       READLU      Externally defined symbol
        LDCA,R0   BUFA        Byte address of buffer
        LDI,R1    48          Number of bytes
        LDI,R2    16          ASCII records in byte format
        LDI,R3    15          Logical unit number
        MON,R0    READLU      Monitor request
```

WRITLU, Write to Logical Unit

WRITDS, Alternate Write to Logical Unit

The WRITLU and WRITDS ESRs initiate a data transfer to a specified logical unit from a buffer residing in the requesting task's memory. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the operation is completed.



Parameter	Definition
ADDRESS	The address of the first element (word or byte) of the data buffer. The format of the data is determined by the MODE parameter.
LENGTH	The number of words (bytes) to be transferred. LENGTH values can be from 0 to 4096 words (0 to 16,384 bytes). A value of zero (0) is treated as the maximum LENGTH value of 4096 words (16,384 bytes).
MODE	The data transmission mode (format) code: 0 = ASCII record, word format 16 = ASCII record, byte format 32 = binary record, word format
LU	Logical unit to be written to.
STATUS	See PARM Status Codes, figure 4-1.

An example of the calling sequence is as follows:

```
        EXT       WRITLU      Externally defined symbol
        LDCA,R0   BUFA        Byte address of buffer
        LDI,R1    48          Number of bytes
        LDI,R2    16          ASCII records in byte format
        LDI,R3    15          Logical unit number
        MON,R0    WRITLU      Monitor request
```

DEVICE CONTROL ESRs

The ability to control a device is provided by the Device Control feature and is invoked by the following ESRs:

BKSP	Backspace unit.
CLEAR	Clear unit.
ERASE	Erase tape segment.
FUNC	Function unit.
REWD	Rewind unit.
SELECT	Select operating mode.
SEOF	Search for end of file.
UINT	Unsolicited interrupt.
ULOC	Locate record on unit.
UNLD	Unload unit.
WEOF	Write end of file on unit.
DIAG	Run diagnostic test on unit.

These ESRs are legal on the devices as specified in appendix L.

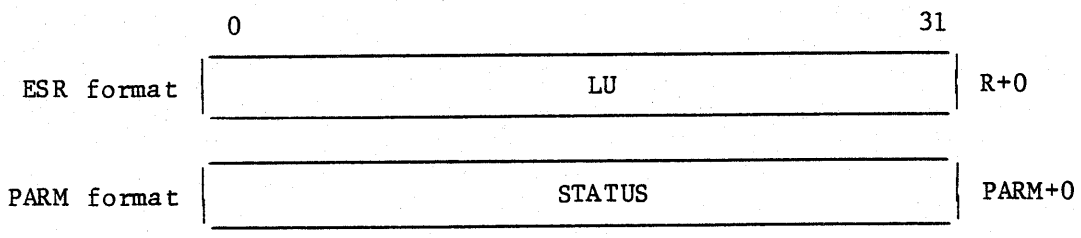
The following abort conditions are possible with these ESRs:

TYPE	ABORT CODE	
1	1	Operator rejected request to ready a unit.
1	2	Buffer size larger than 4096 words.
1	3	Logical unit unassigned.
1	4	Attempt to write on read-only file.
1	5	An input was attempted into a read-only page.
1	6	Hardware reject.
1	7	An input or output was attempted upon a protected page.
1	8	Illegal logical unit number.
1	9	Command is not legal for assigned device.

Hardware and data transmission error recovery procedures are error type and device dependent. The MPX/OS error recovery procedures are described in appendix E.

BKSP, Backspace Unit

The BKSP ESR positions the logical unit before the preceding physical record or block unless the logical unit is already at the beginning of the tape or file. The requesting task is scheduled for execution after the request is initiated and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



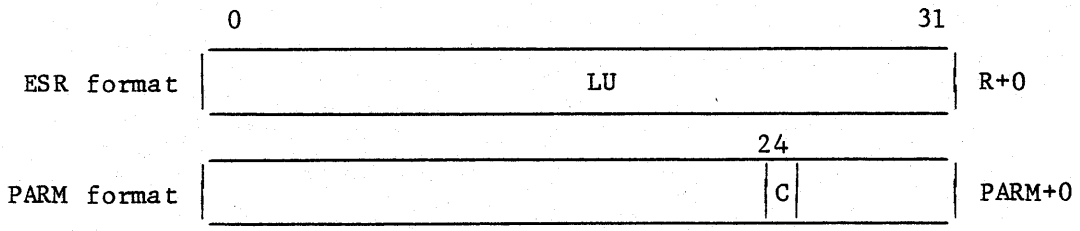
Parameter	Definition
LU	Logical unit to be backspaced.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
        EXT         BKSP        Externally defined symbol
        LDI,R0      10          Logical unit number
        MON,R0      BKSP        Monitor request
```

CLEAR, Clear Unit

The CLEAR ESR is called by a task to clear out the current I/O operation for a specified device. The command or data transfer is terminated at the point of receipt of the clear command. The task is put into I/O wait and not scheduled for execution until an end-of-operation interrupt is received. If the device is not busy, the clear request results in no action, and the task is scheduled for immediate execution.



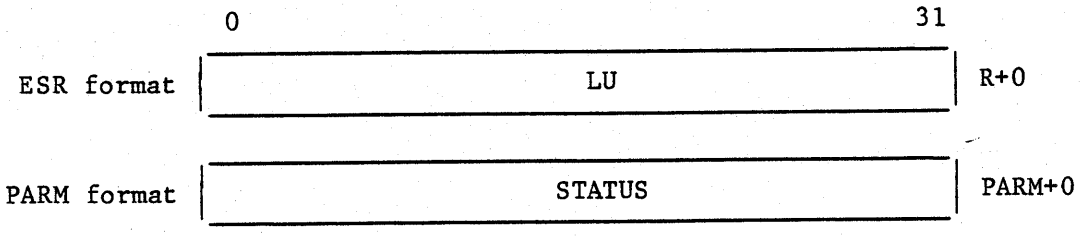
Parameter	Definition
LU	Logical unit.
C	Clear status. If set, an operation in progress was cleared; otherwise, the device was not busy.

An example of a calling sequence is as follows:

```
        EXT         CLEAR           Externally defined symbol
        LDI,R0      20              Logical unit number
        MON,R0      CLEAR           Monitor request
```

ERASE, Erase Tape Segment

The ERASE ESR erases approximately 6 inches of magnetic tape in an effort to bypass faulty material. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



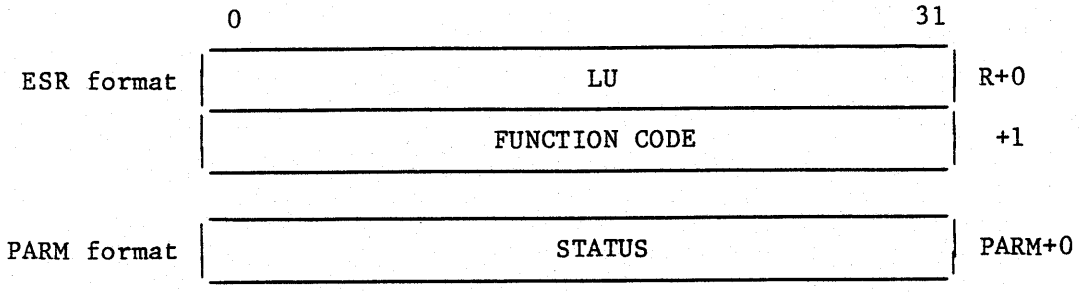
Parameter	Definition
LU	Number of logical unit to which the magnetic tape is assigned.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
EXT       ERASE     Externally defined symbol
LDI,R0    10        Logical unit number
MON,R0    ERASE     Monitor request
```

FUNC, Function Unit

The FUNC ESR is used to perform device dependent functions on a logical unit. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



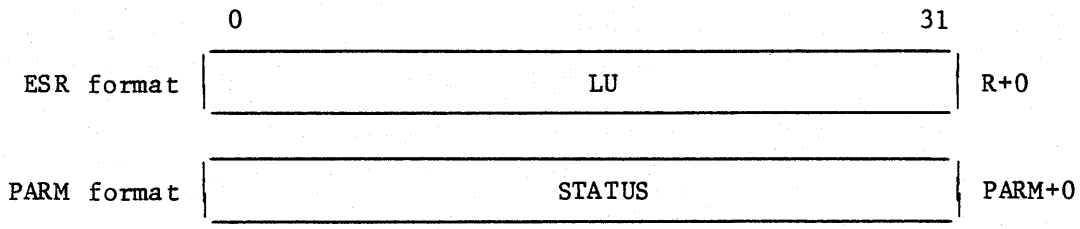
Parameter	Definition
LU	Logical unit.
FUNCTION CODE	Function codes as defined in appendix I.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
        EXT       FUNC        Externally defined symbol
        LDA,R0    ADDR        Buffer starting address
        LDI,R1    2           Buffer length
        LDI,R2    1           Function code
        LDI,R3    10          Logical unit number
        MON,R0    FUNC        Monitor request
```

REWD, Rewind Unit

The REWD ESR repositions a logical unit to the beginning-of-tape (BOT), for magnetic tape devices, and to the first block for mass storage devices. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



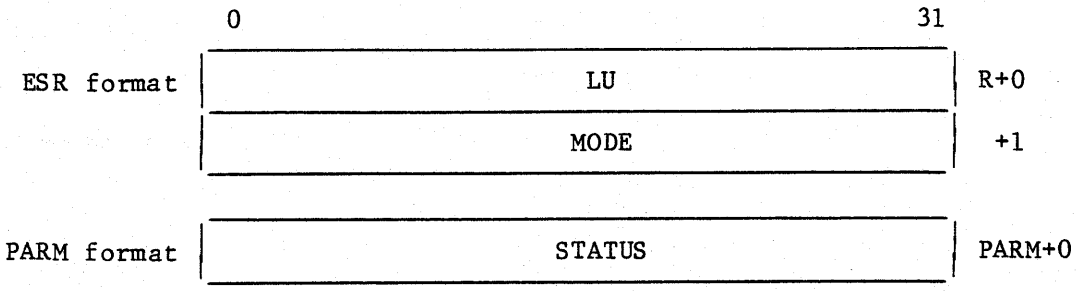
Parameter	Definition
LU	Number of the logical unit to be repositioned.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
EXT       REWD      Externally defined symbol
LDI,RB    21        Logical unit number
MON,RB    REWD      Monitor request
```

SELECT, Select Operating Mode

The SELECT ESR is used to set the operating mode for a logical unit. The available mode selections for each device are specified in appendix I.



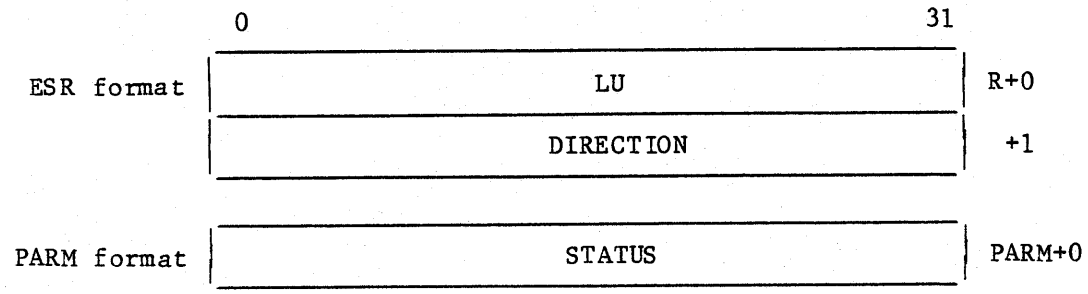
Parameter	Definition
LU	Logical unit.
MODE	The modes for specific devices are listed in appendix I.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
EXT       SELECT      Externally defined symbol
LDI,R0    12          Logical unit number
LDI,R1    1           Select mode
MON,R0    SELECT      Monitor request
```

SEOF, Search for End of File

The SEOF ESR initiates a search operation (forward or backward) on a specified logical unit for the next file marker, initial point of the file for backward searches, or end of the file for forward searches. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



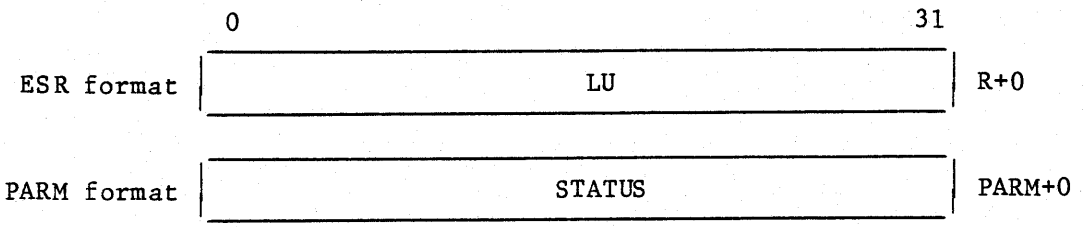
Parameter	Definition
LU	Logical unit to be searched for end-of-file mark.
DIRECTION	= 0, search forward = 1, search backward
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
EXT       SEOF      Externally defined symbol
LDI,R0    10        Logical unit number
LDI,R1    0         Search forward
MON,R0    SEOF      Monitor request
```

UINT, Unsolicited Interrupt

The UINT ESR allows the requesting task to be notified when an unsolicited interrupt occurs on a logical unit. The unsolicited interrupt could be used to signal the occurrence of a malfunction.



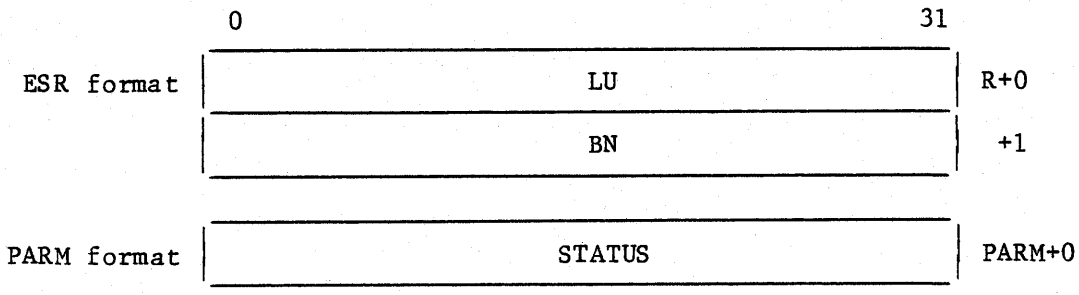
Parameter	Definition
LU	Number of the logical unit to receive unsolicited interrupt.
STATUS	See PARM Status Codes, figure 4-1.

An example of the calling sequence is as follows:

```
        EXT         UINT            Externally defined symbol
        LDI,R0      15              Logical unit number
        MON,R0      UINT            Monitor request
```

ULOC, Locate Record or Unit

The ULOC ESR sets the next block number of the logical unit to the requested block. If the requested block number is greater than the allocated area, the next block number is set to the last block written+1. The requesting task is scheduled for execution after the request is completed.



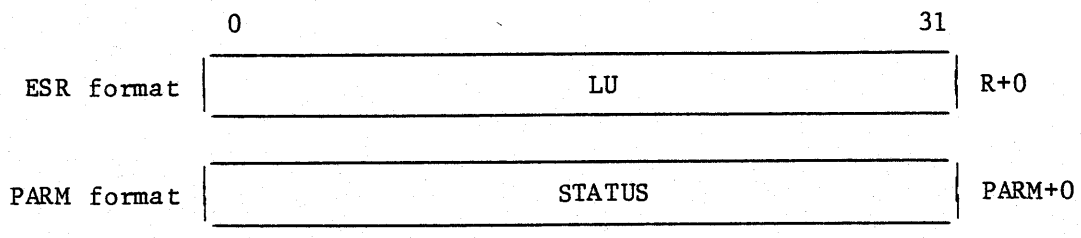
Parameter	Definition
LU	Logical unit to be positioned.
BN	Block number to which the unit is to be set. This value will be the next block read or written. If the value = -1, then the last block written +1 will be used.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
        EXT         ULOC        Externally defined symbol
        LDI,R3      10          Logical unit number
        LDI,R4      24          Block number
        MON,R3      ULOC        Monitor request
```

UNLD, Unload Unit

The UNLD ESR rewinds and makes not ready a magnetic tape unit. The requesting task is scheduled for execution after the request is initiated, and must issue a BSY, UST, or MUST ESR to determine when the request is completed. Following an unload of a tape, the tape drive is still assigned to the job and the logical unit.



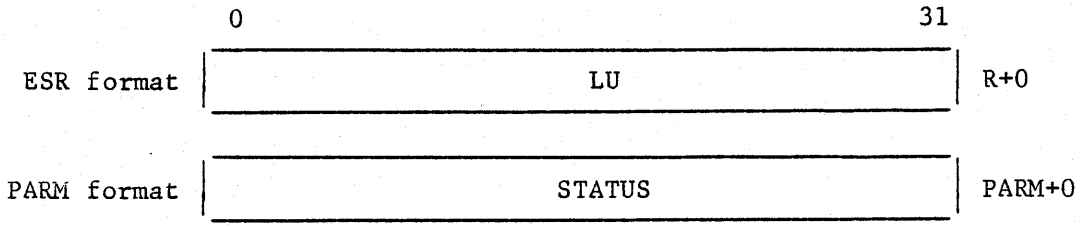
Parameter	Definition
LU	Number of the logical unit to be unloaded.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
        EXT       UNLD      Externally defined symbol
        LDI,R0    10        Logical unit number
        MON,RC    UNLD      Monitor request
```

WEOF, Write End of File on Unit

The WEOF ESR causes an end-of-file mark to be written on the specified logical unit (magnetic tape), or sets the number of the last block written to the current block number (disk file). The requesting task is scheduled for execution after the request is initiated and must issue a BSY, UST, or MUST ESR to determine when the request is completed.



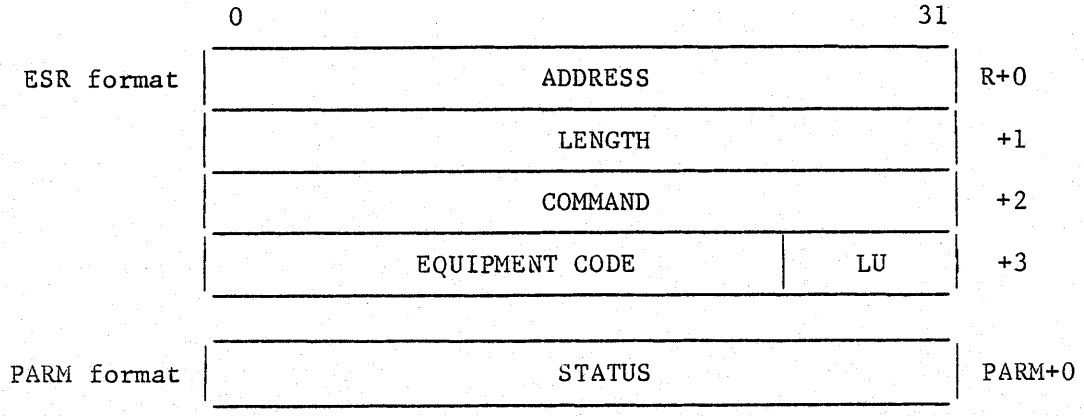
Parameter	Definition
LU	Logical unit end-of-file mark is to be written upon.
STATUS	See PARM Status Codes, figure 4-1.

An example of a calling sequence is as follows:

```
EXT       WEOF      Externally defined symbol
LDI,R0    10        Logical unit number
MON,R0    WEOF      Monitor request
```

DIAG, Run Diagnostic Test on Unit

The DIAG ESR allows a user task to initiate the on-line, self-test diagnostics of the GB138A controller. The DIAG ESR is processed like any I/O command. Upon completion, an end-of-operation interrupt and status are returned to the MP-32 CPU.



Parameter	Definition
ADDRESS	Buffer first word address.
LENGTH	Buffer length.
COMMAND	Commands are device dependent and are listed for the legal devices in appendix L.
EQUIPMENT CODE	The hardware equipment identification.
LU	Logical unit.
STATUS	Status codes are as follows: 0 = Diagnostic started 1 = Device does not exist 2 = Device assigned 3 = LU assigned 4 = No table space available

An example of a calling sequence is as follows:

```
        EXT       DIAG        Externally defined symbol
        LDA,R0    BFWA        Buffer first word address
        LDI,R1    48          Buffer length
        LDI,R2    CMND        Command
        LD,R3     EQUP        Equipment code
        LDI,R3    20,R3       Logical unit number
        MON,R0    DIAG        Monitor request
```

TASK MANAGER ESRS

The capability to control a task's execution is provided by the following ESRS:

ABORT	Voluntary Job Abort
CALL	Establish and Execute Task
DELJOB	Delete Job
DWAIT	Deferred Wait
OPENMEM	Assign Page of Open Memory
RELMEM	Release Memory Pages
RETURN	Terminate Task Execution
TASKRSQ	Resume/Suspend Task
TSCHED	Time Schedule Reactivation of Task
TSKCNGQ	Change Executing Task Parameters
TSTATUS	Return Task Status

The following descriptions explain the relationships between tasks. The initial task entry description explains the relationship between the job manager and tasks that the job manager brings into execution. The CALL ESR description is used by the job manager in response to the *LOAD/*RUN, statement sequence. The remaining descriptions are an extension of the capabilities provided at the job level.

When a task is loaded and placed in execution, a library routine (TSKMON) is loaded with the task and performs a return jump to the task's primary entry point. If the task exits through the primary entry point, TSKMON executes a return with release (the task is released from the system).

```
For example:  MAIN    UJP     **       (primary entry point)
                                .
                                .
                                .
                      UJI     MAIN     (task exits through TSKMON)
```

Alternately, a task may execute its own return operation with or without release. If a task returns without release, and is called again, it regains control after the RETURN monitor call. Such considerations only apply to tasks which multiprogram with each other. Figure 4-3 illustrates a multiprogramming relationship between tasks in a job. In Figure 4-3, task A calls task B and multiprograms with it. Task B calls task C and passes its common memory space. Task B may not multiprogram with task C. While each task has its own PARM area, they share access to the standard files (INP, OUT, and PUN). These files and their data must be accessed through the BLOCKER/DEBLOCKER package.

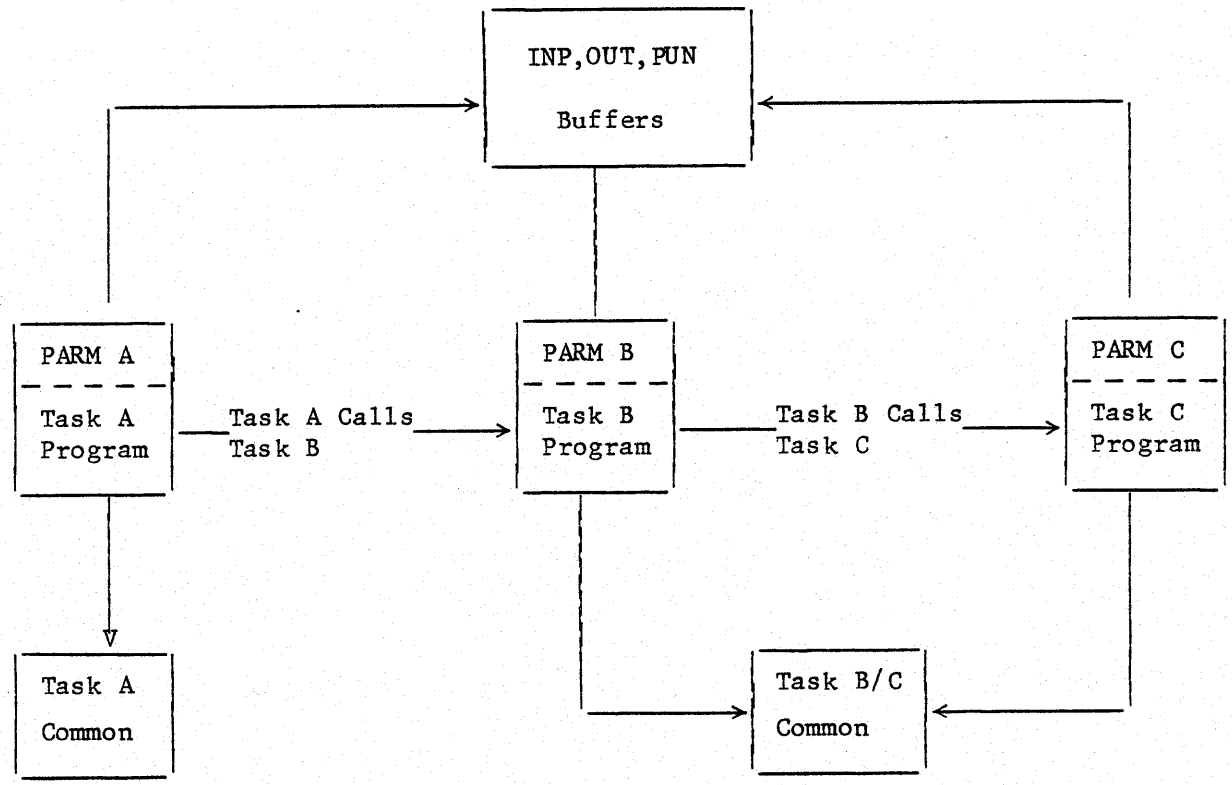


Figure 4-3. Multiprogramming Tasks

INITIAL TASK ENTRY

Upon entering a task from the job manager, the PARM region contains data associated with the task call. This includes task transfer addresses and task name parameters. For example, a task name control statement, such as *TST(I=01,L,X,R), would produce the following data in the PARM area.

PARM format:

		EP1	
		EP2	
		EP3	
		EP4	
		PRI	
I	=	0	1
,	L	,	X
,	R	ETX	

Parameter	Definition
EP	Entry-point addresses obtained from TRA loader directives. The first four encountered are saved. Execution begins at EP_1.
PRI	Priority of the task.
ETX	End-of-text (03) control character. The parameter string begins in PARM+5. The end is defined by the ETX character.

ABORT, Voluntary Job Abort

The ABORT ESR causes the job to enter the abort termination sequence. This sequence results in the production of abort dumps for all active tasks, the release of all job resources, the initiation of post processing of the standard output and standard punch files, and the removal of the job from the system. (This same sequence is entered upon the occurrence of task fault conditions for which the task has not requested return of control).

An example of a calling sequence is as follows:

```
EXT         ABORT       Externally defined symbol
MON,R0      ABORT       Monitor request
```

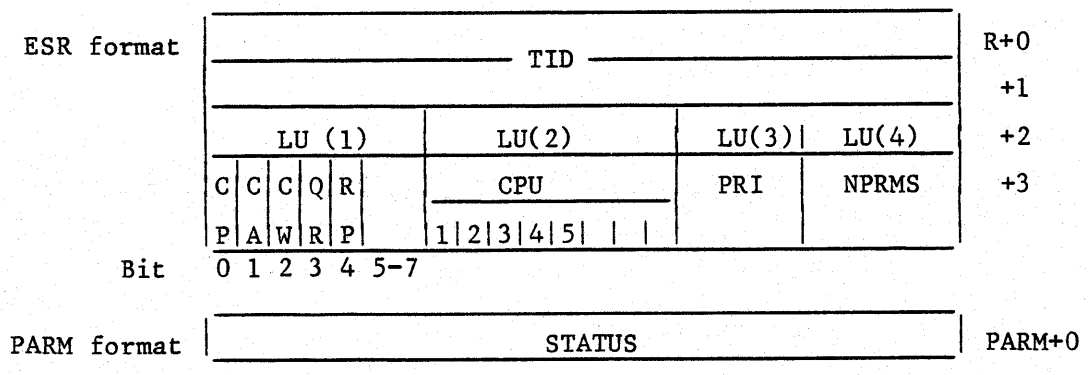
CALL, Establish and Execute Task

The call function is performed by MPX/OS for the user whenever a *LOAD, *RUN control statement is processed by the job manager. In a multiprogramming (or multiprocessing/multiprogramming) structure, the first task is placed into execution in this manner. Additional tasks are placed in execution by the user through the CALL ESR.

A task (the caller) that requires execution of some other task (the callee) issues a CALL ESR to establish the task and initiate its exectuion. The caller has the options of passing caller common to the callee, and/or of passing a copy of caller registers to the callee, and/or passing up to 40 words of parameter information through memory to the callee. If the caller does not pass common and does not expect to receive parameters to be returned from the callee, the caller also has the option to continue execution concurrent with the callee or to await the return of the callee before continuing execution. The caller can use the DWAIT or TSTATUS ESRs to effect synchronization with the callee(s). The caller can issue a maximum number of CALL ESRs as determined by the system configuration. Any attempt to exceed the maximum will cause the job to be terminated.

The CALL ESR can be issued with the callee in one of three states: nonexistent, dormant, or active. If the callee is active, the caller may elect to be scheduled by priority for connection to the callee or the caller may elect to have the CALL ESR rejected. This call status is returned in PARM. If the callee does not exist, it is established by the loader and placed on the ready list. If the callee is dormant, it is simply placed on the ready list.

The caller cannot be placed on the ready list for concurrent execution until the callee is placed on the ready list. During callee loading, or while access to an active callee, the caller has a CALL status. After the call connection is complete, the caller goes to the callee wait status until the callee returns or goes to the ready status for concurrent execution.



Parameter	Definition
TID	8-character task name. This task identifier is maintained by MPX/OS for use in DWAIT and TSTATUS ESRs.
LU	Logical unit numbers of files to be used as loader source if the callee must be loaded. The logical unit number values are expected to be in the range of 1 to 63. Out-of-range values are ignored. The loader examines the bytes in R+2 from left to right and attempts to load from all units with in-range values. The load terminates after all four bytes have been processed or after processing an ABS file. (Loading an ABS file overrides any previously loaded material.) An ABS file is recognized as such from the contents of the file header record.
CP	Common pass flag (bit 0):

Entry	Meaning
=0	Caller common not passed to callee.
=1	Caller common passed to callee.

CA — Common access flag (bit 1):

Entry	Meaning
=0	Common passed with read/write access.
=1	Common passed with read-only access.

CW — Common access flag (bit 2):

Entry	Meaning
=0	Caller waits for callee completion to continue.
=1	Caller can continue execution when call connection is complete.

QR — Queue/reject flag (bit 3):

Entry	Meaning
=0	Caller should be queued by priority for access to active.
=1	Call should be rejected if callee is active.

RP — Register pass flag (bit 4):

Entry	Meaning
=0	Copy of caller registers not passed to callee.
=1	Copy of caller registers is passed to callee.

CPU — A bit map of eligible CPUs. Each bit set enables potential selection of that CPU (numbers 1 through 5).

PRI — Priority designation. Valid values for real-time jobs are 1 through 511, and for non-real-time jobs are 10 through 255. If the priority definition is outside of the allowed range, the value is reset to the nearest permitted value. If the priority is 0, the callee assumes the priority of the caller (bits 16 through 23).

NPRMS — Defines the number of parameter words to be passed to the callee. Maximum value is 40; a zero value indicates that no words are passed. The parameter words are moved from caller memory area PARM+5 through PARM+4+NPRMS to callee memory area PARM+5 through PARM+4+NPRMS; see Initial Task Entry in this section (bits 24 through 31).

STATUS — ESR completion status is returned in PARM as follows:

Entry	Meaning
= -1	Call was rejected (callee active).
= 0	Call was successfully completed.
= 1	None of the selected CPUs are available.
= 2	Call rejected; no TCT table space available.

An example of a calling sequence is as follows:

```
            EXT         CALL                Externally defined symbol
            LDD,R0      TID                 Task name
            LD,R2       LUTBL               Logical unit number
            LD,R3       CONTRL
            MON,R0      CALL                Monitor request
            LD,H0       PARM                Check status
            .
            .
            .
TID         TEXT        8,TASKIDNT
LUTBL       VFD         8/LU1,8/LU2,8/LU3,8/LU4
CONTRL      VFD         1/CP,1/CA,1/CW,1/QR,1/RP,3/0,8/CPU,8/CPU,8/PRI,
                        8/NPRMS
```

NOTES: 1 A callee cannot call its caller nor can a caller call itself (circular calls).

2 Caller must await callee completion if common is passed.

3 Caller must await callee completion if parameters are expected on return of callee.

4 A user attempting to execute two or more tasks concurrently and share the same logical unit between the tasks must exercise caution. For example, if TASKA and TASKB are executing concurrently and both are performing I/O on the same unit, the following conditions can occur:

a) TASKA requests I/O on the unit, making the unit busy.

b) TASKB requests I/O on the unit but is threaded against the unit due to TASKA request.

c) TASKA I/O is completed, and the TASKB request is issued.

If TASKA requests unit status (UST), TASKA will receive a 0 (null) status because the I/O operation is not of TASKA. The safest approach to this type of concurrrent usage problem is to develop a third task, TASKC, through which all job I/O on the shared file is routed.

5 Memory scheduling for tasks within a job should be treated as if the tasks occupy totally separate areas of memory, even if common is passed. For example, assume that TASKA calls TASKB and passes common, that TASKA requires two pages of memory, that TASKB requires two pages of memory, and that common area is two pages of memory. It would appear that the memory requirement would be six pages of memory. However, the following sequence occurs.

a) TASKA is loaded and is put into execution. Four pages of memory (two program and two common pages) are in use.

b) TASKA calls TASKB, passing common. The loading process for TASKB requires three new pages: two for TASKB program code and one for common. Seven pages of memory are now allocated for the job.

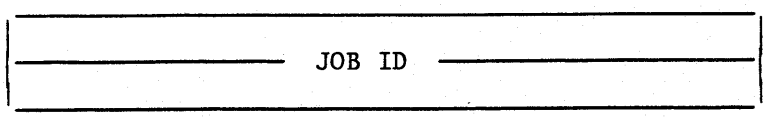
c) TASKB releases its common pages to accept the common pages from TASKA. Six pages of memory are now in use by the job.

6 The job must schedule seven pages of memory, even though six pages are sufficient to run the job.

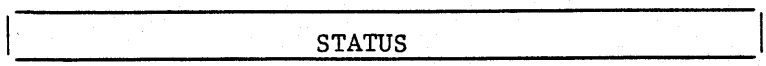
DELJOB, Delete Job

The DELJOB ESR enables a task to abort a job. The job may or may not contain the task initiating the request. Normal abort processing is initiated.

ESR format



PARM format



Parameter	Definition
JOB ID	Job identifier, left justified and blank filled to eight characters; identifier declared on *JOB or *RJOB control card.
STATUS	Numeric value indicating request success:

Entry	Meaning
=0	Job deleted successfully.
=1	Job does not exist.

An example of a calling sequence is as follows:

```
EXT         DELJOB          Externally defined symbol
LDD,R0      ='FIRSTJOB'     Job ID
MON,R0      DELJOB          Monitor request
```

DWAIT, Deferred Wait

A task that has called one or more callees and is executing concurrently with them may reach a point beyond which it should not continue until one or more of its callees have returned. The caller uses the DWAIT to defer the wait for callee completion until the most opportune time. By issuing a DWAIT ESR, the CPU becomes available for reassignment to another task, possibly a task for which the caller is waiting.

The DWAIT can specify one or more tasks. When any task on the wait list issues a RETURN ESR, the caller is placed on the ready list. If all tasks in the wait list have already returned, the caller is immediately rescheduled for execution.

ESR format

ADDRESS	R+0

PARM format

STATUS or RTID (0-3)	PARM+0
RTID (4-7)	PARM+1

ADDRESS format

TID_1(0-3)	ADDRESS+0
TID_1(4-7)	+1
LE or TID_2(0-3)	+2
TID_2(4-7)	+3
LE or TID_i(0-3)	+4

Parameter	Definition
ADDRESS	The full-word address of a list of task identifiers. Each identifier is eight characters. The list is variable in length, the first word following the last entry contains a -1 in place of an identifier. The maximum length of the list depends on the number of tasks allowed per job, an installation parameter.
STATUS	ESR status code:

Entry	Meaning
= -1	No task in the list is active.

	≠ -1	A task on the list has returned. Its identifier is in PARM and PARM+1.
RTID	The 8-character identifier of the returned task.	
LE	List end flag:	
	Entry	Meaning
	= -1	The list has ended.
	≠ -1	Another TID specification begins.
TID	An 8-character string defining the name of a task.	

An example of a calling sequence is as follows:

```
        EXT         DWAIT           Externally defined symbol
        LDA,H1      TIDTBL          Address of list
        MON,H1      DWAIT           Monitor request
        LD,R0       PARM            Check status
        .
        .
        .
TIDTBL  TEXT        8,TASK-ID1
        TEXT        8,TASK-ID2
        GEN         -1              List has ended
```

OPENMEM, Assign Page of Open Memory

The OPENMEM ESR allows a task to expand its scratch common or program area (in multiples of a page) within the limits specified on the *SCHED control statement. MPX/OS supplies the updated memory boundaries after each change. In addition, one call is provided to obtain the next available address in both regions without alteration of the memory limits.

ESR format

AREA	R+0
OPTION	+1

PARM format

STATUS or ADDRESS (1)	PARM+0
ADDRESS (2)	+1

Parameter	Definition
AREA	A flag that selects the program or common limit to be expanded:

Entry	Meaning
= -1	Program area is expanded.
= 0	Common area is expanded.

OPTION — A flag or value that determines the amount of memory increase desired and the content of the response words:

Entry	Meaning
= -1	All memory allowed by *SCHED and task unused space is added to the area selected by the AREA flag (MPX/OS response defines new limits of area selected by AREA flag).
= 0	Memory limits are not changed [MPX/OS response defines the next available program address (PARM+0) and the next available common address (PARM+1)]
= +n	n pages are added to the area selected by AREA flag [MPX/OS (MEM) response defines new limits of area altered].

Parameter	Definition
STATUS	Status of the ESR returned in PARM.

Entry	Meaning
= -1	ESR was rejected. Memory limits were not altered.
≠ -1	Memory limits were returned as per ADDRESS description.

ADDRESS — Except for the OPTION=0 case described above, ADDRESS(1) contains the address of the next available word and ADDRESS(2) contains the address of the last available word of the area selected by AREA flag. The next available address is the address adjacent to the allocated space for the region (small address for common, small for program) - but still in the page already allocated.

In the event the space requested (OPTION.GT.0) is not available, the call is rejected and PARM+0 is set to -1.

An example of a calling sequence is as follows:

```
EXT       OPENMEM     Externally defined symbol
LDI,R0    0           Request common pages
LDI,R1    -1          All scheduled pages
MON,R0    OPENMEM     Monitor request
LD,X7     PARM        Check status
```

RELMEM, Release Memory Pages

The RELMEM ESR is used to return common or program pages to the operating system. For program pages, only pages obtained through the use of the OPENMEM ESR may be released. MPX/OS returns the new memory limit definitions to the task in PARM.

ESR format

AREA	R+0
OPTION	+1

PARM format

STATUS	or	ADDRESS(1)	PARM+0
		ADDRESS(2)	+1

Parameter	Definition
AREA	A flag that selects the program or common area limit to be reduced:

Entry	Meaning
= -1	Program area is reduced.
= 0	Common area is reduced.

Parameter	Definition
OPTION	A flag or value that determines the amount (in pages) of the reduction and the content of the response from MPX/OS:

Entry	Meaning
= -1	Releases all common pages.
= 0	Returns memory limits only (see OPENMEM).
= n	Defines the number of pages to release.

Parameter	Definition
STATUS	A flag that defines the status of the ESR:

Entry	Meaning
= -1	ESR was rejected and memory limits were not altered.
≠ -1	Memory limits were returned as per ADDRESS description.

Parameter	Definition
ADDRESS	Except for the OPTION=0 described above, ADDRESS(1) contains the address of the next available word and ADDRESS(2) contains the address of the last available word of the area selected by AREA flag. The next available address is the address adjacent to the allocated space for the region (small address for common, large address for program), and the last available is the address most distant from the allocated space (large for common, small for program) - but still in the page already allocated.

An example of a calling sequence is as follows:

```
EXT       RELMEM     Externally defined symbol
LDI,R2    -1         Reduce program area
LDI,R3    2          Release two pages
MON,R2    RELMEM     Monitor request
LD,X6     PARM       Check status
```

RETURN, Terminate Task Execution

A task issues a RETURN ESR to notify its caller of completion of execution. When the returning task has active callees, the return cannot be completed until all active callees have returned; it is maintained with a FINIS status.

Every task must issue a RETURN to terminate normally. The loader supplies the module TASKMON from the system library; a subroutine exit from the primary entry point will return control to TASKMON which then issues the RETURN (with release).

ESR format

RELEASE	R+0
NUMBER	+1

Parameter	Definition
RELEASE	The memory release flag:

Entry	Meaning
= 0	Release the task memory and clear task identification from the system.
= -1	Do not release the task memory. The task assumes the dormant status.

NUMBER	The number of words that are to be passed back to the caller. The maximum number is 40. The parameter words are moved from callee memory area PARM+5 through PARM+4+NUMBER to caller memory area PARM+5 through PARM+4+NUMBER. If NUMBER=0, no parameter words are moved. The caller must specify call with wait to receive parameters from its callee.

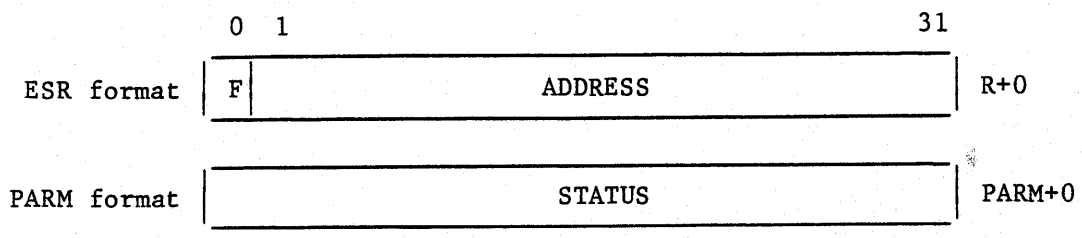
An example of a calling sequence is as follows:

```
EXT       RETURN      Externally defined symbol
LD,R0     0           Release flag
LDI,R1    10          Pass back 10 words
MON,R0    RETURN      Monitor request
```

NOTE: If a task is called after issuing a return without release, execution of the dormant task resumes with the instruction following the RETURN ESR.

TASKRSQ, Resume/Suspend Task

TASKRSQ allows a user to either suspend execution of a nonsystem task or to place a nonsystem task, which has been suspended, back into execution.



Parameter	Definition
F	F = 0 - Suspend request. = 1 - Resume request.
ADDRESS	First word address of the Task Control Table (TCT). The TCT is discussed in section 1, Tasks.
STATUS	0 - No Errors. 1 - Illegal to suspend specified task or resume request is for a task not in suspended status.

An example of the calling sequence is as follows:

```
EXT       TASKRSQ     Externally defined symbol
LD,RO     TCT         TCT address
MON,RO    TASKRSQ     Monitor request
LD,H2     PARM        Check status
```

TSCHED, Time Schedule Reactivation of Task

The TSCHED ESR allows a task to suspend its own execution for a specified length of time (in milliseconds). The issuing task regains control at the instruction following the MON instruction. The task is assigned a TSCHED status until the time period elapses. It is then assigned the READY status and is placed on the ready list to resume execution. The task is not charged for time when in TSCHED status.

ESR format

DELTAT	R+0

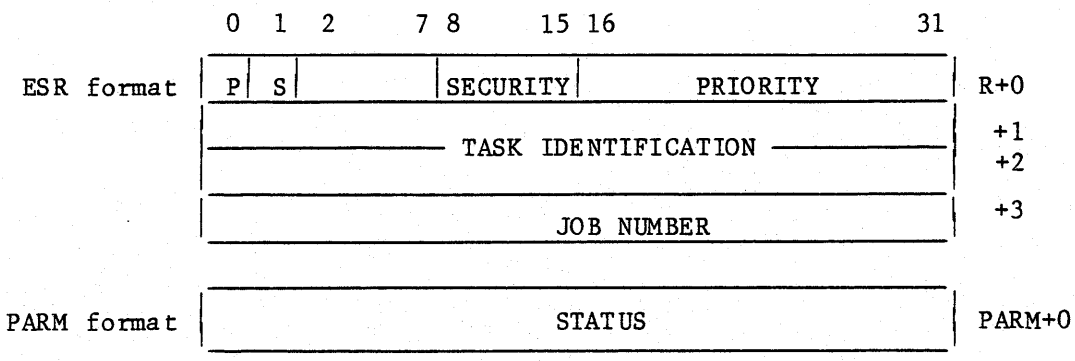
Parameter	Definition
DELTAT	The millisecond time interval that task execution is to be suspended. DELTAT must be positive.

An example of a calling sequence is as follows:

```
EXT       TSCHED      Externally defined symbol
LDI,R0    100         Time interval
MON,R0    TSCHED      Monitor request
```

TSKCNGQ, Change Executing Task Parameters

The TSKCNGQ ESR enables the user to change some parameters of an executing task.



Parameter	Definition
P	P = 0 No change in priority = 1 Change priority
S	S = 0 No change in security = 1 Change security
PRIORITY	New priority if P = 1 Legal PRIORITY values are as follows: 10 .LE. JOB priority .LE. 255 1 .LE. RJOB priority .LE. 511
SECURITY	New security if S = 1 New security must be .LE. Job Security level. Value must be in the range of 0 to 7.
TASKID	Identification of task to be changed.
JOB NUMBER	Used only when caller is System task.
STATUS	STATUS codes are as follows: 0 = ESR executed with no error 1 = JOB not found (invalid JOB Number) 2 = TASK not found (invalid TASK ID) 3 = PRIORITY is invalid 4 = SECURITY is invalid

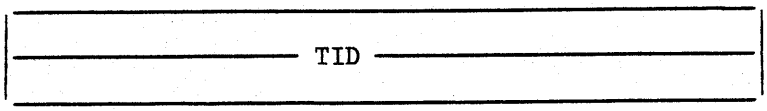
An example of the calling sequence is as follows:

```
        EXT         TSKCNGQ             Externally defined symbol
        LDI,RO      99                  New priority
        SF,RO       8
        SBIT,RO     0                   Set priority flag
        LDD,R2      ='TASKIDNT'         Task Identifier
        MON,RO      TSKCNGQ             Monitor request
```

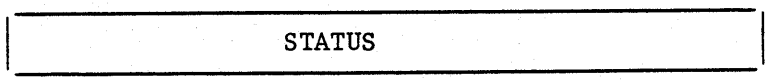
TSTATUS, Return Task Status

The TSTATUS ESR is used to obtain the status of the callee.

ESR format



PARM format



Parameter	Definition
TID	Eight characters defining the identifier of the task, for which the status is to be returned.
STATUS	A code defining the current status of the identified task is returned in PARM as follows:

Entry	Meaning
= -1	Task does not exist within job
= 0	Dormant
= 1	Active
= 2	I/O wait
= 4	File Manager wait
= 5	Call wait
= 6	Callee wait
= 7	Deferred wait
= 8	FINIS
= 9	TSCHED wait
= 10	Operator wait
= 12	MUST wait
= 13	Idle
= 14	SYSQS wait
= 15	Suspended

An example of a calling sequence is as follows:

```
EXT             TSTATUS             Externally defined symbol
LDD,R0          ='TASKIDNT'         Task identifier
MON,R0          TSTATUS             Monitor request
LD,H2           PARM                Check status
```

Refer to table 1-3 for descriptions of each status.

EVENT NOTIFICATION ESRs

The EVENT NOTIFICATION feature provides device and request status information, required by a user to manage an assigned logical unit.

The EVENT NOTIFICATION feature is invoked by the following ESRs:

BSY	Check logical unit busy.
CLREVTQ	Clear user defined event bit.
DATE	Return the current date.
DEFEVTQ	Define user event bit.
DTERCVQ	Define user error recovery routine.
ENABLE	Enable detection of user faults.
MUST	Wait on multiple events.
PFAULT	Enable user processing of page faults.
SETITMQ	Set interval timer event.
SETEVTQ	Set user defined event bit.
STATUS	Status unit.
TIME	Return the current time of day.
TETIME	Return TASKS execution time.
UST	Obtain unit status.
UTYP	Obtain dynamic unit status.

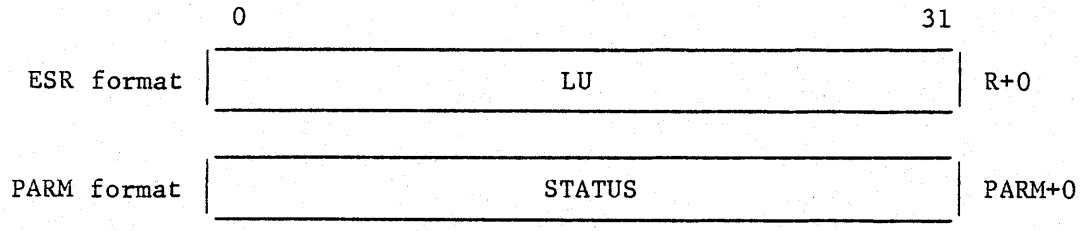
The BSY, UST, MUST, and STATUS ESRs are used to synchronize I/O request processing with task processing and could result in the task being placed in an I/O wait status. Use of the BSY ESR may degrade system performance. The MUST ESR could be used efficiently in place of the BSY ESR.

The following two abort conditions are possible with these ESRs:

ABORT TYPE	CODE	
1	3	Logical unit is not assigned to a device or file.
1	8	Logical unit value is not between 1 and 63.

BSY, Check Logical Unit Busy

The BSY ESR returns the busy/not-busy status of the specified logical unit. The status is not a function of a particular I/O request but rather of the logical unit itself. The requesting task is scheduled after the request is completed.



Parameter	Definition
LU	Number of the logical unit to be tested.
STATUS	Unit busy/not-busy status code:

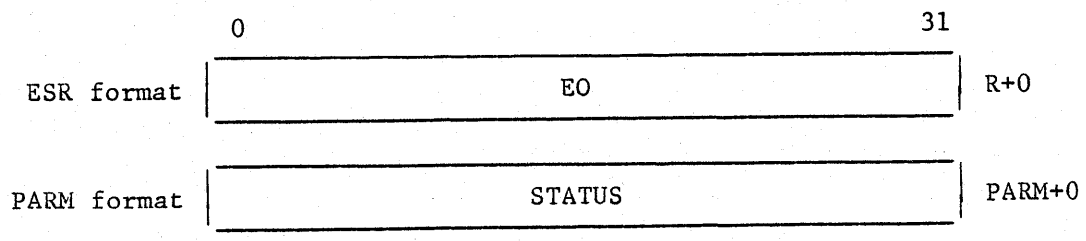
Entry	Meaning
= 0	Unit is not busy.
=-1	Unit is busy.

An example of a calling sequence is as follows:

```
EXT       BSY       Externally defined symbol
LDI,R0    10        Logical unit number
MON,R0    BSY       Monitor request
LD,H2     PARM      Unit busy/not-busy status
```

CLREVTQ, Clear User Defined Event Bit

The CLREVTQ ESR is used to clear an event bit. The clearing of an event bit is done automatically after the waiting task is scheduled. This ESR is used to clear a possible event before going into an event wait condition.



Parameter	Definition
EO	Event Ordinal as returned by DEFEVTQ.
STATUS	0 - Request was accepted. 1 - Invalid Event Ordinal.

An example of the calling sequence is as follows:

```
EXT       CLREVTQ     Externally defined symbol
LD,R0     EVENTORD    Event ordinal
MON,R0    CLREVTQ     Monitor request
LD,H2     PARM        Check status
```

DATE, Return the Current Date

The DATE ESR returns the current date to the user in ASCII format.

PARM format

M	M	/	D	PARM+0
D	/	Y	Y	+1

Parameter	Definition
MM	ASCII codes for month of year (01 through 12).
/	ASCII code for slash graphic.
DD	ASCII codes for the day of the month (01 through 31).
YY	ASCII codes for the year (00 through 99).

An example of a calling sequence is as follows:

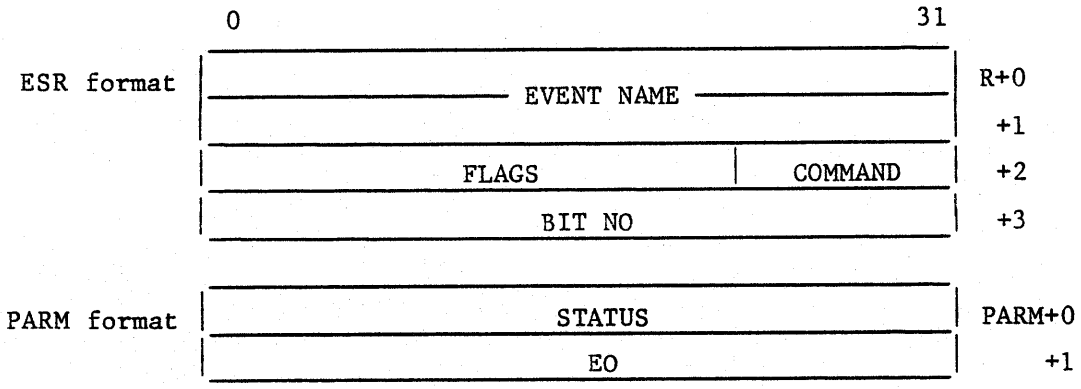
```
        EXT         DATE        Externally defined symbol
        MON,RA      DATE        Monitor request
        LDD,H2      PARM        Date in ASCII
```

DEFEVTQ, Define User Event Bit

The DEFEVTQ ESR creates a mapping from the user defined event name to the user defined event bit. Once the DEFEVTQ ESR is performed, event bits can be used to synchronize processing within a task or between tasks. These event bits can even be used to synchronize events across job streams. An event bit is a number between 0 and 127. An End of Operation event bit is predefined for assigned logical units.

Event bit definitions are automatically cleared upon the termination of the task that defined the bit.

Multiple tasks can set an event, but only the definer can be placed in wait state, pending the occurrence of the event.



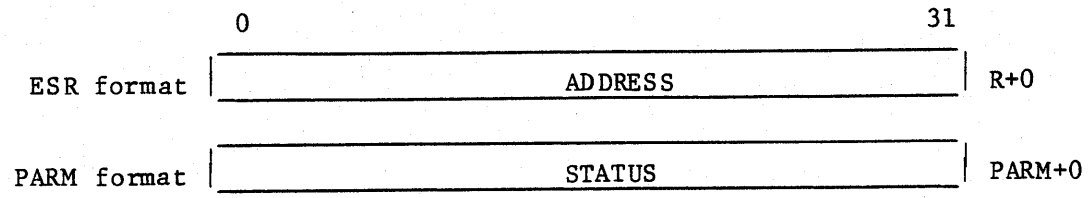
Parameter	Definition
EVENT NAME	User defined event name (8 ASCII characters).
FLAGS	*TBD*
COMMAND	Commands are as follows: 0 - Define event. 1 - Clear definition. 2 - Inquire.
BITNO	Event bit number (64-127) (Ignored for Inquire).
STATUS	Status codes are as follows: 0 - Request was accepted. 1 - Entry of that name already exists (define event). 2 - Entry of that name does not exist (Inquire). 3 - Attempt to clear event not defined by the calling task.
EO	Event Ordinal returned. This Ordinal is used in the SETEVTQ, CLREVTQ, and SETITMQ ESRs.

An example of the calling sequence is as follows:

```
EXT         DEFEVTQ          Externally defined symbol
LDD,R0      ='EVNTNAME'      Event name
LDI,R2      0                Define event
LDI,R3      127              Event bit number
MON,R0      DEFEVTQ          Monitor request
LDD,H2      PARM             Check status
TST,NE      H2,XO,ERR        If error
ST,H3       EVENTORD         Save event ordinal
```

DTERCVQ, Define User Error Recovery Routine

The DTERCVQ ESR allows the user to perform his own error recovery on all devices except mass storage. The user routine is passed the logical unit and error code. The execution of a DTERCVQ ESR overrides all error recovery processing performed by the system (except mass storage).



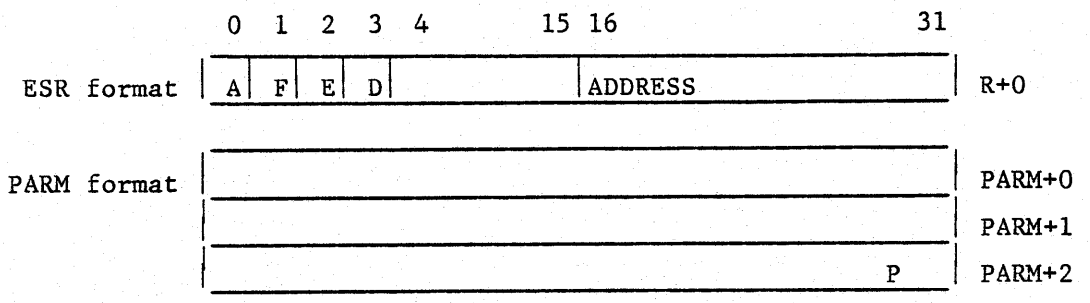
Parameter	Definition
ADDRESS	Error recovery routine address.
STATUS	See PARM Status Codes, figure 4-1.

An example of the calling sequence is as follows:

```
EXT       DTERCVQ     Externally defined symbol
LDA,R0    ERR         Error recovery routine address
MON,R0    DTERCVQ     Monitor request
LD,H2     PARM        Check status
```

ENABLE, Enable Detection of User Faults

The ENABLE ESR enables hardware detection of the arithmetic faults and defines the user interrupt routine which will process the interrupts when they occur. The MPX Operating System provides a default interrupt processor which will abort the job. The user can select one interrupt for each fault, one interrupt routine for all faults or other combinations. The interrupt processor definition can be changed as often as desired, but once interrupt checking has been enabled, it cannot be disabled.



Parameter	Definition
A	Select arithmetic fault detection/control if bit = 1.
F	Select function fault detection/control if bit = 1.
E	Select exponent fault detection/control if bit = 1.
D	Select divide fault detection/control if bit = 1.
ADDRESS	16-bit field containing address of interrupt processor:
P	When control is returned to ADDRESS, the address of the instruction causing the fault is returned in PARM+2.

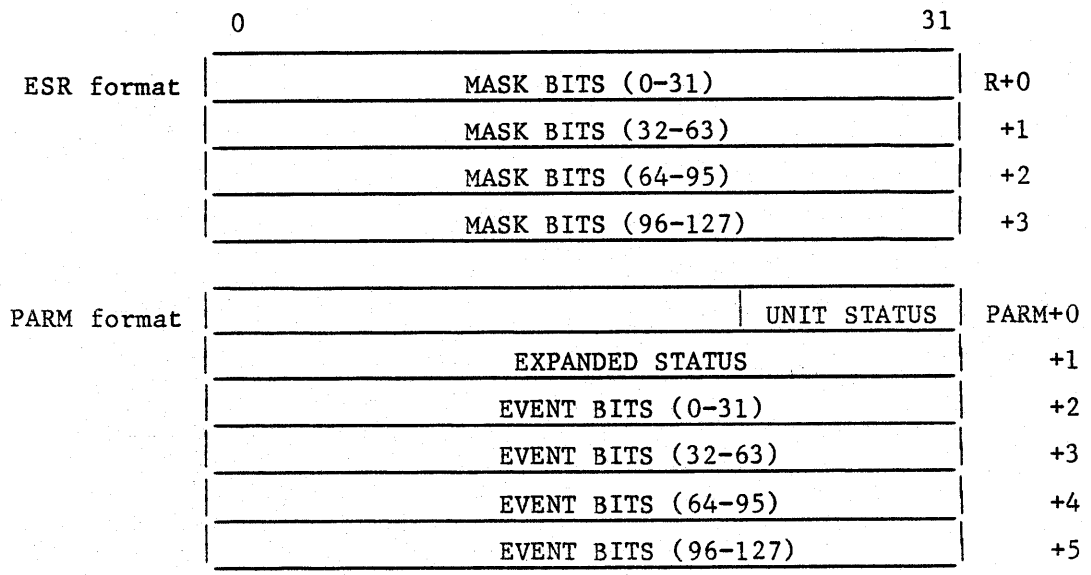
An example of a calling sequence is as follows:

```
        EXT     ENABLE                  Externally defined symbol
        LD,R0   INTRMSK
        MON,R0  ENABLE                  Monitor request

          .
          .
          .

INTRMSK VFD     1/1,1/0,1/1,1/0,12/0,16/INTADR  Go to INTADR on an
                                                arithmetic or exponent
                                                fault.
```

MUST, Wait on Multiple Events

The MUST ESR allows the requesting task to be placed in a wait state pending the occurrence of specified events. The requesting task supplies a 128-bit mask where bits 1 through 63 correspond with the end of operation on logical units 1 through 63 respectively. The other bits (0, 64-127) correspond to the user defined events (DEFEVTQ ESR).

The four registers are treated as a 128-bit mask with each bit corresponding to an event. A set bit enables the detection of the corresponding event bit. The issuing task will be scheduled for execution if an event and event mask bit are both set (one).



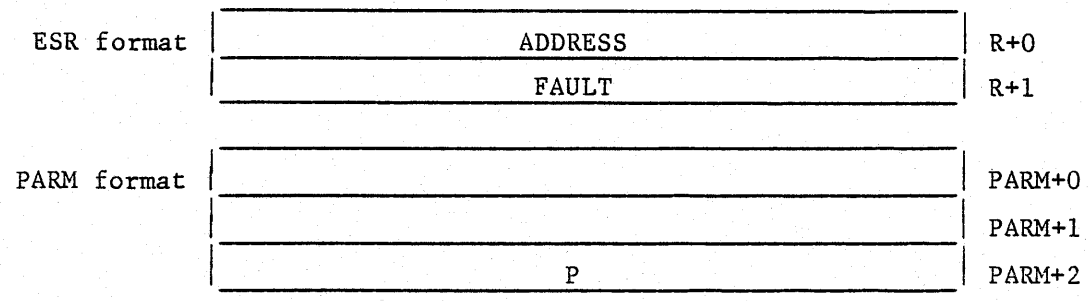
Parameter	Definition
MASK BITS	The event bits (0-127) are set to a one for each event which has occurred.
UNIT STATUS/ EXPANDED STATUS	These parameters are set for the lowest numbered logical unit with an event bit set. See description in appendix I.

An example of a calling sequence is as follows:

```
        EXT       MUST        Externally defined symbol
        LDD,R0    MASK+0      Event Mask Bits
        LDD,R2    MASK+2      Event Mask Bits
        MON,R0    MUST        Monitor Request

MASK    GEN       0
        GEN       0
        GEN       0
        GEN       $8F000000
```

PFAULT, Return Control on Program Faults

The PFAULT ESR defines an address in the issuing task (or in the executive, if address is zero) to which control should be directed upon the occurrence of a page fault or an illegal instruction fault interrupt. Each PFAULT ESR can define one of the conditions and its return-of-control address.



Parameter	Definition
ADDRESS	The address at which the task will be restarted after the fault is detected.
FAULT	A flag defining the fault condition for which the address is valid:

Entry	Meaning
= 0	Page faults return address.
= 1	Illegal instruction return address.

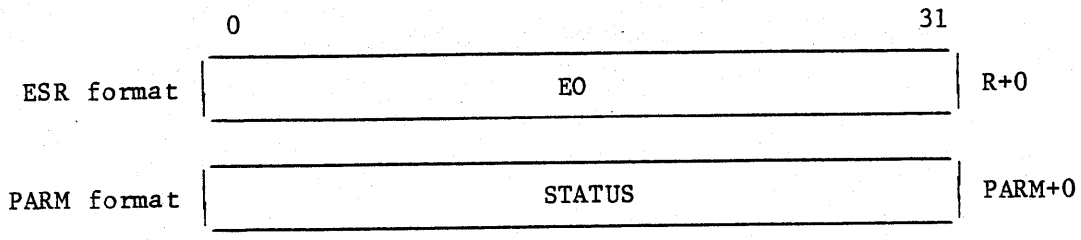
P	Address of instruction executed at the time the fault was detected.

An example of a calling sequence is as follows:

```
EXT         PFAULT          Externally defined symbol
LDA,R1      PFAULTADR       Address for return
LDI,R2      1               An illegal instruction
MON,R1      PFAULT          Monitor request
```

SETEVTQ, Set User Defined Event Bit

The SETEVTQ ESR is used to set a previously defined event bit. As stated previously, event bits can be used to synchronize separate task processing. The SETEVTQ ESR will set the specified event bit. If the defining task is waiting for the occurrence (event bit set) then the waiting task will be scheduled for execution.



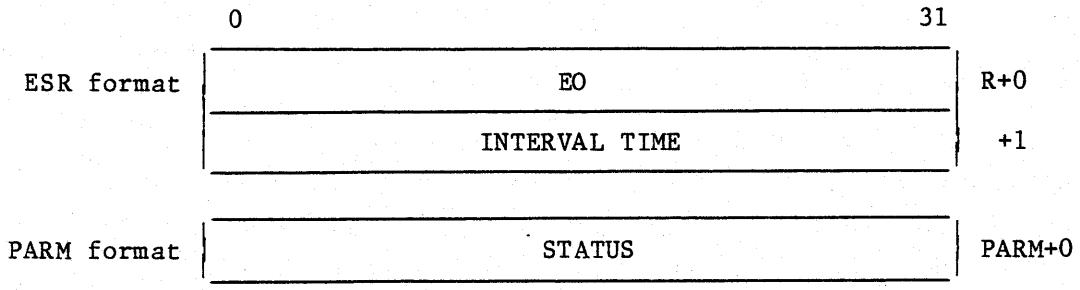
Parameter	Definition
EO	Event ordinal as returned by DEFEVTQ.
STATUS	0 - Request was accepted. 1 - Invalid Event Ordinal.

An example of the calling sequence is as follows:

```
        EXT     SETEVTQ     Externally defined symbol
        LD,R0   EVENTORD    Event ordinal
        MON,R0  SETEVTQ     Monitor request
        LD,H2   PARM        Check status
```

SETITMQ, Set Interval Timer Event

The SETITMQ ESR is used to notify the system that a predefined event bit is to be continually set after the elapse of the specified time interval. This interval time event remains active for the life of the task.



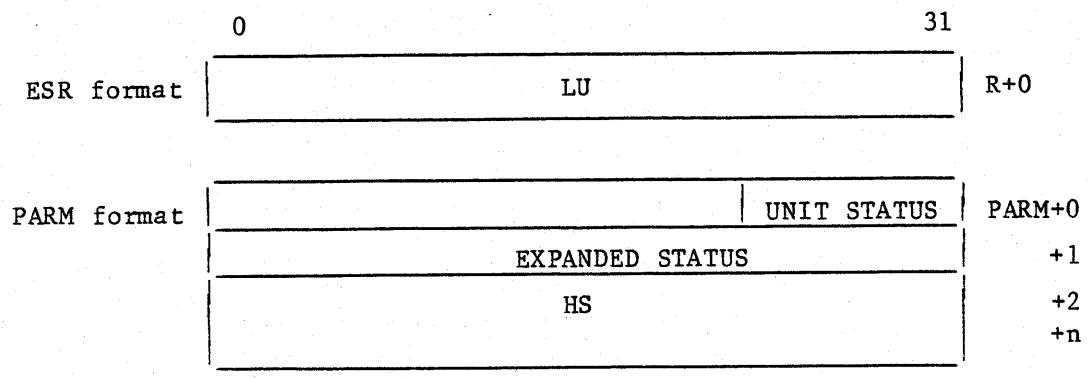
Parameter	Definition
EO	Event ordinal as returned by DEFEVTQ.
INTERVAL TIME	Interval time in milliseconds.
STATUS	0 - Request was accepted. 1 - Invalid Event Ordinal.

An example of the calling sequence is as follows:

```
        EXT       SETITMQ      Externally defined symbol
        LD,R0     EVENTORD     Event ordinal
        LDI,R1    1000         Interval time (milliseconds)
        MON,R0    SETITMQ      Monitor request
        LD,H2     PARM         Check status
```

STATUS, Status Unit

The status of a request on a logical unit, which has an operation in progress, can be tested using the STATUS ESR. The request places the requesting task in the I/O Wait state until the End of Operation event has been processed on the logical unit. If the issuing task has no request pending for the specified unit, then a null status (=0) is returned in the user's parameter area.



Parameter	Definition
LU	Logical unit.
UNIT STATUS	Normal unit status as described in appendix I.
EXPANDED STATUS	Expanded status, as described in appendix I.
HS	Hardware status as described in appendix I.

An example of a calling sequence is as follows:

```
EXT       STATUS      Externally defined symbol
LDI,R0    10          Logical unit number
MON,R0    STATUS      Monitor request
```

TETIME, Task Elapsed Time

The TETIME ESR obtains the number of milliseconds accumulated from the time the task was initialized until the time the monitor call to TETIME was made. The time is returned in PARM.

PARM format

TIME	PARM+0

Parameter	Definition
TIME	Task time (in milliseconds) accumulated

An example of a calling sequence is as follows:

```
        EXT         TETIME          Externally defined symbol
        MON,RO      TETIME          Monitor request
        LD,H2       PARM            Task Elapsed Time
```

TIME, Return Current Time of Day

The TIME ESR returns the current time of day in ASCII and binary formats.

PARM format

H	H	:	M	PARM+0
M	:	S	S	+1
TIME				+2

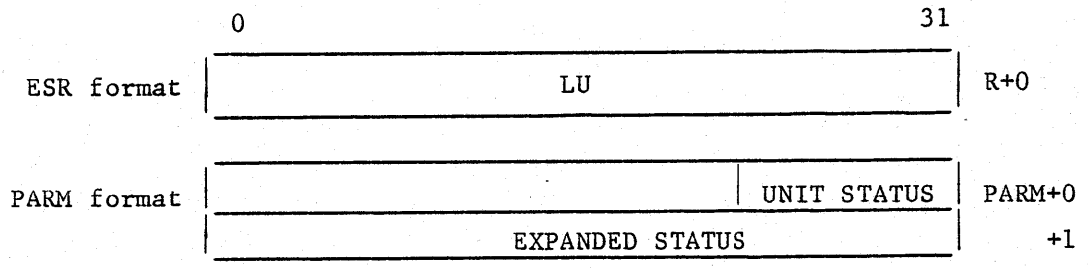
Parameter	Definition
HH	ASCII codes for hour of day (00 through 23).
:	ASCII code for colon graphic.
MM	ASCII codes for minute of hour (00 through 59).
SS	ASCII codes for second of minute (00 through 59).
TIME	Time of day in milliseconds since midnight.

An example of a calling sequence is as follows:

```
        EXT         TIME        Externally defined symbol
        MON,R0      TIME        Monitor request
        LDD,H2      PARM        Current time in ASCII
```

UST, Obtain Unit Status

The UST ESR returns the status of the requesting task's last I/O request on the specified logical unit. If the I/O is still pending, the requesting task is placed in I/O wait until the I/O is completed.



Parameter	Definition
LU	Logical unit to be tested.
UNIT STATUS	See description in appendix I.
EXPANDED STATUS	See description in appendix I.

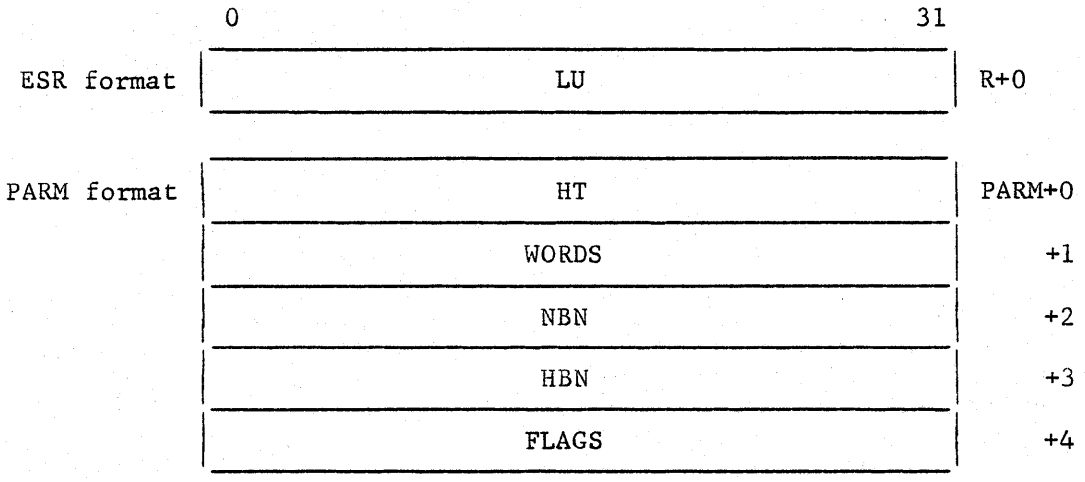
An example of a calling sequence is as follows:

```
EXT       UST       Externally defined symbol
LDI,R0    10        Logical Unit number
MON,R0    UST       Monitor request
LD,H2     PARM      Unit status
```

UTYP, Obtain Dynamic Unit Status

The UTYP ESR returns the hardware type of the specified logical unit. If the hardware type is a disk file, additional file description information is also returned. The requesting task is scheduled for execution after the request is completed.

The values returned in PARM by the UTYP ESR may be modified during system generation. In addition, the peripheral equipment configuration is defined during system generation.



Parameter	Definition
LU	Logical unit for which hardware type is to be returned.
HT	Hardware type. Valid hardware types are defined in appendix K.
WORDS	Number of words per block. Returned for HT = 1 only.
NBN	Number of next block to be read or written (current block number). Returned for HT = 1 only.
HBN	Highest block number written (end-of-file). Returned for HT = 1 only.
FLAGS	Bit 0 - set if device is a disk file. Bit 1 - set if device is a magnetic tape. Bit 2 - set if device is a blocked device. Bit 3 - set if device is an input-only device. Bit 4 - set if device is an output-only device. Bit 5 - set if device is an ASCII-only output device. Bit 6 - set if device is an interactive terminal. Bit 7 - set if device is a remote batch terminal. Bit 8 - set if device is a communication network.

An example of a calling sequence is as follows:

```
EXT         UTYP        Externally defined symbol
LDI,R0      10          Logical unit number
MON,R0      UTYP        Monitor request
LD,H2       PARM        Hardware Type
```

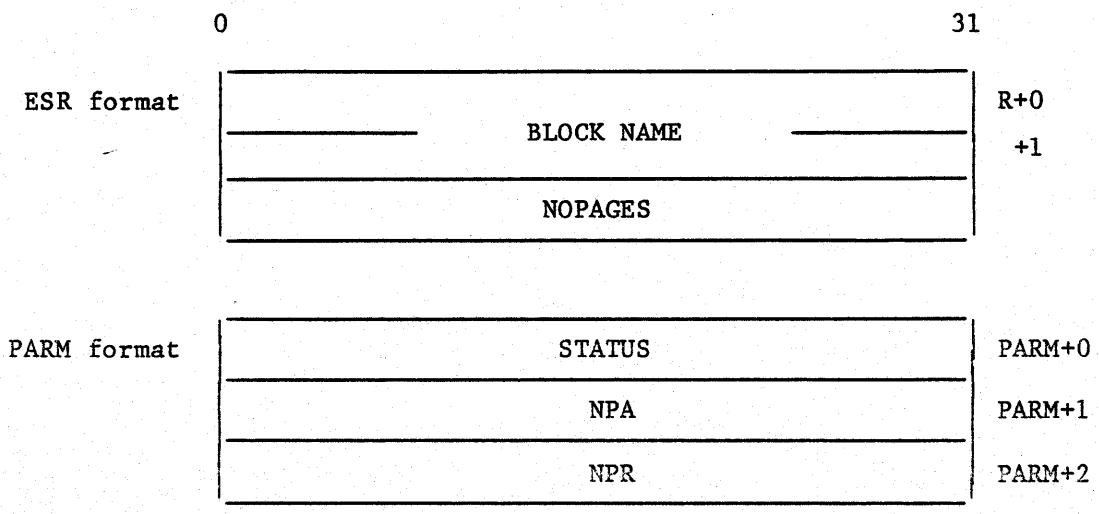
MISCELLANEOUS ESRs

The following ESRs allow a task to communicate with the operator and obtain the date and time, manage global common, and pass messages to and from other tasks.

ASNGC	Assign Global Common
CTOC	Send Command Message to Operator
CTOI	Send Information Message to Operator
CTOR	Receive Message from Operator
DAYFILEQ	Send Message to DAYFILE
DEFGC	Return Defined Global Common Page Numbers to User
GETGC	Attach Global Common
JOBINFO	Return Job Information
MGETGC	Modified Get Global Common
NXTNUM	Get Unique Identifier
RELGC	Release Assigned Global Common
RETGC	Return Global Common
STATGC	Status Global Common

ASNGC, Assign Global Common

The ASNGC ESR causes memory to be assigned as global common. Available memory pages are assigned and the physical page numbers saved. An entry is made containing the specified Global Common name to be associated with the assigned pages, the number of pages assigned, and which physical pages are assigned.



Parameter	Definition
BLOCK NAME	Global common block name, left justified and blank filled to eight characters.
NOPAGES	Number of memory pages to be reserved.
STATUS	Numeric value indicating request status:

Entry	Meaning
= -2	Block name already defined.
= -1	No table space available.
= 0	No memory available.
> 0	Request successful.

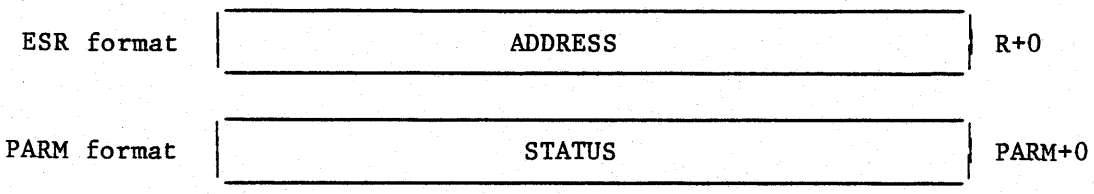
NPA	Number of pages assigned; minimum of number of pages requested and number available.
NPR	Number of pages unassigned pages remaining in system.

An example of the calling sequence is as follows:

```
EXT         ASNGC          Externally defined symbol
LDD,H0      ='BLOCKONE'    Block name
LDI,H2      2              Number of pages
MON,H0      ASNGC          Monitor request
```

CTOC, Send Command Message to Operator

The CTOC ESR allows a task to send a message to the operator and requires a response. Once the ESR is issued, the task does not resume execution until the operator responds. While the task is waiting for operator response, it is assigned operator wait status. When the response is entered by the operator, both the command message and the response are logged in the system dayfile.



Parameter	Definition
ADDRESS	Byte address of the first byte of the message to be displayed. The message is 65 characters in length or is terminated at the occurrence of a 03 (end-of-text, ETX) character value.
STATUS	ESR status or operator response code:

Entry	Meaning
= 0	Message accepted by operator.
= 1	Message rejected by operator.

An example of a calling sequence is as follows:

```
          EXT       CTOC                 Externally defined symbol
          LDCA,R1   ADDRMSG              Address of message
          MON,R0    CTOC                 Monitor request
          LD,X7     PARM                 Check status
                    .
                    .

ADDRMSG   TEXTC     30, THIS IS A MESSAGE TO OPERATOR
          GEN,C     $03                  End of text
```

CTOI, Send Informative Message to Operator

The CTOI ESR allows a task to send a message to the operator and does not require a response. The issuing task is scheduled for execution after the ESR has been processed.

ESR format

ADDRESS	R+0

PARM format

STATUS	PARM+0

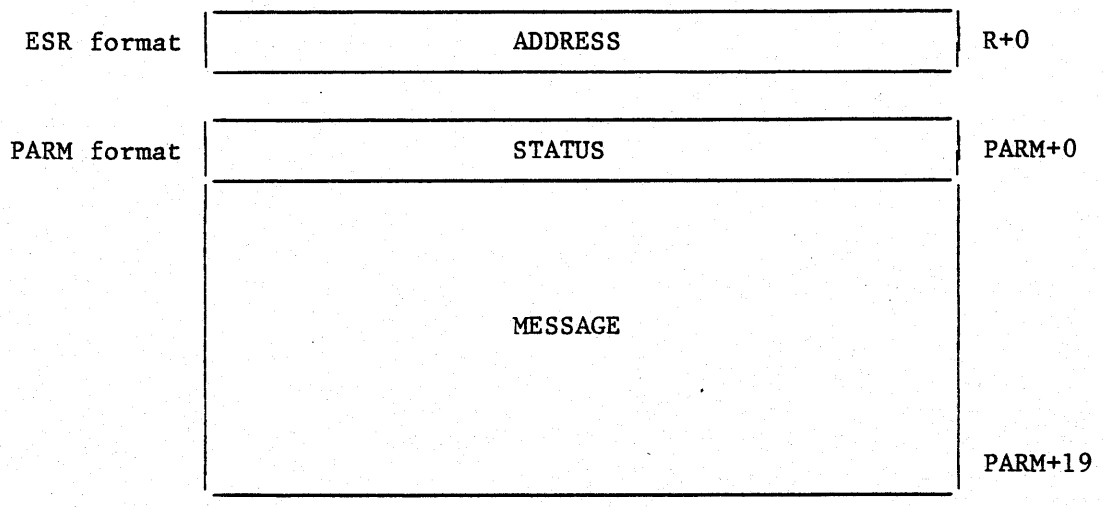
Parameter	Definition
ADDRESS	Byte address of the first byte of the message to be displayed. The message is 65 characters in length or is terminated at the occurrence of a 03 (end-of-text, ETX) character value.
STATUS	See PARM Status Codes, figure 4-2.

An example of a calling sequence is as follows:

```
          EXT       CTOI                Externally defined symbol
          LDCA,R3   ADDRMSG             Address of message
          BSK,S     R3,0,*+1            Set reject
          MON,R3    CTOI                Monitor request
          LD,X7     PARM                Check status
                    .
                    .
ADDRMSG   TEXTC     18, THIS IS A COMMENT
          GEN,C     $03                 End of test
```

CTOR, Receive Message From Operator

The CTOR ESR allows a task to send a message to and receive a message from the operator. Once the ESR is issued, the task does not resume execution until the operator responds with an input message. While the task is waiting for operator response, it is assigned operator wait status. When the response is entered by the operator, both the command message and the response are logged in the system dayfile.



Parameter	Definition
ADDRESS	Byte address of the first byte of the message to be displayed. The message is 65 characters in length or is terminated at the occurrence of an ETX (end-of-text, ASCII code = 03) character.
STATUS	Value defining success of request:

Entry	Meaning
= 0	Message accepted and response message received from operator.
= 1	Message rejected and response message received from operator.

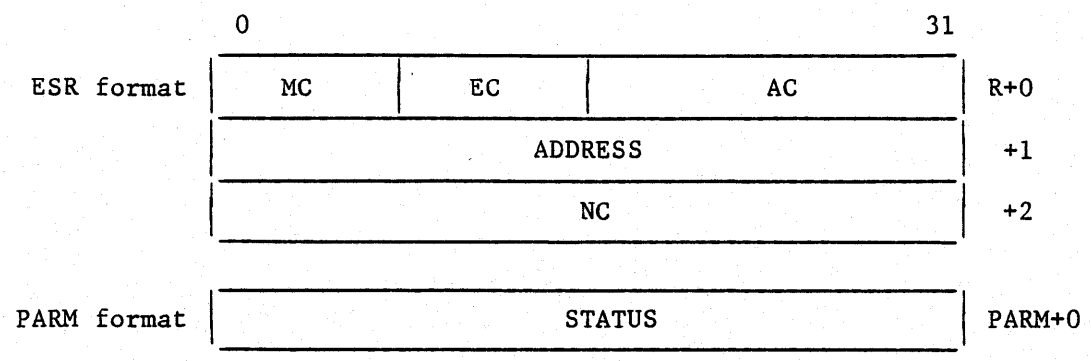
MESSAGE	First 72 characters of respose message.

An example of a calling sequence is as follows:

```
          EXT          CTOR                    Externally defined symbol
          LDCA,R1      ADDRMSG                 Address of message
          MON,R0       CTOR                    Monitor request
          LD,X7        PARM                    Check status
                       .
                       .

ADDRMSG   TEXTC        30, THIS IS A MESSAGE TO OPERATOR
          GEN,C        $03                     End of text
```

DAYFILEQ, Send Message to the Dayfile

The DAYFILEQ ESR provides the capability to write a user message and associated code to the DAYFILE.



Parameter	Definition
MC	Message Identification Code (one ASCII character) A - Accounting C - JOB Control E - Error Log O - Operator Message P - Permanent File Action R - Reserved for Applications S - Statistical Data U - Utilization Z - System Overhead
EC/AC	Event Code/Action Code (three ASCII characters) Accounting (A) TBD Job Control (C) Not used Error Log (E) Event Code: I - Informative Action Code: xx = See device codes, appendix I Operator Message (O) ACC Operator Accept Response C Operator Command Message (CTOC) I Operator Information Message (CTOI) R Receive Operator Message (CTOR) REJ Operator Reject Response

Permanent File Action (P)
  Event Code:
    A - ALLOCATE ESR
    C - CLOSE ESR
    D - DEVICEQ ESR
    E - EXPANDQ ESR
    M - MODIFY ESR
    O - OPEN ESR
    R - RELEASE ESR
    S - SAVE ESR
    X - DEVICEQ ESR
  Action Code:
    xx = Error Code as returned in PARM

Reserved for Applications (R)

Statistical Data (S)
  TBD

Termination (T)

Utilization (U)
  TBD

System Overhead (Z)
  Event Code:
    A - Archive DAYFILE
    B - New DAYFILE
    D - New day
    E - DAYFILE previously archived message
    Z - DEADSTART message
  Action Code:
    xx = Resident edition number

ADDRESS	Word address for start of user message.
NC	Number of ASCII characters in message.
STATUS	See PARM Status Codes, figure 4-2.

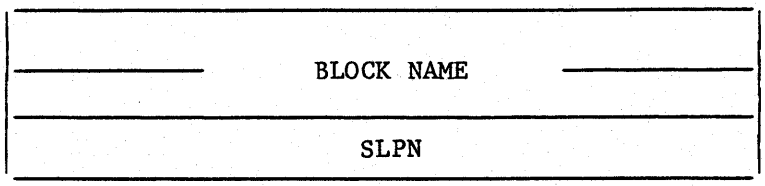
An example of the calling sequence is as follows:

```
        EXT       DAYFILEQ        Externally defined symbol
        LD,R0     ='IEAC'         MC/EC/AC codes
        LDA,R1    MESSAGE         Address of Dayfile message
        LDI,R2    MSGLNGTH        Message length
        MON,R0    DAYFILEQ        Monitor request
```

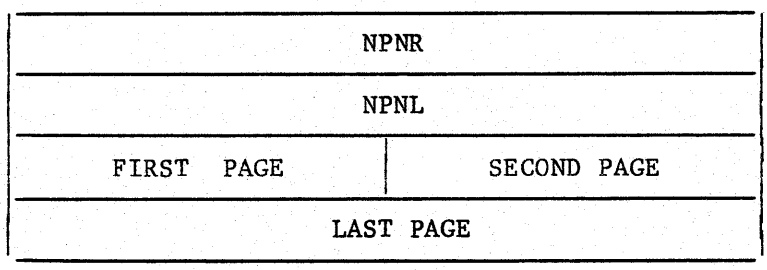
DEFGC, Return Defined Global Common Page Numbers to User

The DEFGC ESR returns the physical page numbers associated with the specified Global Common Name in the caller's PARM. Starting with the physical page number corresponding to the specified Logical Page Number, page numbers are copied from GLOBTAB until either all the defined Global Common Page Numbers or the end of the caller's PARM area is reached.

ESR format



PARM format



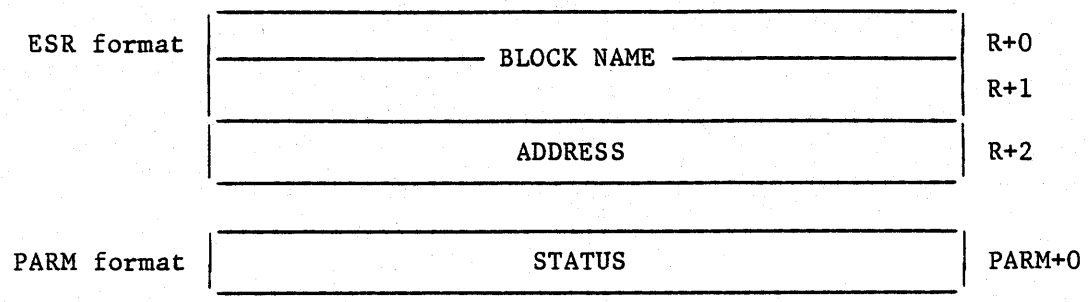
Parameter	Definition
BLOCK NAME	Global common block name, left justified and blank filled to eight characters.
SLPN	Starting logical page number within global common block.
NPNR	Number of page numbers being returned in PARM.
NPNL	Number of page numbers left; unable to return all page numbers.
FIRST PAGE	First page number assigned to global common block.

An example of the calling sequence is as follows:

```
        EXT         DEFGC               Externally defined symbol
        LDD,H0      ='BLOCKONE'         Block name
        LDI,H2      0                   Starting logical page number
        MON,H0      DEFGC               Monitor request
```

GETGC, Attach Global Common

The GETGC ESR enables a task to obtain access to a global common block. The named block is mapped into the task address space beginning at the designated address.



Parameter	Definition
BLOCK NAME	Global common block name, left adjusted and blank filled to eight characters.
ADDRESS	Starting block logical address; must be page boundary.
STATUS	Numeric value defining the call status:

Entry	Meaning
= 0	Request successful.
= 3	Illegal block name.
= 5	Unsuccessful request, not all pages available; no pages assigned.
= 6	Starting address not on a page boundary.

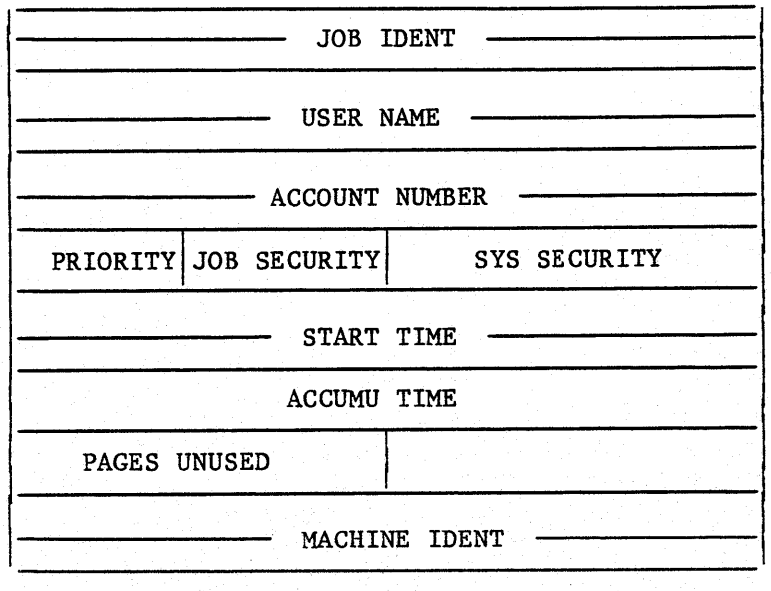
An example of a calling sequence is as follows:

```
EXT       GETGC          Externally defined symbol
LDD,H0    ='BLOCKONE'    Block name
LDA,H2    $1000          Address
MON,H0    GETGC          Monitor request
```

JOBINFO, Return Job Information

The JOBINFO ESR is used to obtain information about the caller's JOB.

PARM format



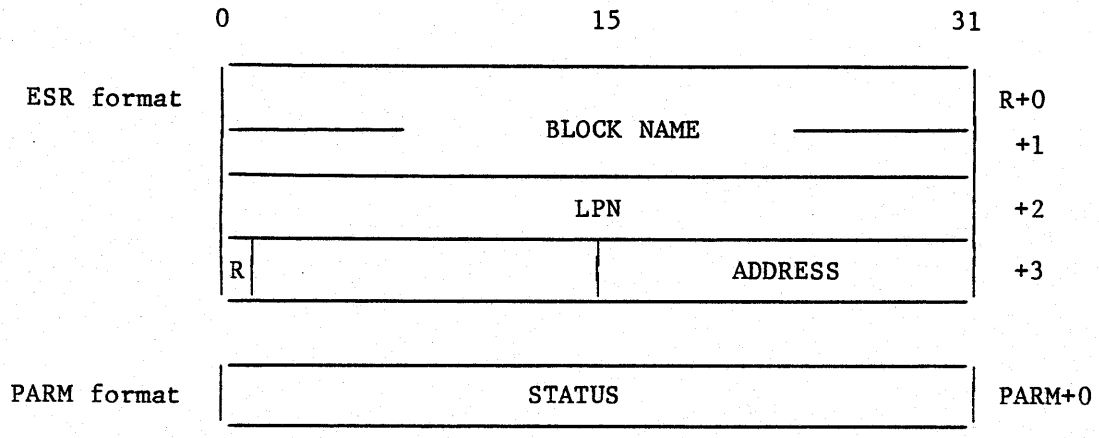
Parameter	Definition
JOB IDENT	Job identifier.
USERNAME	Job owner's username.
ACCOUNT	Account number of job.
PRIORITY	Default priority for tasks.
JOB SECURITY	Maximum job security level or security mask.
SYS SECURITY	Maximum system security level or security mask.
START TIME	Time of day that job was started HH:MM:SS.
ACCUMU TIME	Accumulated execution time of all tasks (in milliseconds).
PAGES UNUSED	Number of scheduled memory pages currently unused.
MACHINE IDENT	Identifies machine that is being used.

An example of the calling sequence is as follows:

```
EXT         JOBINFO     Externally defined symbol
MON,R0      JOBINFO     Monitor request
LDD,H2      PARM+11     Get machine ID
```

MGETGC, Modified Get Global Common

The MGETGC ESR enables a task to obtain access to a page within a global common block. The page specified within the named block is mapped into the task address space at the designated address.



Parameter	Definition
BLOCK NAME	Global common block name, left justified and blank filled to eight characters.
LPN	Logical page number of page within global common block to be mapped into the task.
R	Read only flag:

Entry	Meaning
= 0	Map page as read/write.
= 1	Map page as read only.

ADDRESS	Starting address at which page of global common is to be mapped; must be page boundry.
STATUS	Numeric value indicating request status:

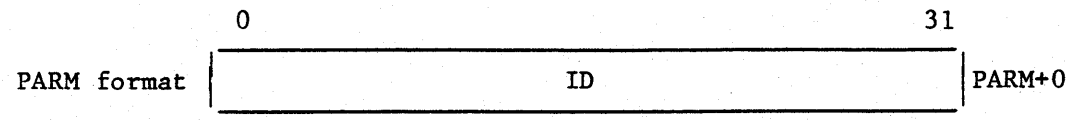
Entry	Meaning
= 0	Request successful.
= 1	Illegal block name.
= 2	Illegal logical page number; attempt to map unassigned page.

An example of the calling sequence is as follows:

```
EXT         MGETGC          Externally defined symbol
LDD,H0      ='BLOCKONE'     Block name
LDI,H2      1               Logical page number
LDA,H3      $1000           Address
MON,H0      MGETGC          Monitor request
```

NXTNUM, Get Unique Identifier

The NXTNUM ESR returns the next available identifier to the user in a series of ASCII alphanumeric characters spanning 00 to ZZ.



Parameter	Definition
ID	Alphanumeric identifier returned by NXTNUM (00-ZZ).

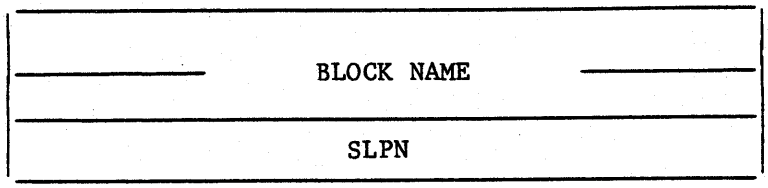
An example of the calling sequence is as follows:

```
EXT         NXTNUM      Externally defined symbol
MON,R0      NXTNUM      Monitor request
LD,H2       PARM        Get identifier
```

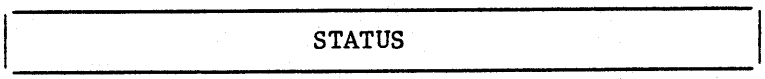
RELGC, Release Assigned Global Common

The RELGC ESR enables a task to release pages assigned to a global common block. All assigned pages greater than or equal to the specified starting logical page number to be released are returned to the system and the number of pages assigned to the global common block is updated.

ESR format



PARM format



Parameter	Definition
BLOCK NAME	Global common block name, left justified and blank filled to eight characters.
SLPN	Starting logical page number of pages associated with global common block to be released.
STATUS	Numeric value indicating request status:

Entry	Meaning
= 0	Request successful.
= 1	Illegal block name.
= 2	Illegal logical page number; attempt to release unassigned page.

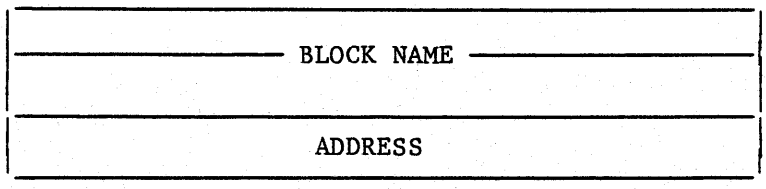
An example of the calling sequence is as follows:

```
EXT       RELGC       Externally defined symbol
LDD,R7    ='BLOCKTWO' Block number name
LDI,R9    2           Starting logical
MON,R7    RELGC       Monitor request
```

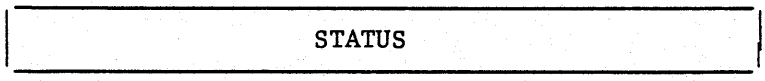
RETGC, Return Global Common

The RETGC ESR enables a task to release pages assigned to a global common block. The pages are set to protected status and become available for subsequent assignment.

ESR format



PARM format



Parameter	Definition
BLOCK NAME	Global common block name, left adjusted and blank filled to eight characters.
ADDRESS	Starting block logical address; must be page boundary.
STATUS	Numeric value defining status of request:

Entry	Meaning
= 0	Request successful.
= 3	Illegal block name.
= 7	Unsuccessful request, not all pages assigned to this block; none released.
= 8	Starting address not on page boundary.

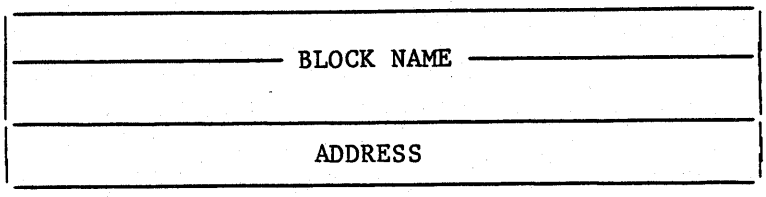
An example of a calling sequence is as follows:

```
EXT         RETGC           Externally defined symbol
LDD,R7      ='BLOCKTWO'     Block name
LDA,R9      $2000           Address
MON,R7      RETGC           Monitor request
```

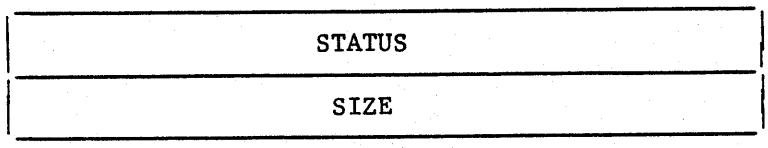
STATGC, Status Global Common

The STATGC ESR enables a task to manage its global common dynamically. Availability of memory can be determined before blocks are attached or detached.

ESR format



PARM format



Parameter	Definition
BLOCK NAME	Global common block name, left adjusted and blank filled to eight characters.
ADDRESS	Starting block logical address; must be page boundary.
STATUS	Numeric value defining status of addressed area:

Entry	Meaning
= 0	All page space needed for the designated block is currently available in user's page map.
= 2	All page space needed for the designated block has previously been assigned.
= 3	Illegal block name.
= 6	All pages needed not assigned.
= 8	Starting address not on page boundary.

An example of a calling sequence is as follows:

```
EXT       STATGC     Externally defined symbol
LDD,H1    BNAME      Block name
LDA,H3    BSTART     Address
MON,H1    STATGC     Monitor request
```

Logical I/O, referred to as blocker/deblocker, consists of library routines that the user calls for transferring logical records to and from user-defined buffer areas. As buffers fill or empty, blocker/deblocker transfers the buffers to or from a physical I/O device. This reduces the actual number of data transfers and allows efficient use of the MPX I/O system.

A double buffering option (that is, the ability to fill or empty one buffer while a second buffer is being transferred to or from a physical I/O device) is provided to allow overlapped operation.

Blocker/deblocker can be used for both mass storage devices (disk) and unit record devices (magnetic tapes, card equipment, etc.). Mass storage and magnetic tape are block devices (accessed in block format) while all other devices are record devices (accessed in record format).

A block number parameter (BN) is required for certain blocker/deblocker functions on mass storage files. Thus a user can, if he wishes, access a mass storage file randomly with blocker/deblocker.

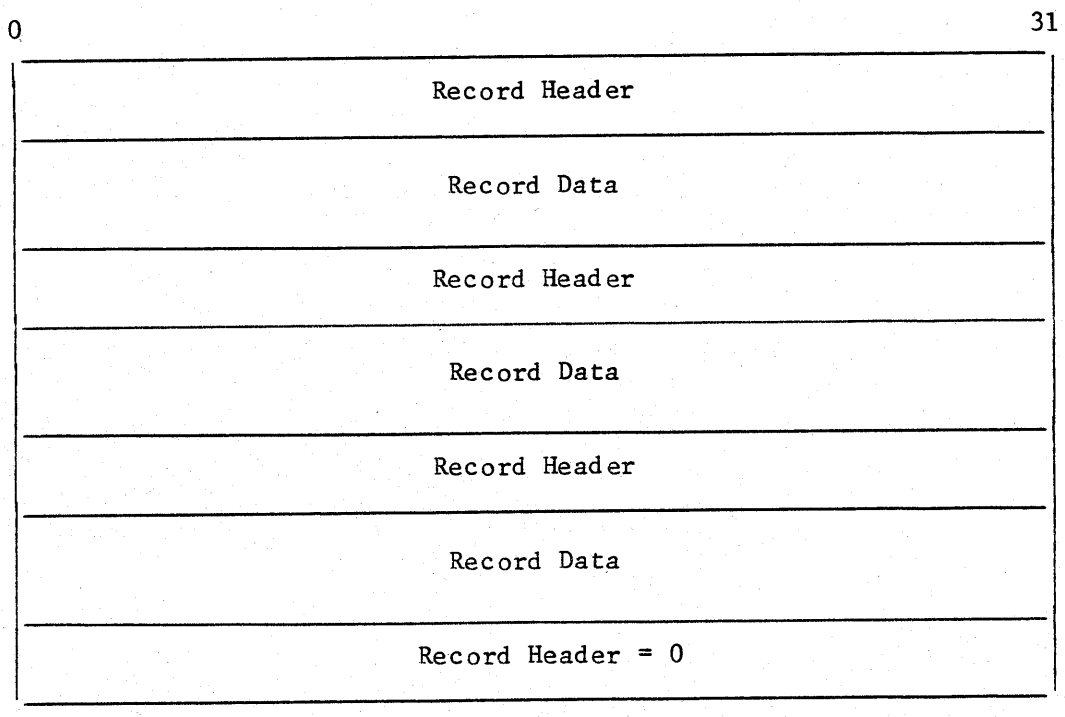
For a working understanding of blocker/deblocker, the user should be familiar with block and buffer formats (refer to appendix C).

AUTOMATIC SCRATCH FILE ALLOCATION

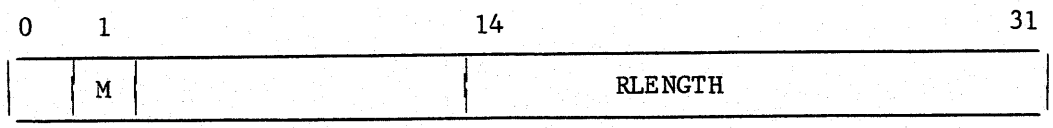
Standard or scratch files are allocated by MPX/OS on an as needed basis. Blocker/deblocker allocates the file the first time it is referenced. Job manager releases the files at job termination. A scratch file can be saved by using the CLOSE and MODIFY ESR or SAVEPF control card (see section 3). The CLOSE ESR returns the file definition parameters needed to release or modify the files. The MODIFY ESR can be used to change the parameters, thereby permanently saving the file.

BLOCK DEVICES

The data format for block devices (disk or magnetic tape) is characterized by a series of alternate record headers and record data areas, ending with a zero record header. One or more records constitute a block. The size of a block, for a file, is determined by the ALLOCATE function of the file manager. The size of a block on magnetic tape is established when a PACKD function is performed. The MPX standard block size is 480 words. A block is transferred to the peripheral device from a blocking buffer, which is specified by PACKD or PICKD. The following is a block format.



The record header is a single word containing information about the record that follows it. A record header appears as follows:

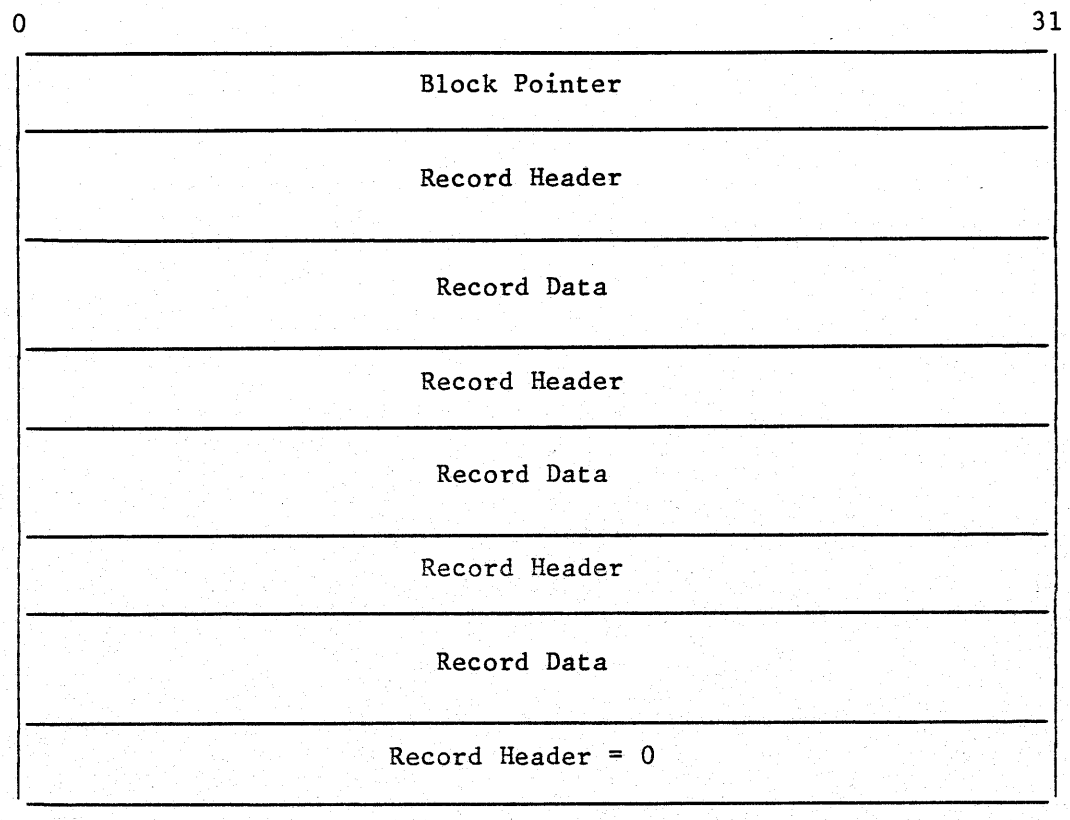


M - Mode of record data (bit 1)

M = 0, ASCII record
M = 1, binary record

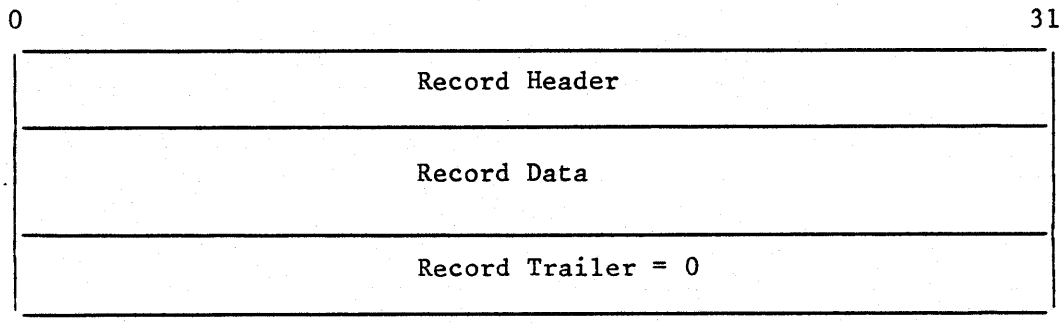
RLENGTH - Number of bytes of data in the record

The block is contained in a user-defined buffer area. For single buffering, the buffer area is one word larger than the block size. The additional word contains a pointer to the next record header and is maintained by the blocker/deblocker. Thus, the user buffer area appears as follows:

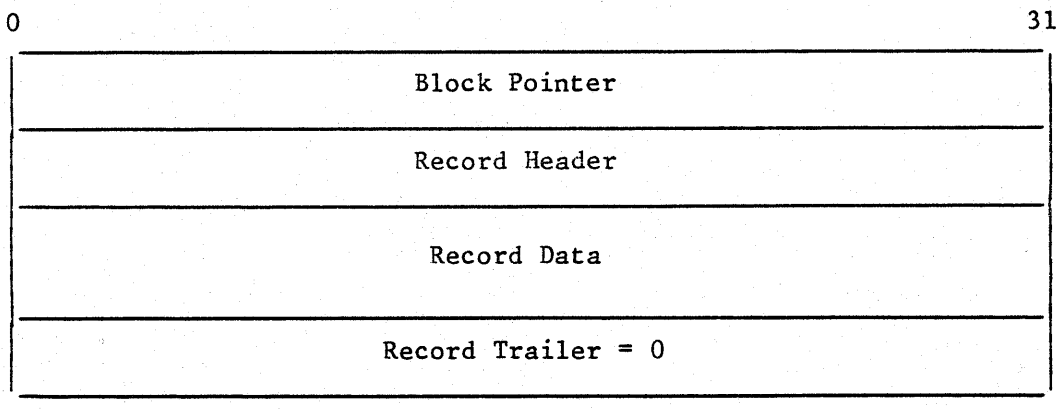


When double buffering is specified, the buffer area must be twice the block size plus two (double the required value for single buffering). The minimum size of a block is four words (pointer, header, one-word record, and zero header).

The data format for record devices is characterized by a record header, followed by a record data area, and ending with a zero record trailer. The maximum size of a record is determined by the size of the user's buffer area but must always be less than 4096 words. The block format appears as follows:



The block is contained in a user-defined buffer area. The buffer area is one word larger than the maximum record size. The additional word contains a pointer to the record header and is maintained by the blocker/deblocker. Thus, the user buffer area appears as follows:



Only the actual record data is transferred to/from the peripheral device (refer to appendix C).

BLOCKER

The blocker is a set of functions that perform blocking on user files/devices. All files to be blocked may have been previously allocated and opened. If a file has not been opened, blocker will automatically allocate and open a mass storage file on the system device(s) with block size set to system standard block size. Unit record devices must be assigned by an EQUIP statement.

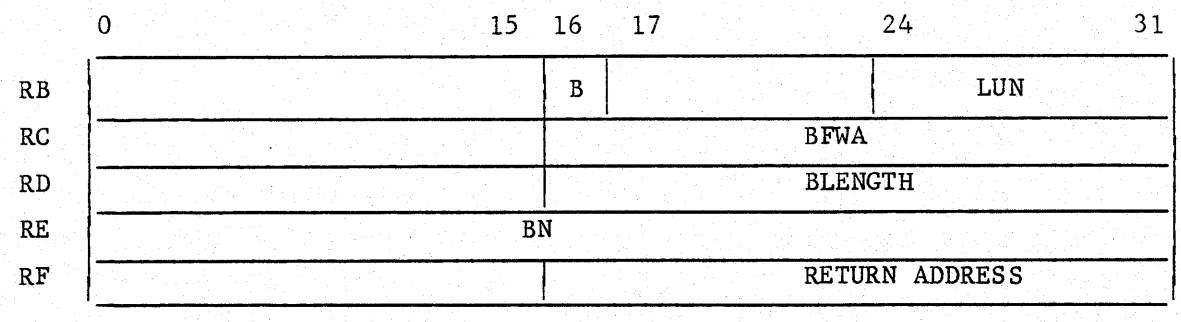
The blocker includes the following functional routines:

- Pack define - PACKD
- Pack - PACK
- Pack output - PACKO
- Pack close - PACKC

PACKD, PACK DEFINE

The PACKD function establishes the blocking area (buffer) to be associated with a file or unit record device. The blocker/deblocker logical unit definition table has space for 63 I/O entries.

Before the user calls PACKD, registers RB through RF should be set as follows:



LUN - Logical unit number of device

B - Type of buffering

B = 0, double buffering
B = 1, single buffering

BFWA - First word address of user's buffer area

BLENGTH - Length of user's buffer area. It must be consistent with block size and buffering requirement

BN - Block number of first write. It pertains to mass storage files only

```
BN .LT. 0, file is positioned to highest block written +1
BN  =   0, file is not positioned
BN .GT. 0, file is positioned to specified block
```

RETURN
ADDRESS - Address in user's program to which PACKD must return.

A calling sequence to PACKD from a user's program is as follows:

```
LDI,RB      B/LUN
LDA,RC      BFWA
LDI,RD      BLENGTH
LDI,RE      BN
JSX         PACKD,RF          Call
LD,X7       PARM              Check for errors
TST,NE      X7,X0,ERRPROC
```

Only one PACKD (output) or PICKD (input) definition can be active for a logical unit at one time. The typical sequence of events for accessing logical unit 10 as a read/write file is as follows:

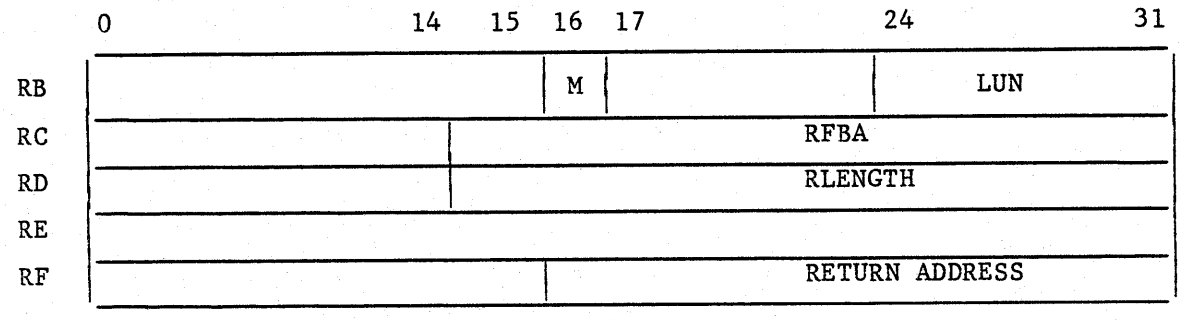
```
LDI,R0      10
MON,R0      REWD              Initial positioning
    .
    .
    .
LDI,RB      10
LDA,RC      BFWA
LDI,RD      BLENGTH
LDI,RE      0
JSX         PACKD,RF          Define output buffer
```

```
        .
        .
        .
LDI,RB      10
LDCA,RD     RFBA
LDI,RD      RLENGTH
JSX         PACK,RF         Output data records
        .
        .
        .
LDI,RB      10
JSX         PACKC,RF        End of output phase, close
                            definition
LDI,RB      10
MON,RB      REWD            Reposition file
        .
        .
        .
LDI,RB      10
LDA,RC      BFWA
LDI,RD      BLENGTH
LDI,RE      0
JSX         PICKD,RF        Define input buffer
        .
        .
        .
LDI,RB      10
LDCA,RC     RFBA
LDI,RD      RLENGTH
JSX         PICK,RF         Input data records
        .
        .
        .
LDI,RB      10
JSX         PICKC,RF        End of operational sequence
```

PACK

The PACK function transfers a record to the buffer area defined by PACKD for the referenced logical unit. In moving the record, the blocker truncates trailing zeros (binary record) or trailing blanks (ASCII record) from the record data to the nearest whole word. When the buffer area is full, the buffer is written on the file/device specified by the logical unit.

Before the user calls PACK, the registers RB through RF should be set as follows:



M - Mode of record

M = 0, ASCII record
M = 1, binary record

LUN - Logical unit number

RFBA - First byte address of the record to be transferred

RLENGTH - Length of the record in bytes

RETURN ADDRESS - Address in user's program to which PACK will return

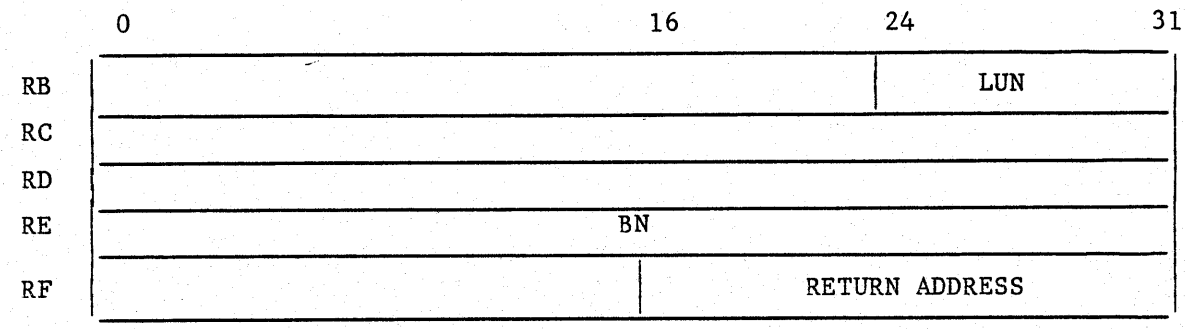
A calling sequence to PACK from a user's program is as follows:

```
LDI,RB      M/LUN
LDCA,RC     RFBA
LDI,RD      RLENGTH
JSX         PACK,RF          Call
LD,X5       PARM             Check for errors
TST,NE      X5,X0,ERRPROC
```

PACKO, Pack Output

The PACKO function is used to output a partially filled buffer. For single-buffered record devices, PACKO has no function. For double-buffered record devices, PACKO outputs the last record.

Before the user calls PACKO, registers RB through RF should be set as follows:



LUN - Logical unit number

BN - Block numbers to which block is output (pertains only to mass storage)
BN .LT. 0, output to highest block written +1
BN = 0, output to next sequential block
BN .GT. 0, output to specified block

RETURN ADDRESS - Address in user's program to which PACKO will return

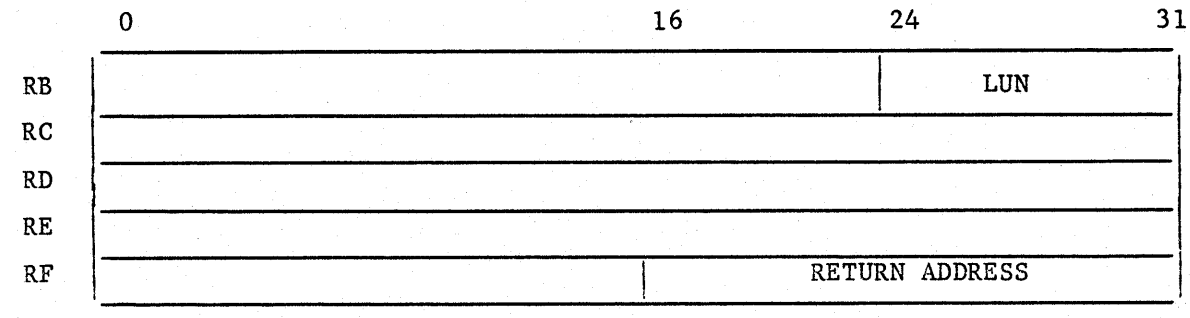
A calling sequence to PACKO from a user's program is as follows:

```
LDI,RB      LUN
LDI,RE      BN
JSX         PACKO,RF          Call
LD,X3       PARM              Check for errors
TST,NE      X3,X0,ERRPROC
```

PACKC, Pack Close

The PACKC function is used to remove a logical unit definition from the blocker/deblocker table. The PACKC function checks to see if any records remain in the buffer and if so, writes them to the file/device before removing the logical unit definition from the blocker/deblocker table. It should be noted that this function only removes the logical unit definition from the blocker/deblocker table and does not close the unit.

Before the user calls PACKC, registers RB through RF should be set as follows:



LUN - Logical unit number

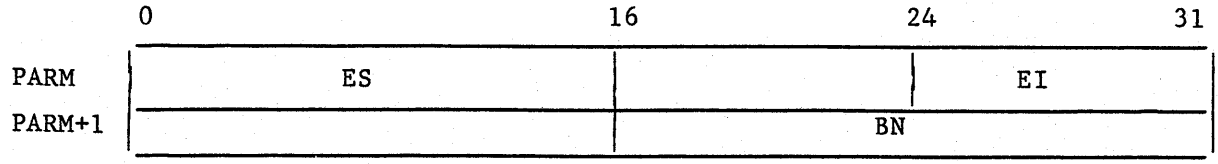
RETURN ADDRESS - Address in user's program to which PACKC will return

A calling sequence to PACKC from a user's program is as follows:

```
        LDI,RB      LUN
        JSX         PACKC,RF        Call
        LD,X6       PARM            Check for errors
        TST,NE      X6,X0,ERRPROC
```

STATUS RETURN

Upon completion of a call, the blocker returns status to the parameter area, which is defined external to the user's program. The parameter area is set as follows:



EI - Error indicator

EI = 0, no error
EI ≠ 0, refer to appendix D for the blocker error indicators and their descriptions

BN - Block number (if EI ≠ 0, BN has no meaning).

Routine	Block Type Device	Record Type Device
PACKD	Block number of next block to be written	Record number of next record to be written
PACK	Block number of block which contains the record	Record number of record
PACKO	Block number of next block to be written	Record number of next record to be written
PACKC	Block number of next block to be written	Record number of next record to be written

ES (bits 0 through 15) - Equipment status, returned if EI = 1 or 12

DEBLOCKER

The deblocker is a set of functions that performs deblocking on user files/devices. All files to be deblocked must have been previously allocated and opened. Unit record devices must be equipped.

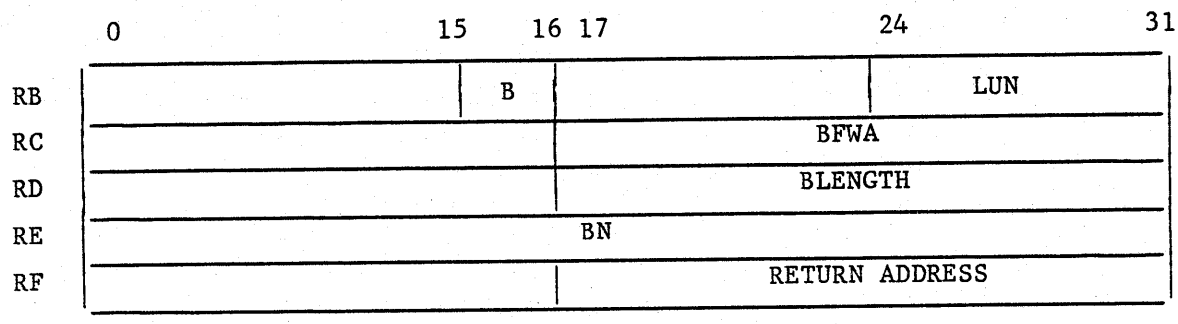
The deblocker includes the following functional routines:

- Pick define - PICKD
- Pick - PICK
- Pick input - PICKI
- Pick close - PICKC

PICKD, Pick Define

The PICKD function establishes the deblocking area (buffer) to be associated with a file or unit record device. The blocker/deblocker logical unit definition table has space for 63 I/O entries.

Before the user calls PICKD, registers RB through RF should be set as follows:



LUN - Logical unit number of device

B - Type of buffering

B = 0, double buffering
B = 1, single buffering

BFWA - First word address of user's buffer area

BLENGTH - Length of user's buffer area. It must be consistent with block size and buffering requirement

BN - Block number of first read, pertains only to a mass storage file
BN .LE. 0, file is not positioned
BN .GT. 0, file is positioned to specified block

RETURN ADDRESS - Address in user's program to which PICKD must return

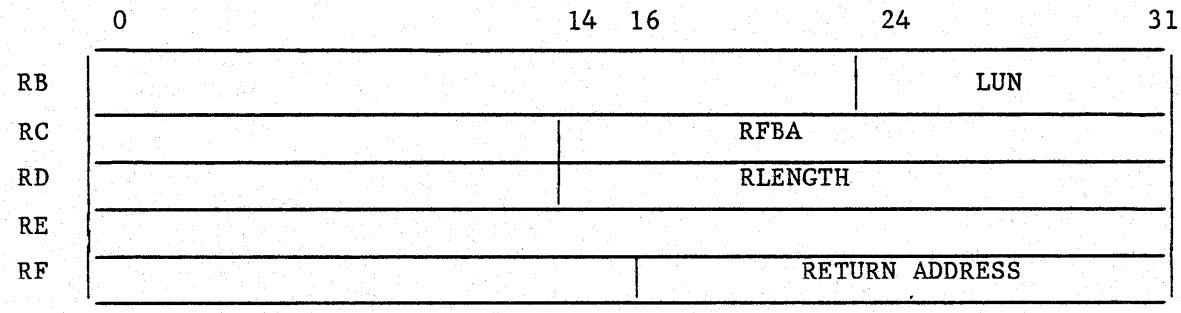
A calling sequence to PICKD from a user's program is as follows:

```
LDI,RB      B/LUN
LDA,RC      BFWA
LDI,RD      BLENGTH
LDI,RE      BN
JSX         PICKD,RF          Call
LD,R0       PARM              Check for errors
TST,NE      R0,X0,ERRPROC
```

PICK

The PICK function transfers a record from the buffer area defined by PICKD to the user's record area established by the PICK call. If the record to be moved is larger than the user's record area, PICK truncates the record. If the record to be moved is smaller than the user's record area, PICK fills the remaining record area with zeros (binary record) or blanks (ASCII record).

Before the user calls PICK, registers RB through RF should be set as follows:



LUN - Logical unit number

RFBA - First byte address of the area the record is to be transferred to

RLENGTH - Length of the record in bytes

RETURN ADDRESS - Address in user's program to which PICK will return

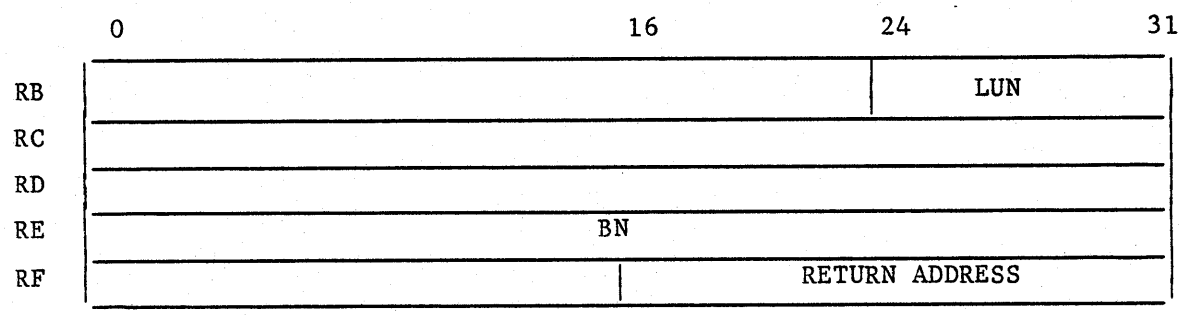
A calling sequence to PICK from a user's program is as follows:

```
        LDI,RB      LUN
        LDCA,RC     RFBA
        LDI,RD      RLENGTH
        JSX         PICK,RF          Call
        LD,R0       PARM             Check for errors
        TST,NE      R0,X0,ERRPROC
```

PICKI, Pick Input

The PICKI function is used to input a new block of data before the last block has been exhausted. For record type devices, PICKI results in skipping one record. For block type devices, PICKI results in skipping one or more records.

Before the user calls PICKI, registers RB through RF should be set as follows:



LUN - Logical unit number

BN - Block number of block to be input (pertains only to mass storage)
BN .LE. 0, input next sequential block
BN .GT. 0, input specified block

RETURN
ADDRESS - Address in user's program to which this PICKI will return

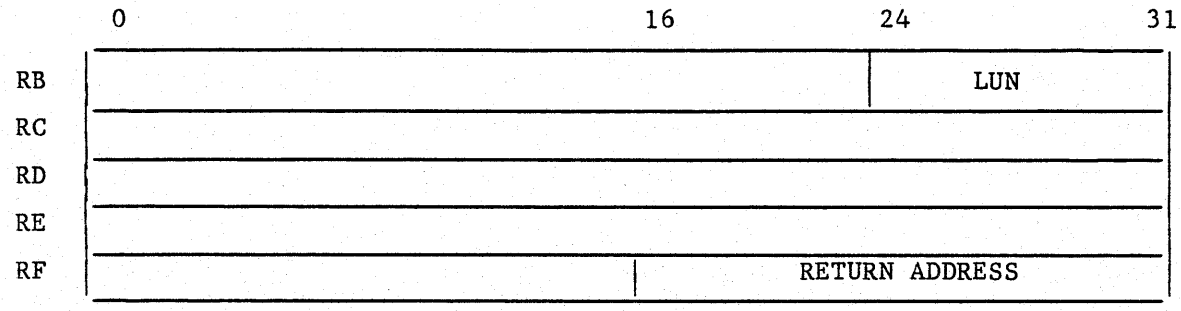
A calling sequence to PICKI from a user's program is as follows:

```
        LDI,RB      LUN
        LDI,RE      BN
        JSX         PICKI,RF        Call
        LD,X7       PARM            Check for errors
        TST,NE      X7,X0,ERRPROC
```

PICKC, Pick Close

The PICKC function is used to remove a logical unit definition from the blocker/deblocker table. It should be noted that this function only removes the logical unit definition from the blocker/deblocker table and does not close the unit.

Before the user calls PICKC, registers RB through RF should be set as follows:



LUN - Logical unit number

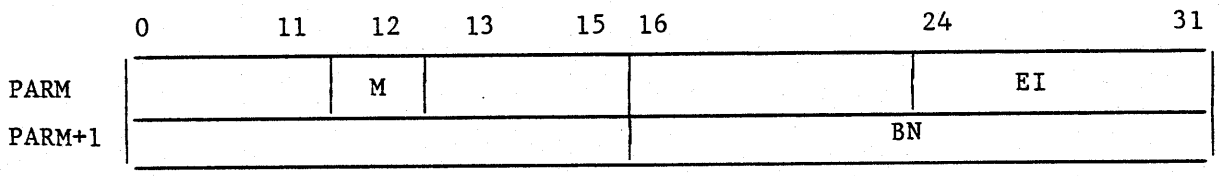
RETURN ADDRESS - Address in user's program to which the PICKC will return

A calling sequence to PICKC from a user's program is as follows:

```
LDI,RB      LUN              P1
JSX         PICKC,RF         Call
LD,X4       PARM             Check for errors
TST,NE      X4,X0,ERRPROC
```

STATUS RETURN

Upon completion of a call, the deblocker returns status to the parameter area, which is defined external to the user's program. The parameter area is set as follows:



M - Mode bit
M = 0, record passed by PICK is ASCII
M = 1, record passed by PICK is binary

EI - Error indicator

EI = 0, no error
EI ≠ 0, refer to appendix D for the deblocker error indicators and their description

NOTE: If EI = 1 or 12, bits 0 through 15 of PARM contain the equipment status.

BN - Block number (if EI ≠ 0, BN has no meaning).

Routine	Block Type Device	Record Type Device
PICKD	Block number of next block to be read	Record number of next record to be read
PICK	Block number of block containing the record	Record number of record
PICKI	Block number of next block to be read	Record number of next record to be read
PICKC	Block number of next block to be read	Record number of next record to be read

The MPX relocatable loader performs the following services for the user.

- Loads relocatable binary information into memory from the sources named in the call to the loader (*name, *LOAD, or binary decks).
- Loads absolute tasks created by the *ABS control statement.
- Links independently compiled or assembled subprograms that reference each other through symbolically named entry points.
- Loads and links any externally referenced library routines into a task.
- Detects and records format errors and/or violations of loading procedures detected during the loading process.
- Prepares a memory map of all subprograms, entry points, and common data areas (except for library tasks).

Programs are loaded from specified files and the system library file in blocked card image form.

Each subprogram loaded must contain a binary identification card (IDC). The information from the IDC is used to allocate subprogram storage in upper memory. Subprogram allocation begins in logical page 14 and continues downward as needed. If the program name on the IDC has been previously encountered during the load process, the current program is not loaded.

The information from the block common table (BCT) card is used by the loader to allocate data and scratch common blocks. Data common is loaded in the same way as subprograms, while scratch common is allocated upward beginning with logical page 0.

As subprograms are loaded, a table of subprogram names, block common names, entry point names, and external symbol names is created. This table is

referred to as the loader symbol table (LST). When the transfer (TRA) card of a subprogram is reached, an attempt is made to link the externals declared by the subprogram with previously loaded entry points. Upon detection of an end-of-load condition, the LST is checked for any external symbols for which no corresponding entry point symbol was declared. The system library is then searched in an attempt to satisfy these external declarations. This is done by comparing unlinked externals against entry points of the loader directory cards contained on the system library. Each library program has a directory card associated with it containing all of its unprotected (accessible) entry points. If a match is found, the library program is loaded as a subprogram. After all external symbols have been linked, or (having not located all externals) after two searches through the library, library loading is ceased. Any external symbols still not linked to an entry point are listed as undefined in the memory map.

The loader requests physical memory as needed on a page basis during the loading process. If the number of physical pages needed to complete the load exceeds the number of pages scheduled by the job, the job is aborted. Regardless of the number of pages scheduled, only the physical page needed to satisfy the loading process are assigned to the resources of the job. To gain access to other pages requested on the associated job *SCHED control statement, the user must utilize the OPENMEM call.

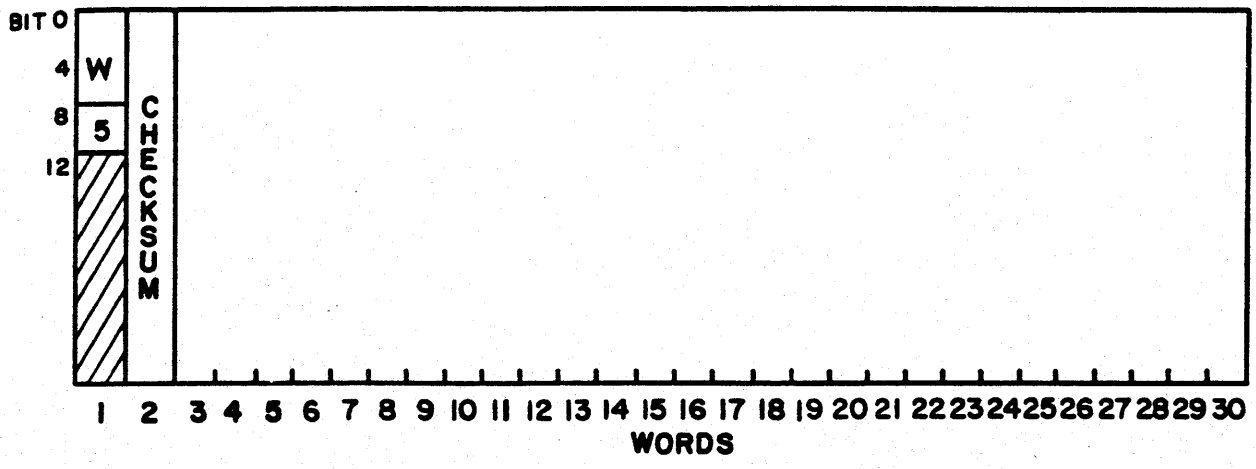
LOADER CARDS

The loader accepts the binary cards produced by assmeblers and compilers in the following order.

1) IDC Program identification
2) BCT Data and common block declarations
3) EPT Entry point names
4) RIF Relocatable information
5) EXT External names
6) TRA Transfer address

BINARY CARD STRUCTURE

The binary record occupies 30 computer words of 32 bits each. The general format of a binary record is:



Word 1

Bits 0 through 7 = Two hexadecimal digits identifying card type and, for an RIF card (W = 1 through 16_{16}), the number of words of information on the card.

Bits 8 through 11 = Hexadecimal 5, indicating binary card.

Bits 12 through 31 = Defined for types as required. See individual cards in this section.

Word 2

32-bit 2's complement sum of all other words contained on the card.

Words 3-30

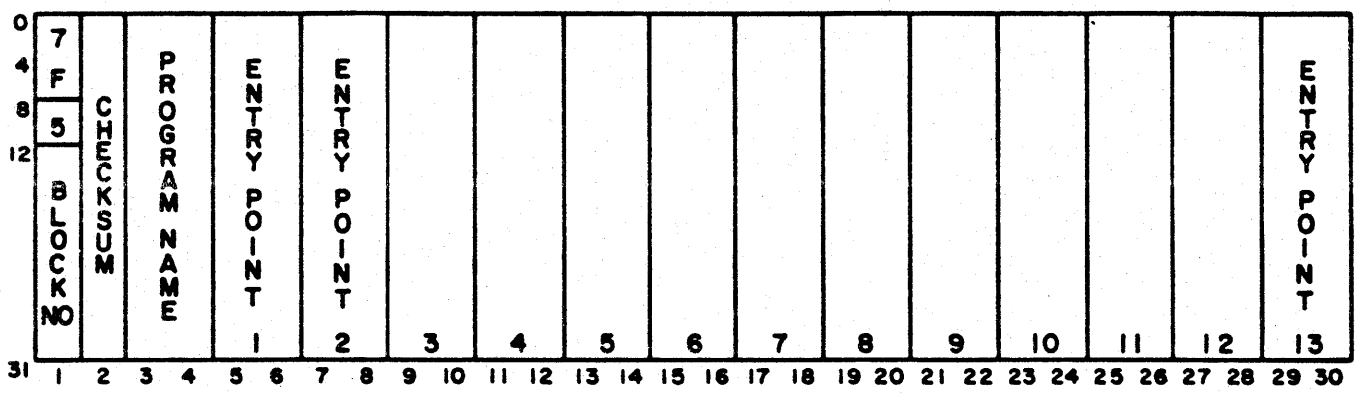
Defined for individual types. See remainder of this section.

LOADER DIRECTORY CARD

When the user calls subprograms from LIB, the loader refers to a directory card that aids the loader in searching for entry points. Only those entry points in a program that are unprotected can be referenced by a user program or another LIB file program. The loader processes the directory card and determines if the associated program is to be loaded. Every entry point name on the directory card is unprotected and can be referenced by the user program. All other entry points for that program are protected entry points.

The directory card also aids the loader in finding the next program on the library.

The directory consists of a binary card placed on the library ahead of the IDC card for the associated program. The format for the directory card is as follows:



Word 1

Bits 0 through 7 = 7F.
Bit 10, if set. Library routine is absolute.
Bits 12 through 31 = Block number of next LIB entry. Zero indicates end of LIB.

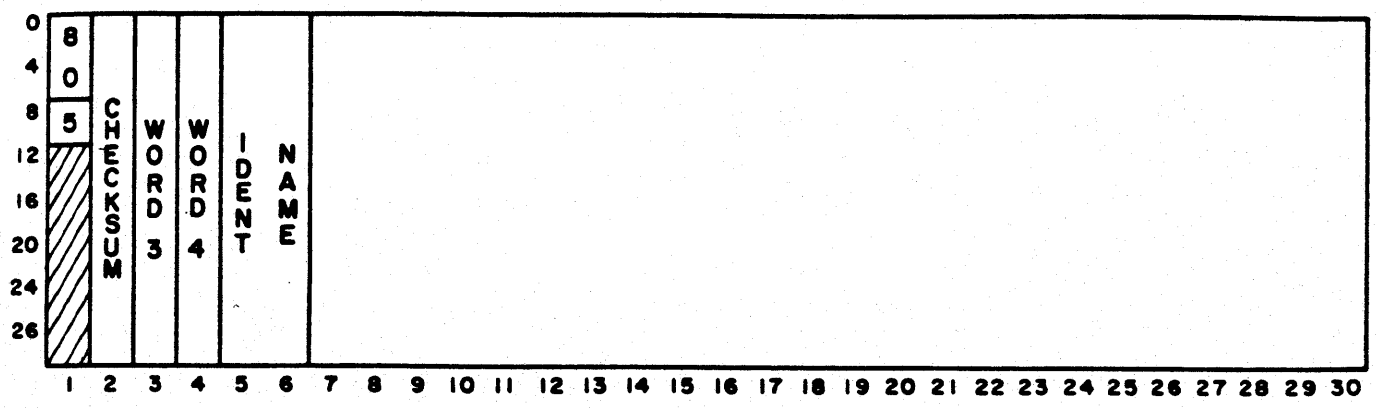
Word 3 and 4

8-character program name

Words 5 and 6 (7 and 8 etc. for remainder of card)

8-character entry point names (maximum of 13)

IDENTIFICATION CARD



Word 1

Bits 0 through 7 = 80

Word 3

Bit 0 = 0, absolute program
= 1, relocatable program

Bits 6 and 7 = Addressing type

0 = Word
1 = Half word
2 = Byte
3 = Bit

Bits 11 through 31 = Start address

Word 4

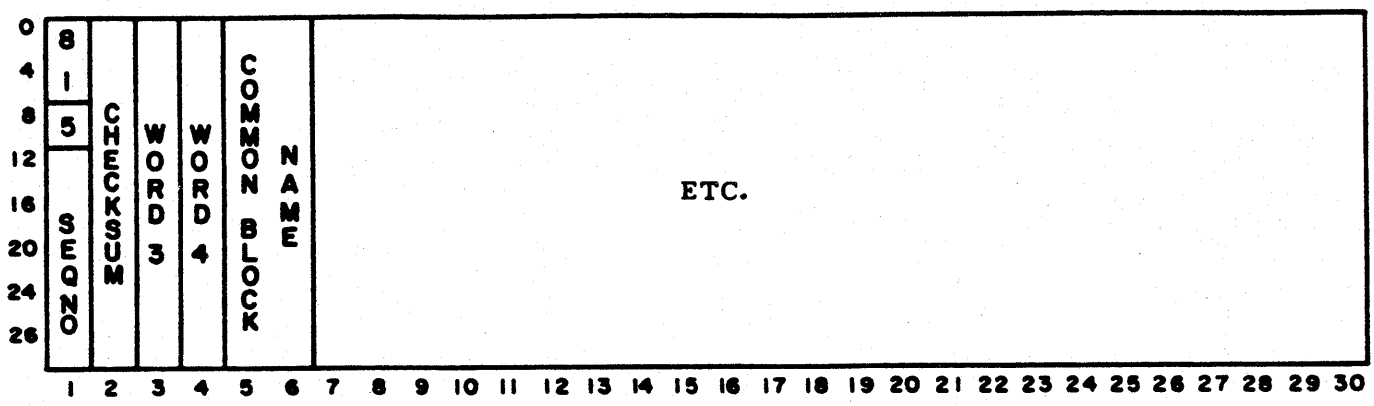
Bits 6 and 7 = Addressing type, see word 3

Bits 11 through 31 = End address

Words 5 and 6

8-character name in ASCII, left adjusted

BLOCK COMMON TABLE CARD



Word 1

Bits 0 through 7 = 81

Bits 16 through 31 = Sequence number (1 to 5)

Word 3

Bit 0 = 0, DCOM area
= 1, SCOM area

Bits 6 and 7 = Addressing type

0 = Word
1 = Half word
2 = Byte
3 = Bit

Bits 11 through 31 = Starting address of the common block

Word 4

Bits 6 and 7 = Addressing type

Bits 11 through 31 = Ending address of the common block

Words 5 and 6 (7 and 8 etc. for remainder of card)

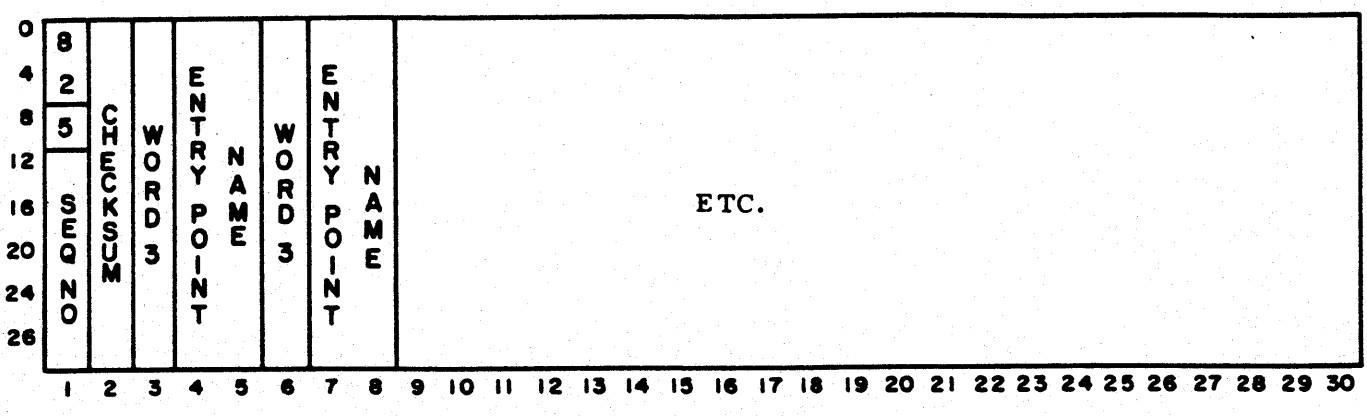
8-character common block name in ASCII, left justified

NOTES: 1 A maximum of five BCT cards is permitted per module.

2 Words 3 through 6 are repeated for each common block defined by the BCT card.

3 The information content of the card image is terminated by a zero field.

ENTRY-POINT CARD



Word 1

Bits 0 through 7 = 82

Bits 12 through 31 = Sequence number (1 to 5)

Word 3

Bit 0 = 1, negative relocatable

Bit 1 = 0, absolute address
= 1, relocatable address

Bits 6 and 7 = Addressing type

0 = Word
1 = Half word
2 = Byte
3 = Bit

Bits 11 through 31 = Address of entry point

Words 4 and 5

8-character entry-point name in ASCII, left justified

Word N

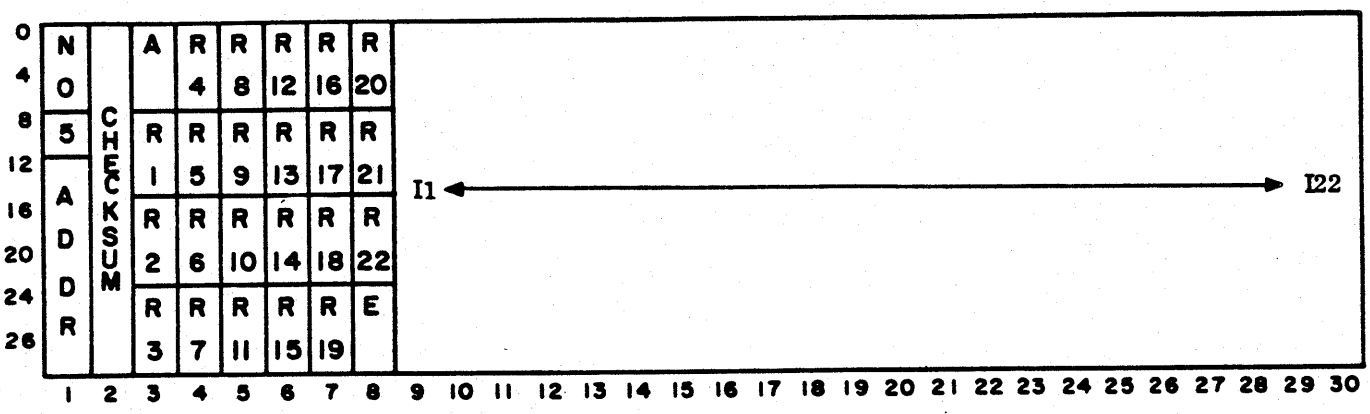
Same information as word 3

Words N+1 and N+2

Same information as words 4 and 5

NOTE: The information content of the card image is terminated by a zero field.

RELOCATABLE INFORMATION CARD



Word 1

Bits 0 through 7 = Number of words on card, 1 through 22_{10}.

Bit 8 and bits 12 through 31 = Address of first loadable information.

Word 3

Bits 0 through 7 = Address information: bit 0 = 0, bits 1 and 2 are unused and bits 3 through 7 = relocation of the starting word address. Bit 0 = 1, first word on card is not a full word and address field gives the starting bit address.

Words 3 through 8

Relocation bytes for each address field on the card (R1 through R22):

Bit 0 = 1, if negative relocation, else 0

Bits 1 and 2 = Addressing type

0 = Word
1 = Half word
2 = Byte
3 = Bit

Bits 3 through 7 = Relocation of each address field

0 = Nonrelocatable (absolute)
1 = Program relocatable
2 through 31 = Common block relocatable (defined by BCT)

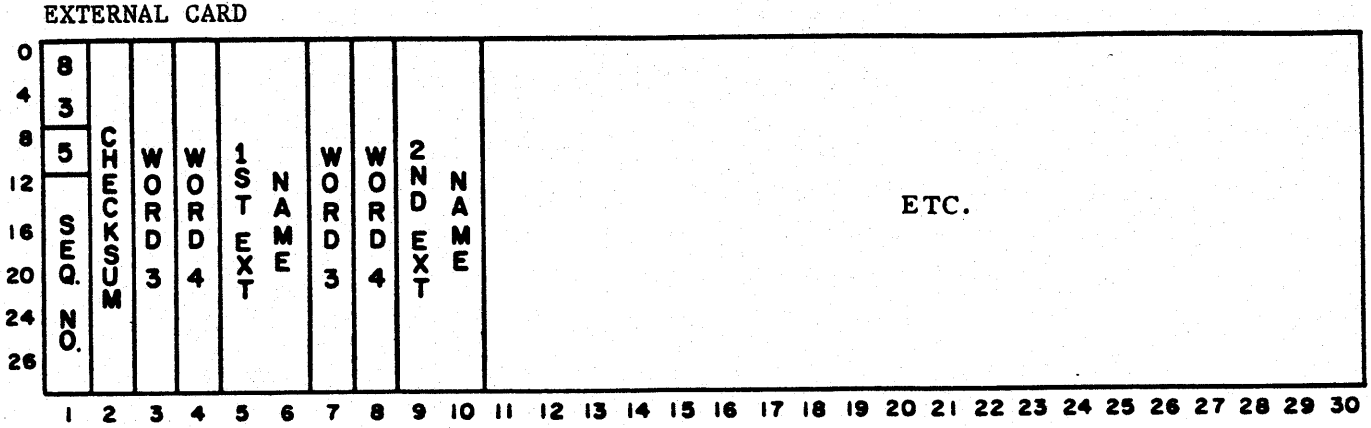
Word 8

Bits 24 through 31 = End information: bit 0 = 1, last word on card not a full word.
Bits 3 through 7 = last used bit (0 through 30)

Words 9 through 30 (I1 through I22)

Loadable information corresponding to R1 through R22.

EXTERNAL CARD



Word 1

Bits 0 through 7 = 83

Bits 12 through 31 = Sequence number (1 to 5)

Word 3

Bit 0 = 1, negative relocatable

Bits 14 and 15 = Addressing type of string

0 = Word
1 = Half word
2 = Byte
3 = Bit

Bits 16 through 31 = Word address of end of string

Word 4

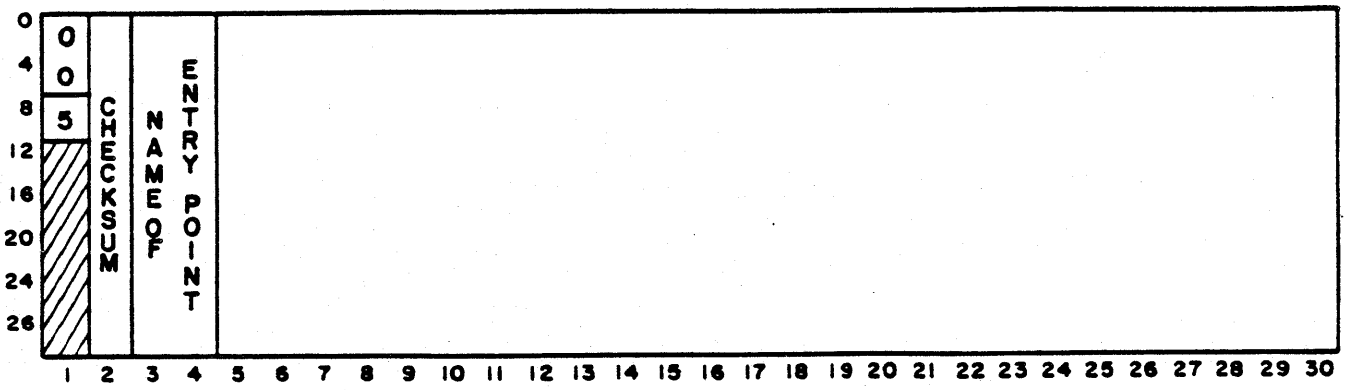
Bit 0 = 1, negative additive

Bits 11 through 31 = Additive

Words 5 and 6 (7 and 8 etc. for remainder of card)

8-character external name in ASCII, left justified

TRANSFER ADDRESS CARD



Word 1

Bits 0 through 7 = 00

Word 2

Checksum is running checksum of IDC through TRA cards.

Words 3 and 4

Name of primary entry point in 8-character ASCII, left justified. Zero implies no transfer address

HEXADECIMAL CORRECTION CARDS

General format:

*HCC, location hexadecimal correction and relocation factor, ..

Hexadecimal corrections may be made to binary subprograms after loading. *HCC statements may be used to enter corrections or to add code through the establishment of a program extension area. The program extension area is created after the subprogram area. Corrections to subprograms referring to the extension area, or additional instructions that are to be stored in the extension area, may not be submitted until all subprograms have been loaded.

The location field symbolically defines the location to be amended through the use of subprogram names and displacement values. The extension area is given a unique identification. The values that are allowed in the location field of the card are as follows:

Location Contents	Interpretation
(Program name + K)	Corrections on this *HCC card are loaded beginning with location K in the named subprogram.

Hexadecimal Correction	Interpretation
(D/data name + K)	Corrections are loaded beginning with location K in the named data area.
(XK)	First occurrence - defines a program extension area of length K. Corrections on the first card of this type are ignored. Subsequent occurrences - corrections are loaded beginning with location K of the program extension area.
(+K)	Continuation *HCC cards. +K is an increment from the last location plus one corrected by the previous *HCC statement.

Hexadecimal corrections of up to eight characters and their relocation factors, if any, follow the location term. Fields are separated by commas. All values are hexadecimal. Each value is stored right justified in successive words. Values of less than eight digits are zero filled. Acceptable values for this field are:

Hexadecimal Correction	Interpretation
Hexadecimal correction	The correction replaces the contents of the memory location determined by the location defined on the card and the position of this hexadecimal correction field.
Contiguous commas	Commas do not alter the location.
Hexadecimal correction with relocation factor	Replaces the contents of memory determined by the location stated on the card and the position of this field on the card. The address portion of the hexadecimal correction is to be relocated as dictated by the relocation factor.

This relocation factor can take any of the following forms:

Relocation Factor*	Interpretation
No relocation factor	Correction is stored as absolute correction.

*A relocation factor followed by H, C, or B indicates that half-word, character (byte), or bit addressing modes are to be performed.

Relocation Factor* (Contd)	Interpretation (Contd)
(Subprogram name)	Relocate the word address portion of the correction relative to the address of the first location in the subprogram enclosed in parentheses.
(D/data block) (C/common block)	Relocate the address portion of the correction relative to the address of the first location of the named common or data block.
(X)	Relocate the address portion of the correction relative to the first location of the previously defined extension area.
($)	Relocate the address portion of the correction relative to the last relocation factor defined in any field of this or any preceding *HCC statement.

HCC EXAMPLES

The following are examples of various formats and uses of the *HCC statement.

1) *HCC,(PROG1+70)21600100($)

Enter hexadecimal correction 2160XXXX at address 0070 relative to subprogram PROG1. The $ relocation factor relocates address 0100 relative to subprogram PROG1.

2) *HCC,(SUB1+7F)21600100($),47310101(SUB2)

Enter correction 2160XXXX in location SUB1+7F. Relocate 0100 relative to subprogram SUB1. Enter correction 4731XXXX in location SUB1+80. Relocate address 0101 relative to subprogram SUB2.

3) *HCC,(SUB1+20)00000036,000036,00036,0036,036,36

Enter the hexadecimal value 00000036 into locations 20, 21, 22, 23, 24, and 25 of subprogram SUB1. All corrections are right justified and stored in memory as 00000036.

4) *HCC,(X2E)

Assign 2E locations to the program extension area.

*A relocation factor followed by H, C, or B indicates that half-word, character (byte), or bit addressing modes are to be performed.

5) *HCC,(X)20000100(SUB1),40000101($),20000102($),40000103($)

Enter 2000XXXX into the first location of the extension area. XXXX is the relocated address relative to subprogram SUB1. 4000XXXX goes to the second location of the extension area. 2000XXXX goes to the third and 4000XXXX to the fourth. All XXXX addresses are relocated relative to subprogram SUB1.

6) *HCC,(+)20000400(SUB2),40000401($),20000402(SUB3),40000403($)

Continue inserting corrections in the program extension area. The addresses of the first two corrections are relocated relative to subprogram SUB2. The last two are relative to subprogram SUB3.

7) *HCC,(X1F)20000420(SUB4),40000621($),20000622(SUB5),40000623($)

Load the corrections with relocation factors of subprograms SUB4 and SUB5 into the extension area beginning with location 1F.

8) *HCC,(D/DATA1)5,10,15,20,25,30,35,40

9) *HCC,(+)45,50,55,60,65,70

Examples 8 and 9 - Enter the 14 hexadecimal values 5 through 70 into the data block DATA1 in successive locations starting with location zero.

10) *HCC,(D/DATA1+20)75,100,105,110,115,,125,,13C

Enter the five hexadecimal values 75 through 115 in successive locations starting with location 20 of the data area DATA1. Loation 25 will be unchanged, 26 will hold 00000125, 27 will be unchanged, and 28 will hold 0000013C.

11) *HCC,(SUB1+70)01000010(X),20000005(C/COM1),40000007(D/DATA1)

Enter correction 0100XXXX into location 70 of SUB1. XXXX is modified relative to the program extension area. Put 2000XXXX into SUB1+71. XXXX is modified relative to the common area COM1. Put 4000XXXX into SUB1+72. XXXX is modified relative to the data area DATA1.

12) *HCC,(+2)20000007(SUB1)C

Put hexadecimal correction 20000007 into SUB1+75. Modify the 18-bit character address relative to subprogram SUB1.

13) *HCC,(+)20000030($)H

Put hexadecimal correction 20000030 into SUB1+76. Modify the 17-bit half-word address relative to subprogram SUB1.

MAP, MEMORY ALLOCATION PRINTOUT

The loader automatically produces a map of memory allocation of a loaded program at the time the load operation is complete. The MAP consists of information from the loader symbol table and appears as follows:

Heading	Category
SUBP	Name of each subprogram and the absolute address of the first location in each subprogram.
ENTR	Entry-point symbols in the program and the absolute address of each entry point in the subprograms.
COMM	Each common block name and the absolute starting address of each common block.
DATA	Each data block name and the absolute starting address of each data block.

The MAP is illustrated in figure 6-1.

```
MEMORY MAP

PROGRAM NAMES

ENLARGE     EBD2   A8CIIINP   EA46   ASCIIOUT    E861    CONTROL    EE19
FORMAT      E528   Q8QERROR   E4E0   Q8QSTP      E4CC    AMATHER    E49E
BLKDEBLK    E1B9   TSKMON     E1AE

SCRATCH COMMON BLOCKS

Q8QBUF      0000

ENTRY POINT NAMES

ABORT       0128   ABORTJM    0129   ABPACKD     E692    ABPICKD    E68A
ACTIVECK    E66D   ALLOCATE   0111   ARCHKI%     E4B3    BDBWTIME   0054
BKSP        0105   BSY        013C   CALL        011D    CLOSE      0110
CTOC        C115   CT01       0114   DATE        0117    DEVICE     0118
DVCHKI%     E4B1   DWAIT      013E   ENABLE      0124    ENABLE%    E49E
ENLARGE     EEE8   ERASE      012F   FNCHKI%     E4B2    ILLUNIT    E61E
JACC        F0E0   JCIJNUM    F08D   JCIJPC      F09A    JCIJPL     F099
JCIJSCHL    F09A   LUNITBL    F3ED   MATHE%P     E4C2    MATHE%S    E4BD
MODIFY      0113   OPEN       010F   OPENMEM     0119    OVCHKI%    E4B4
PACK        E28E   PACKC      E2F8   PACKD       E1BF    PACKO      E2DF
PARM        EFCE   PFAULT     0123   PICK        E230    PICKC      E280
PICKD       E1B9   PICKI      E266   Q8QENGIN    EA60    Q8QENGOT   E87A
Q8QENTRY    E646   Q8QERROR   E4E0   Q8QEXIIS    E65C    Q8QIFRMT   E528
Q8QINDEC    EA58   Q8QINENC   E872   Q8QINGIN    EA46    Q8QINGOT   E861
Q8QIOINT    E619   Q8QIOTAB   E732   Q8QISCAN    E53D    Q8QITERM   E539
Q8QLGINI    EA67   Q8QLGIN2   EA6B   Q8QLGOT1    E882    Q8QLOOT2   E886
Q8QPAUSE    E4CC   Q8QSTOP    E408   Q8QTABLE    E80F    READIN     E6C0
READLU      0102   RELEASE    0112   RELMEM      011F    RESTREG    E717
RETURN      011C   REWD       0106   SAVREG      E704    SEOF       0104
STATUS      E6D5   STDFEXPS   0002   STDPSEG     0010    TIME       0116
TSCHED      0127   TSKMON     E1AE   TSTATUS     013D    ULOC       010A
UNLD        0107   UST        0108   USTUP       E6FA    UTYP       0109
WEOF        0103   WRITLU     0101   WRITOUT     E6C5

TRANSFER ADDRESS NAMES

ENLARGE     EEEB   Q8QENTRY   E646
```

Figure 6-1. MAP Example

APPENDIX A

CHARACTER SET

Table A-1 illustrates the MP-60 System Character Set for certain types of peripheral equipment. The MP-60 Code column shows the hexadecimal byte code (8 bits) used internally and sent to or received from a peripheral device in ASCII mode. the ASCII graphic column shows the printed or displayed graphic corresponding to the internal code. The default Standard 026 Card Punch column shows the Hollerith code punched to obtain the internal code in 026 mode. The ANSI Standard 029 Card Punch column shows the Hollerith code punched to obtain the internal code in the 029 mode. The ASD Card Punch column shows the Hollerith code in the MPP compatible ASD mode.

As cards are being read, the presence of a Card Reader Code Switch card causes subsequent cards to be read in a new mode. The presence of a *EOJ card causes a switch back to the site default mode. The format of the card Reader Code Switch card is as follows:

Column 1 contains a 12-11-0-1-2-3-4-5 multiple punch.

Column 2 contains a:

6	(6)	to switch to 026 mode.
9	(9)	to switch to 029 mode.
A	(12-1)	to switch to ASD mode.
space		to switch to site default mode.

The above applies only to the locally connected card reader. Translation of cards read into remote terminal card readers is governed by the particular remote terminal in use.

Table A-1 contains the matrix for the ASCII-coded character set.

TABLE A-1. MP-60 CHARACTER SET

MP-60 Code	ASCII Graphic	Default Standard 026 Card Punch Code	ANSI Standard 029 Card Punch Code	ASD Punch Code
$00	NUL	12-0-1-8-9	12-0-1-8-9	12-0-1-8-9
$01	SOH	12-1-9	12-1-9	12-1-9
$02	STX	12-2-9	12-2-9	12-2-9
$03	ETX	12-3-9	12-3-9	12-3-9
$04	EOT	7-9	7-9	7-9
$05	ENQ	0-5-8-9	0-5-8-9	0-5-8-9
$06	ACK	0-6-8-9	0-6-8-9	0-6-8-9
$07	BEL	0-7-8-9	0-7-8-9	0-7-8-9
$08	BS	11-6-9	11-6-9	11-6-9
$09	HT	12-5-9	12-5-9	12-5-9
$0A	LF	0-5-9	0-5-9	0-5-9
$0B	VT	12-3-8-9	12-3-8-9	12-3-8-9
$0C	FF	12-4-8-9	12-4-8-9	12-4-8-9
$0D	CR	12-5-8-9	12-5-8-9	12-5-8-9
$0E	SO	12-6-8-9	12-6-8-9	12-6-8-9
$0F	SI	12-7-8-9	12-7-8-9	12-7-8-9
$10	DLE	12-11-1-8-9	12-11-1-8-9	12-11-1-8-9
$11	DC1	11-1-9	11-1-9	11-1-9
$12	DC2	11-2-9	11-2-9	11-2-9
$13	DC3	11-3-9	11-3-9	11-3-9
$14	DC4	4-8-9	4-8-9	4-8-9
$15	NAK	5-8-9	5-8-9	5-8-9
$16	SYN	2-9	2-9	2-9
$17	ETB	0-6-9	0-6-9	0-6-9
$18	CAN	11-8-9	11-8-9	11-8-9
$19	EM	11-1-8-9	11-1-8-9	11-1-8-9
$1A	SUB	7-8-9	7-8-9	7-8-9
$1B	ESC	0-7-9	0-7-9	0-7-9
$1C	FS	11-4-8-9	11-4-8-9	11-4-8-9
$1D	GS	11-5-8-9	11-5-8-9	11-5-8-9
$1E	RS	11-6-8-9	11-6-8-9	11-6-8-9
$1F	US	11-7-8-9	11-7-8-9	11-7-8-9
$20	space			
$21	!	11-0	12-7-8	12-7-8
$22	"	4-8	7-8	7-8
$23	#	0-6-8	3-8	6-8
$24	$	11-3-8	11-3-8	11-3-8
$25	%	6-8	0-4-8	12-5-8
$26	&	0-7-8	12	12-6-8
$27	'	11-5-8	5-8	4-8
$28	(	0-4-8	12-5-8	0-4-8
$29	)	12-4-8	11-5-8	12-4-8
$2A	*	11-4-8	11-4-8	11-4-8
$2B	+	12	12-6-8	12
$2C	,	0-3-8	0-3-8	0-3-8
$2D	-	11	11	11
$2E	.	12-3-8	12-3-8	12-3-8
$2F	/	0-1	0-1	0-1

TABLE A-1. MP-60 CHARACTER SET (Contd)

MP-60 Code	ASCII Graphic	Default Standard 026 Card Punch Code	ANSI Standard 029 Card Punch Code	ASD Punch Code
$30	0	0	0	0
$31	1	1	1	1
$32	2	2	2	2
$33	3	3	3	3
$34	4	4	4	4
$35	5	5	5	5
$36	6	6	6	6
$37	7	7	7	7
$38	8	8	8	8
$39	9	9	9	9
$3A	:	2-8	2-8	2-8
$3B	;	12-7-8	11-6-8	11-6-8
$3C	<	12-0	12-4-8	11-5-8
$3D	=	3-8	6-8	3-8
$3E	>	11-7-8	0-6-8	0-6-8
$3F	?	11-6-8	0-7-8	0-7-8
$40	@	5-8	4-8	5-8
$41	A	12-1	12-1	12-1
$42	B	12-2	12-2	12-2
$43	C	12-3	12-3	12-3
$44	D	12-4	12-4	12-4
$45	E	12-5	12-5	12-5
$46	F	12-6	12-6	12-6
$47	G	12-7	12-7	12-7
$48	H	12-8	12-8	12-8
$49	I	12-9	12-9	12-9
$4A	J	11-1	11-1	11-1
$4B	K	11-2	11-2	11-2
$4C	L	11-3	11-3	11-3
$4D	M	11-4	11-4	11-4
$4E	N	11-5	11-5	11-5
$4F	O	11-6	11-6	11-6
$50	P	11-7	11-7	11-7
$51	Q	11-8	11-8	11-8
$52	R	11-9	11-9	11-9
$53	S	0-2	0-2	0-2
$54	T	0-3	0-3	0-3
$55	U	0-4	0-4	0-4
$56	V	0-5	0-5	0-5
$57	W	0-6	0-6	0-6
$58	X	0-7	0-7	0-7
$59	Y	0-8	0-8	0-8
$5A	Z	0-9	0-9	0-9
$5B	[	7-8	12-2-8	12-0
$5C	\	12-5-8	0-2-8	0-2-8
$5D	]	0-2-8	11-2-8	11-0
$5E	^	12-6-8	11-7-8	11-7-8
$5F	_	0-5-8	0-5-8	0-5-8

TABLE A-1. MP-60 CHARACTER SET (Contd)

MP-60 Code	ASCII Graphic	Default Standard 026 Card Punch Code	ANSI Standard 029 Card Punch Code	ASD Punch Code
$60	`	1-8	1-8	1-8
$61	a	12-0-1	12-0-1	12-0-1
$62	b	12-0-2	12-0-2	12-0-2
$63	c	12-0-3	12-0-3	12-0-3
$64	d	12-0-4	12-0-4	12-0-4
$65	e	12-0-5	12-0-5	12-0-5
$66	f	12-0-6	12-0-6	12-0-6
$67	g	12-0-7	12-0-7	12-0-7
$68	h	12-0-8	12-0-8	12-0-8
$69	i	12-0-9	12-0-9	12-0-9
$6A	j	12-11-1	12-11-1	12-11-1
$6B	k	12-11-2	12-11-2	12-11-2
$6C	l	12-11-3	12-11-3	12-11-3
$6D	m	12-11-4	12-11-4	12-11-4
$6E	n	12-11-5	12-11-5	12-11-5
$6F	o	12-11-6	12-11-6	12-11-6
$70	p	12-11-7	12-11-7	12-11-7
$71	q	12-11-8	12-11-8	12-11-8
$72	r	12-11-9	12-11-9	12-11-9
$73	s	11-0-2	11-0-2	11-0-2
$74	t	11-0-3	11-0-3	11-0-3
$75	u	11-0-4	11-0-4	11-0-4
$76	v	11-0-5	11-0-5	11-0-5
$77	w	11-0-6	11-0-6	11-0-6
$78	x	11-0-7	11-0-7	11-0-7
$79	y	11-0-8	11-0-8	11-0-8
$7A	z	11-0-9	11-0-9	11-0-9
$7B	{	12-2-8	12-0	12-2-8
$7C	|	12-11	12-11	12-11
$7D	}	11-2-8	11-0	11-2-8
$7E	~	11-0-1	11-0-1	11-0-1
$7F	DEL	12-7-9	12-7-9	12-7-9

APPENDIX B

GLOSSARY

Abort	The premature termination of a process whenever an irrecoverable situation (either hardware or software) occurs.
Absolute	Refers to actual machine addresses (i.e., not relocated).
Assemble	The process by which an object (binary) module is created from a symbolic language program (e.g., COMPASS assembler).
Asynchronous	Refers to a type of serial transmission in which bit synchronization is accomplished for each character.
Batch	Class of tasks which run on a time-available basis.
BCLA	Buffered Communications Line Adaptor - the RS-232C asynchronous communication lines.
Block	A grouping of machine words or bytes. Usually a collection of one or more records used in I/O to reduce the number of physical operations.
Buffer	A portion of memory used to collect data in order to compensate for speed differences between the processor and peripheral devices.
CALL	The transference of control to a closed routine or task. A monitor function, CALL, is used to activate a specific task.
Callee	The task called by a caller.
Caller	A task that calls another task.
CLA	Communication line adaptor between the CPU and the communications line.
Compile	The process by which an assembly (and usually an object) module is created from a problem solving language such as FORTRAN. A compiler usually generates several machine instructions from a single symbolic statement.
Common	An area of memory that can be shared between programs. Tasks can communicate through common areas.

CPU	Central processing unit.
Data	An area of memory that can be restored with data at load time and can be shared only between programs of one task; not between tasks.
Dispatcher	An operating system routine that unthreads a task from the top of the ready list and places it into execution.
ESR	Executive Service Request - MPX/OS routines invoked by the MON instruction.
Establish	Acquire a task control table for a task and initiate the loading process.
File	A collection of blocks and/or records, usually of related data. Each mass storage file has an entry in the system file label directory.
Interrupt	A break in the normal processing flow usually caused by a hardware-generated signal (involuntary interrupt). Interrupts can be enabled or disabled and occur with an associated priority. Processes that are interrupted are later resumed at the point of interruption. A software-generated interrupt occurs when a task makes a monitor request (voluntary interrupt). An exchange package describing machine conditions at the point of interruption is generated by the hardware/firmware.
ITS	Interactive Terminal Subsystem.
Job	The sequential and/or parallel execution of tasks. Begins with *JOB card and ends with *EOJ card.
Job Control Table (JCT)	An area of storage containing information for controlling a given job.
Job Manager	A task that processes the input stream of the job. The job manager is a set of re-entrant programs shared by all user jobs.
Library	A collection of frequently used, checked-out programs maintained on an external device that can be loaded and executed separately (by a control card) or in conjunction with a user's program (via an external). Libraries must be arranged to minimize searching (one library program can declare another external, etc.).

Linkage	The interconnection between routines. The loader matches externals and entry points to establish linkage.
Loader	A subtask of the job manager that is used to load, relocate, and link binary object modules.
Logical Units	A number from 1 to 63 that is used to identify a physical unit or a file. Logical unit assignments correspond to a specific job and are in effect only during the life of the job.
MPX/OS	The MPX Operating System.
MP-32	MP-60 CPU Emulation Device.
MPP	MP-60 CPU Emulation Device (Militarized or Ruggedized).
Ordinal	The relative location of an entry in a table. The absolute location of an entry can be obtained by multiplying the ordinal by the number of machine words per entry and adding the starting address of the table.
Page	A block (4K words) of main memory. Paging is a technique where a logical address is transformed via a set of page registers to a physical address.
Port	The communications line between the CPU and the user.
Priority	A value (0 to 511) assigned to a task that facilitates scheduling and processing within the operating system.
Queue	A first-in, first-out list used to control, for example, the work to be done. (See Stack).
Ready List	A prioritized list of tasks waiting for control of the CPU. (See Schedule and Dispatcher).
Re-entrant	A routine coded such that it can be called while executing at a higher priority or during a wait and resume processing later at the point of interruption. Usually, all intermediate results are maintained in registers.
Relocatable	Refers to a program that has been prepared by a source language compiler or assembler to be loaded into any area of available memory.
Resident	The portion of the operating system which resides permanently in main memory.

RETURN	A monitor function that terminates a task and transfers control to the point in the caller where the call originated. A task can return with or without release of memory.
Schedule	The process of placing a task on the ready list by priority. A task can be scheduled at the top or behind other tasks of equal priority.
Spooling	Refers to the simultaneous I/O of standard units while the CPU is processing other tasks.
Stack	A last-in, first-out list (see Queue).
Status	A stage or condition of an I/O request or a task itself (e.g., busy, ready, etc.).
Synchronous	Serial transmission in which characters are sent bit wise without start and stop bits.
System Initialize	Refers to the initial system load process where the resident is loaded, memory initialized, and the system tasks are started.
Task	An independent unit of work that can compete for the resources of the system. A task can call and be called by other tasks.
Task Control Table (TCT)	An area of memory containing information used to control a task.
Terminal	The device connected to the user end of the Port.
Terminate	The process of completing a job. A job can terminate normally or abnormally.
Thread	A linked list of elements, the contents of each thread cell contains the address of the next thread cell and so on until a thread cell of zero which indicates the end of the list.
Utility	A routine or procedure that supports the operation of a system (e.g., an I/O transfer routine).

APPENDIX C

BLOCKER/DEBLOCKER

Figure C-1 illustrates the relationship of records, buffers, and blocks for a block type device. Block i is within the user-defined core buffer area. Block i contains records 1 through n with appropriate headers.

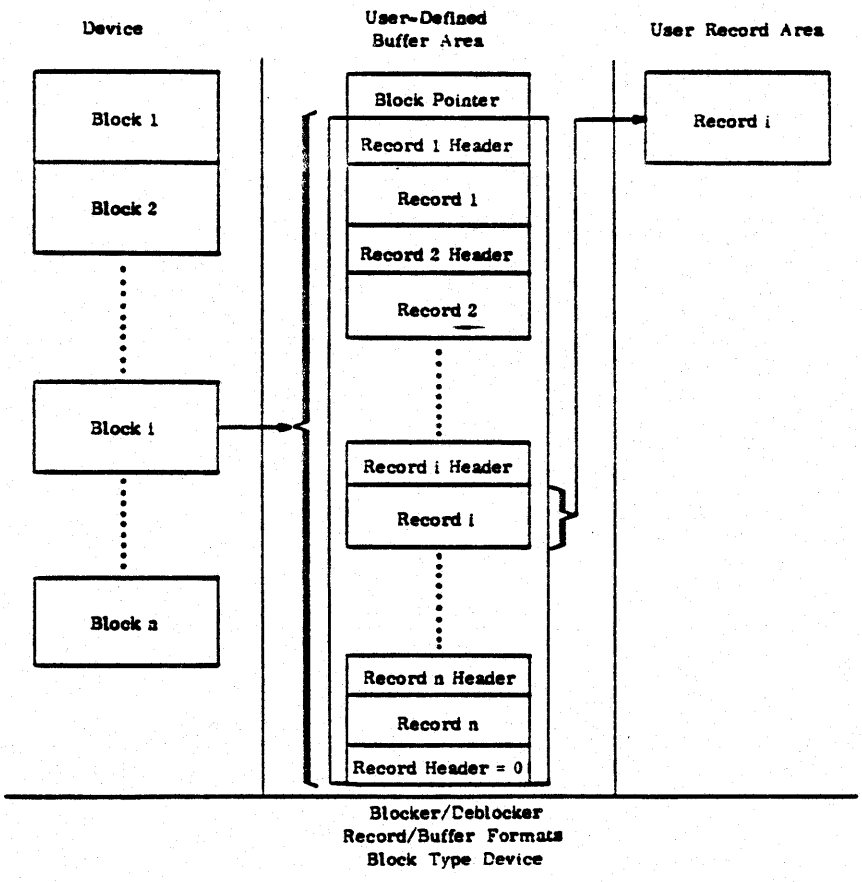


Figure C-1. BLOCKER/DEBLOCKER records/buffer/blocks on block type device

Figure C-2 illustrates the relationship of user records, buffers, and physical records (that is, data actually transferred to or from an I/O device). Physical record i is within the user-defined buffer area with appropriate headers. Physical record i is the same as record i in the user record area.

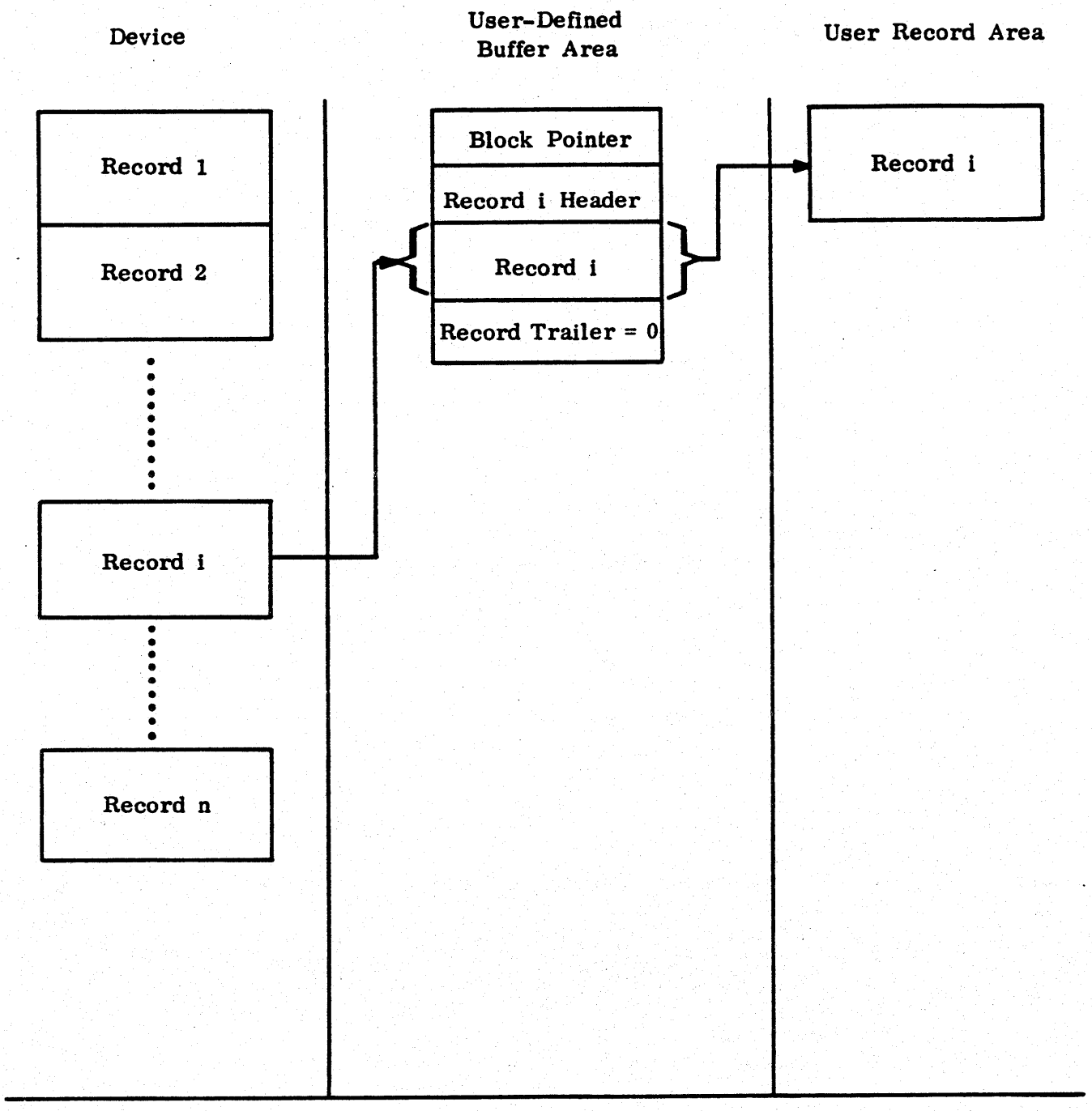


Figure C-2. BLOCKER/DEBLOCKER Physical/Logical Records

Figure C-3 illustrates double buffering when blocking a block or record type device. For a blocking type device, block i is transferred physically to the device at the same time records are being passed to block i+1. For a record type device, record i is transferred physically to the device at the same time record i+1 is being transferred to the user-defined buffer area.

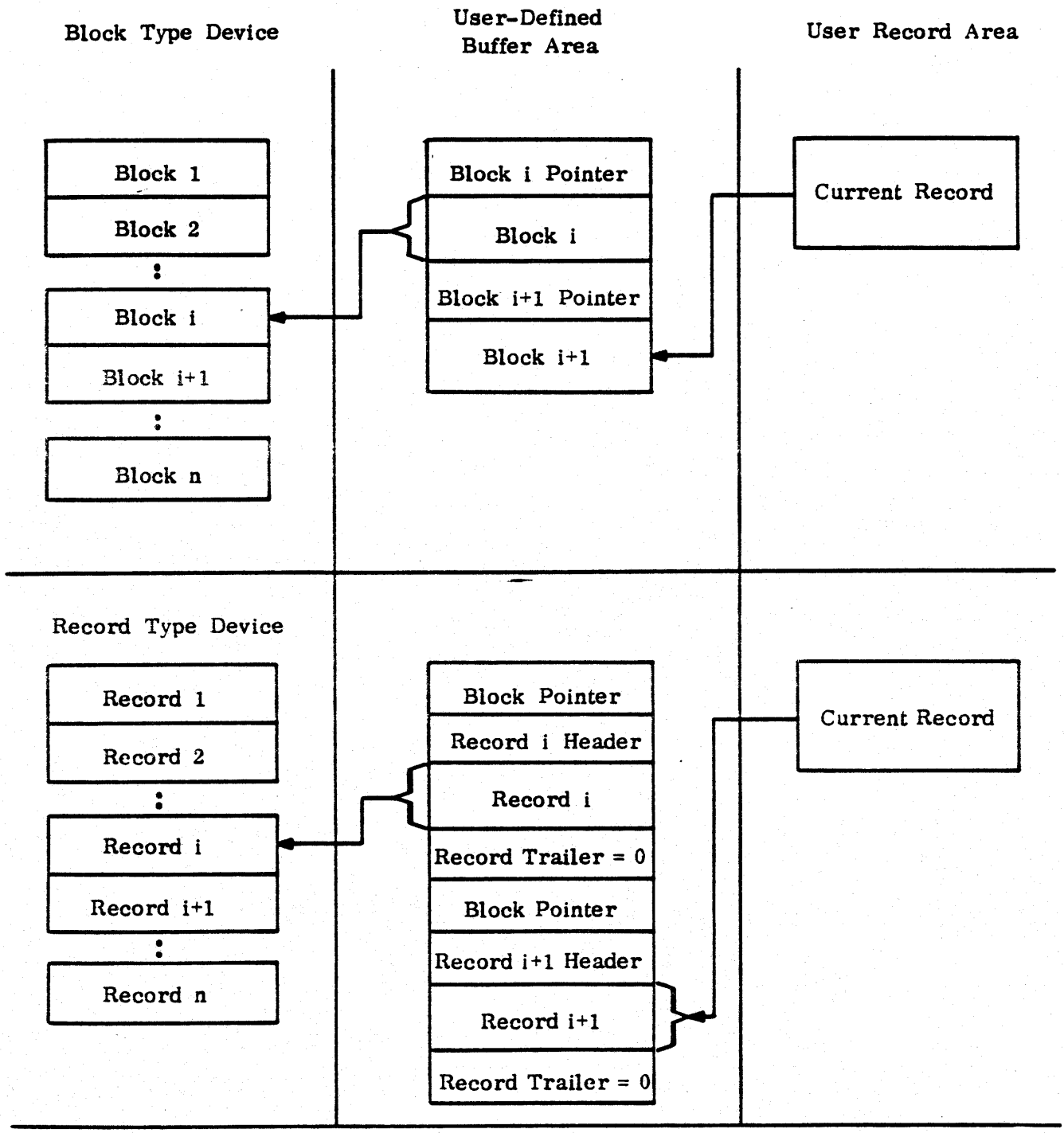


Figure C-3. BLOCKING

Figure C-4 illustrates double buffering when deblocking a block or record type device. For a blocking type device, block i+1 is transferred physically from the device at the same time records are being passed from block i. For a record type device, record i+1 is transferred physically from the device to the user-defined buffer area at the same time record i is being transferred from the user defined buffer area.

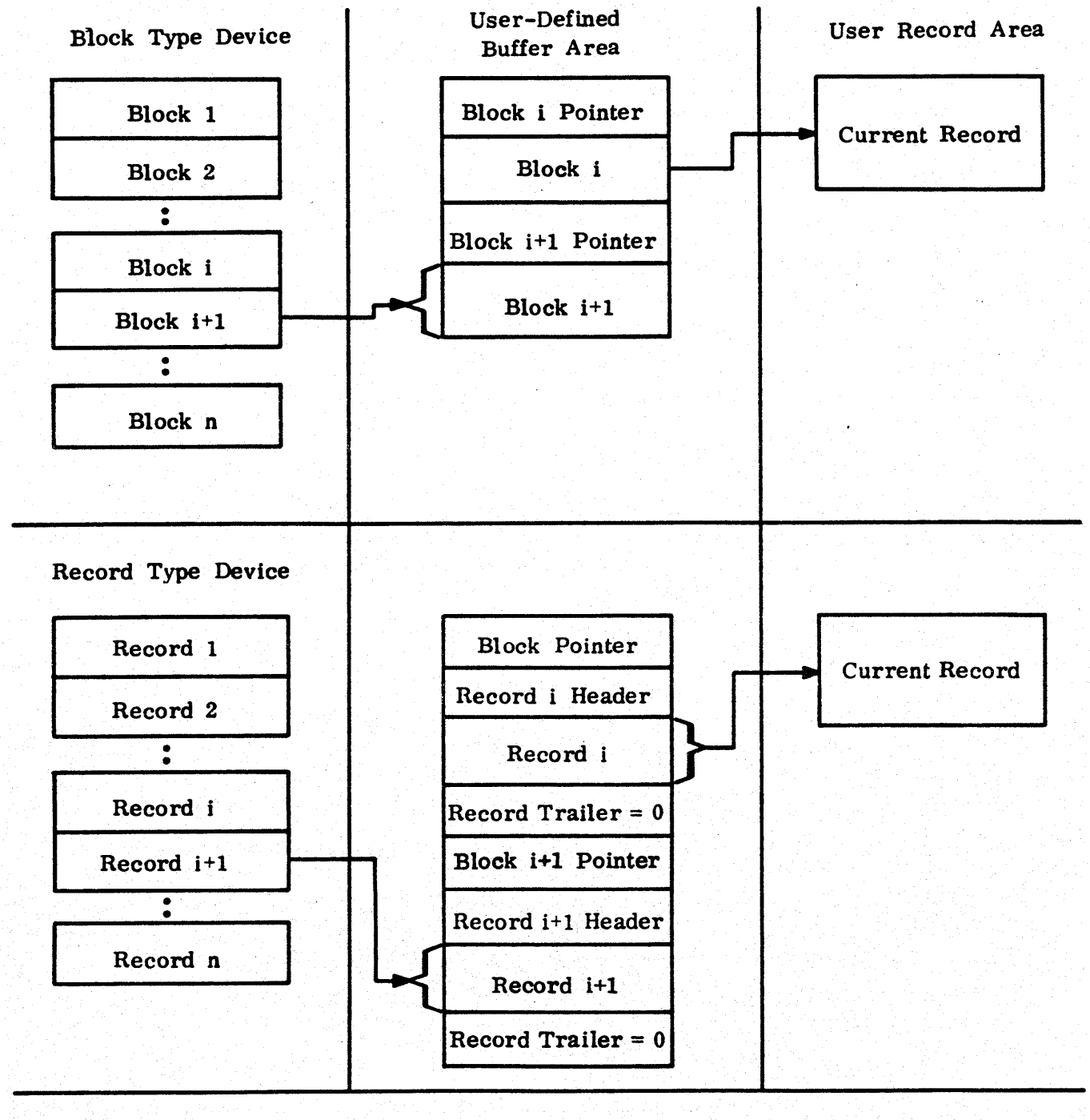


Figure C-4. DEBLOCKING

APPENDIX D

SYSTEM ERROR CODE DEFINITIONS

ABORT TYPES AND CODES

Abort Type	Abort Code	Description
1	YY	I/O abort.
	1	Operator rejected request to ready a unit for operation.
	2	Buffer size larger than 4096 words.
	3	Logical unit unassigned.
	4	Attempt to write on read-only file.
	5	An input was attempted into a read-only page.
	6	Hardware reject.
	7	An input or output was attempted upon a protected page.
	8	Illegal logical unit number.
	9	Illegal command.
	10	No space in memory pool for RET initialization.
	11	Attempt to locate to block number not open.
2	YY	Operator abort.
	1	Operator aborted the job.
	2	Job time limit expired.
3	YY	Page fault.
	1	Read-only violation.
	2	Protect violation.
	3	Read-only or fully protected page violation.
4	YY	Memory problem.
	1	Memory parity error - instruction.
	2	Memory reject - instruction.
	3	Memory parity error - operand.
	4	Memory reject - operand.
5	YY	Arithmetic fault.
	1	Arithmetic fault.
	2	Function fault.
	3	Exponent fault.
	4	Divide fault.
6	YY	Illegal instructions.
	1	Privileged instruction encountered in program state.
	2	Illegal address encountered in a monitor call.
	3	Illegal monitor call.

Abort Type	Abort Code	Description
7	YY	Voluntary abort.
	1	User's program made a monitor call to ABORT.
	2	Task deleted job (DELJOB).
8	YY	Parameter address error.
	1	Caller's parameter address is in a protected page.
	2	Parameter address in unassigned page for CTOI.
12	YY	Control card errors.
	1	Unrecognized card.
	2	Irrecoverable error on OUT.
	3	Incomplete parameter list.
	6	Logical unit number already assigned.
	7	Invalid logical unit number.
	9	Unidentified parameters on dump request.
	10	Unrecognized parameter.
	11	LUN equated to unassigned logical unit number.
	12	Exceeded scheduled hardware.
	13	Operator rejected request on PAUSE card.
	14	Operator rejected EQUIP request.
	15	Parameter list improperly terminated.
	20	PACKD on standard output unit.
	21	Illegal control card.
	22	Too many *TASK cards.
	23	Error on ABS file.
	25	Standard file error.
	26	Print lines limit exceeded.
	27	Punch cards limit exceeded.
	28	Blocker/deblocker error.
	30	Scratch limit exceeded.
	31	Illegal call to library task.
	32	All queue entries full.
	33	Job evicted from input queue.
13	YY	Loader error. (see MPX Loader Diagnostics below.)
14	YY	Hexadecimal correction card error.
	1	Location is undefined.
	2	Location is in common.
	3	Location field is missing.
	4	Program is undefined.
	5	Illegal program name.
	6	Program name too large.
	7	Illegal hexadecimal field.
	8	Extension area overflows memory.

15	YY	Task call error.
	1	Caller calling itself.
	2	Caller not waiting on common pass.
	3	Too many tasks.
	4	Circular call.
	5	Caller not waiting on parameter pass.
	6	Not enough memory.

MPX LOADER DIAGNOSTICS

Several conditions can arise during the loading of a program that result in a diagnostic. Some conditions result in the job being aborted while others are merely reported. The format of the loader diagnostic is as follows:

(program name) ERROR NO. YY WORDS 1 AND 2 = XXXXXXXXXXXXXXXX

The error conditions and the actions taken are described below:

Error No.	Cause	Action C = Continue A = Abort
0001	Checksum error (IDC,BCT,EPT,RIF,or EXT).	C
0002	LST table overflowed loaded program.	A
0003	Doubly defined program name - not loaded.	C
0004	Current program will overlay LST table.	A
0005	Current absolute program overlays loaded program.	A
0006	Mixed program types - absolute and relocatable.	C
0007	Current absolute program LST table.	A
0008	Current program will overlay scratch common.	A
0009	BCT name occurred before as other type - use %%000000.	C
0010	BCT card out of sequence.	A
0011	Data common block overlays LST table.	A
0012	Data common block overlays scratch common area.	A
0013	Scratch common overlays loaded program.	A

Error No.	Cause	Action C = Continue A = Abort
0014	Second SCOM of same name greater than first SCOM.	A
0015	Second DCOM of same name greater than first DCOM.	A
0016	EPT card out of sequence.	C
0017	Two entry points of same name - ignore second.	C
0018	EXT card out of sequence.	C
0019	Next string address out of program area.	A
0020	EXT string in tight loop.	A
0021	Next string address out of memory.	A
0022	Running checksum error - card missing perhaps.	C
0023	More than three transfer addresses - use last three found.	C
0024	Transfer name not an entry point - ignore name.	C
0025	IDC card not first card of deck.	A
0026	Nonbinary card between IDC and TRA cards.	A
0027	Second IDC card before TRA card.	A
0028	Current card out of sequence.	A
0029	Unrecognized card - out of sequence perhaps.	A
0030	*(library name) not found in library directory.	A
0031	No TRA card.	A
0032	Program size exceeds scheduled memory.	A
0033	Irrecoverable error on load unit.	A
0034	Library sequence error.	A
0035	Task monitor not loaded.	A

FILE MANAGER ERRORS

Error Code	Description
3	Incomplete parameter list.
4	No file name specified.
5	No block size specified.
6	No block count specified.
7	Illegal logical unit number.
11	Label file read error. *
12	File previously allocated.
13	Insufficient label file space. *
14	Illegal device type.
15	Too many devices.
16	Insufficient contiguous space.
17	Insufficient space available on the specified devices.
18	File size exceeds system limits.
19	Number of blocks to release exceeds the number of allocated blocks.
20	File not allocated.
21	Operator cannot place devices on-line.
22	Device label read error. *
23	Invalid logical unit number.
24	Logical unit previously defined by EQUIP or OPEN.
25	File is allocated as read-only and the OPEN call specifies read/write use.

* Consult system analyst.

Error Code	Description
26	File was previously opened and the use in the OPEN call conflicts with the previous OPEN use. A file can be opened only once with read/write usage.
27	Insufficient table space. *
28	File is open. A file cannot be modified or released while it is open.
29	Illegal access key.
30	Too many DIDs.
31	Label file cannot be closed.
32	Block size is 0.
33	Number of blocks is 0.
34	Segment count exceeded.
35	Partial open on block not in file.
36	Label checksum error.
37	User tried to close OUT, PUN, and INP files.
38	Unused.
39	DEVICEQ Equivalance; Equated LUN not assigned.
40	DEVICEQ; Device/Port unavailable or immediate port request not unique.
41	DEVICEQ; hardware not scheduled.

* Consult system analyst.

BLOCKER ERROR INDICATORS

Indicator	Routine	Description
1	PACK, PACKO, PACKC	Write attempted beyond file limits.
2	PACK, PACKO, PACKC	Logical unit is not open.
3	PACK, PACKO, PACKC	Write attempted on a read-only file or device.
4	PACKD	Buffer area already defined by previous PACKD or PICKD.
5	PACKD	Buffer size too large; inconsistent with file blocking definition.
6	PACKD	Buffer size too small.
7	PACK, PACKO, PACKC	Buffer area not defined by PACKD.
8	PACK	Record size (after removal of trailing blanks or zeros) is greater than buffer size.
9	PACK, PACKO, PACKC	Buffer area has been defined by PICKD.
10	PACKD, PACKC	Cannot perform these functions on logical unit 61 or 62.
11	PACKD, PACK, PACKO, PACKC	Logical unit invalid (not 1-63).
12	PACK, PACKO, PACKC	Irrecoverable I/O error.
13	PACK	Blocker pointer out of bounds; BLOCKER/DEBLOCKER pointers have been modified.
14	PACK	Record length = 0
15	PACK, PACKD, PACKC	Not enough disk scratch or memory pool space for job to run. Resubmit when resources are available (or try increasing the SCR value on the *SCHED card).

DEBLOCKER ERROR INDICATORS

Indicator	Routine	Description
1	PICKD, PICK, PICKI	End-of-file.
2	PICK, PICKI, PICKC	Logical unit is not open.
3	PICKD	Device is a write-only device.
4	PICKD	Buffer area already defined by previous PICKD or PACKD.
5	PICKD	Buffer size too large.
6	PICKD	Buffer size too small; inconsistent with file blocking definition.
7	PICK, PICKI, PICKC	Buffer area not defined by PICKD.
9	PICK, PICKI, PICKC	Buffer area has been defined by PACKD.
10	PICKD, PICKC	Cannot perform these functions on logical unit 63.
11	PICKD, PICK, PICKI, PICKC	Logical unit invalid (not 1-61)
12	PICKD, PICK	Irrecoverable I/O error.
13	PICK	Blocker pointer out of bounds; BLOCKER/DEBLOCKER pointers have been modified.
14	PICK, PICKI	Record length = 0.
15	PICKD	Not enough memory pool space for job to run. Resubmit job when resources are available.

APPENDIX E

MPX/OS ERROR RECOVERY PROCEDURES

The MPX/OS error recovery procedures are dependent upon the device and error status codes returned by the device managers. The recovery techniques employed for each error status code and associated devices are described in the following paragraphs.

Busy - Bit 31 set and no other error bits set.

All Devices

The request is retried periodically until the associated device manager generates an end of operation.

Not Ready - Bit 31 and 30 set, and not other error bits.

All Devices

The operator is notified via an Informative message and the request is retried periodically until the operator clears the condition.

Data error - Bit 31 and 25 set.
Lost Data - Bit 31 and 23 set.

Mass Storage and Flexible Disk

The request is retried six (6) times.

Magnetic Tape

The following sequence is executed:

1) Backspace
2) Retry operation
3) Repeat 1&2 six times

Read	Write
4) Skip back 3 records	4) Backspace
5) Skip 2 records forward	5) Erase
6) Retry operation	6) Repeat 1-5 six times
7) Repeat 1-6 six times	

All others

The operator is notified via an Informative message and the error is considered irrecoverable.

Hardware Error - Bit 31 and 24 set.

All Devices

The operator is notified via an Informative message and the error is considered irrecoverable.

Address Error - Bit 31 and 22 set.

Mass Storage Only

The following sequence is executed:

1) Return to zero seek.
2) Retry operation.
3) Repeat 1-2 six (6) times.

Seek Error - Bit 31 and 21 set.

Mass Storage and Flexible Disk Only

The operation is retried six (6) times.

Write Protect Fault - Bit 31 and 27 set.

Mass Storage and Magnetic Tape Only

The operator is notified via an Informative message and the request is retried periodically until the operator clears the condition.

Feed Failure - Bit 31 and 27 set.
Stacker Full - Bit 31 and 25 set.
Input Tray Empty - Bit 31 and 23 set.

Card Reader Only

The operator is notified via an Informative message and the request is retried periodically until the operator clears the condition.

Paper Fault - Bit 31 and 21 set.

Printer Only

The operator is notified via an Informative message and the request is retried periodically until the operator clears the condition.and awaits response.

End of Tape - Bit 31 and 21 set.

Magnetic Tape Only

The tape is unloaded and the operator is notified via an Informative message.

APPENDIX F

MASS STORAGE DEVICES

CONTROL DATA 9425 CARTRIDGE DISK DRIVE

The Control Data 9425 Cartridge Disk Drive contains a removable and a nonremovable device. Each device must contain a device label. The two devices have the following identical characteristics:

Sector size	100 words
Track size	16 sectors
Number of tracks	408 tracks
Allocation unit size	1 track
Capacity	652,800 words
Cylinder size	2 tracks

CONTROL DATA 844 DISK STORAGE UNIT

The Control Data 844 Disk Storage Unit contains one removable device. Each device has the following characterics:

Sector size	120 words
Track size	24 sectors
Number of tracks	7,806 tracks
Allocation unit size	80 sectors
Capacity	22,483,200 words
Cylinder size	19 tracks

CONTROL DATA 9427 CARTRIDGE DISK DRIVE

The Control Data 9427 Cartridge Disk Drive contains a removable and nonremovable device. Each device must contain a device label. The two devices have the following identical characteristics:

Sector size	100 words
Track size	16 sectors
Number of tracks	816 tracks
Allocation unit size	1 track
Capacity	1,305,600 words
Cylinder size	2 tracks

CONTROL DATA 9760 DISK UNIT

The Control Data 9760 Disk Unit contains one removable device. Each device has the following characteristics:

Sector size	128 words
Track size	32 sectors
Number of tracks	2,055 tracks
Allocation unit size	1 track
Capacity	8,417,280 words
Cylinder size	5 tracks

CONTROL DATA 9762 DISK UNIT

The Control Data 9762 Disk Unit contains one removable device. Each device has the following characteristics:

Sector size	128 words
Track size	32 sectors
Number of tracks	4,110 tracks
Allocation unit size	2 tracks
Capacity	16,834,560 words
Cylinder size	10 tracks

CONTROL DATA 1867-1 DISK STORAGE UNIT

The Control Data 1867-1 Disk Storage Unit contains one removable device. Each device has the following characteristics:

Sector size	48 words (192 bytes)
Track size	64 sectors
Number of tracks	2,020 tracks
Allocation unit size	16 tracks (768 words)
Capacity	6,205,440 words
Cylinder size	5 tracks

CONTROL DATA 1867-2 DISK STORAGE UNIT

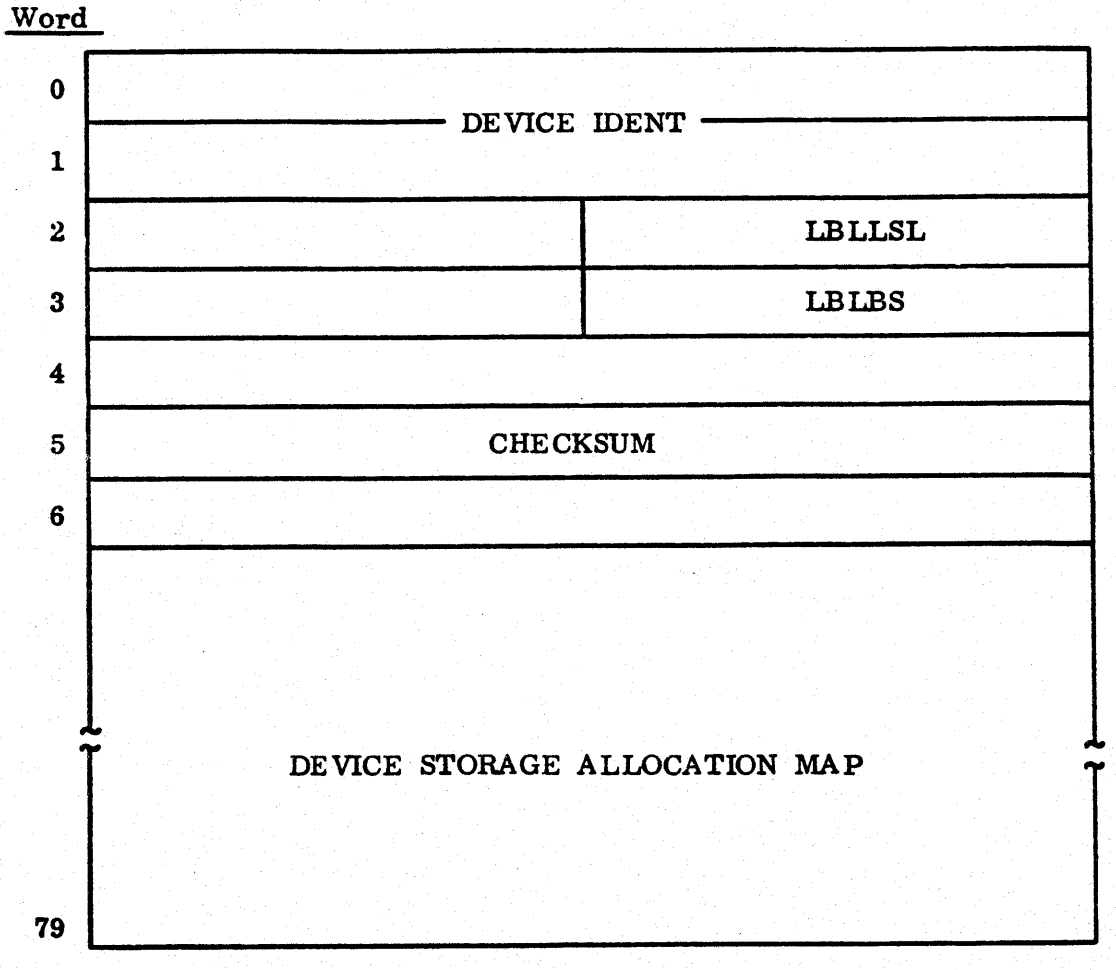
The Control Data 1867-2 Disk Storage Unit contains one removable device. Each device has the following characteristics:

Sector size	48 words (192 bytes)
Track size	64 sectors
Number of tracks	4,040 tracks
Allocation unit size	16 tracks (768 words)
Capacity	12,410,880 words
Cylinder size	5 tracks

APPENDIX G

MASS STORAGE LABELS

Two types of labels are associated with mass storage; the device label, which defines a physical disk pack (fixed or removable), and the file label, which defines files on mass storage devices. Labels can be listed in various formats by the FMP utility program. These labels are shown on the following pages.

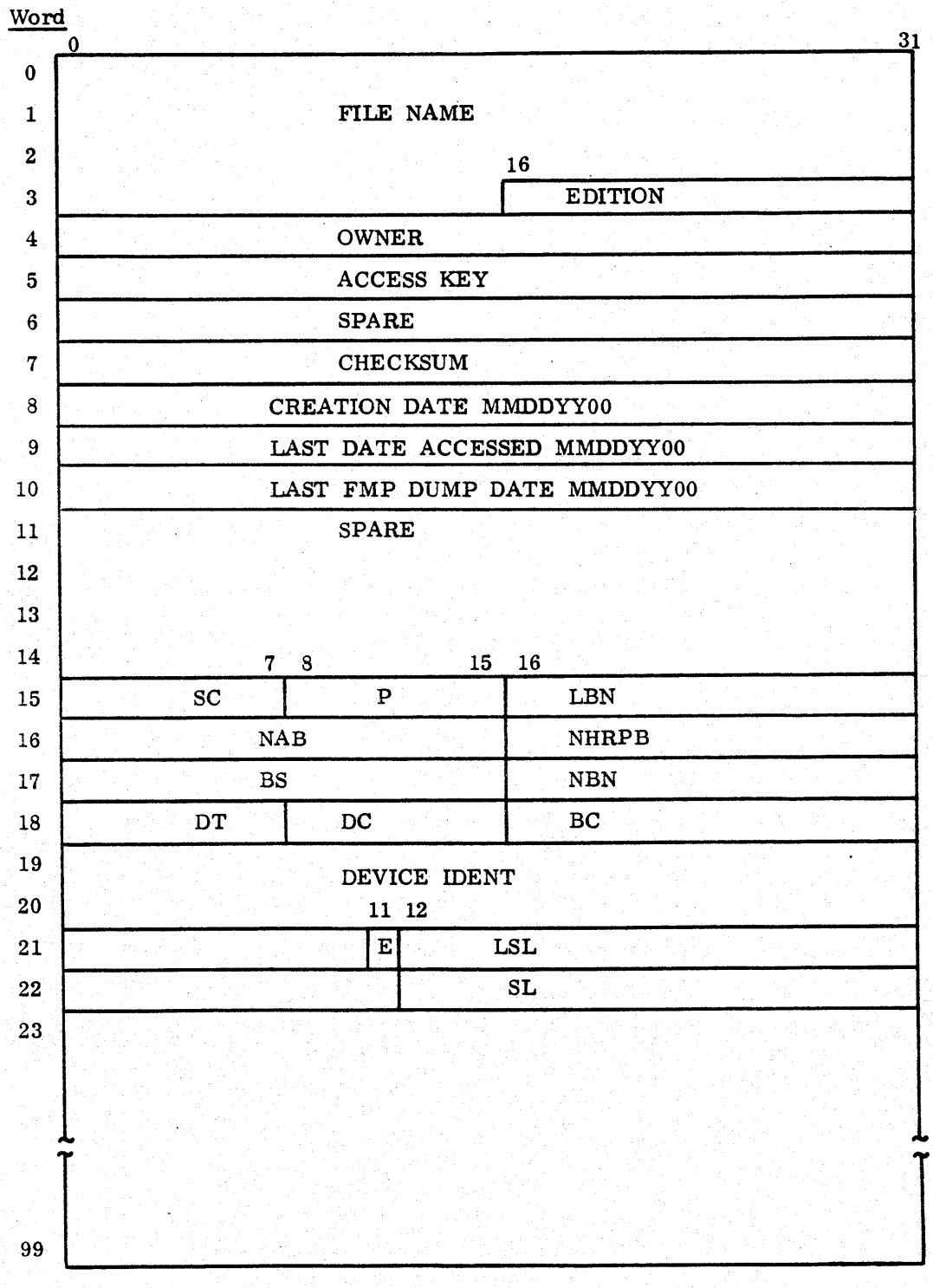


Words 6 through 79 contain a bit mapping of allocation units on the device and represent units 0 through 2304 of the device. The size of an allocation unit is device dependent (see appendix F for mass storage device characteristics). A bit set to 1 indicates the corresponding allocation unit is assigned. A bit set to 0 indicates the allocation unit is available.

FIELD DESCRIPTIONS

Field	Size	Description
DEVICE IDENT	8 bytes	Identity of the device pack.
LBLLSL	16 bits	Sector address of the beginning of the label file (only appears on the primary system device, disk unit 0).
LBLBS	16 bits	Block size, in words, of the label file.
CHECKSUM	32 bits	Binary checksum of the entire device label.

FILE LABEL


Word
0
31
0
1
FILE NAME
2
16
3
EDITION
4
OWNER
5
ACCESS KEY
6
SPARE
7
CHECKSUM
8
CREATION DATE MMDDYY00
9
LAST DATE ACCESSED MMDDYY00
10
LAST FMP DUMP DATE MMDDYY00
11
SPARE
12
13
14
7 8
15 16
15
SC
P
LBN
16
NAB
NHRPB
17
BS
NBN
18
DT
DC
BC
19
DEVICE IDENT
20
11 12
21
E
LSL
22
SL
23
99

FIELD DESCRIPTIONS

Field Name	Size	Description
FILE NAME	14 bytes	Identifies the file and is used in file manager references to the file.
EDITION NO.	2 bytes	Parameter to identify different versions of the same file.
OWNER	4 bytes	Identity of the owner of a file.
ACCESS KEY	4 bytes	Controls access to the file.
CHECKSUM	4 bytes	32-bit binary checksum of the entire device label.
CREATION DATE	4 bytes	Each byte represents binary value.
LAST DATE ACCESSED	4 bytes	Each byte represents binary value.
LAST FMP DUMP DATE	4 bytes	Each byte represents binary value.
SC	1 byte	Number of segments in the file.
P	4 bits	Security level of file.
	4 bits	Protection flag used by the I/O system: = 0, file is read or write. = 1, file is read only.
LBN	2 bytes	Block number of the label in the label file.
NAB	2 bytes	Number of blocks allocated to the file.
NHRPB	2 bytes	Number of sectors per block.
BS	2 bytes	Block size; number of words per block.
NBN	2 bytes	Next block number; next block number to read from or to be written into.

Field Name	Size	Description
DT	1 byte	8-bit code to indicate the type of mass storage device containing the file: = 1, 9425 = 2, 844 = 3, 9427 = 4, 1867-1 = 5, 1867-2
DC	1 byte	Number of devices on which the file resides.
BC	2 bytes	Highest block written.
DEVICE IDENT *	2 words	Identity of the device containing the segment map following the device identification.
E	1 bit	Flag to indicate end of device map; 1 = end of device segments.
LSL **	20 bits	Lower sector address; sector address at which this segment begins.
SL **	20 bits	Segment length; number of sectors in this segment.

NOTE: A maximum of eight devices and/or 38 segments may be specified for one file.

* Repeated for each device.
** Repeated for each segment on device.

APPENDIX H

PROGRAMMING CONVENTIONS

REGISTER NAMING CONVENTIONS

Operand/index registers used in coding examples in this document are given symbolic names as specified below. These symbolic symbols are preequated by the COMPASS assembler.

Name	Register	Name	Register
X0	0	R0	16
X1	1	R1	17
X2	2	R2	18
X3	3	R3	19
X4	4	R4	20
X5	5	R5	21
X6	6	R6	22
X7	7	R7	23
X8	8	R8	24
X9	9	R9	25
H0	10	RA	26
H1	11	RB	27
H2	12	RC	28
H3	13	RD	29
H4	14	RE	30
H5	15	RF	31

FORTRAN CALLING SEQUENCE CONVENTIONS

The calling sequence generated by FORTRAN for external subroutines and functions is as follows:

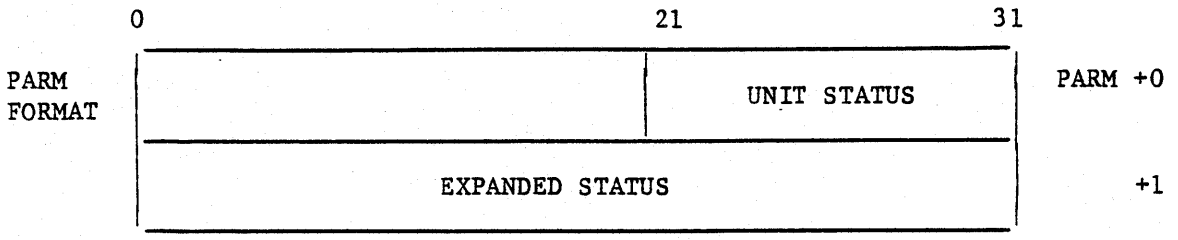
```
RTJ     name
UJP     *+n+1       (n = number of parameters)
NOP     ap1         (ap = actual parameter addres)
NOP     ap2
 .
 .
 .
NOP     apn
```

Function subprograms expect the results to be returned in register RE (single-precision result) or registers RE and RF (double precision results).

APPENDIX I

HARDWARE/DEVICE CODES

The information returned in PARM for the UST, MUST and STATUS ESRs is as follows:



The unit status for each device is as follows:

Dummy

No error bits or status used.

Files

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set then the device is busy.

30 Not Ready -- When this bit and bit 31 and set, the mass storage device is in a not ready condition.

29 End of File -- When this bit and bit 31 are set, an attempt was made to read beyond the end of the file.

28 Transmission Mode -- After a data transfer this bit is one.

27 End of Device -- When this bit and bit 31 are set a transfer beyond the highest available sector was attempted.

26 End of Allocated Blocks -- When this bit and bit 31 are set a write beyond the highest allocated block was attempted and the file requires expansion.

25 Data Error -- When this bit and bit 31 are set a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set a hardware condition was encountered.

23 Lost Data -- When this bit and bit 31 are set a lost data condition was encountered.

Files (Contd)

22 Address Error -- When this bit and bit 31 are set a sector address error was encountered.

21 Seek Error -- When this bit and bit 31 are set a seek error was encountered.

Pipes

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the pipe has been disconnected.

29 End of File -- When this bit and bit 31 are set, an attempt was made to read beyond the end of the data.

28 Transmission Mode -- After a data transfer, this bit is set.

27-26 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

Interactive and Communication Network

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the terminal connection was lost.

29 Data fills buffer (transparent mode only).

28-27 Not used.

26 Checksum Error (transparent mode only).

25 Character Parity Error (transparent mode only).

24 Not used.

23 Lost data since last read. MPX was forced to discard the data as the linked task had not requested any data. The data returned is valid and Bit 31 is not set as a result of this status.

22 BREAK was detected.

21 Framing error was detected (transparent mode only).

Expanded Status -- If no error conditions are set, then Expanded Status will contain the following:

After READ opeation = Number of characters received.
After WRITE operation = Number of characters not transmitted.

1867 SMD/MMD

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the mass storage device is in a not ready condition.

29 Not used.

28 Transmission Mode -- After a data transfer this bit is set. After a UINT instruction only, this bit is set when an unsolicited interrupt occurs.

27 Write Protect Fault -- When this bit and bit 31 are set, a write to a protected unit was attempted.

26 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set, a hardware error condition was encountered.

23 Lost Data -- When this bit and bit 31 are set, a lost data condition was encountered.

22 Address Error -- When this bit and bit 31 are set, a sector address error was encountered.

21 Seek Error -- When this bit and bit 31 are set, a seek error was encountered.

1833-5 FDD

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the device is in a not ready condition.

29 Not used.

28 Transmission Mode -- After a data transfer this bit is set. After a UINT request, only this bit is set to indicate an unsolicited interrupt occurrence.

1833-5 FDD (Contd)

26-27 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set, a hardware error condition was encountered.

23 Lost Data -- When this bit and bit 31 are set, a lost data condition was encountered.

22 Not used.

21 Seek Error -- When this bit and bit 31 are set, a seek error was encountered.

1829 Reader

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the device is in a not ready condition.

29 End of File -- When this bit and bit 31 are set, a card with a 7 and 8 punch in the first column was read.

28 Transmission Mode -- After a data transfer, this bit is set to indicate the type of card (0 - ASCII, 1 - Binary).

27 Feed Failure -- When this bit and bit 31 are set, a card failed to feed through the read station. The card reader also goes not ready.

26 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set, a hardware error condition was encountered.

23 Input Tray Empty -- When this bit and bit 31 are set, the input hopper is empty, the card reader is not ready, and no card was read.

21-22 Not used.

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

1827 Printer

30 Not Ready -- When this bit and bit 31 are set, the device is in a not ready condition.

29 Not used.

28 Transmission Mode -- After a data transfer this bit is zero.

26-27 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set, a hardware error condition was encountered.

22-23 Not used.

21 Paper Fault -- When this bit and bit 31 are set, the printer is out of paper and not ready.

1860-4 Tape

31 Reject -- always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the device is in a not ready condition.

29 End of File -- when this bit and bit 31 are set, read or write of a tape mark occurred.

28 Transmission Mode -- after a data transfer this bit is set.

27 Write Protect Fault -- when this bit and bit 31 are set, a write to a protected unit was attempted.

26 Not used.

25 Data Error -- when this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- when this bit and bit 31 are set, a hardware error condition was encountered.

23 Lost Data -- when this bit and bit 31 are set, a lost data condition was encountered.

22 Load Point -- when this bit is set, the tape unit is resting at load point.

1860-4 Tape (Contd)

21 End of Tape -- when this bit and bit 31 are set, the End of Tape foil was encountered.

2558-3 Coupler

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

30 Not Ready -- When this bit and bit 31 are set, the switch on the coupler card is switched in the off line position.

29 End of file -- When this bit and bit 31 are set, an end of file condition was sensed.

28 Transmission Mode -- After a data transfer, this bit is set to indicate the mode of the data transfer (0 - ASCII, 1 - Binary).

27 End of Record -- When this bit and bit 31 are set, an end of record was encountered.

26 Not used.

25 Data Error -- When this bit and bit 31 are set, a data error was encountered.

24 Hardware Error -- When this bit and bit 31 are set, a hardware error condition was encountered.

23 Not used.

22 Not used.

21 Not used.

Local CRT

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

21-30 Not used -- The status information is contained in the Peripheral Information Table (PIT).

BCLA/MUX

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

21-30 Not used. The status information is contained in the PIT.

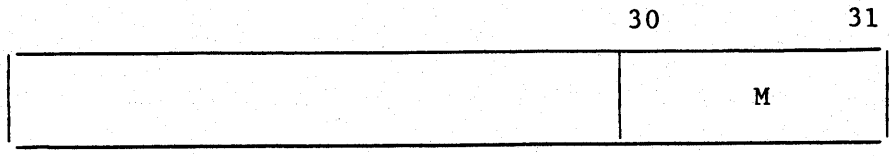
If no error conditions are set, then the Expanded Status will contain the number of bytes not transferred; otherwise the Expanded Status will contain further delineation of the error condition.

31 Reject -- Always set when an error occurs. When no other error bits are set and this bit is set, the device is busy.

21-30 Not used. The status information is contained in the PIT.

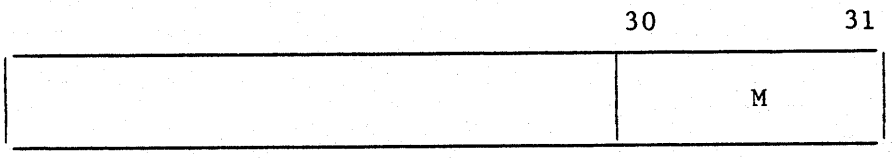
If no error conditions are set, then the Expanded Status will contain the number of bytes not transferred; otherwise the Expanded Status will contain further delineation of the error condition.

The MODE parameter in the SELECT ESR is defined for the 1829 card reader as follows:



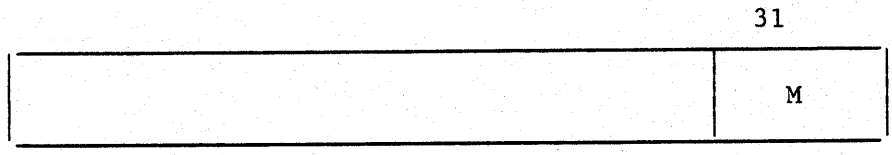
M = 0, Select 026 Hollerith conversion mode
= 1, Select 029 Hollerith conversion mode
= 2, Select ASD Hollerith conversion mode

The MODE parameter in the SELECT ESR is defined for the 1827 line printer as follows:



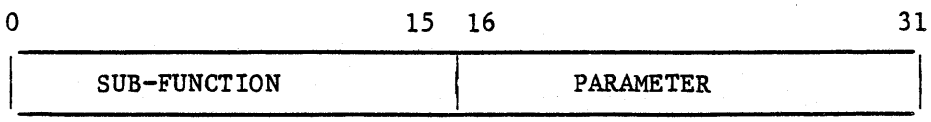
M = 0, Select 96 ASCII character set
= 1, Select 64 ASCII character set or fold 96 into 64 ASCII character set

The MODE parameter in the SELECT ESR is defined for the 1860-4 magnetic tape as follows:



M = 0, Select 1600 B.P.I. recording density.
= 1, Select 800 B.P.I. recording density.
= 2, Select 556 B.P.I. recording density.
= 3, Select 200 B.P.I. recording density.

The CODE parameter in the FUNC ESR is defined for the Local CRT, BCLA, and MPCLA as follows:



Sub-function

0 Define Peripheral Interface Table (PIT), the parameter specifies the first word address of the PIT.

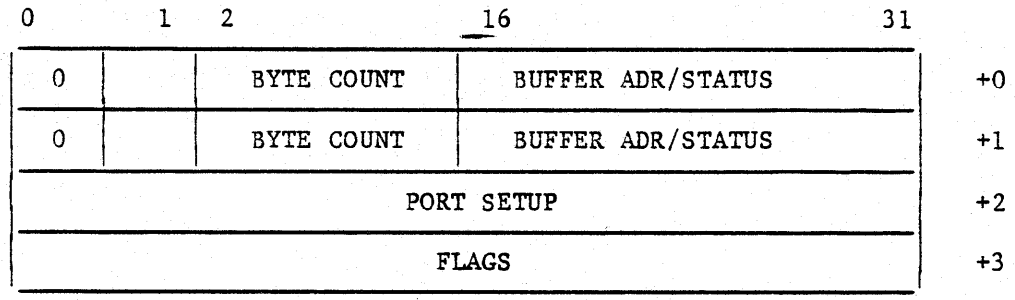
1 Read address in PIT, the parameter specifies the port.

2 Write buffer address in PIT, the parameter specifies the port.

3 Port Setup in PIT, the parameter specifies the port.

4 Clear a port, the parameter specifies the port.

The Peripheral Interface Table (PIT) is used to communicated port (unit) information to the device manager. The PIT contains an entry for each port or unit and an entry has the following format:



Word	Bits	Definition
0	0	Control bit (0=system, 1-device manager).
	2-13	Number of bytes not used in received buffer.
	14-31	Receive buffer first byte address.
	16-31	Status (see below).
1	0	Control bit (0=system, 1=device manager).
	2-13	Number of bytes not transmitted.
	14-31	Transmit buffer first byte address.
	15-31	Status (see below).
2		Port/Unit setup information (see below).
3		Flags usable by Device Manager.

The status field in the PIT has the following definitions for the associated device:

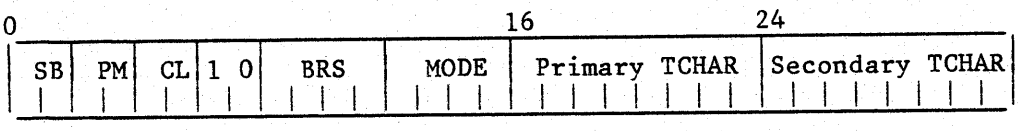
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
LOCAL CRT							B	L	H	P		C			R	E
BCLA						F	B	L	H	P		C			R	E
MPCLA						F	B	L	H	P	X	C			R	E

Bit Definition

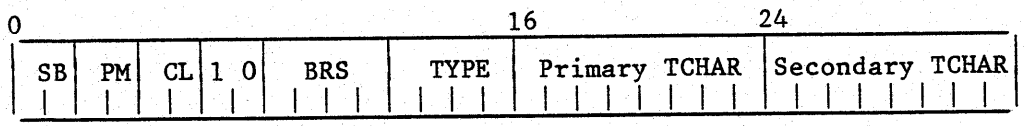
21 Framing error.
22 Break detected.
23 Overrun/Underrun.
24 Hardware Error.
25 Parity Error.
26 Checksum error.
27 Carrier not on.
30 Data set not ready.
31 Error flag.

The port setup field in the PIT has the following definitions for the associated device:

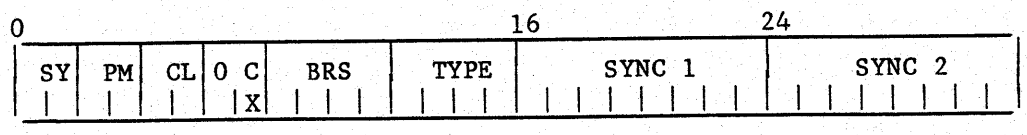
BCLA Port Setup (Asynchronous)



MPCLA Port Setup (Asynchronous)



MPCLA Port Setup (Synchronous)



SB - Stop bits

00 - invalid
01 - One stop bit
10 - 1.5 Stop bits
11 - 2 Stop bits

PM - Parity Mode

00 - No Parity
01 - Odd Parity
10 - No Parity
11 - Even Parity

CL - Character Length

00 - 5 bits
10 - 6 bits
01 - 7 bits
11 - 8 bits

SY - Sync type

00 - One Sync character
01 - Two Sync characters
10 - Reserved
11 - External Sync

CX - clock control

0 - internal clock generator
1 - external clock signal

BRS - Baud Rate Select (CX=0)

	Async	Sync
0000	50 BPS	800 BPS
0001	75	1200
0010	110	1760
0011	134.5	2152
0100	150	2400
0101	300	4800
0110	600	9600
0111	1200	19200
1000	1800	24743.0
1001	2000	31916.8
1010	2400	38400
1011	3600	57825.8
1100	4800	76800
1101	7200	114306
1110	9600	153600
1111	19200	307200

MODE

XXX0 - Echoplex off (half)
XXX1 - Echoplex on (full)

TYPE

0 - Debug mode
1 - Diagnostic mode (transparent)
2 - AWN
3 - NEDN
4 - Mode 4A
5 - NMC
6 - ID-50
7-15 - Reserved

Primary TCHAR - primary termination character
Secondary TCHAR - secondary termination character

SYNC1 & SYNC2 - two possible SYNC characters that are used when SY = 0 or 1.

For AWN PROTOCOL Only:

SYNC1 FIELD:	01	Selects Receive SYNC pattern 7106
	02	Selects Receive SYNC pattern 0606
SYNC2 FIELD:	01	Selects Transmit SYNC pattern 7106
	02	Selects Transmit SYNC pattern 0606

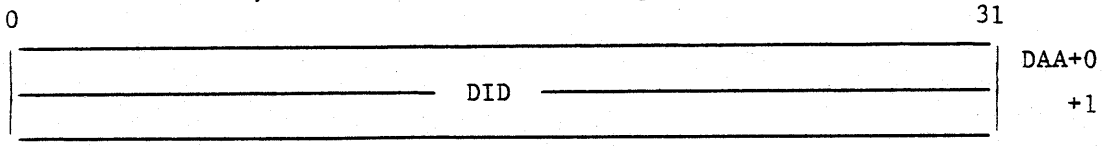
APPENDIX J

DEVICE ASSIGNMENT AREAS

There exist three DEVICE ASSIGNMENT AREA (DAA) descriptions, each to be used for the assignment of a different category of hardware types as follows:

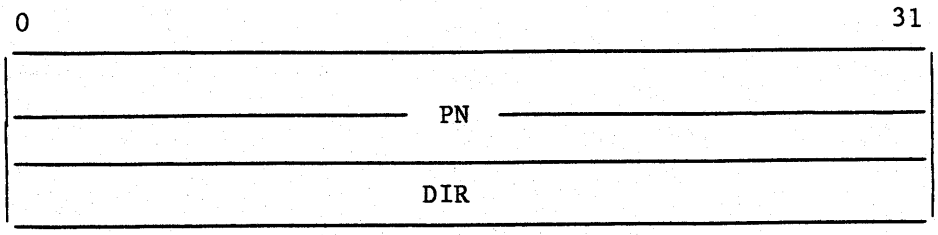
DAA1 - Unit Record
DAA2 - Data Pipe
DAA3 - Interactive Terminal

DAA1, Unit Record Device Assignment Area



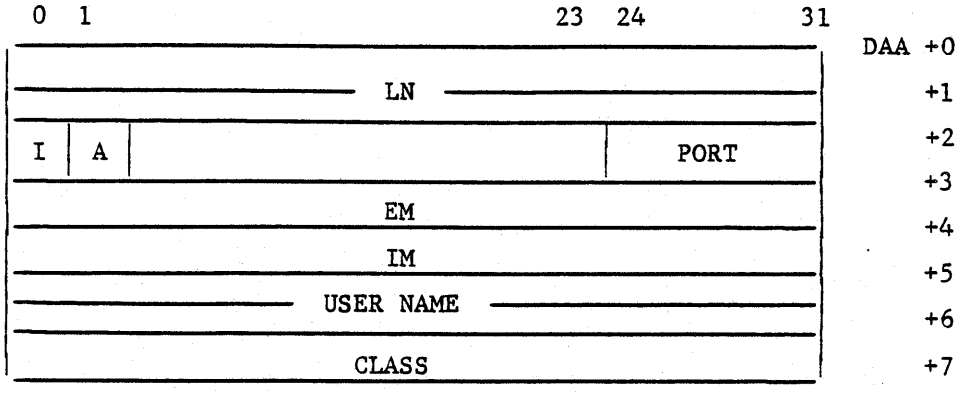
Parameter	Description
DID	Device Identification (ASCII) as given in appendix K Valid Hardware types. If DID is set to -1, the default device will be the first available device of the type specified.

DAA2, Data Pipe Device Assignment Area



Parameter	Description
PN	Pipe Name (ASCII, left justified, blank fill)
DIR	Data Direction 0 = Outbound 1 = Inbound

DAA3, Interactive Terminal Device Assignment Area



Parameter	Description
LN	Linkage name (ASCII, left justified, blank filled).
I	Immediate Connection Flag as follows: I=0, connect when requested by terminal I=1, immediate connection
A	Any Port Flag as follows: A=0, port selection desired A=1, any port; the PORT parameter is ignored.
PORT	Port number, a 16-bit field specifying a system port number 0-256. Legal port numbers are determined at System Installation. Port 0 is always the port number for the console on the master CPU.
EM	Exclusion Mask The Exclusion Mask (EM) limits the port being requested by excluding ports for which specified bits are set in the PORT and USER validation masks. The PORT and USER validation masks are ANDed to form a single validation mask (VM). If EM .AND. VM = 0, a port will be considered for assignment. A description of the PORT/USER Validation mask is given in the Interactive Terminal Subsystem User's Manual.
IM	Inclusion Mask The Inclusion Mask (IM) limits the port being requested by accepting only those ports for which specified bits are set in the PORT or USER validation mask. The PORT and USER validation masks are ANDed to form a single validation mask (VM). If IM .AND. .NOT. VM = 0, a port will be considered for

assignment. A description of the PORT/USER validation mask is given in the Interactive Terminal Subsystem User's Manual.

USER NAME — User Name, an 8 ASCII character identifier. An ASCII NUL character in any character position indicates that a match is not required in that position.

CLASS — Terminal Class Mask. The class mask limits port selection according to the following equation: .NOT. Class Mask .AND. 2**(CLASS-1) = 0. If a port fails this equation, it cannot be selected. Terminal classes are defined as follows:

Unmanned Classes:

1 Mode 4 Line (200UT)
2 X.25 Packet Network line
3 CPU-CPU Async Protocal (undefined)
4 Undefined
5 Undefined
6 Awn Network Line
7 Neds Network Line
8 NMC Network Line
9 ID50 Network Line
10 Undefined
11 200UT Card reader (RBTMGR)
12 200UT Line printer (RBTMGR)
13 Output only port (RO terminal)
14 Input only terminal (CR)
15 Undefined

Manned Classes:

16 Generic - Glass Teletype
17 Generic - Printing Terminal
18 CDC 751
19 CDC 752
20 CDC 756
21 Undefined
22 Undefined
23 200UT CONSOLE
24 X.25 Packet Assembly/Disassembly
25 Cyber Virtual Terminal

If Class Mask = 0, any class may be selected.

APPENDIX K

VALID HARDWARE TYPES

The valid hardware types which can be requested with the DEVICEQ ESR and the EQUIP control request or returned by the UTYP ESR are as follows:

Device Identification	Type	Description
* MEM	0	Memory
* DP	1	Disk
MT9	2	Nine Track Tape
* CR	3	Card Reader
* CP	4	Card Punch
* LP	5	Line Printer
* PR	5	Line Printer
* CRT	6	Keyboard Display
TT	7	Teletype
CT	8	Cartridge Tape
PLT	9	Plotter
FDD	10	Flexible Disc Drive
CCC	11	CYBER Coupler
MT7	12	7-Track Tape
IT	13	Interactive Terminal
* RBT	14	Remote Batch Terminal
CN	15	Communication Network
PI	16	Data Pipe
* MUX	17	BCLA/MUX
* SMX	18	MPCLA
* OPF	19	OPF Pseudo device

* These hardware types are available only to Operating System Tasks; they cannot be assigned to a User Task.

APPENDIX L

ESR AND DEVICE CROSS REFERENCE CHART

The following chart cross references ESRs and devices; ESRs not in the chart are legal on all devices.

	FORMAT	READLU	READDS	WRITLU	WRILDS	BKSP	ERASE	FUNC	REWD	SELECT	SEOF	ULOC	UNLD	WEOF	DIAG
Logical Devices															
Dummy	N	N	N	N	N	N	N	N	N	N	N	N	N	N	N
Data Pipes	I					I	I		I		I		I		N
Files	I		I		I		I	I		I			I		N
Interactive	I		I		I	I	I	I	I		I	I	I	I	N
Com. Network	I		I		I	I	I	I	I		I	I	I	I	N
Physical Devices															
1867 SMD/MMD						I	I	I	I	I	I		I	I	N
1833-5 FDD			I		I		I	I		I	I		I		N
1829 CARD READER	I		I	I	I	N	I	I	N		I	I	I	I	N
1827 PRINTER	I	I	I		I	N	I	I	N		I	I	I		N
1860-4 MAG TAPE	I		I		I			I		N		I			N
2558-3 COUPLER	I						I				I		I		N
LOCAL CRT	I	I	I	I	I	I	I		I	I	I	I	I	I	N
BCLA/MUX	I	I	I	I	I	I	I		I	I	I	I	I	I	N
MPCLA	I	I		I		I	I		I		I		I	I	N

I = ILLEGAL   N = NULL   BLANK = LEGAL

COMMENT SHEET

TITLE: MP-60 MPX/OS

Version 3 Reference Manual

PUBLICATION NUMBER: 17329125 REVISION: A

NAME:

COMPANY:

STREET ADDRESS:

CITY: STATE: ZIP CODE:

Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

NO POSTAGE STAMP NECESSARY IF MAILED IN U.S.A.

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Systems Technology Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

FOLD

FOLD