17329110

CONTROL DATA CORPORATION

# MP-60 MPX/OS REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

# MPX/OS CONTROL CARD INDEX

17329110

CONTROL DATA
CORPORATION

# MP-60
# MPX/OS
# REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

# REVISION RECORD

| REVISION | DESCRIPTION |
|---|---|
| A | Original Release. |
| (78-02-24) | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| | |
| Publication No.<br>17329110 | |

REVISION LETTERS I, O, Q AND X ARE NOT USED

Address comments concerning this manual to:

CONTROL DATA CORPORATION

215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

or use Comment Sheet in the back of this manual

# PREFACE

This document describes CONTROL DATA® MP-60 Computer Systems in general, and the CONTROL DATA® MPX Operating System (MPX/OS) specifically.

Although this document contains sufficient information to be studied independently, the following documents are required to achieve a thorough understanding of total MP-60 systems:

| CDC Publication No. | Title |
|---|---|
| 41618300 | MP-60 Emulation Reference Manual |
| 14061300 | MP-60 Computer System COMPASS Reference Manual |
| 14061100 | MP-60 Computer System FORTRAN Reference Manual |
| 14062100 | MP-60 Computer System COSY Reference Manual |
| 14062200 | MP-60 Computer System PRELIB Reference Manual |
| 14063800 | MP-60 Computer System Utility Reference Manual |
| 14063900 | MP-60 Computer System Peripheral Equipment Reference Manual |
| 17328900 | MASS/MPSIM Reference Manual |

# CONTENTS

Section Page

# CONTENTS (CONT.)

# CONTENTS (CONT.)

Section Page

# CONTENTS (CONT.)

Section Page

# CONTENTS (CONT.)

# CONTENTS (CONT.)

## FIGURES

## TABLES

# CONTENTS (CONT.)

## APPENDICES

# INTRODUCTION 1

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

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

The microprogrammable processor (MP32) 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-60 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.

## CONFIGURATIONS

Table 1-1 presents the minimum and maximum configurations supported by MPX/OS for single- and multiple-processor systems. Figure 1-1 illustrates a two-slave multiprocessor configuration as an example.

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

| | Single Processor Configurations | | Multiple Processor Configurations | |
|---|---|---|---|---|
| | Minimum | Maximum | Minimum | Maximum |
| Program states | 3 | 6 | 6 | 48 |
| Main memory | 388K Bytes<br>96K Words | 16768 Bytes<br>4096K Words | 524K Bytes<br>131K Words | 16768 Bytes<br>4096K Words |
| Card reader | 1 | 1 | 1 | 1 |
| Line printer | 1 | 1 | 1 | 1 |
| Display console | 1 | 1 | 1 | 1 |
| Display station | 0 | 6* | 0 | 6* |
| Mass storage | 10M Bytes | 720M Bytes* | 20M Bytes | 720M Bytes* |
| Magnetic tape | 1 | 8* | 1 | 8* |
| Card punch | 0 | 1 | 0 | 1 |
| Teletype | 0 | 2* | 0 | 2* |
| Number of processors | 1 | 1 | 1 | 5 |

*These values represent typical limits.

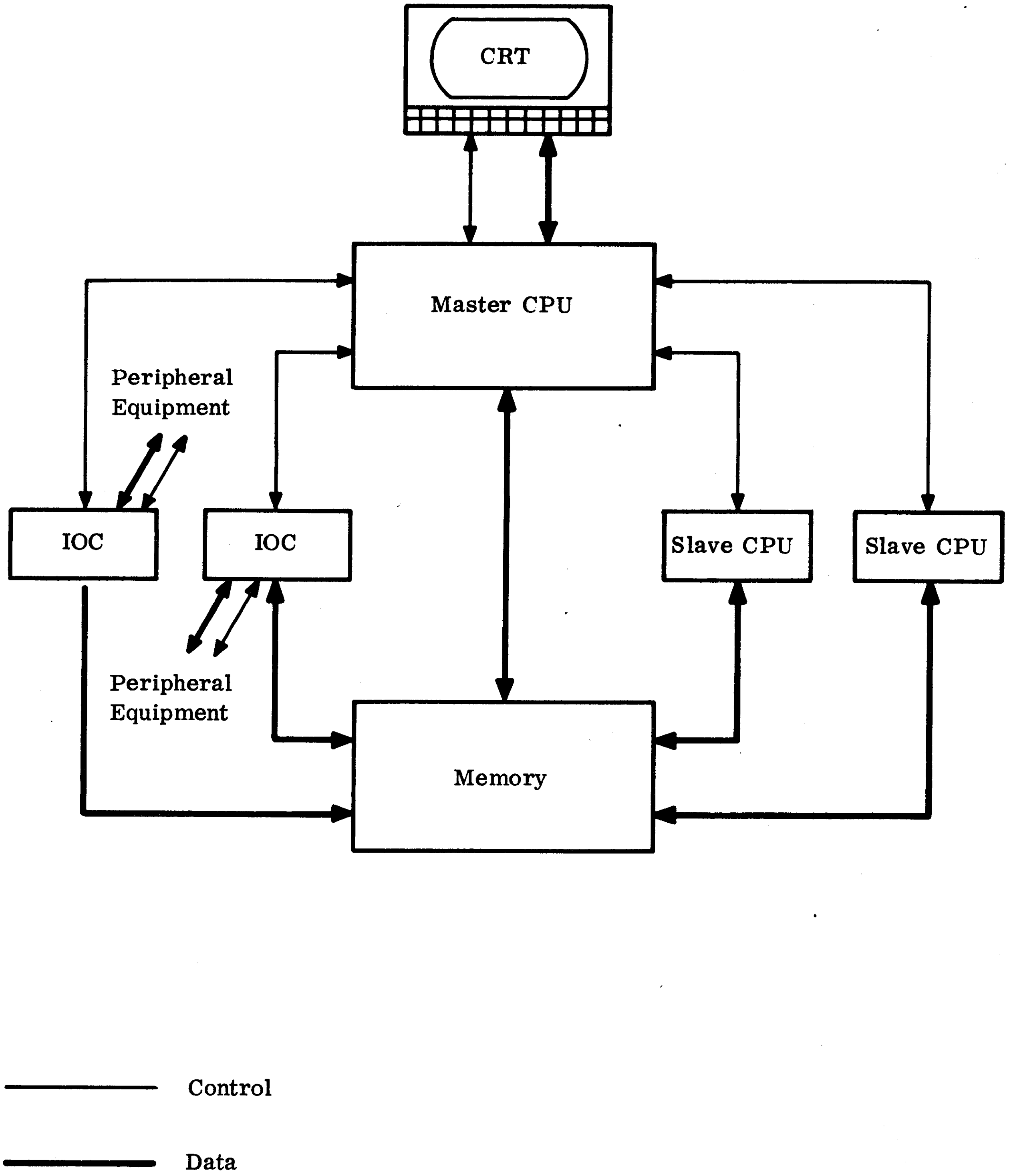


Figure 1-1. A Multiprocessor Configuration

## MAIN MEMORY

The main memory of the MP-60 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-60 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, has all connections to the IOCs, 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, divide, 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 3, 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 used as synonyms for state 0. State 0 on each CPU provides the CPU interrupt recognition and interrupt service processing. State 0 on the master CPU additionally provides system resource management, console CRT management, job accounting, and real-time executive processing.

•MPX/OS uses states 1, 2, 3, 5, 6, and 7 to execute nonsystem code. These states, together with state 4, are referred to as program states. Execution in these seven states is referred to as program mode or task mode execution. State 4 is used to support execution of most system tasks.

## TABLE 1-2. INTERRUPT PRIORITY

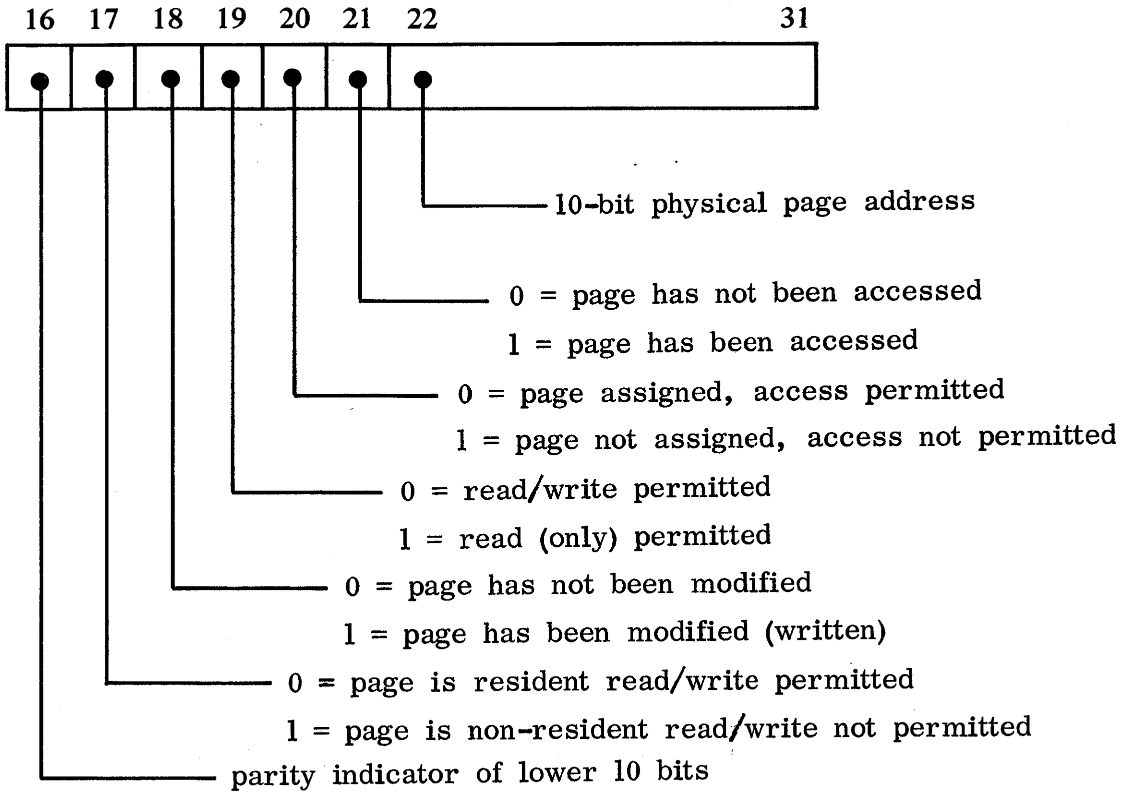
| LEVEL | INTERRUPT GROUPS |
|---|---|
| 1 | Power Failure |
| 2 | CPU Memory Errors |
| 3 | DMA Memory Errors |
| 4 | Illegal Instruction |
| 5-12 | Micro I/O 0-8 |
| 13-28 | Macro I/O 0-15 |
| 29-44 | Real-Time 0-15 |
| 45-50 | Open |
| 51 | Open |
| 52 | Clock Interval |
| 53-55 | Open |
| 56 | Inter-Processor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


16
17
18
19
20
21
22
31
10-bit physical page address
0 = page has not been accessed
1 = page has been accessed
0 = page assigned, access permitted
1 = page not assigned, access not permitted
0 = read/write permitted
1 = read (only) permitted
0 = page has not been modified
1 = page has been modified (written)
0 = page is resident read/write permitted
1 = page is non-resident read/write not permitted
parity indicator of lower 10 bits

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.

## INPUT/OUTPUT CONTROLLERS

Input/output controllers (IOCs) are logically connected to the master CPU, to the main memory, and to the peripheral devices they control. The master CPU invokes the IOC to request status, to initiate I/O data transfers, or to initiate device functions. The IOC interrupts the master CPU to signal the completion of a requested service.

The IOCs are not operating system components in the same sense that MPX/OS coded modules are. They do, however, provide services traditionally provided by CPU code modules. IOCs allow MPX/OS to program I/O operations at a high level, providing a level of device independence, error detection and recovery functions, and I/O functions parallel with normal system operation.

The IOCs accept and process one or more CPU requests, maintaining a list of requests in their own memory. When the list becomes full, additional CPU requests are rejected. In addition to accepting multiple requests, the IOCs return request status to CPU-designated areas of main memory. These features allow MPX/OS to maximize system effectiveness within acceptable overhead limits. MPX/OS sends requests to the IOC until the IOC cannot accept new requests. Rejected requests are saved and reissued one at a time as the IOC completes processing accepted requests. Since the status data is placed at request-designated areas of main memory, minimal system overhead is required to save status information and process the interrupt to a point where additional interrupts can be accepted. By maintaining a list of requests in the IOCs, the peripheral devices are not left unused while the CPU processes interrupts, ensuring minimal system delays.

## MPX/OS OPERATING SYSTEM OVERVIEW

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.

## MPX/OS CONCEPTS

The following sections describe the significant design philosophies adopted for MPX/OS. These descriptions cut across the physical/functional organization of the operating system code.

### 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. The card deck is processed by an operating system task, the job manager. The job manager interprets the control statements in the card deck 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 total number of JCTs is an installation-controlled parameter. The contents of a JCT include the following:

- Job identification (from *JOB or *RJOB control card).
- 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.
- Standard file disposition.

### 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 executive or task code. The executive executes on demand, either from user-issued 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 the core, on the library, or in a data set accessible to the user's job. Core 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 3, RETURN, Terminate Task Execution).

Tasks that reside in the system library data set 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 2, Library Task Statement).

Tasks that reside in data sets accessible to and maintained by the user are referred to as user tasks. User tasks are brought into execution by *LOAD and *RUN control cards, or by a CALL ESR from within an executing task, by specifying the data set(s) containing the task.

Library tasks and user tasks may be maintained in two forms: relocatable and absolute binary. The relocatable binary form normally originates from an 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 duration of the task. That is, when the task returns to the system after completing execution, the identifier continues to exist if the RETURN ESR specifies retention in memory. It does not continue to exist if the RETURN does not specify retention in memory.

Library tasks are assigned their library names as task identifiers when placed into execution. Library tasks specify release of memory in their RETURN ESRs.

## 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, that are transformed into physical addresses through the MP-60 paging hardware.

Figure 1-2 illustrates the important features of loaded tasks. Logical page 15 of each task in a job is the same physical page of memory. Logical page 15, the intrajob data area, is used to support MPX/OS functions provided as a part of each task (through the blocker/deblocker library subroutines) and functions provided by the job manager system task.

Logical page 14 contains an area reserved for communications between the system and tasks and, on CALL and RETURN ESRs, between the tasks. (The job manager task communicates through an area of job manager tables in logical page 15.) The communication area (PARM) is 50 words* in length. 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, and with blank and numbered common blocks from logical page 0 upward. If some pages are left unused after loading is complete, the corresponding page registers are protected 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.

*An installation option.

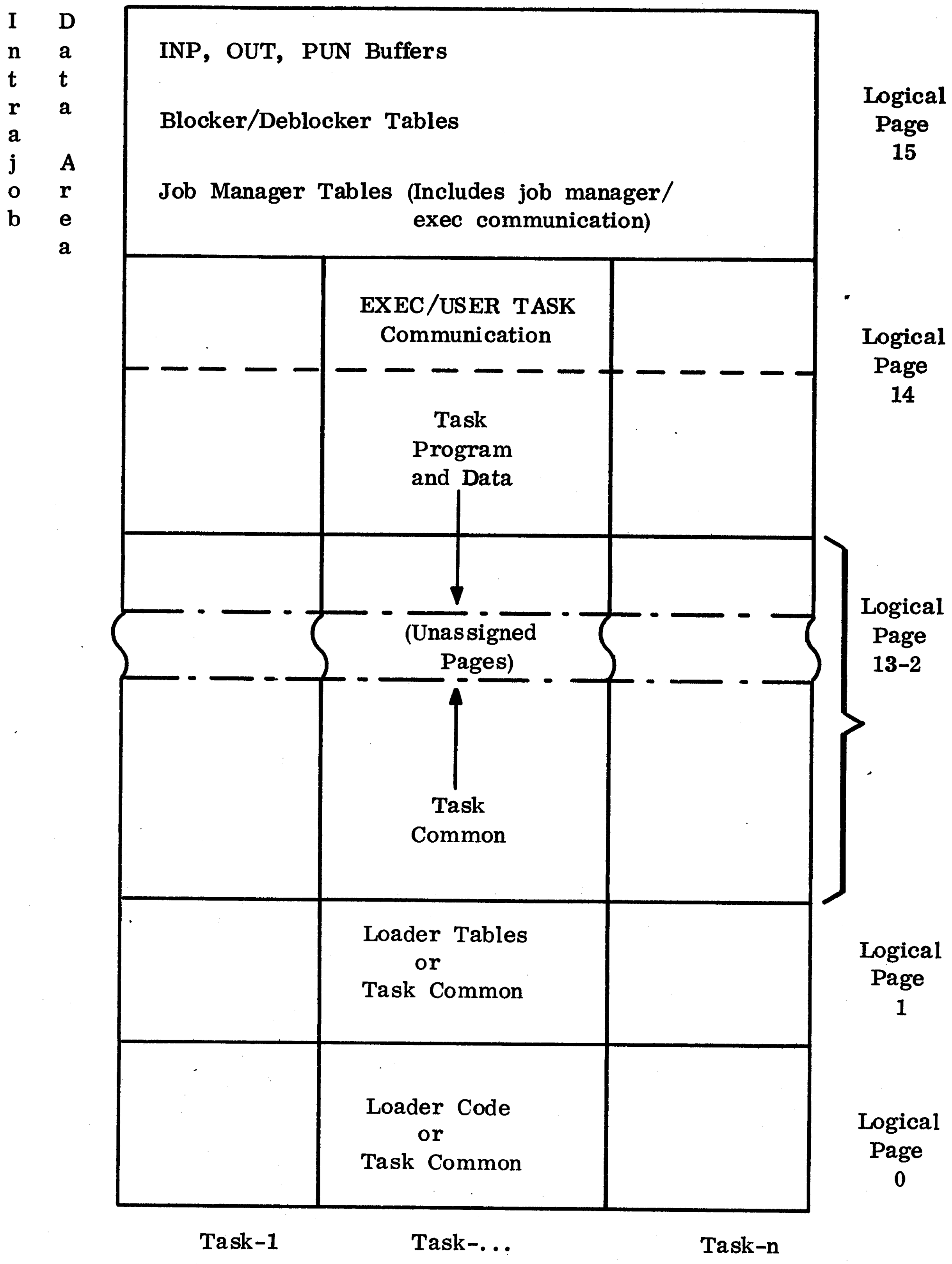


Figure 1-2. Task Memory Layout

### 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 total number of TCTs per job is an installation-controlled parameter. The contents of each TCT include the following:

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

The TCT is the task identification 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. The maximum number of concurrently established tasks within a single job is controlled through an installation parameter at system build time.

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

| Status | Description |
|---|---|
| Dormant | The task has completed its work and returned, without release, to its caller. Status remains dormant until the task is called again. |
| Active, Ready | The task is currently executing or ready to resume execution. |
| 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. |
| 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 manager. |
| Call Wait | The task has called another task. Until the call can be connected, the caller remains in call status; it cannot resume execution. |
| Callee Wait | The task has 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. |
| 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. |
| Finis | The task has returned but has outstanding callees. |
| TSCHED Wait | The task issued a TSCHED request. After the specified time interval has elapsed, the task status will be set to active. |
| CRT Wait | The task is waiting for the operator to respond to a pending message. Task becomes ready after the operator responds. |
| TCT Wait | The task has attempted to call another task and the call cannot be connected due to an insufficiency of TCTs. |

MPX/OS provides a multiprocessing capability, which means that the system services are 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 provides 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, IOC, 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 255 (highest) to 0 (lowest) with ranges 240 through 255 and 1 through 9 reserved for real-time and system tasks. 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 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 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.

TABLE 1-4. TASK PRIORITY ASSIGNMENTS

| Priority Use | Priority Range | Tasks |
|---|---|---|
| | 0 | Idle |
| Low priority real-time and system tasks | 1<br>9 | Real-time |
| Real-time and system and nonreal-time tasks | 10<br>239 | Job Manager<br>Nonreal-time<br>Real-time |
| High priority real-time and system tasks | 240<br>255 | BKI<br>BKO<br>BKP<br>File manager<br>Real-time |

Unit record devices are accessible to the user through the *SCHED and *EQUIP control statements in the batch job control statement card deck. The *SCHED control statement reserves the device for the job, and the *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 may 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) may 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 may 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.
- Performing selected executive functions in program state 4.
- Servicing I/O requests by priority.
- Dispatching tasks for execution by priority.

A real-time environment consists of a real-time executive, real-time interrupt processors, and real-time tasks. The real-time executive shares monitor state with the MPX/OS executive. A set of monitor-state machine registers is reserved for real-time executive use when responding to real-time interrupts.

Real-time tasks are established in program states and communicate with the real-time executive via the normal MPX/OS ESRs mechanism. Real-time ESRs are routed to the real-time executive for processing. 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) through the normal standard input unit. The real-time job must load real-time tasks. Execution of loaded real-time tasks includes initializing calls to the real-time executive to establish interrupt linkage. Following initialization steps, the real-time tasks remain in memory in a dormant state until activated by the real-time executive in response to real-time interrupts, or until called by another real-time task. Real-time activity is sustained through task calls, time scheduling, real-time interrupts, and any variety of combinations of the above. The bulk of real-time processing is performed as tasks scheduled by the real-time executive. Real-time tasks can be executed at high-system reserved priority.

The real-time executive is integrated into the MPX/OS resident system by defining real-time ESRs and their necessary processing codes. In this fashion, the real-time executive becomes an integral part of the MPX/OS operating system.

## 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 facilitates prioritized delivery of services.

Figures 1-3 and 1-4 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. Among the operating system initialization activities are the scheduling of two system tasks: IDLE and BKI. The IDLE system task is scheduled for all CPUs, and BKI is scheduled for the master CPU only.

Slave startup accomplishes CPU initialization (firmware loading) and slave identity definition and awaits the signal to start normal activity (Section 3, 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 higher-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 BKI system task to be placed into execution since its priority exceeds that of the IDLE system task (see Table 1-4).

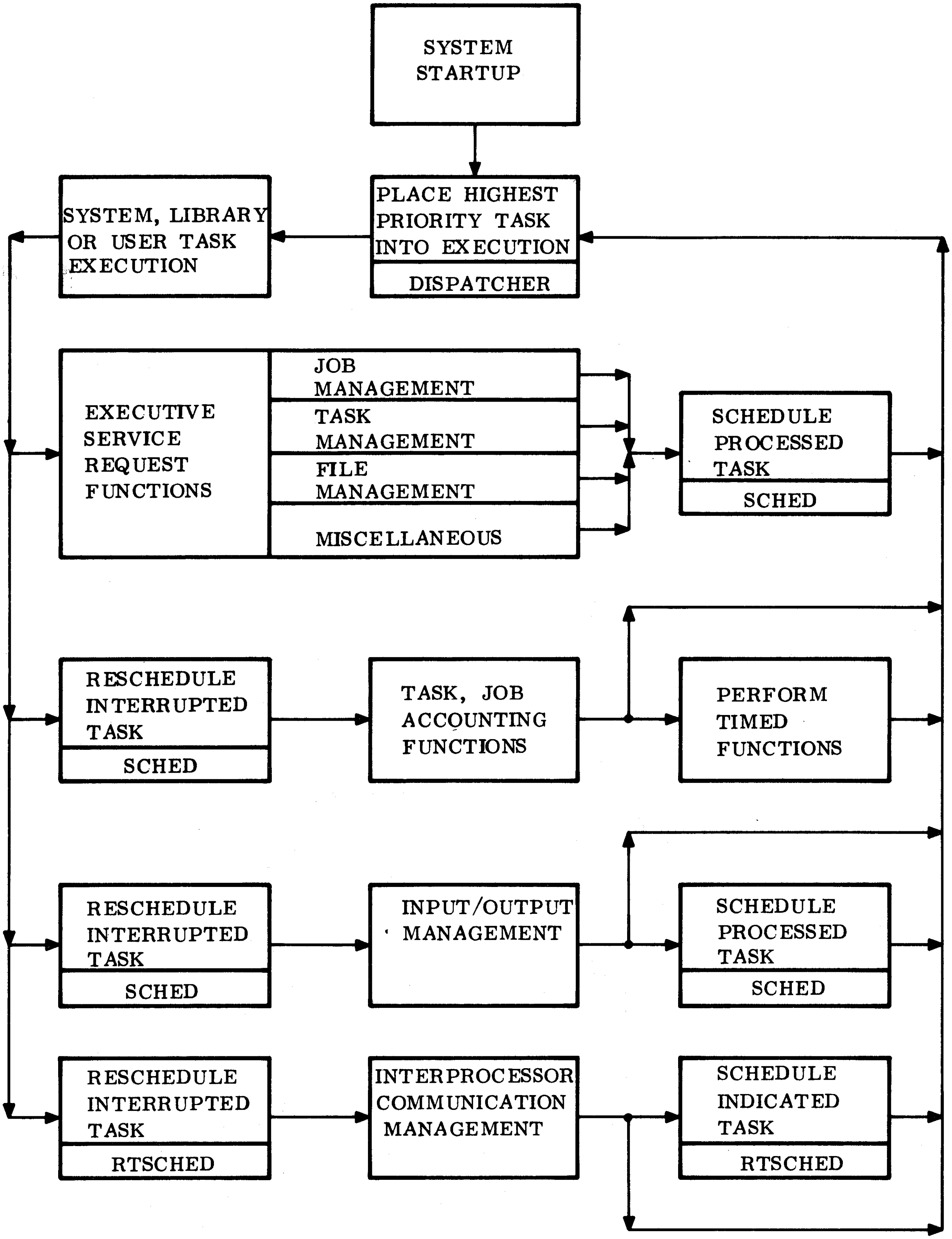

SYSTEM STARTUP
PLACE HIGHEST PRIORITY TASK INTO EXECUTION
DISPATCHER
SYSTEM, LIBRARY OR USER TASK EXECUTION
EXECUTIVE SERVICE REQUEST FUNCTIONS
JOB MANAGEMENT
TASK MANAGEMENT
FILE MANAGEMENT
MISCELLANEOUS
SCHEDULE PROCESSED TASK
SCHED
RESCHEDULE INTERRUPTED TASK
SCHED
TASK, JOB ACCOUNTING FUNCTIONS
PERFORM TIMED FUNCTIONS
RESCHEDULE INTERRUPTED TASK
SCHED
INPUT/OUTPUT MANAGEMENT
SCHEDULE PROCESSED TASK
SCHED
RESCHEDULE INTERRUPTED TASK
RTSCHED
INTERPROCESSOR COMMUNICATION MANAGEMENT
SCHEDULE INDICATED TASK
RTSCHED

Figure 1-3. Normal System Flow (Master Processor)

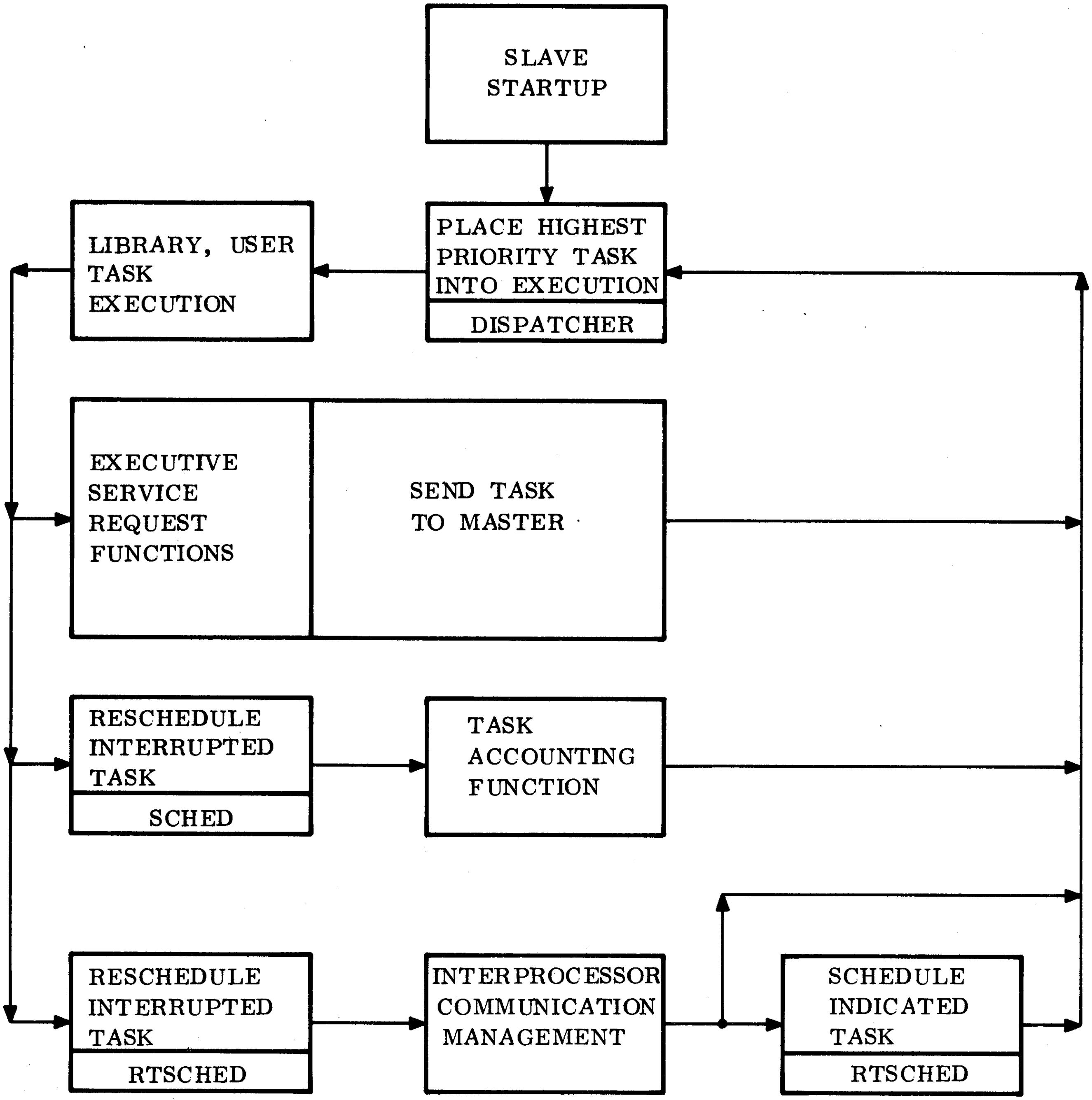

SLAVE STARTUP
PLACE HIGHEST PRIORITY TASK INTO EXECUTION
DISPATCHER
LIBRARY, USER TASK EXECUTION
EXECUTIVE SERVICE REQUEST FUNCTIONS
SEND TASK TO MASTER
RESCHEDULE INTERRUPTED TASK
SCHED
TASK ACCOUNTING FUNCTION
RESCHEDULE INTERRUPTED TASK
RTSCHED
INTERPROCESSOR COMMUNICATION MANAGEMENT
SCHEDULE INDICATED TASK
RTSCHED

Figure 1-4. 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 signaling progress on services underway in IOCs or in other CPUs, or signaling a time interval lapse which may allow a task to be scheduled for execution.

## TASK EXECUTION

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 arises 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 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 are maintained by the blocker/deblocker modules in logical page 15 of the job tasks. The same physical page is logical page 15 of 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 (BKI system task), identifying and initiating job requested work (job manager system task), and returning the job output listings and punched cards to the user (BKO and BKP system tasks).

## INPUT (BKI) SYSTEM TASK

BKI obtains job control decks from the card reader and establishes the job manager system task. When the card reader is empty, a message is sent to the operator. When the operator responds to the message, BKI is activated and again processes card reader data.

When a deck becomes available, BKI reads the deck and saves it in a file. The *JOB or *RJOB control statements and any *SCHED statements are read and interpreted for system resource requirements. The content of these two control cards form the initial content of the JCT entry created for the job.

BKI attempts to fill the resource requirements of the job. If the resources are not available, BKI assumes a dormant status and remains inactive until the necessary resources become available. When all resources are reserved for the job, BKI establishes the job manager and returns to the card reader to process/await additional input.

BKI implements the job management philosophy as described, which influences the level of system activity. MPX/OS does not oversubscribe any system resource except program states. This means that once a job starts into execution, it cannot be delayed due to insufficient resources. Introduction of more tasks than there are program states to directly support them simply causes state swapping, an activity which increases system overhead but which does not prevent task execution.

## 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 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 and PUN to the BKO and BKP system tasks for post processing.

*The externally observable features of the job manager are the subject of Section 2 and are not described here.

## OUTPUT AND PUNCH (BKO AND BKP) SYSTEM TASKS

BKO prints the OUT file. BKP punches the PUN file. These two post processors are completely independent of each other but are treated alike by MPX/OS. Once activated, the post processor processes all files of the specified type that exist, and goes dormant. They are activated by the job terminator when necessary, if not already active.

## 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, MODIFY, OPEN, and RELEASE functions. The servicing of such requests may involve disk reads/writes and task queueing, which result in unpredictable 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 3, 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 core 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 and controls usage of the IOCs. The I/O manager accepts the data set number from the ESRs, determines the device and/or file and/or IOC access conflicts, 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 3, UST, Unit Status Test).

## INTERPROCESSOR COMMUNICATION MANAGEMENT

Figure 1-5 illustrates the flow of a service request originating on the master CPU and on a slave CPU. Requests that originate on a slave CPU (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.

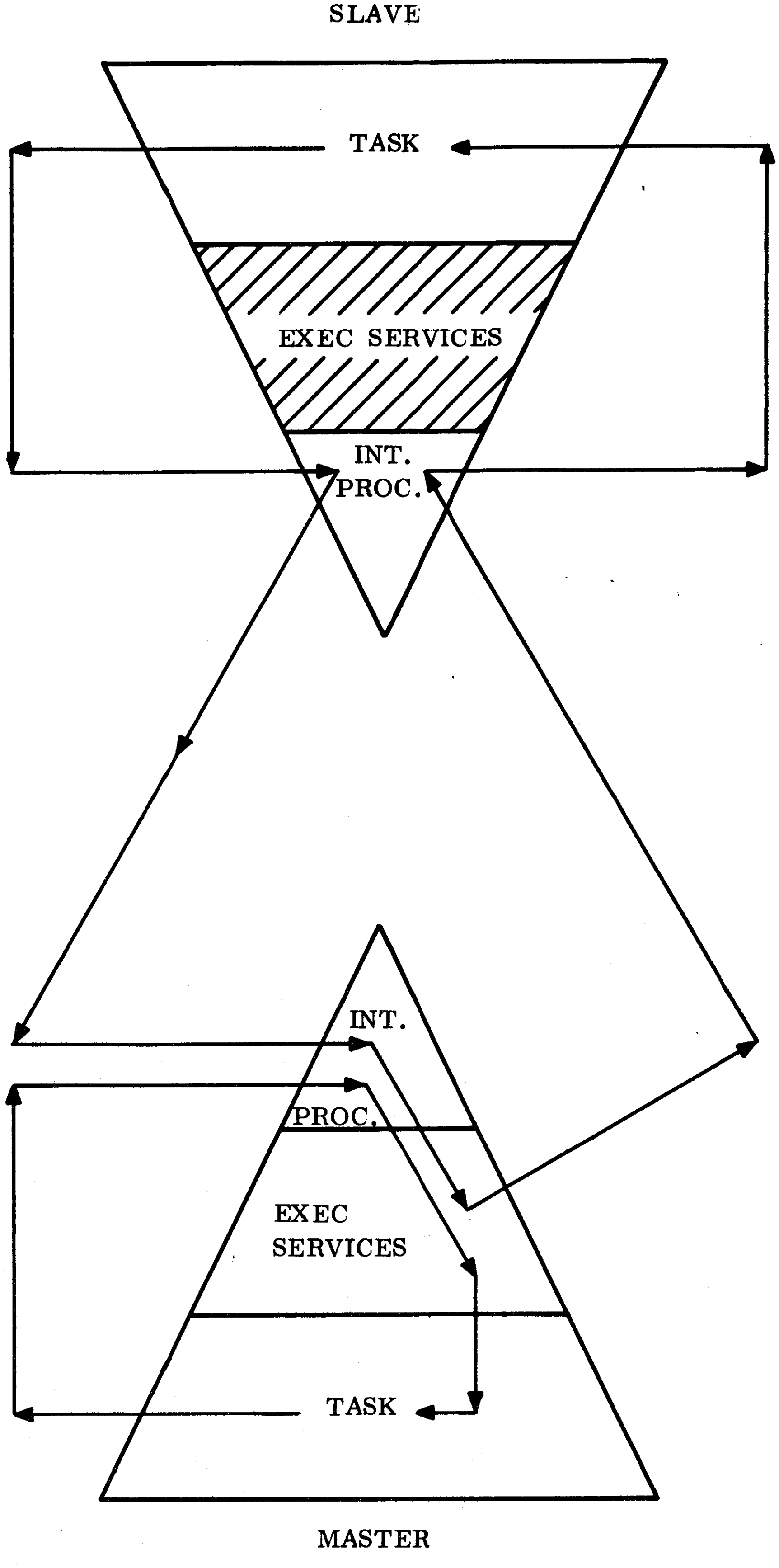


Figure 1-5. ESR Processing Flow

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 (waiting for file access, for example).

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.

## 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.

# 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 or by modifying the system executive (EXEC).

## DEVICE LABELS

MPX/OS uses device labels to identify all mass storage devices. Each removable disk pack, as well as each drum and nonremovable disk, has a device label written on its first hardware address. Device labels are written by the MPX/OS utility routine (*FMP). prior to using the device.

Device labels contain information pertaining to mass storage devices, including a 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 monitor calls to the MPX/OS system task ALLOCATE to create a file label. These calls provide the file identification, access code, block size, block count, etc. 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 maintains a threaded map of segments and inserts it in the file label. MPX/OS allows files to contain up to 32 segments. One or more segments of a file may 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 may specify that the space be contiguously allocated. If insufficient contiguous space is available on the device, ALLOCATE rejects the request.

## FILE ALLOCATION METHOD

ALLOCATE 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 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 on.

# JOB PROCESSING 3

Job processing flow is illustrated by Figure 3-1. The processing of a job is initiated by loading the job deck into the card reader (standard input unit). From this point on, MPX/OS assumes control of the job.

The operating system task, BKI, reads a batch job card deck and places it in a mass storage file (job input file). BKI examines the job (both real- and nonreal-time) and the schedule statements, determines the resource requirements for the job, and attempts to secure the necessary resources. When all of the resources have been acquired, BKI establishes the job in the system and initiates execution of the job manager system task (JMTR). BKI is now free to process any new jobs that might appear at the card reader.

The job manager system task reads and interprets the job control statements in the sequence they appear in the batch job deck (now 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 (TSKMGR and TASK1 ... TASKn), and for job termination (JTRM).

Control cards contain an asterisk (*) in column 1, followed by the requested function name. The parameter list extends through the remaining columns of the card. One additional card may be used to further extend the parameter list, if necessary. The additional card does not contain the asterisk in column 1, but continues the parameter list from the preceding card. The parameter list is enclosed in parentheses with commas separating each parameter. Comments are permitted on the control cards, but must follow the corresponding parentheses that terminate the parameter list.

The batch job card deck must be organized in three sections: job definition, job activity, and job termination. Figure 3-2 defines a batch job card deck 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. The job termination section releases resources assigned to the job, adds job accounting information to the OUT file, eliminates the job from the system, and initiates processing by the BKO and BKP system tasks. (BKO prints the OUT file. BKP punches the PUN file.)

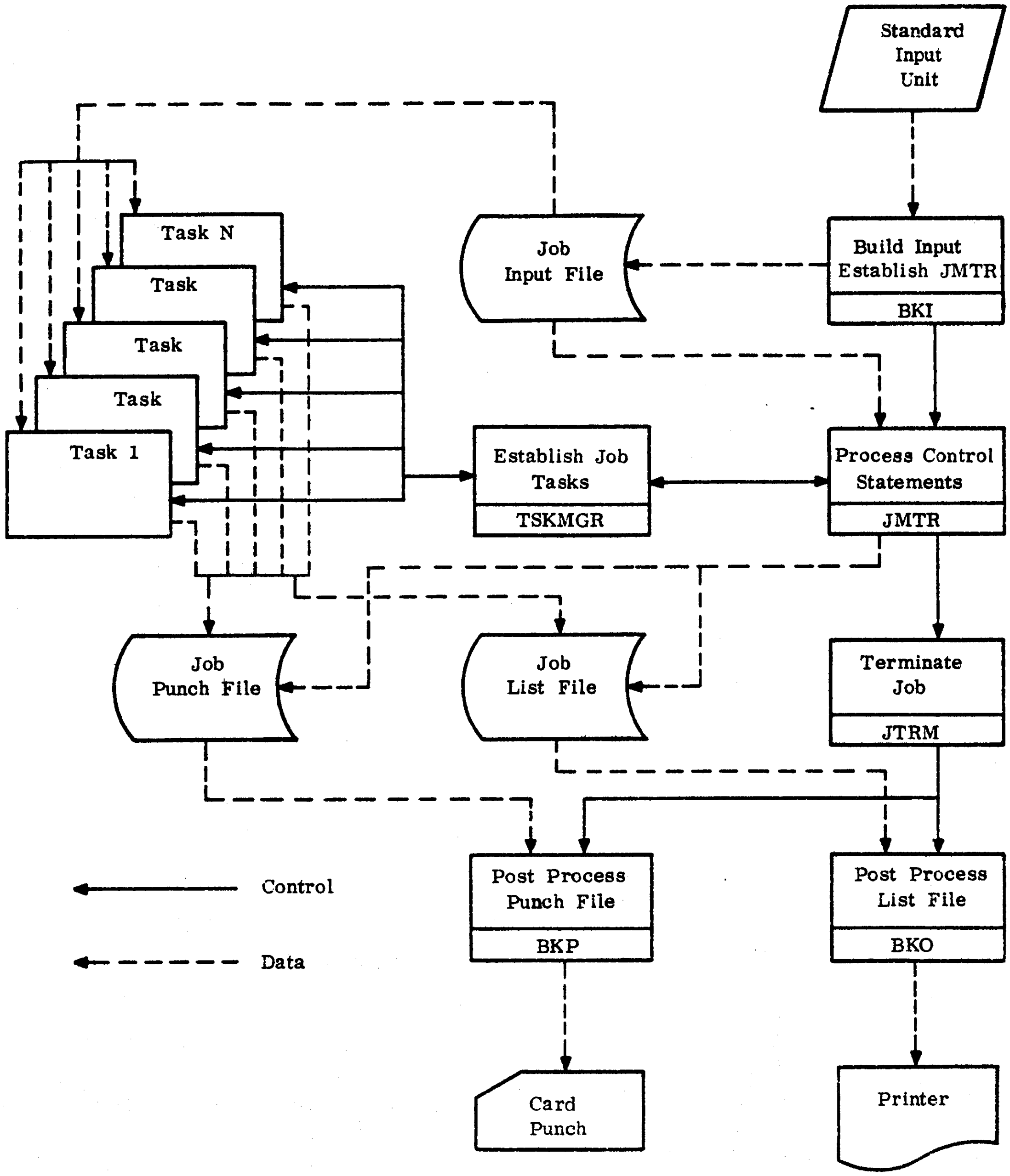


Figure 3-1. Job Processing Flow

Job Definition Section

```
*JOB(ID=EXAMPLE,AC=1234)
*SCHED(CM=12,MT=1)
*SCHED(9427=SCRATCH1)
```

Job Activity Section

```
*EQUIP(1=MT)
*ALLOCATE(FN,OWNR,01,Q00Q,480,50,,RW,3,SCRATCH1)
*OPEN(2,FN,OWNR,01,Q00Q,W)
*FTN(I, L, X)
              PROGRAM RW
              INTEGER CARD (20)
10            READ (1,99) CARD
              WRITE (2,99) CARD
              IF (CARD (1).EQ.4H*END) STOP
              GO TO 10
99            FORMAT (20A4)
              END
              FINIS
*LOAD(57)
*RUN
```

Job Termination Section

```
*EOJ
```

Figure 3-2. Batch Job Deck Example

## JOB DEFINITION STATEMENTS

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

### NONREAL-TIME JOB STATEMENT

```
*JOB(ID=      ,AC=    )
```

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

<u>Parameter</u> <u>Definition</u>

ID One to eight characters indicating the job identification. This parameter is optional. If omitted, ID=.JOB. is supplied by MPX/OS.

AC One to eight characters indicating the job account number. This is an optional parameter. If omitted, blanks are supplied by MPX/OS.

### REAL-TIME JOB STATEMENT

```
*RJOB(ID=      ,AC=    )
```

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 nonreal-time job in its ability to use the reserved priorities (1 to 9, 240 to 255) 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.

## SCHEDULE STATEMENT

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

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

| Parameter | Definition |
|---|---|
| CM= | Core 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 may 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 99,999. A value of 99999 is regarded as infinity. Time charged against this limit is CPU usage only. The parameter may be omitted, in which case an MPX/OS-defined time limit is used. |
| PL= | Print line limit assigned to job, value ⩽ 65,535. The parameter may be omitted, in which case an MPX/OS-defined print limit is used. |
| PC= | Punch card limit assigned to job, value ⩽ 65,535. The parameter may be omitted, in which case an MPX/OS-defined punch limit may be used. |
| 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 may 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. |

| Mmemonic | Hardware Type |
|---|---|
| MT7 | Seven-track magnetic tape |
| MT | Nine-track magnetic tape (800 bpi) |
| MT9 | Same as MT |
| MT9M | Same as MT |

Parameter (Cont.) Definition (Cont.)

| Mmemonic | Hardware Type |
|---|---|
| MT9H | Nine-track tape, 1600 bpi |
| MT7M | Same as MT7 |
| MT7L | Seven-track tape, 556 bpi |
| TT | Teletypewriter |
| CT | Cartridge tape |
| CP | Card punch |
| DP | CRT display |
| PR | Printer |
| PL | Plotter |
| CR | Card reader |

dt=DID The device types and device identifications on which class B files will be accessed in the job (for example, 9427=SCR1, 844=SCR2).

## JOB ACTIVITY CONTROL STATEMENTS

The job activity section of a batch job consists of four types of control statements: miscellaneous, data set identification, data set modification, and task preparation and use control statements. A batch 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 cathode-ray tube (CRT).

## COMMENT-TO-OPERATOR STATEMENT

```
*CTO message
```

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

## PAUSE STATEMENT

```
*PAUSE message
```

The pause (*PAUSE) statement causes job processing to be suspended. The message is copied to the console CRT. 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, an *EQUIP statement is used. A new logical unit number can also be defined as being equivalent to an already-defined logical unit number with an *EQUIP statement.

## EQUIPMENT ASSIGNMENT STATEMENT

*EQUIP(lu=$d_1$, lu=$d_2$, ... , lu=$d_n$)

The equipment assignment (*EQUIP) statement assigns physical equipments to logical units for the job. The statement includes declarations (d) regarding logical units (lu). The legal logical unit assignments are 1 through 63, with the following fixed assignments.

| Logical Unit Number | Assignment |
|---|---|
| 63 | Standard input (INP) |
| 62 | Standard output (OUT) |

| Logical Unit Number (Cont.) | Assignment (Cont.) |
|---|---|
| 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 |

## ASSIGNING UNIT RECORD DEVICES

```
*EQUIP(lu=hh, .. ,lu=hh)
```

The parameters of this statement assign a hardware type (hh) to a logical unit number (lu). The following hardware mnemonics are used in making the assignments.

| Mnemonic | Type |
|---|---|
| MT | 800 bpi density selection, 9-track tape format (CONTROL DATA® 659 and 669 Magnetic Tape Units) |
| MT9 | Same as MT |
| MT9M | Same as MT |
| MT9H | 1600 bpi density selection, 9-track tape format |

| Mnemonic (Cont.) | Type (Cont.) |
|---|---|
| MT7 | 800 bpi density selection, 7-track tape format (CONTROL DATA® 667 Magnetic Tape Unit) |
| MT7M | Same as MT7 |
| MT7L | 556 bpi density selection, 7-track tape format |
| CR | Card reader |
| PR | Line printer |
| CP | Card punch |
| TT | Teletypewriter |
| DP | CRT display |
| CT | Cartridge tape |
| PL | Plotter |

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, a diagnostic will result.

## ALLOCATE STATEMENT

```
*ALLOCATE(FN, OWNER, ED, AK, BLKSIZE, NOBLKS, S, USE, 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 | One to 14 characters specifying the file name |
| OWNER | One to four characters specifying the file owner |
| ED | One or two characters specifying the edition number |

Parameter (Cont.) Definition (Cont.)

AK

One to four 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 65,535

NOBLKS

Number of logical blocks in the file. Decimal constant in the range of 1 to 65,535

S

Segmentation flag:

| Entry | Indicates |
|---|---|
| Blank | File may be segmented |
| S | File may be segmented |
| NS | File may not be segmented |

USE

Protection flag:

| Entry | Indicates |
|---|---|
| Blank | File may be accessed as read/write |
| R | File may be accessed as read only. It may not be written until modified |
| RW | File may be accessed as read/write |

DT

Device type:

| Entry | Indicates |
|---|---|
| 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® 9760/9762 Storage Module Drive |

DID

One to eight characters identifying the device to be used for the file. Up to eight devices may 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.

## 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 | One to 14 characters specifying the file name |
| OWNER | One to four characters specifying the file owner |
| ED | One or two characters specifying the edition number of file to be opened |
| AK | One to four characters specifying the access privacy key. This field is not copied on the job OUT file |
| USE | Protection flag: |

| Entry | Indicates |
|---|---|
| Blank | File may be accessed as read/write |
| R | File may be accessed as read only |
| RW | File may be accessed as read/write |
| W | File may 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) |

## LOGICAL UNIT EQUIVALENCING

*EQUIP(lu$_1$=lu$_2$,..,lu$_n$=lu$_m$)

This statement equates 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.

## DATA SET MODIFICATION STATEMENT

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.

## 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. Logical units with any other type of assignment are ignored.

## 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 block of the file. A search forward positions to the block past the highest block written.

## 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 block of the file. The units to be rewound are indicated by the parameters $lu_1$ through $lu_n$, which are logical unit numbers.

If a logical unit number is illegal, unassigned, or not assigned to magnetic tape or a file, the parameter is ignored and the remainder of the statement is processed.

## 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.

## 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.

## MODIFY STATEMENT

*MODIFY(FN, OWNER, ED, AK, NFN, NOWNER, NED, NAK, NOBLKS, S, USE, $\text{DID}_1$, . . . , $\text{DID}_n$)

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 | One to 14 characters specifying the file name |
| OWNER | One to four characters specifying the file owner |
| ED | One or two characters specifying the edition number |
| AK | One to four 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 | One to 14 characters specifying the new file name (blank = no change) |
| NOWNER | One to four characters specifying the new owner (blank = no change) |
| NED | One or two characters specifying the new edition number (blank = no change) |
| NAK | One to four 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 65,534. The total number of blocks in the expanded file may not exceed 65,535. If blank, there is no change. |

Parameter (Cont.) Definition (Cont.)

S Segmentation flag:

| Entry | Meaning |
|---|---|
| Blank | Added blocks may be segmented |
| S | Added blocks may be segmented |
| NS | Added blocks may not be segmented |

USE New protection flag for the modified file:

| Entry | Meaning |
|---|---|
| Blank | Does not modify existing usage parameter |
| R | File may be accessed as read only |
| RW | File may be accessed as read/write |

DID One to eight 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 may not exceed eight

## RELEASE STATEMENT

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

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

Parameter Definition

FN One to 14 characters specifying the file name

OWNER One to four characters specifying the file owner

ED One or two characters specifying the file edition number

AK One to four characters specifying the access privacy key for the file. This field is not copied to the OUT file

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| 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 |

## 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 the assembler or compiler, is loaded and executed by a library name control statement.

```
*name(parameter list)
```

A program may 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)
```

## TASK STATEMENT

```
*TASK(ID=   ,PCC=   ,DMP   ,PTY=   ,CPU=  )
```

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

Parameter — Definition

ID= One to eight characters indicating the task identification; this parameter is optional. If not present, the system default DUMMYTASK 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. The value assigned to PCC is the logical unit for I/O. The logical unit must have previously had a CRT assigned to it with an *EQUIP statement.

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= One to three numeric characters indicating the task priority. This parameter is optional. It must be a number, n, where $1 \leq n \leq 255$ for real-time jobs and $10 \leq n \leq 239$ for nonreal-time jobs. If $n \leq 10$ or $\geq 239$ for a nonreal-time job, the task priority is set respectively to 10 or 239. 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 (Section 3, 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 operating system determination chosen to level the CPU loading. A nondefault value constrains the execution of the task to the designated CPU except for ESR processing.

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 after the logical unit number [for example, PCC = 10 (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 3-3 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, since it does not have a corresponding *TASK statement.

## 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 may 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).

## BUILD ABSOLUTE TASK STATEMENT

*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. An absolute file may be used in a task call sequence to decrease the load time of the called task.

## 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

```
*JOB (. . .)
*SCHED (. . .)
.
.
.
*TASK (ID=TASK1,PCC=10,DMP, PRTY=25)
.
.
.
*LOAD(20) Load Task 1
*RUN
.
.
.
*LOAD(30) Load Task 2
*RUN
.
.
*TASK(ID=TASK3,PCC=10,DMP, PRTY=26)
.
.
.
*LOAD(40) Load Task 3
*RUN
.
.
.
*EOJ
```

Figure 3-3. *TASK Control Statement Example

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).

## 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.

## 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 3-4.

## JOB ACCOUNTING STATISTICS

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

- Job name
- Account number
- Date (mm/dd/yy)
- MPX/OS resident edition number

```
            ┌── name of task      ┌── status of task when job aborted
TASK NAME = JMTR            STATUS = 01

Page Registers    04050200  02000200  02000200  02000200  02000200  02000200  02000200     0200001F  ┐
               ┌─[2B3D043C] 00000000  001E71A0  00000000  00000002  00000CAD  00000001  X7 00000005  │ repeated for
               │ H0 50000000  0000FC3E  000003C2  00000000  B4000002  00000005  0000F022  H7 00000014  > each task
               │ R0 434F5359  20202020  0000003A  00001901  00000003  00000000  00000000  R7 00000384  │ in job
               │ R8 00000198  00000063  454F4A20  0000003E  0003C000  00000088  000001E0  Rf 00000609  ┘
               │
               └── 043C - abort address + 1                X0 register is not displayed

                   2B   - start address of stack

                   3D   - end address of stack

                          add 1 at two digits of
                          abort address to look
                          ahead

                          stack pointers to
                          compute stack address

                          Actual stack address
                             042B and
                             043D
```

Figure 3-4. Examples of Abort Listing of Registers.

- Library edition number
- Time on (hh/mm/ss)
- Time off (hh/mm/ss)
- Time used (hh/mm/ss.sss)
- Facilities not used:
    - Core memory
    - Scratch segments
    - Print lines
    - Punch cards

The job accounting statistics are illustrated in Figure 3-5.

```
  JOB NAME = ENLARGE   ACCT. NO. =      DATE = 03/22/77  TIME ON = 14/16/36   TIME OFF = 14/18/17
RESIDENT EDITION = 03    VERSION = 01A    LIBRARY EDITION = 02
ACCUMULATING TIME = 00/00/58.200
ACCOUNTING INFORMATION
  RESOURCES NOT USED
  CORE = 000
  SCR  = 012
  LINE  = 7104
  CARD = 0000
JOB ABORTED   ABORT TYPE = 02    ABORT CODE = 01    TASK = JMTR
```

Figure 3-5. Abort Message and Accounting Statistics Example

# EXECUTIVE SERVICE REQUESTS 4

All executive service request (ESR) descriptions follow one basic format illustrated by Figure 4-1. Each description assumes that the ESR was issued by the monitor call instruction, MON,R ESRID. R is the first of four 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 labeled and explained for each ESR.

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.)

## FILE MANAGER ESRs

The following file manager ESRs provide the executing task with the same mass storage management facilities available to the batch job. The batch job control statements have the same names as the file manager ESRs. For example, after processing file manager returns a code to PARM. A nonzero code indicates an error response (see Appendix D).

ESR Title, Written Description

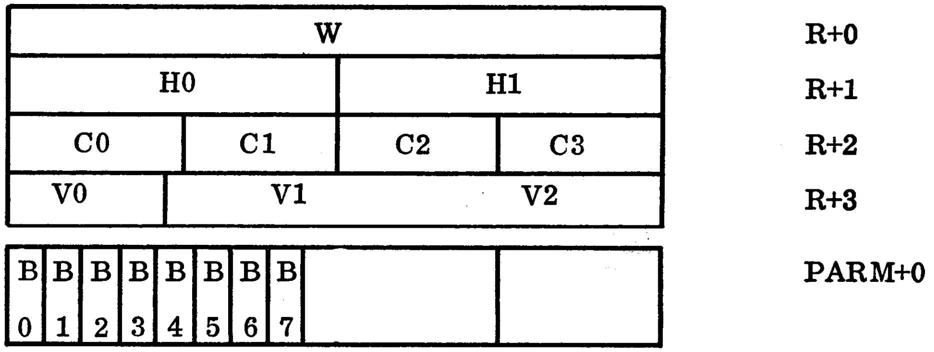
## ESRID, ESR IDENTIFIER

This portion of the description explains the options and basic function of the executive service request being presented. The written description is followed by a drawing of all data passed between the executive and the task. The drawing fields are labeled and an explanation of each labeled field follows the drawings.

Drawings with Labeled Fields

ESR format

PARM format



Field Descriptions

| Parameter | Definition |
|---|---|
| R | The set of register values passed to the executive. |
| W | Full-word parameter value (no subdivisions). |
| H0-H1 | Two half-word (16-bit) fields (two subdivisions and no explicit mention of nonstandard sizes). |
| C0-C3 | Four character (8-bit) fields (four subdivisions and no explicit mention of nonstandard sizes). |
| V0-V2 | Three variable (nonstandard) size fields. Their descriptions would specify the field sizes. |
| PARM | Executive normal response area. |
| B1-B8 | Bit fields. When occuring in words with standard field sizes, and if all bit subdivisions are explicit, their size is not specified. |
| □ | Unused areas are normally left blank. Field should be filled with zero. |

Note Area

NOTES Additional usage hints or constraints not covered by the field descriptions and not evident from the basic ESR description are presented here. All values are expressed in decimal unless otherwise stated.

Figure 4-1. ESR Description Format

## 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).

ESR format

| | |
|---|---|
| ADDRESS | R+0 |

ADDRESS format

<table>
<tr><td colspan="3">FN (0-3)</td><td>ADDRESS+0</td></tr>
<tr><td colspan="3">FN (4-7)</td><td>ADDRESS+1</td></tr>
<tr><td colspan="3">FN (8-11)</td><td>ADDRESS+2</td></tr>
<tr><td colspan="2">FN (12-13)</td><td>ED</td><td>ADDRESS+3</td></tr>
<tr><td colspan="3">OWNER</td><td>ADDRESS+4</td></tr>
<tr><td colspan="3">ACCESS</td><td>ADDRESS+5</td></tr>
<tr><td>USE</td><td>S</td><td>NOBLKS</td><td>ADDRESS+6</td></tr>
<tr><td colspan="2">BLKSIZE</td><td>DT</td><td>ADDRESS+7</td></tr>
<tr><td colspan="3">LE or $DID_1$ (0-3)</td><td>ADDRESS+8</td></tr>
<tr><td colspan="3">$DID_1$ (4-7)</td><td>ADDRESS+9</td></tr>
<tr><td colspan="3">⋮</td><td>⋮</td></tr>
<tr><td colspan="3">END OF LIST</td><td>ADDRESS+n</td></tr>
</table>

| Parameter | Definition |
|---|---|
| ADDRESS | The full-word address of the first word of a file definition |
| FN | Fourteen-character string that defines the file name |
| ED | Two-character string that defines the file edition |
| OWNER | Four-character string that defines the file owner |
| ACCESS | Four-character string that defines the file access privacy key |
| USE | Binary value that defines the allowed file usage: |

| Entry | Indicates |
|---|---|
| =0 | File may be opened for read/write use |
| =1 | File may be opened for read use only |

Parameter (Cont.) Definition (Cont.)

S Binary value defining acceptable segmentation mode for file allocation:

| Entry | Indicates |
|---|---|
| =0 | File may be allocated in segments |
| =1 | File may not be segmented when allocated |

NOBLKS Binary value defining the size of the file in each logical block. Value may be from 1 to 65,535.

BLKSIZE Binary value defining the number of words in each logical block. Value may be from 1 to 65,535.

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

| Entry | Indicates |
|---|---|
| =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 9760/9762 Storage Module Drive |

DID Eight-character string defining the device or devices to be used for file allocation

LE The list-end flag:

| Entry | Indicates |
|---|---|
| = -1 | The list has ended |
| ≠ -1 | Another DID specification begins |

An example of a calling sequence is as follows:

```
LDA,R0          ALLOCTBL              Define file

MON,R0          ALLOCATE              Call

LD,R7           PARM

TST,NE          R7,X0,ERRPROC         Test for errors
      .
      .
      .
ALLOCTBL    TEXTC    14,FILE-NAME         File name

            TEXTC    2,01                 Edition

            TEXTC    4,OWNR               Owner

            TEXTC    4,AKEY               Access key

            VFD      8/0,8/0,16/1000      Read/write, segment, 1000 blocks

            VFD      16/480,16/0          480-word block, system device

            TEXTC    8,SYSTEM01           Device identification

            GEN      -1                   List has ended
```

NOTE: A maximum of eight devices and between 20 to 40 segments may be specified for one file.

## 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 two things: it frees a logical unit number, and it may remove restrictions on file usage. Since all files are closed by the system when a job terminates, the CLOSE ESR is used for overall efficiency.

| | | |
|---|---|---|
| ESR format | LU | R+0 |

| Parameter | Definition |
|---|---|
| LU | Number of logical unit to be closed |

An example of a calling sequence is as follows:

```
LDI,R1      10                  Logical unit number
MON,R1      CLOSE               Call
LD,R2       PARM                Check for errors
TST,NE      R2,X0,ERRORPROC
```

## 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 the control parameters of the file.

ESR format

| | |
|---|---|
| ADDRESS | R+0 |

ADDRESS format

| | | | |
|---|---|---|---|
| OFN (0-3) | | | ADDRESS+0 |
| OFN (4-7) | | | ADDRESS+1 |
| OFN (8-11) | | | ADDRESS+2 |
| OFN (12-13) | OED | | ADDRESS+3 |
| OOWNER | | | ADDRESS+4 |
| OAK | | | ADDRESS+5 |
| NFN (0-3) | | | ADDRESS+6 |
| NFN (4-7) | | | ADDRESS+7 |
| NFN (8-11) | | | ADDRESS+8 |
| NFN (12-13) | NED | | ADDRESS+9 |
| NOWNER | | | ADDRESS+10 |
| NAK | | | ADDRESS+11 |
| 0 7<br>USE | 8 15<br>S | 16 31<br>NOBLKS | ADDRESS+12 |
| LE or DID$_1$ (0-3) | | | ADDRESS+13 |
| DID$_1$ (4-7) | | | ADDRESS+14 |
| ⋮ | | | ⋮ |
| END OF LIST | | | ADDRESS+n |

| Parameter | Definition |
|---|---|
| ADDRESS | Full-word address of the first word of the file-definition modification specifications |
| OFN | Fourteen-character string that is the name of the file to be redefined (old file name) |
| OED | Two-character string that is the edition number of the file to be redefined (old edition) |
| OOWNER | Four-character string that is the file owner identification of the file to be redefined (old owner) |
| OAK | Four-character string that is the access key of the file to be redefined (old access key) |
| NFN | Fourteen-character string that defines the new file name (no change if blank) |
| NED | Two-character string that defines the new edition number of the file (no change if blank) |
| NOWNER | Four-character string that defines the new owner identification of the file (no change if blank) |
| NAK | Four-character string that defines the new access key for the file (no change if blank) |
| USE | Binary value that defines the (new) allowed file usages: |

| Entry | Indicates |
|---|---|
| =0 | File may be opened for read/write use |
| =1 | File may be opened for read-only use |

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

| Entry | Indicates |
|---|---|
| =0 | Addition may be allocated in segments |
| =1 | Addition may not be segmented |

| | |
|---|---|
| NOBLKS | Binary value defining the number of blocks to be added to the file. Total file allocation may not exceed 65,535 blocks. |

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| LE | The list-end flag: |
| DID | Eight-character string defining the device or devices to be used for the expanded blocks |

The list-end flag (LE):

| Entry | Indicates |
|---|---|
| = -1 | The list has ended |
| ≠ -1 | Another DID specification begins |

An example of a calling sequence is as follows:

```
LDA,RC       MODLIST                      Address of modify list

MON,RC       MODIFY                       Call

LD,X7        PARM                         Check for errors

TST,NE       X7,X0,ERRPROC
   .
   .
   .
MODLIST   TEXTC     14,FILE-NAME         File name - old

          TEXTC     2,01                 Edition - old

          TEXTC     4,OWNR               Owner - old

          TEXTC     4,AKEY               Access key - old

          TEXTC     14,FILE-NAME         Keep same file name

          TEXTC     2,02                 New edition

          TEXTC     4,OWNR               Keep same owner

          TEXTC     4,NKEY               New access key

          VFD       8/0,8/0,16/1000      Read/write, segmented, add 1000 blocks

          TEXTC     8,SYSTEM02           Device for addition

          GEN       -1                   List has ended
```

NOTE: A maximum of eight devices and 20 to 40 segments may be specified for one file.

## OPEN, ESTABLISH ACCESS TO MASS STORAGE FILE

The OPEN ESR is used to prepare an existing file for data transmission by locating the file and requesting the device to be put on-line, if necessary.

| | | |
|---|---|---|
| ESR format | ADDRESS | R+0 |
| | LU | R+1 |

| | | | | |
|---|---|---|---|---|
| ADDRESS format | FN (0-3) | | | ADDRESS+0 |
| | FN (4-7) | | | ADDRESS+1 |
| | FN (8-11) | | | ADDRESS+2 |
| | FN (12-13) | | ED | ADDRESS+3 |
| | OWNER | | | ADDRESS+4 |
| | AK | | | ADDRESS+5 |
| | USE | | BLOCK | ADDRESS+6 |
| Bit | 0 7 | 8 15 | 16 31 | |

Parameter — Definition

ADDRESS — Full-word address of the first word of the file identification specification

LU — Number of the logical unit to be assigned to the file being opened

FN — Fourteen-character string that is the name of the file to be opened

ED — Two-character string that is the edition number of the file to be opened

OWNER — Four-character string that is the owner identification of the file

AK — Four-character string that is the access key of the file

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 | Indicates |
|---|---|
| =0 | File to be used for read/write |

Parameter (Cont.) Definition (Cont.)

| Entry (Cont.) | Indicates (Cont.) |
|---|---|
| =1 | File to be used for read only |
| =2 | File to be used for read/write, set the highest block written count to 0 (next block written will be the first block of the file) |

BLOCK The first block to be read/written is defined by the value of block:

| Entry | Indicates |
|---|---|
| =0 | First block of file to be accessed next |
| ≠0 | The number of the block to be accessed next (The device containing block number BLOCK is opened in a partial open of the file) |

An example of a calling sequence is as follows:

```
         LDA,R3    OPENTBL           Address of open parameters

         LDI,R4    10                Logical unit number

         MON,R3    OPEN

         LD,X7     PARM              Check for errors

         TST,NE    X7,X0,ERRPROC

              .
              .
              .

OPNTBL   TEXTC     14,FILE-NAME      File name

         TEXTC     2,02              Edition

         TEXTC     4,OWNR            Owner

         TEXTC     4,NKEY            Access key

         VFD       8/0,8/0,16/0      Read/write, access first block
```

## RELEASE, RELEASE MASS STORAGE FILE SPACE

The RELEASE ESR is used to release some or all of the space allocated to a file.

| | | |
|---|---|---|
| ESR format | ADDRESS | R+0 |

| | | | |
|---|---|---|---|
| ADDRESS format | FN (0-3) | | ADDRESS+0 |
| | FN (4-7) | | ADDRESS+1 |
| | FN (8-11) | | ADDRESS+2 |
| | FN (12-13) | ED | ADDRESS+3 |
| | OWNER | | ADDRESS+4 |
| | AK | | ADDRESS+5 |
| | NOBLKS | | ADDRESS+6 |

| Parameter | Definition |
|---|---|
| ADDRESS | The full-word address of a file release specification |
| FN | Fourteen-character string that is the name of the file |
| ED | Two-character string that is the edition number of the file |
| OWNER | Four-character string that is the owner identification of the file |
| AK | Four-character string that is the access key of the file |
| NOBLKS | The number of blocks by which the file is to be reduced in length. If NOBLKS = -1, all blocks beyond the highest block written are released. If NOBLKS = 0, all blocks are released |

An example of a calling sequence is as follows:

```
LDA,R0      RELTBL        Address of release parameters

MON,R0      RELEASE       Call

LD,X5       PARM          Check for errors
```

```
          TST,NE    X5,X0,ERRPROC
              .
              .
              .
RELTBL    TEXTC     14,FILE-NAME     File name
          TEXTC     2,02             Edition
          TEXTC     4,OWNR           Owner
          TEXTC     4,NKEY           Access key
          GEN       -1               Release unused space
```

### STANDARD UNIT

Standard units such as INP, OUT and PUN (see Section 3, EQUIP Assignment) may be accessed by the user. The user should access these units through blocker/deblocker with PICK and PACK (see Section 6). The block pointer, record headers and trailers, block numbers, etc. are defined for the job by the system (block size is 480 words).

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

## PHYSICAL I/O ESRs

The following ESRs are more extensive than the functions provided at the batch-job control-statement level. All of these ESRs may be used on unit record devices and mass storage devices. The ESRs are summarized in Table 4-1.

### BKSP, BACKSPACE LOGICAL UNIT ONE RECORD

The BKSP ESR repositions the logical unit before the immediately-preceding record. If the logical unit is at the beginning of the file, no action is taken.

```
ESR format   ┌──────────────────────────────────────┐
             │                           LU         │  R+0
             └──────────────────────────────────────┘
```

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be repositioned |

TABLE 4-1. PHYSICAL I/O ESRs AS A FUNCTION OF DEVICE TYPE

| Routine | Operation | Applicable Unit Record Devices* | Mass Storage Device Operation |
|---|---|---|---|
| READLU | Data transfer from logical unit | MT, CR, TT, DP, CT | Read record |
| WRITLU | Data transfer to logical unit | MT, PR, CP, TT, DP, CT, PL | Write record |
| WEOF | Write end-of-file | MT, CP, CT | Set highest block written count equal to current block position |
| ERASE | Erase 6 inches of tape | MT, CT | Illegal |
| SEOF | Search for end of file | MT, CT | Set next block number to block 1 or block count +1 |
| UTYP | Return hardware type | MT, PR, TT, CT, CR, CP, DP, PL | Return hardware type and file attributes |
| BKSP | Backspace one record | MT, CT | Reduce the next block number one block |
| UST | Await completion of I/O | MT, PR, TT, CT, CR, CP, DP, PL | Await completion of I/O |
| REWD | Reposition to starting point | MT, CT | Set the next block number to first block |
| UNLD | Unload unit | MT | NOP |
| BSY | Return busy/not-busy status | MT, PR, TT, CT, CR, CP, DP, PL | Return busy/not-busy, status |
| ULOC | Locate | NOP | Set the next block number to specified block |
| SELDEN | Select density | MT | NOP |
| SELTRK | Select track | CT | NOP |

| | |
|---|---|
| *MT = Magnetic Tape | CR = Card Reader |
| PR = Line Printer | CP = Card Punch |
| TT = Teletypewriter | DP = CRT Display |
| CT = Cartridge Tape | PL = Plotter |

NOTE: MPX/OS performs error recovery on standard peripheral units when an error is detected during data transmission (see Appendix E). If the error is not recoverable, the status indicates the error type (Section 4, UST, Unit Status Test).

An example of a calling sequence is as follows:

```
LDI,R4      15        Logical unit number

MON,R4      BKSP      Call
```

## BSY, BUSY LOGICAL UNIT STATUS TEST

The busy/not-busy status of a logical unit may be tested using the status test ESR, BSY. The status is not a function of a particular I/O request, but rather of the unit itself. The requestor is immediately placed on the ready list after the request is processed.

| | | |
|---|---|---|
| ESR format | LU | R+0 |
| PARM format | STATUS | PARM+0 |

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

| Entry | Indicates |
|---|---|
| =0 | Unit is not busy |
| ≠0 | Unit is busy |

An example of a calling sequence is as follows:

```
LDI,R2      15        Logical unit number

MON,R2      BSY       Call

LD,X5       PARM      Check status
```

## ERASE, ERASE MAGNETIC TAPE SEGMENT

The ERASE ESR erases approximately 6 inches of magnetic tape in an effort to bypass faulty material.

ESR format

| | |
|---|---|
| LU | R+0 |

| Parameter | Definition |
|---|---|
| LU | Number of logical unit to which the magnetic tape is assigned |

An example of a calling sequence is as follows:

```
        LDI,RA      33          Logical unit number

        MON,RA      ERASE       Call
```

## READLU, READ RECORD FROM LOGICAL UNIT

The READLU ESR initiates a data transfer from the specified logical unit to a buffer allocated in the issuing task address space. After the read is initiated, the issuing task is placed on the ready list. The UST or BSY ESRs are provided to determine when the transmission is complete.

ESR format

| | |
|---|---|
| ADDRESS | R+0 |
| LENGTH | R+1 |
| MODE | R+2 |
| LU | R+3 |

| Parameter | Definition |
|---|---|
| ADDRESS | The address of the buffer that is to receive the transmitted record. The value is an 18-bit byte address or a 16-bit full-word address (see MODE definition on next page). |
| LENGTH | The number of elements to be transmitted to the buffer. The number is a byte or word count depending on the specification of MODE |

Parameter (Cont.) — Definition (Cont.)

MODE — Selects the data element size and recording method of the record:

| Entry | Indicates |
|---|---|
| =0 | ASCII record, word format |
| =16 | ASCII record, byte format |
| =32 | Binary record, word format |

LU — Number of the logical unit to be read

NOTES: 1/ The maximum buffer size is 4096 words (16,384 bytes).

2/ If LENGTH is specified as zero, the maximum length buffer is transmitted.

3/ If LU idenfifies the CRT, a LENGTH value of zero is used to request that the manual interrupt be enabled. The detection of a manual interrupt from the operator is accomplished with the UST ESR (returns key code value of nonzero).

An example of a calling sequence is as follows:

```
LDA,R0     BUFF      Address of data
LDI,R1     20        Number of words
LDI,R2     0         ASCII record, words
LDI,R3     10        Logical unit number
MON,R0     READLU    Call
```

## REWD, REWIND LOGICAL UNIT

The REWD ESR repositions the logical unit to the initial position (magnetic tape loadpoint, disk file first block).

| | | |
|---|---|---|
| ESR format | LU | R+0 |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be repositioned |

An example of a calling sequence is as follows:

```
LDI,RB     21      Logical unit number
MON,RB     REWD    Call
```

## 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 the end of the file (for forward searches). The issuing task is scheduled for execution after the search is initiated. The issuing task must issue a BSY or UST ESR to determine when the operation is complete.

ESR format

| | |
|---|---|
| LU | R+0 |
| DIRECTION | R+1 |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be repositioned |
| DIRECTION | Flag word that defines the direction of the search: |

| Entry | Indicates |
|---|---|
| =0 | Search forward |
| =1 | Search backward |

An example of a calling sequence is as follows:

```
LDI,R0     10      Logical unit number
LDI,R1     0       Search forward
MON,R0     SEOF    Call
```

## SELDEN, SELECT DENSITY

The SELDEN ESR initiates a select density on the specified logical unit.

ESR format

| | |
|---|---|
| LU | R+0 |
| DENSITY | R+1 |

| Parameter | Definition |
|---|---|
| LU | Number of logical unit to which the tape is assigned |
| DENSITY | Density control: |

| Entry | Indicates |
|---|---|
| =0 | Low density (556 bpi NRZI - 667) |
| =1 | High density (800 bpi NRZI - 667/669) |
| =2 | Hyper density (1600 bpi NRZI - 669) |

An example of a calling sequence is as follows:

```
LDI,RA      11        Logical unit number
LDI,RB      1         Select 800 bpi
MON,RA      SELDEN    Call
```

## SELTRACK, SELECT TRACK

The SELTRACK ESR initiates a select track on the specified logical unit.

ESR format

| | |
|---|---|
| LU | R+0 |
| TRACK | R+1 |

| Parameter | Definition |
|---|---|
| LU | Number of logical unit |
| TRACK | Track control: |

| Entry | Indicates |
|---|---|
| =0 | Select track 0 |
| =1 | Select track 1 |
| =2 | Select track 2 |
| =3 | Select track 3 |

An example of a calling sequence is as follows:

```
LDI,R0      10           Logical unit number
LDI,R1      3            Select track 3
MON,R0      SELTRACK     Call
```

## ULOC, LOCATE BLOCK ON LOGICAL UNIT

The ULOC ESR sets the next block number of a logical unit to a 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.

| ESR format | | |
|---|---|---|
| | LU | R+0 |
| | NUMBER | R+1 |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be positioned |
| NUMBER | Number of the block to which the unit is to be set. The block identified will be the next block read or written. NUMBER = -1 is used to request the NBN to be set to the last block written +1. |

An example of a calling sequence is as follows:

```
            LDI,R3      10          Logical unit number

            LDI,R4      24          Block number

            MON,R3      ULOC        Call
```

## UNLD, UNLOAD LOGICAL UNIT

The UNLD ESR rewinds and dismounts a magnetic tape reel. If applied to a disk file, the ESR is functionally a nonoperation. Note that the magnetic tape drive is still assigned to the job and to the unit number.

| ESR format | | |
|---|---|---|
| | LU | R+0 |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be dismounted |

An example of a calling sequence is as follows:

```
            LDI,R3      24          Logical unit number

            MON,R3      UNLD        Call
```

## UST, UNIT STATUS TEST

The status of a logical unit for which the user has an operation in progress may be tested using the UST ESR. The request places the issuing task in the I/O WAIT status until the end-of-operation interrupt has been processed on the unit, but only if the issuer has a request pending for that unit; otherwise a null status (=0) is returned in PARM. The status is returned in PARM and PARM+1.

| ESR format | | |
|---|---|---|
| | LU | R+0 |

|  | 0 20 | 21 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |  |
|---|---|---|---|---|---|---|---|---|---|---|
| PARM format | Unused | IOC | I<br>D<br>E | E<br>O<br>A | E<br>O<br>D | M | E<br>O<br>F | N<br>R | R | PARM+0 |
|  | ES |  |  |  |  |  |  |  |  | PARM+1 |

**Parameter** — **Definition**

LU — Number of the logical unit to be tested

IOC — Four-bit field (bits 21 through 24) defining the status of the IOC to which the unit is attached. Bit interpretation is device dependent (see Table 4-2). (Also, see MP-60 Computer System Peripheral Reference Manual, CDC Publication No. 14063900.)

TABLE 4-2. BIT INTERPRETATION PER DEVICE TYPE

| Bit / Device | 21 | 22 | 23 | 24 |
|---|---|---|---|---|
| Magnetic tape | EOT | Load-point | Lost data | Hardware error |
| Disk | SEEK error | Address error | Lost data | Hardware error |
| Line printer | Out of paper/ paper fault | N/A | N/A | Hardware error |
| Card punch | N/A | N/A | Hopper empty | Hardware error |
| Card reader | N/A | N/A | Input tray empty | Hardware error |
| CRT display | FUNCTION KEY CODE |  |  |  |
| Cartridge tape | End-of-tape | Beginning of tape | Lost data | Hardware error |

Parameter (Cont.) — Definition (Cont.)

IDE — Irrecoverable data error flag (bit 25):

| Entry | Indicates |
|---|---|
| =0 | No error |
| =1 | Error (bit 31 also set) |

EOA — End-of-allocated blocks (mass storage files, only) or irrecoverable memory error flag (bit 26):

| Entry | Indicates |
|---|---|
| =0 | Not end-of-allocated blocks |
| =1 | End-of-allocated blocks (bit 31 also set) |

EOD — End-of-device (system use for mass storage) or EOT (magnetic tape files) flag (bit 27):

| Entry | Indicates |
|---|---|
| =0 | Not end-of-device |
| =1 | End-of-device (bit 31 also set) |

M — Mode of transmission flag (bit 28):

| Entry | Indicates |
|---|---|
| =0 | ASCII transmission |
| =1 | Binary transmission |

EOF — End-of-file flag (bit 29):

| Entry | Indicates |
|---|---|
| =0 | Not end-of-file |
| =1 | End-of-file (bit 31 also set) |

Parameter (Cont.) Definition (Cont.)

NR Not ready flag (bit 30):

| Entry | Indicates |
|---|---|
| =0 | Unit ready |
| =1 | Unit not ready (bit 31 also set) |

R Reject flag (bit 31):

| Entry | Indicates |
|---|---|
| =0 | Request accepted |
| =1 | Request rejected |

ES Extended status word. Definition is device dependent. A complete explanation can be found in the MP-60 Computer System Peripheral Equipment Reference Manual (CDC Publication No. 14063900)

An example of a calling sequence is as follows:

```
LDI,H4     10      Logical unit number
MON,H4     UST     Call
```

## UTYP, RETURN LOGICAL UNIT HARDWARE TYPE

The UTYP ESR obtains the hardware type of a specified logical unit. In addition, if the device type is a disk file, additional file description information is returned.

NOTE: Standard files such as INP, OUT, and PUN will be defined as mass storage files and not as card reader, printer, or punch.

| ESR format | LU | R+0 |
|---|---|---|

PARM format

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | HT | | | | | | | PARM+0 |
| | WORDS | | | | | | | PARM+1 |
| | NBN | | | | | | | PARM+2 |
| | LBN | | | | | | | PARM+3 |
| | M<br>S | T | B | I | O | A | | PARM+4 |
| Bit | 0 | 1 | 2 | 3 | 4 | 5 | | |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit for which information is to be returned |
| HT | Hardware type definition code: |

| Entry | Indicates |
|---|---|
| =0 | No assignment |
| =1 | Disk |
| =2 | Magnetic tape |
| =3 | Card reader |
| =4 | Card punch |
| =5 | Line printer |
| =6 | Keyboard/display |
| =7 | Teletypewriter |
| =8 | Cartridge tape |
| =9 | Plotter |
| =12 | Magnetic tape (7-track) |

| | |
|---|---|
| WORDS | The number of words per block of file. Returned for HT=1 only |
| NBN | The number of the next block to be read or written (that is, the current block position of the file). Returned for HT=1 only |
| LBN | The number of the last block written for the file. Returned for HT=1 only |

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| MS | Bit 0, if set means the device is a mass storage unit (disk) |
| T | Bit 1, if set means the device is a tape (9-track, 7-track, cartridge, cassette, etc.) |
| B | Bit 2, if set means the device is a blocked device |
| I | Bit 3, if set means the device is an input-only device |
| O | Bit 4, if set means the device is an output-only device |
| A | Bit 5, if set means the device is an ASCII-only output device |

An example of a calling sequence is as follows:

```
LDI,X7      11       Logical unit number

MON,X7      UTYP     Call
```

## WEOF, WRITE END-OF-FILE MARK

The WEOF ESR causes an end-of-file mark to be written on the specified logical unit (magnetic tape), or sets the last block written value to the current block number (disk file).

| | | |
|---|---|---|
| ESR format | LU | R+0 |

| Parameter | Definition |
|---|---|
| LU | Number of the logical unit to be written |

An example of a calling sequence is as follows:

```
LDI,R3      33       Logical unit number

MON,R3      WEOF     Call
```

## WRITLU, WRITE RECORD TO LOGICAL UNIT

The WRITLU ESR initiates a data transfer to a specified logical unit from a buffer in task memory. Once the data transfer is initiated, the issuing task is again placed on the ready list for continued execution. The task issues a BSY or UST ESR to determine when the transfer is complete.

ESR format

| | |
|---|---|
| ADDRESS | R+0 |
| LENGTH | R+1 |
| MODE | R+2 |
| LU | R+3 |

Parameter — Definition

ADDRESS — The address of the first word or the first character of the buffer to be written. The address type is determined by the MODE parameter

LENGTH — The number of words (characters) to be transferred from the buffer. LENGTH values may be from 1 to 4096 words (1 to 16,384 characters). A value of 0 is treated as the maximum field-length value. The count type (word versus character) is determined by the MODE parameter

MODE — The data transmission mode (format) code:

| Entry | Indicates |
|---|---|
| =0 | ASCII records in word format are transmitted |
| =16 | ASCII records in character format are transmitted |
| =32 | Binary records in word format are transmitted |

LU — Number of the logical unit to which the data is transmitted

An example of a calling sequence is as follows:

```
LDCA,R0    BUFA       Character address of buffer
LDI,R1     48         Number of characters
LDI,R2     16         ASCII records in character format
LDI,R3     15         Logical unit number
MON,R0     WRITLU     Call
```

## TASK MANAGER ESRs

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 batch-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 4-2 illustrates a multiprogramming relationship between tasks in a job. In Figure 4-2, 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.

### 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.

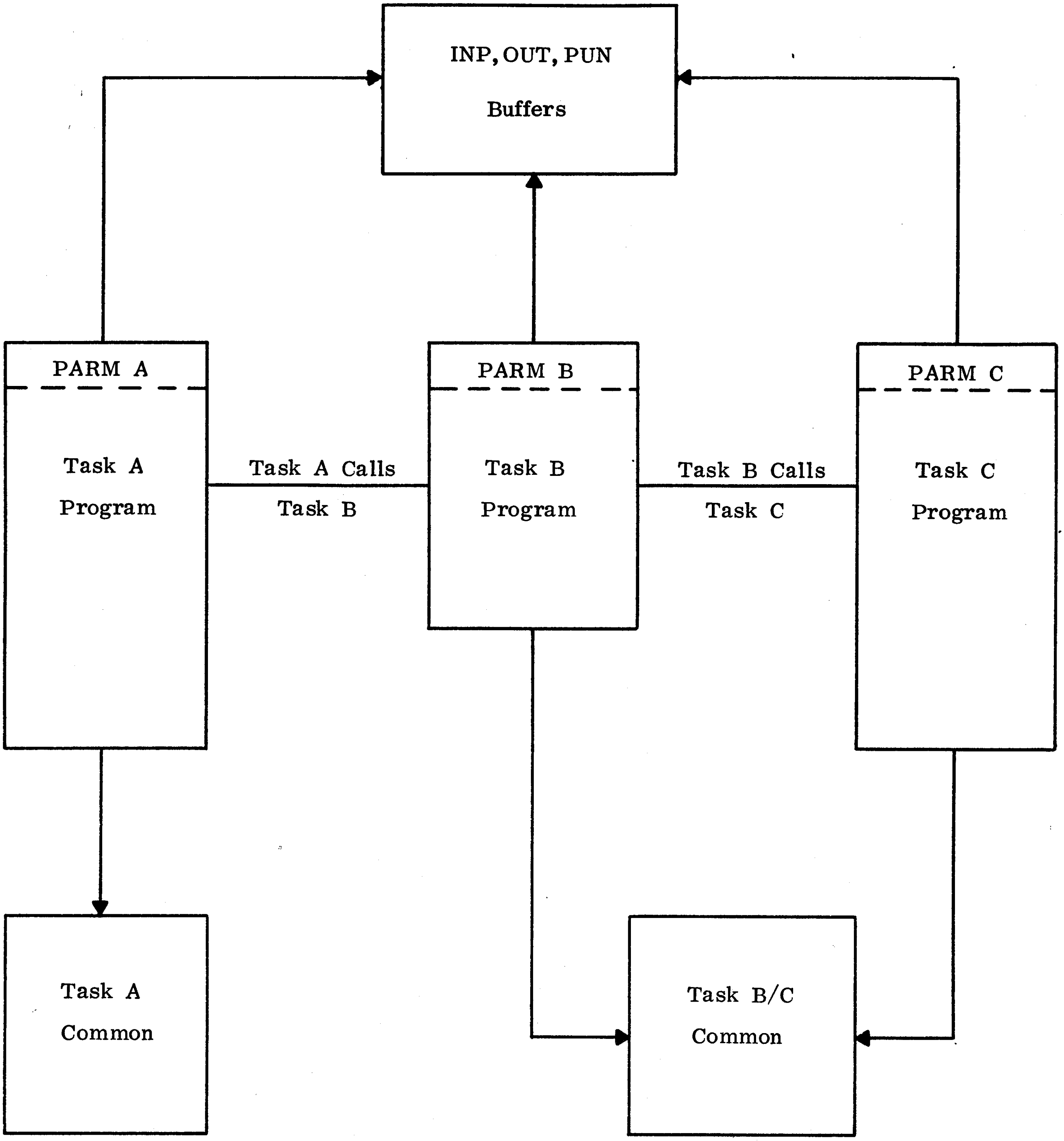


Figure 4-2. Multiprogramming Tasks

PARM format

| | | | | |
|---|---|---|---|---|
| | | EP$_1$ | | PARM+0 |
| | | EP$_2$ | | PARM+1 |
| | | EP$_3$ | | PARM+2 |
| | | EP$_4$ | | PARM+3 |
| | | PRI | | PARM+4 |
| I | = | 0 | 1 | PARM+5 |
| , | L | , | X | PARM+6 |
| , | R | ETX | | PARM+7 |
| | | | | • |
| | | | | • |
| | | | | • |

| 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:

```
          MON,R0       ABORT        Call
```

## 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 execution. 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 may use the DWAIT or TSTATUS ESRs to effect synchronization with the callee(s). The caller may 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 may 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 awaiting access to an active callee, the caller has a CALL status. If no TCT is available, the task is assigned a TCT wait 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.

<table>
<tr><td>ESR format</td><td colspan="9">TID (0-3)</td><td>R+0</td></tr>
<tr><td></td><td colspan="9">TID (4-7)</td><td>R+1</td></tr>
<tr><td></td><td colspan="6">LU(1)</td><td>LU(2)</td><td>LU(3)</td><td>LU(4)</td><td>R+2</td></tr>
<tr><td></td><td>C<br>P</td><td>C<br>A</td><td>C<br>W</td><td>Q<br>R</td><td>R<br>P</td><td></td><td>CPU</td><td>PRI</td><td>NPRMS</td><td>R+3</td></tr>
<tr><td>Bit</td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5-7</td><td></td><td></td><td></td><td></td></tr>
</table>

| | | |
|---|---|---|
| PARM format | STATUS | PARM+0 |

| Parameter | Definition |
|---|---|
| TID | Eight-character task name. This task identifier is maintained by MPX/OS for use in DWAIT and TSTATUS ESRs |

Parameter (Cont.) Definition (Cont.)

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 | Indicates |
|---|---|
| =0 | Caller common not passed to callee |
| =1 | Caller common passed to callee |

CA Common access flag (bit 1):

| Entry | Indicates |
|---|---|
| =0 | Common passed with read/write access |
| =1 | Common passed with read-only access |

CW Callee wait flag (bit 2):

| Entry | Indicates |
|---|---|
| =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 | Indicates |
|---|---|
| =0 | Caller should be queued by priority for access to active callee |
| =1 | Call should be rejected if callee is active |

Parameter (Cont.) Definition (Cont.)

RP Register pass flag (bit 4):

| Entry | Indicates |
|---|---|
| =0 | Copy of caller registers not passed to callee |
| =1 | Copy of caller registers is passed to callee |

CPU CPU designation. Valid values are 0 to the number of processors in the configuration. The value 0 requests default assignment. Values 1 through n require the task execution to occur on processor number CPU (bits 8 through 15).

PRI Priority designation. Valid values for real-time jobs are 1 through 255, and for nonreal-time jobs are 10 through 239. 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 0 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 | Indicates |
|---|---|
| = -1 | Call was rejected (callee active) |
| = 0 | Call was successfully completed |

An example of a calling sequence is as follows:

```
LDD,R0      TID         Task name
LD,R2       LUTBL       Logical unit number
LD,R3       CONTRL
MON,R0      CALL        CALL
```

```
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/PRI,8/NPRMS
```

NOTES: 1/ A callee may not call its caller nor may 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 concurrent usage problem is to develop a third task, TASKC, through which all job I/O on the shared file is routed.

5/ Core 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 core 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.

## DWAIT, DEFERRED WAIT

A task that has called one or more callees and is executing concurrent 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) | ADDRESS+1 |
| | LE or TID$_2$ (0-3) | ADDRESS+2 |
| | TID$_2$ (4-7) | ADDRESS+3 |
| | LE or TID$_i$ (0-3) | ADDRESS+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 | Indicates |
|---|---|
| = -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 eight-character identifier of the returned task

LE — List end flag:

| Entry | Indicates |
|---|---|
| = -1 | The list has ended |
| ≠ -1 | Another TID specification begins |

TID — Eight-character string defining the name of a task

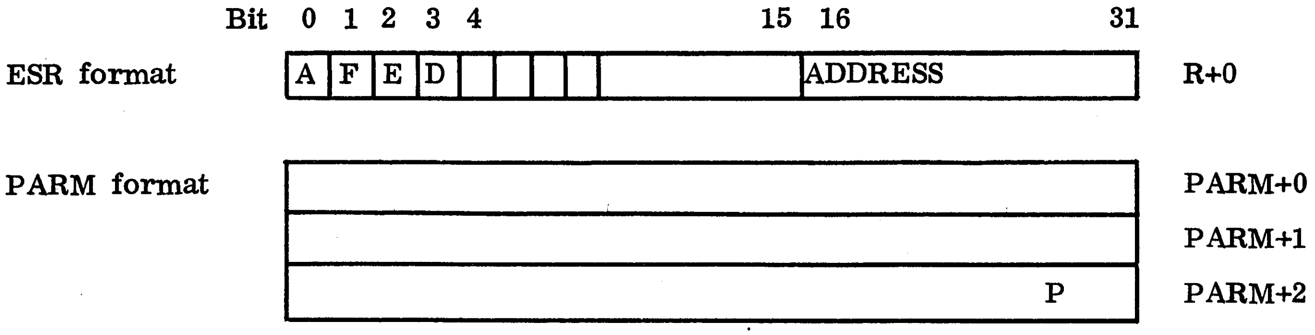
An example of a calling sequence is as follows:

```
          LDA,H1     TIDTBL       Address of list
          MON,H1     DWAIT        Call
          LD,R0      PARM         Check status
            .
            .
            .
TIDTBL    TEXT       8,TASK-ID1
          TEXT       8,TASK-ID2
          GEN        -1           List has ended
```

## ENABLE, ENABLE AND SELECT INTERRUPT CONTROL

The ENABLE ESR enables hardware detection of the arithmetic faults and defines the interrupt routine that will process the interrupts when they occur. MPX/OS provides a default interrupt processor that will abort the job. The user may select one interrupt for each fault, one interrupt routine for all faults or other combinations. The interrupt processor definition may be changed as often as desired but once the interrupt checking has been enabled, it cannot be disabled (see also PFAULT, Return Control on Program Faults in this section).

The ENABLE ESR does not receive a response from MPX/OS. The response format described applied to a return of control upon the occurrence of a fault.



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 — Sixteen-bit field containing address of interrupt processor:

| Entry | Indicates |
|---|---|
| =0 | MPX/OS default routine is used (job abort) |
| ≠0 | Task is placed in ready list with program counter set to ADDRESS. (ADDRESS applies to one or more fault conditions as selected by A, F, E, and D settings) |

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:

```
                    LD,R0          INTRMSK

                    MON,R0         ENABLE         CALL

                      •
                      •
                      •

         INTRMSK    VFD     1/1,1/0,1/1,1/0,12/0,16/INTADR    Go to INTADR on an arith-
                                                               metic fault or exponent fault
```

## 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 | R+1 |
| | | | | |
| PARM format | STATUS | or | ADDRESS(1) | PARM+0 |
| | | | ADDRESS(2) | PARM+1 |

Parameter | Definition

AREA

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

| Entry | Indicates |
|---|---|
| = -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:

Parameter (Cont.) Definition (Cont.)

| Entry | Indicates |
|---|---|
| = -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)] |
| > 0 | OPTION number of pages are added to the area selected by AREA flag [MPX/OS (MEM) response defines new limits of area altered] |

STATUS

Status of the ESR returned in PARM.

| Entry | Indicates |
|---|---|
| = -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, large address for program) and the last available address is the address most distant from the allocated space (large for common, small for program) - but still in the page already allocated.

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

An example of a calling sequence is as follows:

```
LDI,R0      0          Request common pages

LDI,R1      -1         All scheduled pages

MON,R0      OPENMEM    Call

LD,X7       PARM       Check status
```

## 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 may define one of the conditions and its return-of-control address.

ESR format

| | |
|---|---|
| ADDRESS | R+0 |
| FAULT | R+1 |

PARM format

| | |
|---|---|
| | PARM+0 |
| | PARM+1 |
| P | PARM+2 |

**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 | Indicates |
|---|---|
| =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:

```
LDA,R1      PFALTADR      Address for return
LDI,R2      1             An illegal instruction
MON,R1      PFAULT        Call
```

## 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 | R+1 |

PARM format

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

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

| Entry | Indicates |
|---|---|
| = -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 | Indicates |
|---|---|
| = -1 | Releases all common pages |
| = 0 | Returns memory limits only (see OPENMEM) |
| > 0 | Defines the number of pages to release |

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

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

Parameter (Cont.) Definition (Cont.)

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, large address for program), and the last available address 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:

```
LDI,R2     -1          Reduce program area
LDI,R3     2           Release two pages
MON,R2     RELMEM
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 | R+1 |

Parameter Definition

RELEASE The memory release flag:

| Entry | Indicates |
|---|---|
| = 0 | Release the task memory and clear task identification from the system |

| Parameter (Cont.) | Definition (Cont.) | |
|---|---|---|
| | **Entry (Cont.)** | **Indicates (Cont.)** |
| | = -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:

```
        LD,R0       0           Release flag

        LDI,R1      10          Pass back 10 words

        MON,R0      RETURN      Call
```

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.

## 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:

```
          LDI,R0      100         Time interval

          MON,R0      TSCHED      Call
```

## TSTATUS, RETURN TASK STATUS

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

ESR format

| | |
|---|---|
| TID (0-3) | R+0 |
| TID (4-7) | R+1 |

PARM format

| | |
|---|---|
| STATUS | PARM+0 |

<u>Parameter</u> — <u>Definition</u>

TID — Eight characters defining the identifier of the task, of 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 | Indicates |
|---|---|
| = -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 |

| Entry (Cont.) | Indicates (Cont.) |
|---|---|
| = 9 | TSCHED wait |
| = 10 | CRT wait |
| = 11 | TCT wait |

An example of a calling sequence is as follows:

```
          LDD,R0       ='TASKIDNT'      Task identifier

          MON,R0        TSTATUS         Call

          LD,H2         PARM            Check status
```

Refer back to Table 1-3 for descriptions of each status.

## MISCELLANEOUS ESRs

The following ESRs allow a task to communicate with the operator and obtain the date and time from the system.

### CTOC, SEND COMMAND MESSAGE TO OPERATOR

The CTOC ESR allows a task to send a message to the operator and requires a response. The message will not be accepted by MPX/OS if there is not room to display the message. If the message is accepted, the issuing task is assigned the I/O WAIT status. The task cannot resume execution until the operator responds to the message with either an Accept or Reject.

| | | |
|---|---|---|
| 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

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| STATUS | ESR status or operator response code: |

| Entry | Indicates |
|---|---|
| = -1 | CRT screen full or busy (ESR reject) |
| = 0 | Accept response from operator |
| = +1 | Reject response from operator |

An example of a calling sequence is as follows:

```
         LDCA,R0     ADDRMSG     Address of message
         MON,R0      CTOC        Call
         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 ESR will be rejected if MPX/OS cannot display the message (screen full or CRT busy). 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 03 (end-of-text, ETX) character value. |

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| STATUS | ESR status code: |

| Entry | Indicates |
|---|---|
| = -1 | CRT screen full or busy (ESR rejected) |
| = 0 | Message accepted and displayed |

An example of a calling sequence is as follows:

```
              LDCA,R3      ADDRMSG       Address of message
              MON,R3       CTOI          Call
              LD,X7        PARM          Check status
                      •
                      •
                      •
    ADDRMSG   TEXTC        18,  This is a comment
              GEN,C        $03
```

## DATE, RETURN CURRENT DATE

The DATE ESR obtains the current date in ASCII format.

PARM format

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

| Parameter | Definition |
|---|---|
| MM | ASCII codes for the month of the 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:

```
                    MON,RA          DATE            Call
```

## TETIME, TASK ELAPSED TIME

The TETIME ESR obtains the time in milliseconds accumulated from the time the task was called until the time the monitor call to TETIME was made. Each time the task is called the accumulated time is initialized at zero. 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:

```
                    MON,R0          TETIME            Call
```

## TIME, RETURN CURRENT TIME OF DAY

The TIME ESR obtains the current time of day from the executive in ASCII and binary formats and returns them in PARM, PARM+1, and PARM+2.

| | | | | | |
|---|---|---|---|---|---|
| PARM format | H | H | / | M | PARM+0 |
| | M | / | S | S | PARM+1 |
| | TIME | | | | PARM+2 |

| Parameter | Definition |
|---|---|
| HH | ASCII codes for hour of day (00 through 23) |
| / | ASCII code for slash graphic |
| MM | ASCII codes for minute of hour (00 through 59) |

| Parameter (Cont.) | Definition (Cont.) |
|---|---|
| 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:

```
        MON,R0      TIME      Call
```

# BLOCKER/DEBLOCKER 5

Logical I/O, referred to as blocker/deblocker, consists of library routines the user calls for transferring logical records to and from user-defined core 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 may 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).

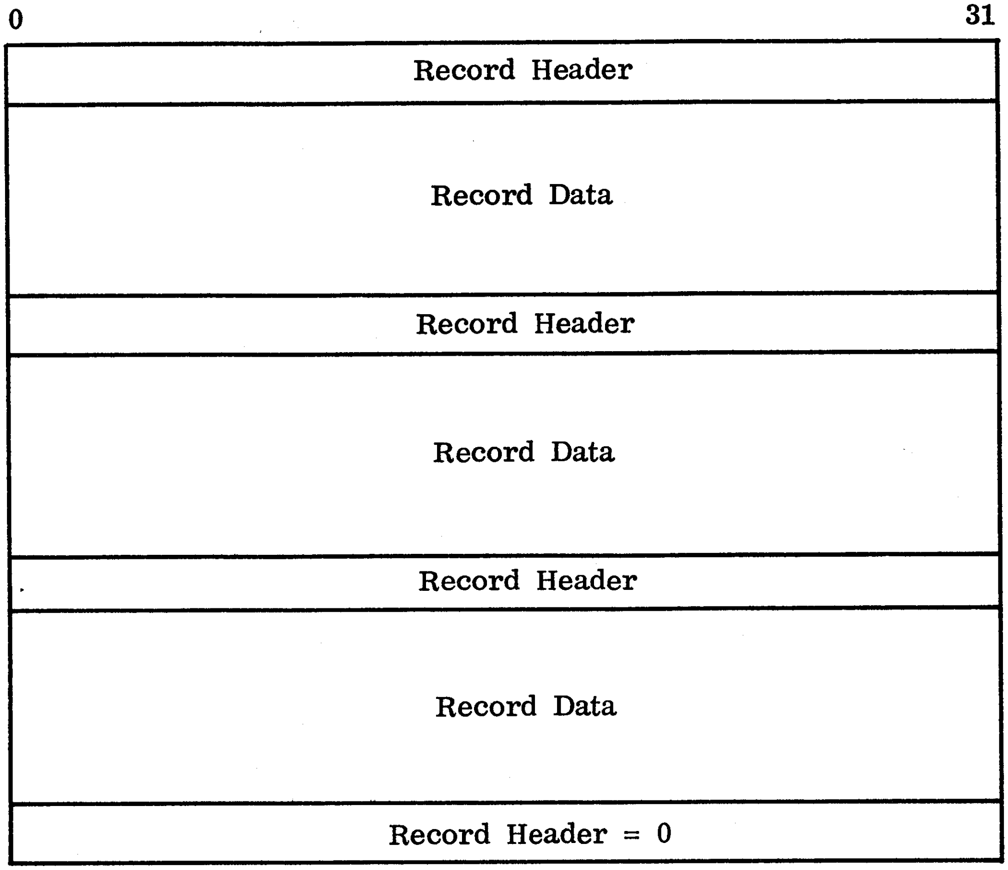
A block number parameter (BN) is required for certain blocker/deblocker functions on mass storage files. Thus a user may, if he wishes, access a mass storage file randomly with blocker/deblocker.

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

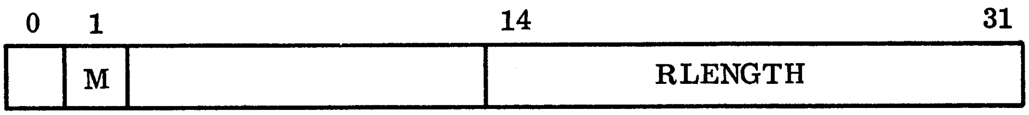
## BLOCK AND BUFFER FORMATS

### 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:

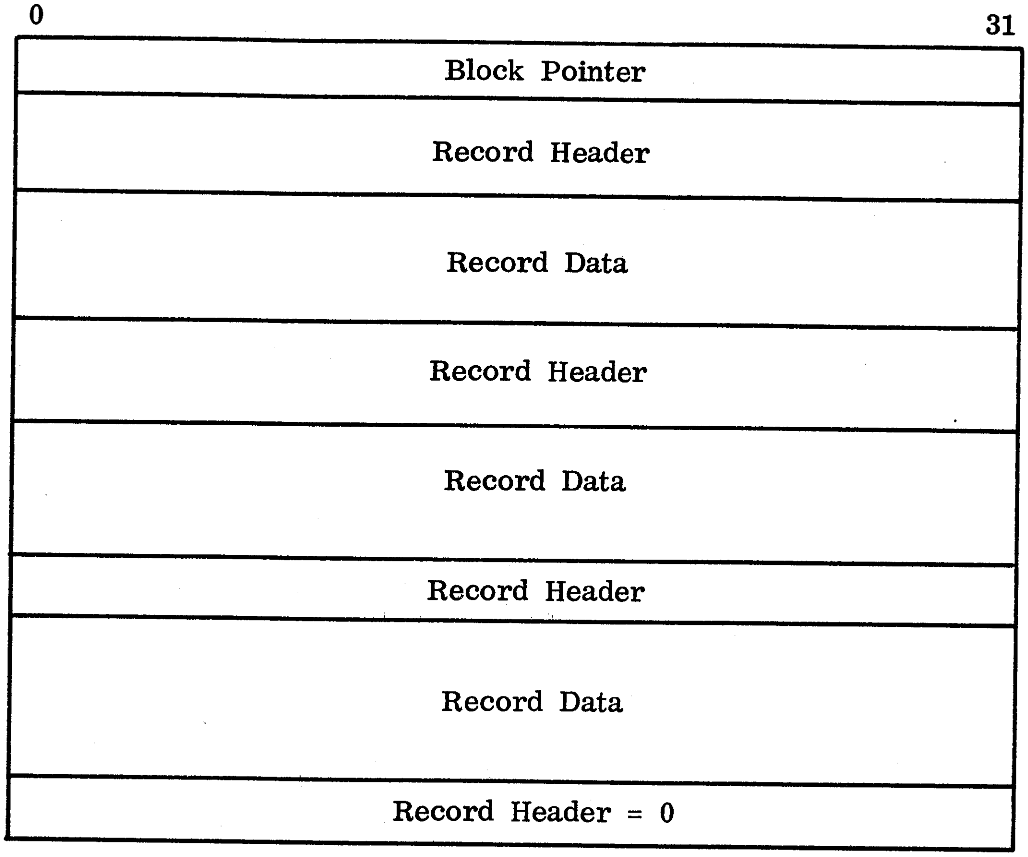


M - Mode of record data (bit 1)

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).

## RECORD DEVICES

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:

| 0 31 |
|---|
| Record Header |
| Record Data |
| Record Trailer = 0 |

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:

| 0 31 |
|---|
| Block Pointer |
| Record Header |
| Record Data |
| Record Trailer = 0 |

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 must have been previously allocated and opened. Unit record devices must be equipped.

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:

| | 0 15 | 16 | | 24 31 |
|---|---|---|---|---|
| RB | | B | | LUN |
| RC | | BFWA | | |
| RD | | BLENGTH | | |
| RE | BN | | | |
| RF | | RETURN ADDRESS | | |

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 < 0, file is positioned to highest block written +1
BN = 0, file is not positioned
BN > 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 may 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,RC        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 removes trailing zeros (binary record) or trailing blanks (ASCII record) from the record data. 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:

| | 0 15 | 16 | | 24 31 |
|---|---|---|---|---|
| RB | | M | | LUN |
| RC | | RFBA | | |
| RD | | RLENGTH | | |
| RE | | | | |
| RF | | RETURN ADDRESS | | |

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

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 OUTFIT

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:

| | 0 | 16 | 24 | 31 |
|---|---|---|---|---|
| RB | | | LUN | |
| RC | | | | |
| RD | | | | |
| RE | BN | | | |
| RF | | RETURN ADDRESS | | |

LUN - Logical unit number

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

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:

| | 0 | 16 | 24 31 |
|---|---|---|---|
| RB | | | LUN |
| RC | | | |
| RD | | | |
| RE | | | |
| RF | | RETURN ADDRESS | |

LUN - Logical unit number

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:

| | 0 15 | 16 | 24 31 |
|---|---|---|---|
| PARM | ES | | EI |
| PARM+1 | | BN | |

EI - Error indicator

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

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 |

| Routine | Block Type Device | Record Type Device |
|---|---|---|
| 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 00 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:

| | 0 15 | 16 | 24 31 |
|---|---|---|---|
| RB | | B | LUN |
| RC | | BFWA | |
| RD | | BLENGTH | |
| RE | BN | | |
| RF | | RETURN ADDRESS | |

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 ≤ 0, file is not positioned
BN > 0, file is positioned to specified block

RETURN
ADDRESS - Address in user's program in 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:

| | 0 | 14 | 16 | 24 | 31 |
|---|---|---|---|---|---|
| RB | | | | LUN | |
| RC | | RFBA | | | |
| RD | | RLENGTH | | | |
| RE | | | | | |
| RF | | | RETURN ADDRESS | | |

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

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:

| | 0 | 16 | 24 | 31 |
|---|---|---|---|---|
| RB | | | LUN | |
| RC | | | | |
| RD | | | | |
| RE | BN | | | |
| RF | | RETURN ADDRESS | | |

LUN - Logical unit number

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

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:

| | 0 | 16 | 24 31 |
|---|---|---|---|
| RB | | | LUN |
| RC | | | |
| RD | | | |
| RE | BN | | |
| RF | | RETURN ADDRESS | |

LUN - Logical unit number

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:

| | 0 | 12 | 15 | 16 | 24 31 |
|---|---|---|---|---|---|
| PARM | | M | | | EI |
| PARM+1 | | | | BN | |

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 $\neq$ 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 |

# MPX LOADER 6

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 pages 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.

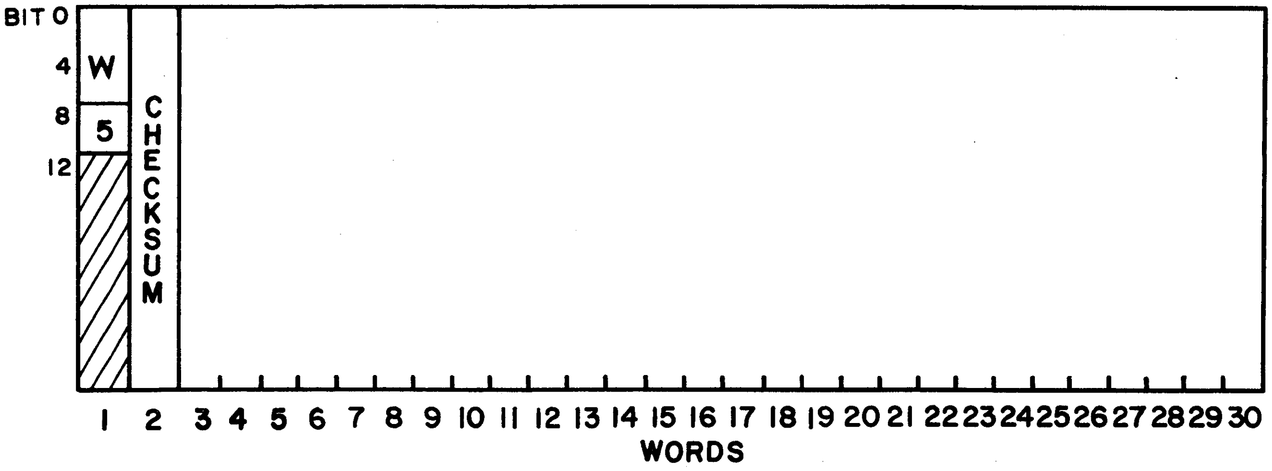
## LOADER CARDS

The loader accepts the binary cards produced by assemblers 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 binard card

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

Word 2

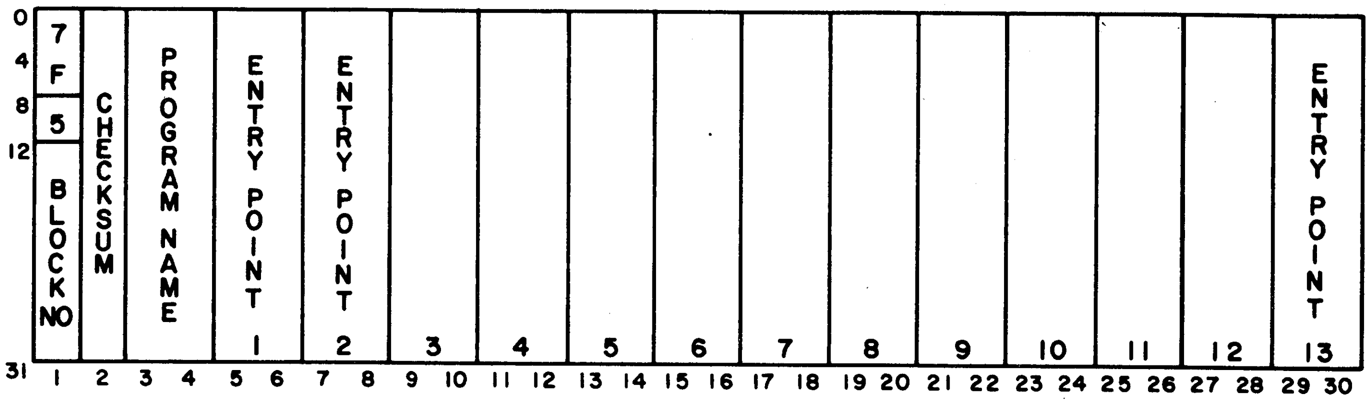
32-bit 2's complement sum of all other words contained on the card

# 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

Bit 10, if set. Library routine is absolute

Bits 12 through 31 = Block number of next LIB entry. Zero indicates end of LIB

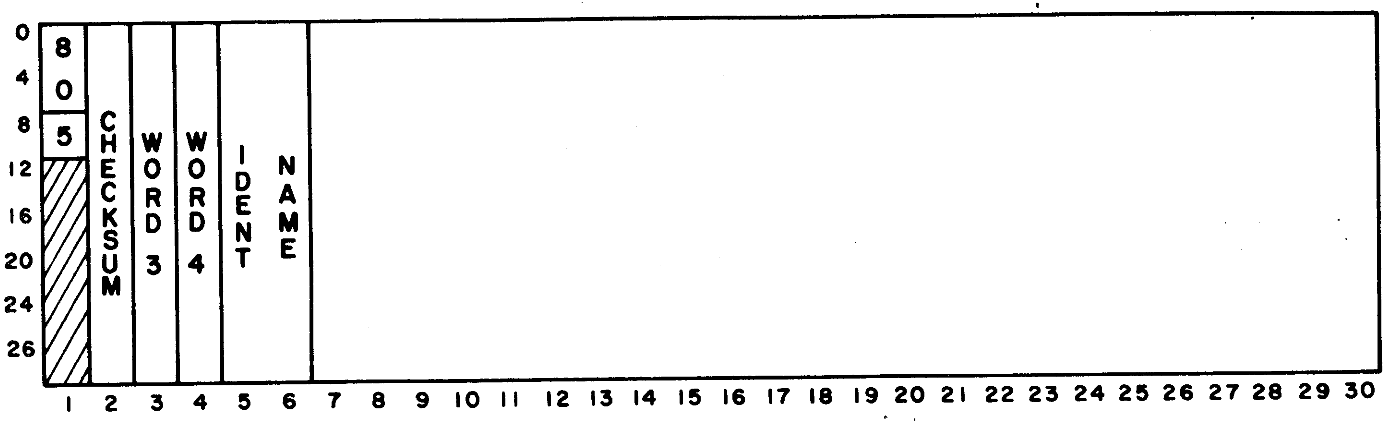
## Words 3 and 4

Eight-character program name

## Words 5 and 6

Eight-character entry point name

## IDENTIFICATION CARD



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

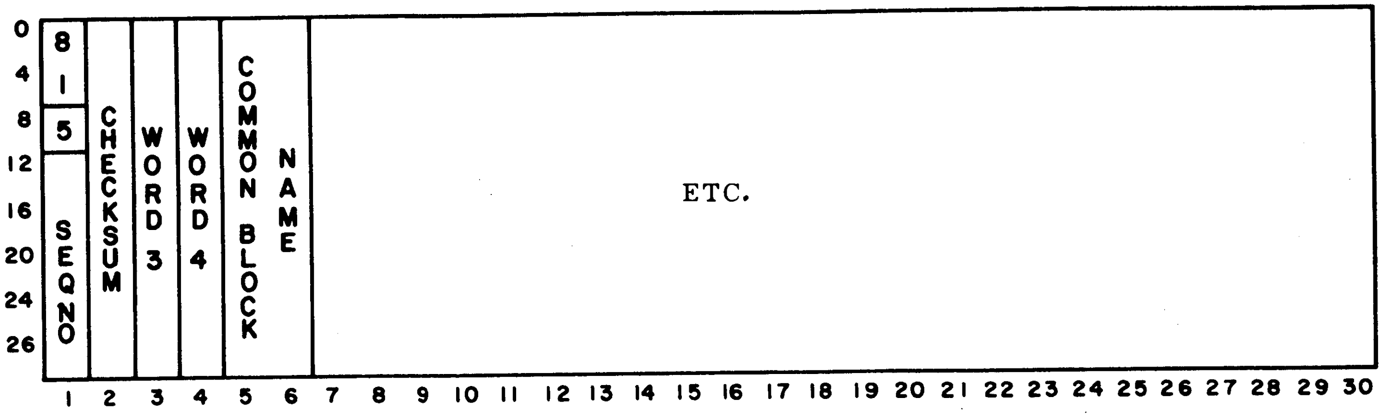
Bits 6 and 7 = Addressing type

Bits 11 through 31 = End address

Words 5 and 6

Name in ASCII codes, left adjusted

## BLOCK COMMON TABLE CARD



Word 1

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

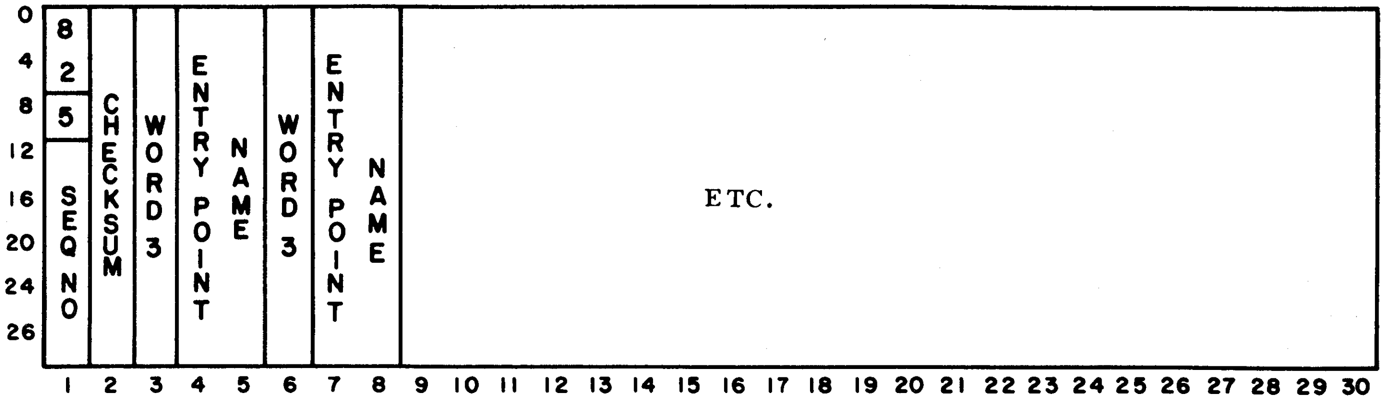
Common block name in ASCII codes, left justified

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 0

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

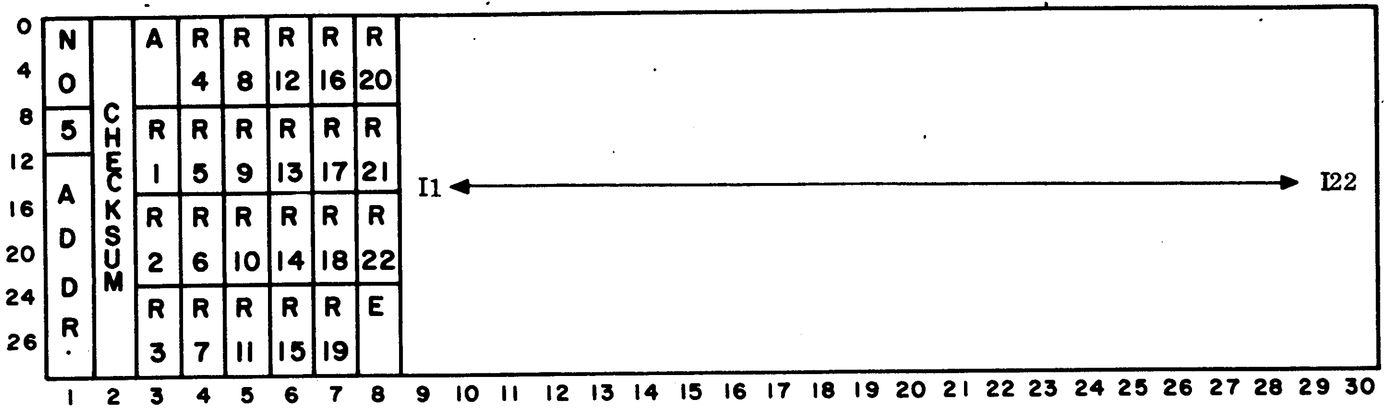
Bits 11 through 31 = Address of entry point

Words 4 and 5

Entry-point name in ASCII codes, left justified

NOTE: The information content of the card image is terminated by a zeros 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

Bit 0 = 1, negative relocation

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)

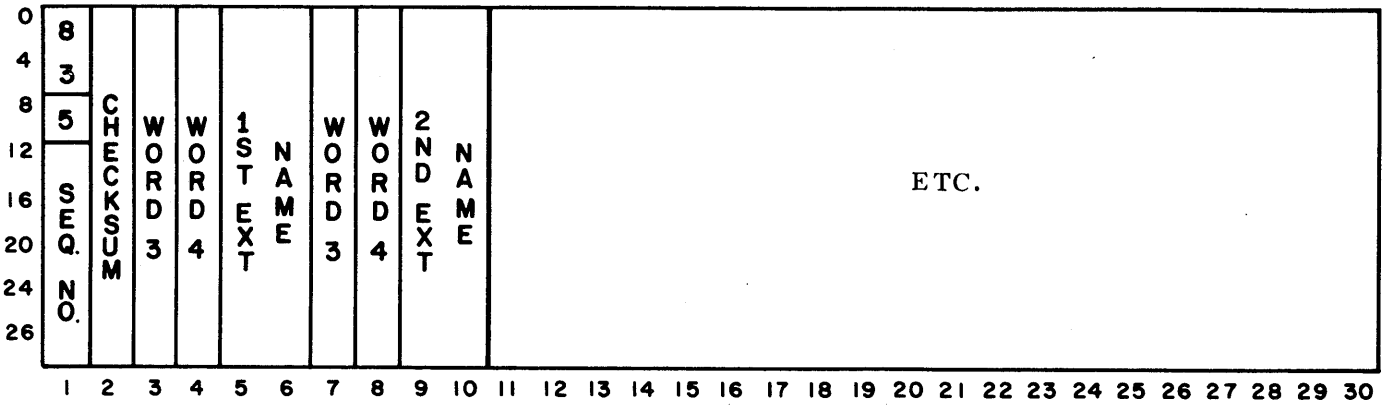
Word 8

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

Loadable information

## EXTERNAL CARD



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

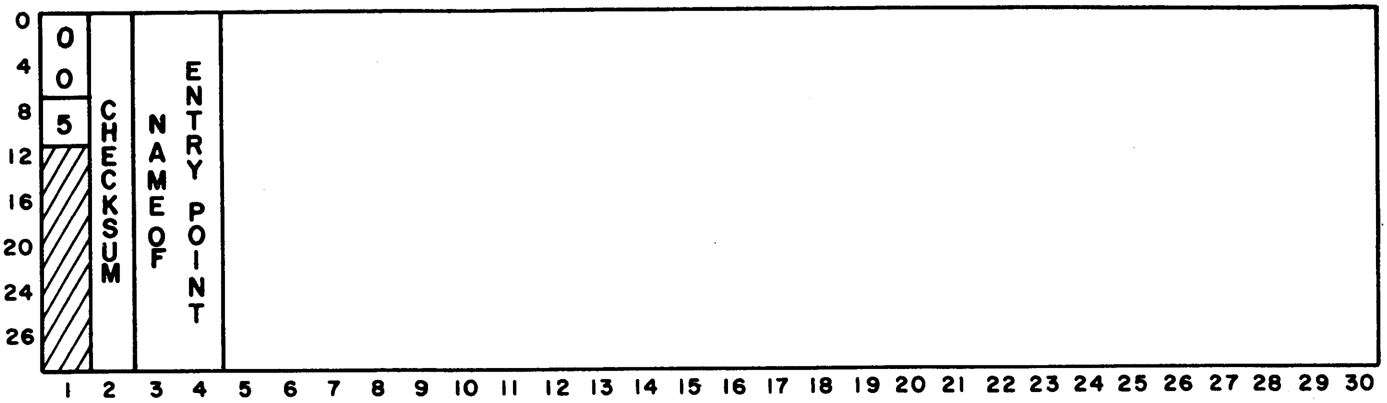
Bit 0 = 1, negative additive

Bits 11 through 31 = Additive

Words 5 and 6

External name in ASCII codes, left justified

## TRANSFER ADDRESS CARD



Checksum is running checksum of IDC through TRA cards.

Words 3 and 4

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

## HEXADECIMAL CORRECTION CARDS

*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:

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

| Location Contents (Cont.) | Interpretation (Cont.) |
|---|---|
| (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.<br>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 factor, 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 may take any of the following forms:

| Relocation Factor* | Interpretation |
|---|---|
| No relocation factor | Correction is stored as absolute correction. |
| (Subprogram name) | Relocate the word address portion of the correction relative to the address of the first location in the subprogram enclosed in parentheses. |

*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* (Cont.) | Interpretation (Cont.) |
|---|---|
| (D/data block)<br>(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. Location 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   ASCIIINP   EA46   ASCI IOUT  E861   CONTROL    EE19
FORMAT     E528   Q8QERROR   E4E0   Q8QSTP     E4CC   AMATHER    E49E
BIKDEBLK   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   0064
BKSP       0105   BSY        013C   CALL       011D   CLOSE      0110
CTOC       C115   CTOI       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
JCIJSCRL   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   Q8QEXI IS  E65C   Q8QIFRMT   E528
Q8QINDEC   EA58   Q8QINENC   E872   Q8QINGIN   EA46   Q8QINGOT   E861
Q8QIOINT   E619   Q8QIOTAB   E732   Q8QISCAN   E53D   Q8QITERM   E539
Q8QLGIN1   EA67   Q8QLGIN2   EA6B   Q8QLGOT1   E882   Q8QLGOT2   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

*RUN
```

Figure 6-1. MAP Example

# CHARACTER SET

A

The following table illustrates the MP-60 system character set for certain types of peripheral equipment. The MP-60 Internal Code column shows the hexadecimal byte code (8 bits) sent to or received from a peripheral device in ASCII mode. The Card Punch 026 Code column shows the Hollerith code punched for the MP-60 code. The Printer Graphic column shows the print character for a CONTROL DATA® 595-4 Print Chain. The Card Reader Code column shows the transmitted code for the corresponding 026 code. Only codes 20 through 5F are defined for the card reader. The Console CRT Graphic column shows central codes and graphic characters (refer to Appendix E of the MP-60 Computer System Reference Manual, CDC Publication No. 14306500). The ASCII Character column shows the relationship between MP-60 internal codes and the standard definition.

| MP-60 Internal Code | Card Punch 026 Code | Printer Graphic | Card Reader Code | Console CRT Graphic | ASCII Character |
|---|---|---|---|---|---|
| 00 | 12 0 9 8 1 | Blank | None | Control | NUL |
| 01 | 12 9 1 | | | Codes | SOH |
| 02 | 12 9 2 | | | | STX |
| 03 | 12 9 3 | | | | ETX |
| 04 | 9 7 | | | | EOT |
| 05 | 0 9 8 5 | | | | ENQ |
| 06 | 0 9 8 6 | | | | ACK |
| 07 | 0 9 8 7 | | | | BEL |
| 08 | 11 9 6 | | | | BS |
| 09 | 12 9 5 | | | | HT |
| 0A | 0 9 5 | | | | LF |
| 0B | 12 9 8 3 | | | | VT |
| 0C | 12 9 8 4 | | | | FF |
| 0D | 12 9 8 5 | | | | CR |
| 0E | 12 9 8 6 | | | | SO |
| 0F | 12 9 8 7 | | | | SI |
| 10 | 12 11 9 8 1 | | | | DLE |
| 11 | 11 9 1 | | | | DC1 |
| 12 | 11 9 2 | | | | DC2 |
| 13 | 11 9 3 | | | | DC3 |
| 14 | 9 8 4 | | | | DC4 |
| 15 | 9 8 5 | | | | NAK |
| 16 | 9 2 | | | | SYN |
| 17 | 0 9 6 | | | | ETB |
| 18 | 11 9 8 | | | | CAN |
| 19 | 11 9 8 1 | | | | EM |
| 1A | 9 8 7 | | | | SUB |
| 1B | 0 9 7 | | | | ESC |
| 1C | 11 9 8 4 | | | | FS |
| 1D | 11 9 8 5 | | | | GS |
| 1E | 11 9 8 6 | | | Control | RS |
| 1F | 11 9 8 7 | Blank | None | Codes | US |

| MP-60 Internal Code | Card Punch 026 Code | Printer Graphic | Card Reader Code | Console CRT Graphic | ASCII Character |
|---|---|---|---|---|---|
| 20 | Blank | Blank | 20 | Space | Space |
| 21 | 12 8 7 | ! | 21 | ! | ! |
| 22 | 8 7 | " | 22(1) | " | " |
| 23 | 8 6 | # | 23 | # | # |
| 24 | 11 8 3 | $ | 24 | $ | $ |
| 25 | 12 8 5 | % | 25 | % | % |
| 26 | 12 8 6 | & | 26 | & | & |
| 27 | 8 4 | ' | 27 | ' | ' |
| 28 | 0 8 4 | ( | 28 | ( | ( |
| 29 | 12 8 4 | ) | 29 | ) | ) |
| 2A | 11 8 4 | * | 2A | * | * |
| 2B | 12 | + | 2B | + | + |
| 2C | 0 8 3 | , | 2C | , | , |
| 2D | 11 | - | 2D | - | - |
| 2E | 12 8 3 | . | 2E | . | . |
| 2F | 0 1 | / | 2F | / | / |
| 30 | 0 | 0 | 30 | 0 | 0 |
| 31 | 1 | 1 | 31 | 1 | 1 |
| 32 | 2 | 2 | 32 | 2 | 2 |
| 33 | 3 | 3 | 33 | 3 | 3 |
| 34 | 4 | 4 | 34 | 4 | 4 |
| 35 | 5 | 5 | 35 | 5 | 5 |
| 36 | 6 | 6 | 36 | 6 | 6 |
| 37 | 7 | 7 | 37 | 7 | 7 |
| 38 | 8 | 8 | 38 | 8 | 8 |
| 39 | 9 | 9 | 39 | 9 | 9 |
| 3A | 8 2 | : | 3A | : | : |
| 3B | 11 8 6 | ; | 3B | ; | ; |
| 3C | 11 8 5 | < | 3C | < | < |
| 3D | 8 3 | = | 3D | = | = |
| 3E | 0 8 6 | > | 3E | > | > |
| 3F | 0 8 7 | ? | 3F | ? | ? |

(1) Interpreted as end of file in column 1

| MP-60 Internal Code | Card Punch 026 Code | Printer Graphic | Card Reader Code | Console CRT Graphic | ASCII Character |
|---|---|---|---|---|---|
| 40 | 8 5 | @ | 40 | @ | @ |
| 41 | 12 1 | A | 41 | A | A |
| 42 | 12 2 | B | 42 | B | B |
| 43 | 12 3 | C | 43 | C | C |
| 44 | 12 4 | D | 44 | D | D |
| 45 | 12 5 | E | 45 | E | E |
| 46 | 12 6 | F | 46 | F | F |
| 47 | 12 7 | G | 47 | G | G |
| 48 | 12 8 | H | 48 | H | H |
| 49 | 12 9 | I | 49 | I | I |
| 4A | 11 1 | J | 4A | J | J |
| 4B | 11 2 | K | 4B | K | K |
| 4C | 11 3 | L | 4C | L | L |
| 4D | 11 4 | M | 4D | M | M |
| 4E | 11 5 | N | 4E | N | N |
| 4F | 11 6 | O | 4F | O | O |
| 50 | 11 7 | P | 50 | P | P |
| 51 | 11 8 | Q | 51 | Q | Q |
| 52 | 11 9 | R | 52 | R | R |
| 53 | 0 2 | S | 53 | S | S |
| 54 | 0 3 | T | 54 | T | T |
| 55 | 0 4 | U | 55 | U | U |
| 56 | 0 5 | V | 56 | V | V |
| 57 | 0 6 | W | 57 | W | W |
| 58 | 0 7 | X | 58 | X | X |
| 59 | 0 8 | Y | 59 | Y | Y |
| 5A | 0 9 | Z | 5A | Z | Z |
| 5B | 12 0 | { | 5B | [ | [ |
| 5C | 0 8 2 | \ | 5C | \ | \ |
| 5D | 11 0 | } | 5D | ] | ] |
| 5E | 11 8 7 | ^ | 5E | ^ | ^ |
| 5F | 0 8 5 | _ | 5F | _ | _ |

| MP-60 Internal Code | Card Punch 026 Code | Printer Graphic | Card Reader Code | Console CRT Graphic | ASCII Character |
|---|---|---|---|---|---|
| 60 | 8 1 | Blank | None | \ | \ |
| 61 | 12 0 1 | | | a | a |
| 62 | 12 0 2 | | | b | b |
| 63 | 12 0 3 | | | c | c |
| 64 | 12 0 4 | | | d | d |
| 65 | 12 0 5 | | | e | e |
| 66 | 12 0 6 | | | f | f |
| 67 | 12 0 7 | | | g | g |
| 68 | 12 0 8 | | | h | h |
| 69 | 12 0 9 | | | i | i |
| 6A | 12 11 1 | | | j | j |
| 6B | 12 11 2 | | | k | k |
| 6C | 12 11 3 | | | l | l |
| 6D | 12 11 4 | | | m | m |
| 6E | 12 11 5 | | | n | n |
| 6F | 12 11 6 | | | o | o |
| 70 | 12 11 7 | | | p | p |
| 71 | 12 11 8 | | | q | q |
| 72 | 12 11 9 | | | r | r |
| 73 | 11 0 2 | | | s | s |
| 74 | 11 0 3 | | | t | t |
| 75 | 11 0 4 | | | u | u |
| 76 | 11 0 5 | | | v | v |
| 77 | 11 0 6 | | | w | w |
| 78 | 11 0 7 | | | x | x |
| 79 | 11 0 8 | | | y | y |
| 7A | 11 0 9 | | | z | z |
| 7B | 12 0 | | | { | { |
| 7C | 12 11 | | | ¦ | ¦ |
| 7D | 11 0 | | | } | } |
| 7E | 11 0 1 | | | ~ | ~ |
| 7F | 12 9 7 | Blank | None | Del | Del |

# GLOSSARY

B

Abort — The premature termination of a process whenever an irrecoverable situation (either hardware of 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).

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 core 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.

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 may be shared between programs. Tasks may communicate through common areas.

Data — An area of memory that may be prestored 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.

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.

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 reentrant 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 may 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.

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 core memory. Paging is a technique where a logical address is transformed via a set of page registers to a physical address.

Post Processor

A system task that spools the standard output of a job.

Preprocessor

A system task that spools the standard input of a job.

Priority

A value (0 to 255) 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).

Reentrant

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 core memory.

Return

A monitor function that terminates a task and transfers control to the point in the caller where the call originated. A task may return with or without release of memory.

Schedule

The process of placing a task on the ready list by priority. A task may 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 (see Preprocessor and Post Processor).

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.).

System Initialize

Refers to the initial system load process where the resident is loaded, memory initialized, and the input preprocessor task is started.

Task

An independent unit of work that can compete for the resources of the system. A task may call and be called by other tasks.

Task Control Table (TCT)

An area of memory containing information used to control a task.

Terminate

The process of completing a job. A job may terminate normally or abnormally.

Thread

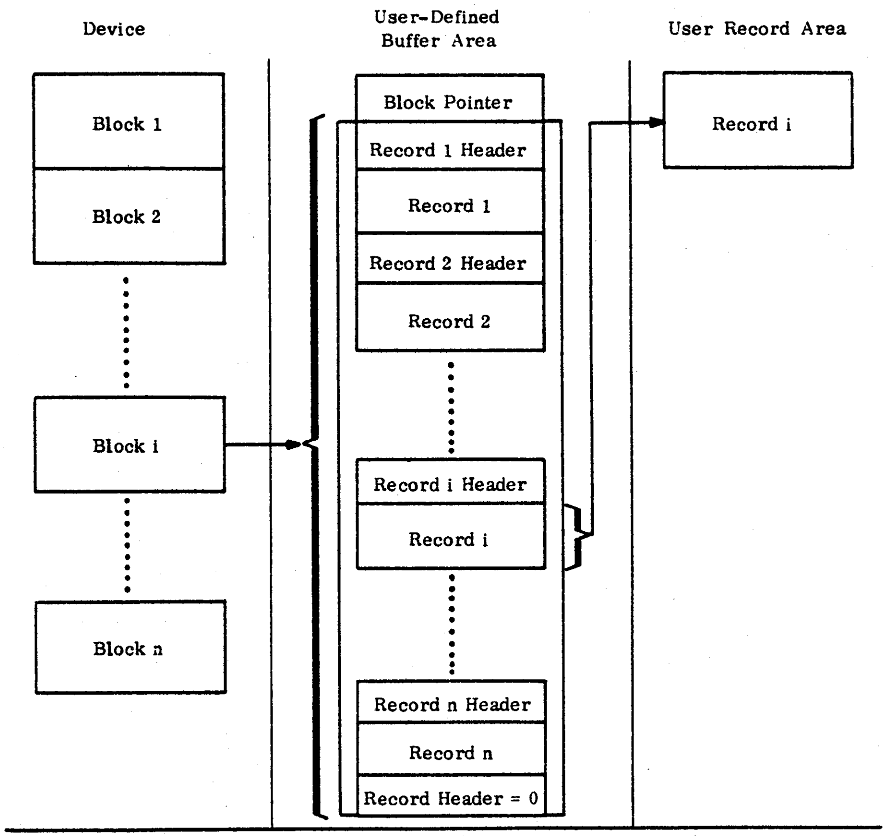
A linked list of elements, the contents of each thread cell contains the address of the next thread cell and so on until a thread cell of zero which indicates the end of the list.

Utility

A routine or procedure that supports the operation of a system (e.g., an I/O transfer routine).

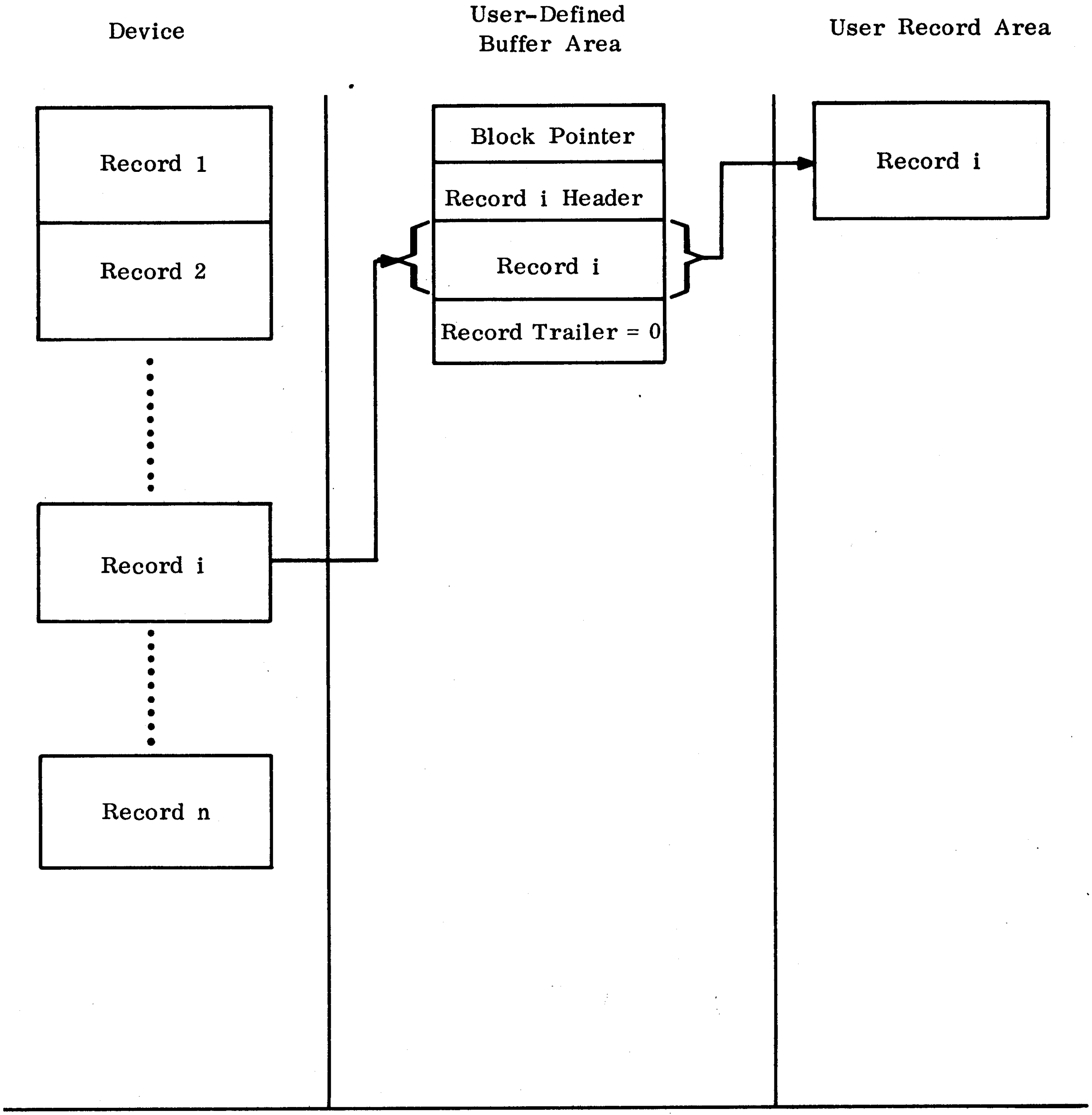
# BLOCKER/DEBLOCKER

The following diagram 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.



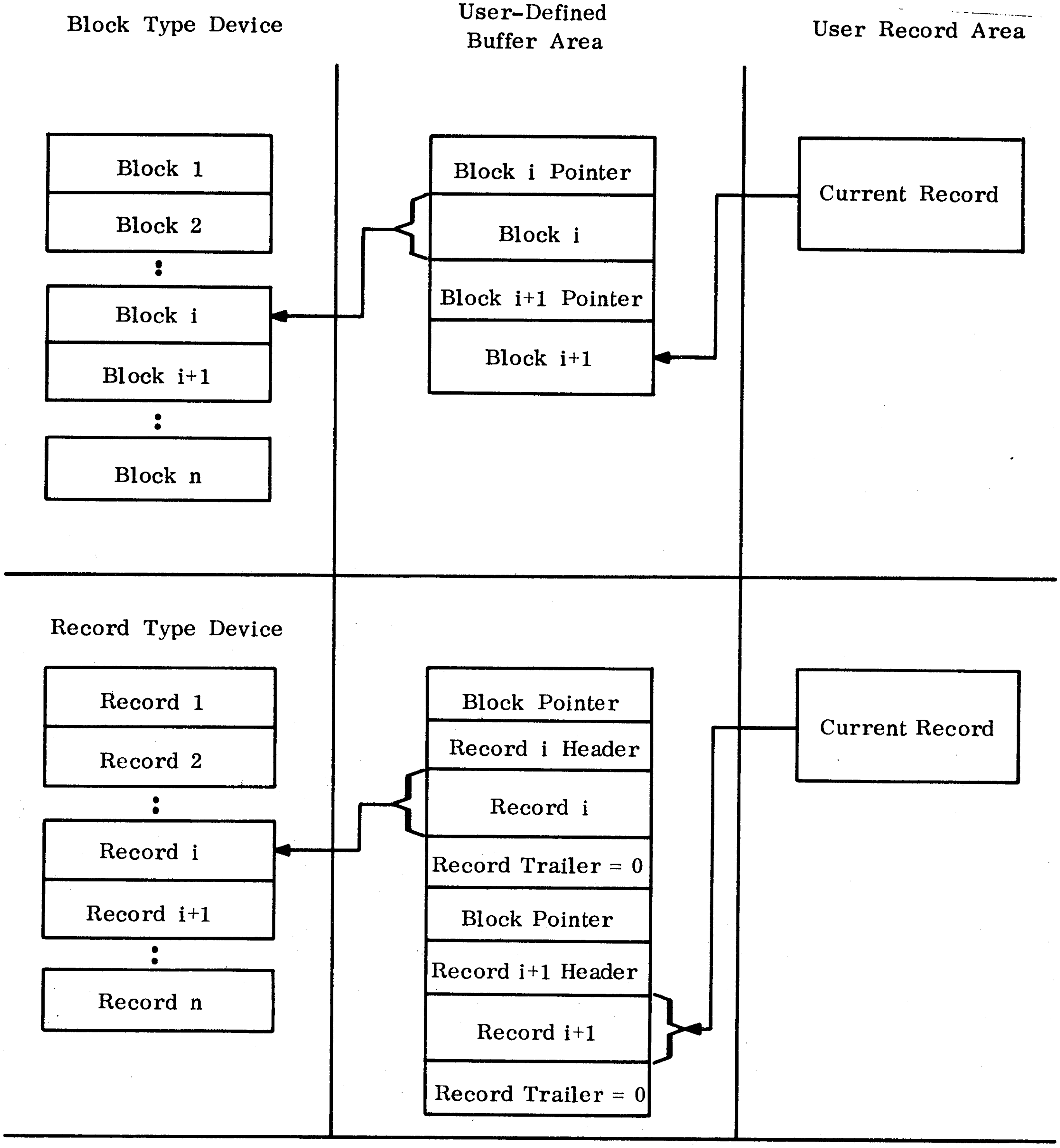
Blocker/Deblocker
Record/Buffer Formats
Block Type Device

The following diagram illustrates the relationship of user records, buffers, and physical records (i.e., 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.



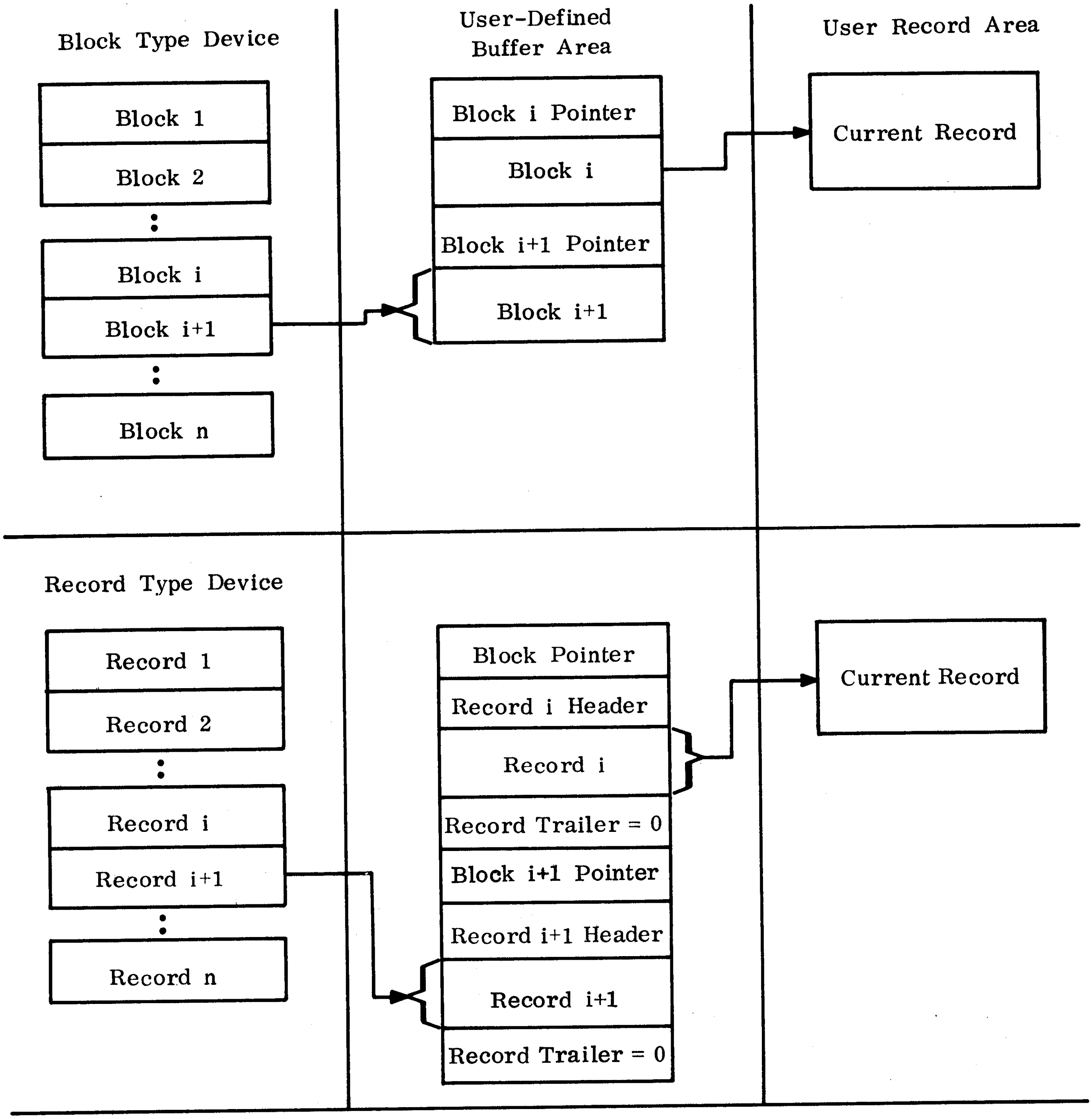
Blocker/Deblocker
Record/Buffer Formats
Record Type Device

The following diagram 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.



Blocker
Double Buffering

The following diagram 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.



Deblocker
Double Buffering

# SYSTEM ERRORS CODE DEFINITIONS D

## 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 | IOC reject |
|  | 7 | An input or output was attempted upon a protected page |
|  | 8 | Illegal logical unit number |
|  | 9 | Illegal command |
| 2 | YY | Operator abort |
|  | 1 | Operator aborted the job from the console CRT |
|  | 2 | Job time limit expired |
| 3 | YY | Page fault |
|  | 1 | Read-only violation |
|  | 2 | Protect violation |
| 4 | YY | Memory problem |
|  | 1 | Memory parity error - instruction |
|  | 2 | Memory reject - instruction |

| Abort Type | Abort Code | Description |
|---|---|---|
| | 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 |
| 7 | YY | Voluntary abort |
| | 1 | User's program made a monitor call to ABORT |
| 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 |

| Abort Type | Abort Code | Description |
|---|---|---|
| | 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 |
| 13 | YY | Loader error |
| 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 | 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. 02 WORDS 1 AND 2 = XXXXXXXXXXXXXXXX
```

The error conditions and the actions taken are described below:

| Error No. | Cause | Action<br>C = Continue<br>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 overlays 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 |
| 0014 | Second SCOM of same name greater than first SCOM | A |
| 0015 | Second DCOM of same name greater than first DCOM | A |

| Error No. | Cause | Action<br>C = Continue<br>A = Abort |
|---|---|---|
| 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

When a file manager control card error is encountered, a diagnostic is output on the OUT file, and the job processing continues.

When a file manager call error is detected, an error code is returned to the parameter area, PARM. PARM must be defined as an external in the user's program. An error code of 0 specifies normal processing.

## CONTROL CARD ERROR DIAGNOSTICS

The format of the control card error diagnostic is:

```
FILE MANAGER ERROR XX
```

where XX is the error code.

## FILE MANAGER ERROR CODES

| 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 |

*Consult system analyst

| Error Code | Description |
|---|---|
| 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 |
| 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 |

*Consult system analyst

## BLOCKER ERROR INDICATORS

| Indicator | Routine | Description |
|---|---|---|
| 1 | PACK, PACKO, PACKC | Write attempted beyond file limits |
| 2 | PACKD, PACK<br>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) 62 |
| 12 | PACK, PACKO, PACKC | Irrecoverable I/O error |
| 13 | PACK | Block pointer out of bounds; BLOCKER/DEBLOCKER pointers have been modified |
| 14 | PACK | Record length = 0 |

## DEBLOCKER ERROR INDICATORS

| Indicator | Routine | Description |
|---|---|---|
| 1 | PICKD, PICK, PICKI | End-of-file |
| 2 | PICKD, 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 1-63) |
| 12 | PICKD, PICK | Irrecoverable I/O error |
| 13 | PICK | Block pointer out of bounds; BLOCKER/DEBLOCKER pointers have been modified |
| 14 | PICK, PICKI | Record length = 0 |

# ERROR RECOVERY

MPX/OS performs error recovery on standard peripheral units when an error is detected during data transmission. If the error is not recoverable, the status indicates the error type (see subtitle, "UST, Unit Status Test" in Section 3). Error recovery for specific error conditions is a function of the device and the command issued. Standard error recovery by device is given in the following:

| Device | Error | Command | Procedure |
|---|---|---|---|
| Disk | Memory | All | 1. Repeat operation three times; if error persists, it is irrecoverable. |
| | Data | Read | 1. Repeat operation three times; if error persists, it is irrecoverable. |
| | Hardware | All | 1. Irrecoverable. |
| | Lost data | All | 1. Repeat operation three times; if error persists, it is irrecoverable. |
| | Address | All | 1. Locate to sector 0.<br>2. Repeat operation three times; if error persists, it is irrecoverable. |
| | Seek | All | 1. Repeat operation three times; if error persists, it is irrecoverable. |
| Magnetic tape | Memory | Read/Write | 1. Backspace one record.<br>2. Repeat operation.<br>3. Repeat procedures 1 and 2 three times; if error persists, it is irrecoverable. |

| Device | Error | Command | Procedure |
|---|---|---|---|
| | | Erase | 1. Irrecoverable. |
| | Data | Read/Write/ Write Tape Mark | 1. Backspace one record.<br>2. Repeat operation.<br>3. Repeat procedures 1 and 2 three times.<br>4. If error persists, backspace three records; check for load point after each backspace (irrecoverable if loadpoint).<br>5. Skip forward two records and repeat operation.<br>6. If error persists repeat procedures 1 through 5 three times; if error persists, it is irrecoverable. |
| | | Erase | 1. Irrecoverable. |
| | Hardware | All | 1. Irrecoverable. |
| | Lost data | Read/Write | 1. Backspace one record.<br>2. Repeat operation.<br>3. Repeat procedures 1 and 2 three times; if error persists, it is irrecoverable. |
| | Write protect | Write/Write Tape Mark/ Erase | 1. Request that the operator enable write function; the operator responds with accept or reject. |
| | End-of-tape | Write/Write Tape Mark/ Erase | 1. Request that the operator mount a new reel. |
| | | Read/SEOF | 1. Request that the operator mount a new reel. |
| Card reader | Memory | Read | 1. Irrecoverable. |
| | Data | Read | 1. Request operator to recycle card.<br>2. Repeat operation. |

| Device | Error | Command | Procedure |
|---|---|---|---|
|  | Hardware | Read | 1. Irrecoverable. |
|  | Feed failure | Read | (See Not Ready.) |
|  | Input tray empty | Read | (See Not Ready.) |
| Card punch | Memory | Write | 1. Repeat operation three times; if error persists, it is irrecoverable. |
|  | Stacker full | Write | 1. Notify operator. (No repeat necessary.) |
|  | Hardware | Write | 1. Irrecoverable. |
|  | Hopper empty | Write | (See Not Ready.) |
| Printer | Memory | Write | 1. Irrecoverable. |
|  | Data | Write | 1. Notify operator. (No repeat.) |
|  | Hardware | Write | 1. Irrecoverable. |
|  | Paper fault | Write | 1. Notify operator. (No repeat.) |
| CRT | Memory | Read/Write | 1. Irrecoverable. |
| Teletypewriter | Memory | Read/Write | 1. Irrecoverable. |
|  | Data | Read/Write | 1. Irrecoverable. |
|  | Hardware | Read/Write | 1. Irrecoverable. |
| All | Not ready | All | 1. Request operator ready unit.<br>2. Repeat operation. |
|  | Reject (only) | All | 1. Thread request against IOC.<br>2. Repeat operation on time basis. |

# MASS STORAGE DEVICES F

## 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 characteristics:

| | |
|---|---|
| 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 1867-1 DISK UNIT

The Control Data 9760 Disk Unit contains one removable device. Each device has the following characteristics:

| | |
|---|---|
| Sector size | 48 words |
| Track size | 64 sectors |
| Number of tracks | 2055 tracks |
| Allocation unit size | 1 track |
| Capacity | 25 Mega-byte |
| Cylinder size | 5 tracks |

## CONTROL DATA 1867-2 DISK UNIT

The Control Data 9762 Disk Unit contains one removable device. Each device has the following characteristics:

| | |
|---|---|
| Sector size | 48 words |
| Track size | 64 sectors |
| Number of tracks | 4110 tracks |
| Allocation unit size | 2 tracks |
| Capacity | 50 Mega-byte |
| Cylinder size | 10 tracks |

# MASS STORAGE LABELS G

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.

# DEVICE LABEL

| Word | | |
|---|---|---|
| 0 | DEVICE IDENT | |
| 1 | | |
| 2 | | LBLLSL |
| 3 | | LBLBS |
| 4 | | |
| 5 | CHECKSUM | |
| 6 | | |
| | DEVICE STORAGE ALLOCATION MAP | |
| 79 | | |

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 | 0–7 | 8–10 | 11 | 12–15 | 16–31 |
|---|---|---|---|---|---|
| 0 | FILE NAME | | | | |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | EDITION |
| 4 | OWNER | | | | |
| 5 | ACCESS KEY | | | | |
| 6 | SPARE | | | | |
| 7 | CHECKSUM | | | | |
| 8 | SPARE | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | SC | P | | | LBN |
| 16 | NAB | | | | NHRPB |
| 17 | BS | | | | NBN |
| 18 | DT | DC | | | BC |
| 19 | DEVICE IDENT | | | | |
| 20 | | | | | |
| 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 |
| SC | 1 byte | Number of segments in the file |
| P | 1 byte | Protection flag used by the I/O system:<br>= 0, file is read or write<br>= 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 |
| DT | 1 byte | 8-bit code to indicate the type of mass storage device containing the file:<br>= 1, 9425<br>= 2, 844<br>= 3, 9427 |
| 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 |
| BC | 2 bytes | Highest block written |
| NHRPB | 2 bytes | Number of sectors per block |
| DC | 1 byte | Number of devices on which the file resides |
| CHECKSUM | 4 bytes | 32-bit binary checksum of the entire device label |

| Field Name | Size | Description |
|---|---|---|
| DEVICE IDENT* | 2 words | Identity of the device containing the segment map following the device identification |
| LSL** | 20 bits | Lower sector address; sector address at which this segment begins |
| E | 1 bit | Flag to indicate end of device map; 1 = end of device segments |
| SL** | 20 bits | Segment length; number of secters 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

# PROGRAMMING CONVENTIONS H

## REGISTER NAMING CONVENTIONS

Operand/index registers used in coding examples in this document are given symbolic names as specified below:

| 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 |
| H0 | 8 | R8 | 24 |
| H1 | 9 | R9 | 25 |
| H2 | 10 | RA | 26 |
| H3 | 11 | RB | 27 |
| H4 | 12 | RC | 28 |
| H5 | 13 | RD | 29 |
| H6 | 14 | RE | 30 |
| H7 | 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 address)
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).

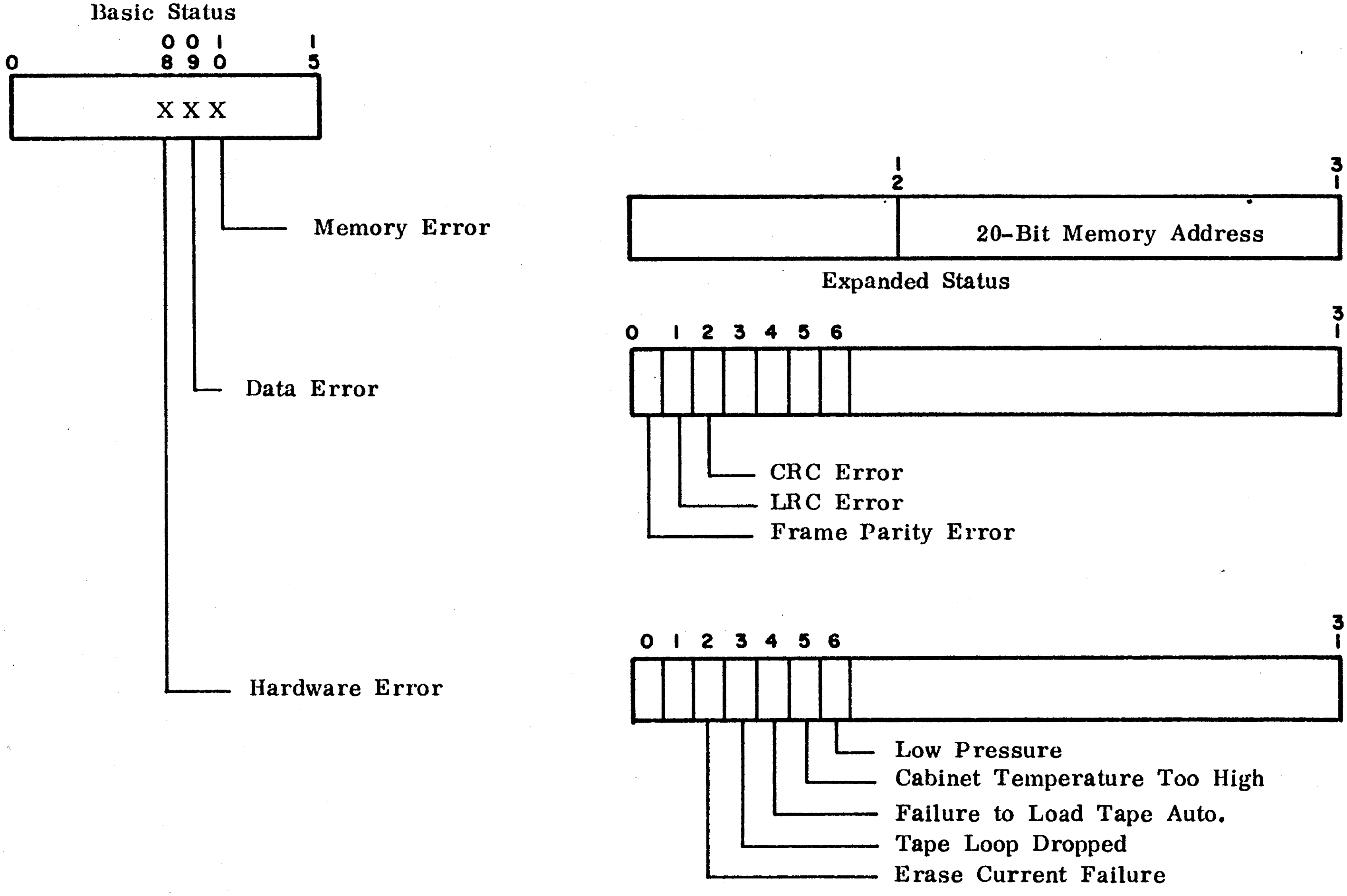
# ENGINEERING FILE

This appendix describes the system engineering file facility of MPX/OS. The engineering file is provided to collect system error information, such as device or memory errors for equipment maintenance. The file is allocated at autoload unless it is already allocated. The I/O interrupt processor (IOIP) collects information for the file everytime an I/O operation indicates an error condition. The information content and the format of an engineering file record are shown in Figure I-1. IOIP blocks the engineering file records (12 entries per block), which when written to the disk, occupies one sector. Before each write, the file is opened and after each write, the file is closed. This ensures all blocks are physically transferred to the disk and the file block number is updated. In the case of a system abort from MPX, or if the system reautoloaded, the last partial block may be lost. IOIP monitors the number of blocks written to the engineering file and when the file is within 10 blocks of the end of allocated area, the operator is informed that he should dump the file. The operator is informed when the file is full and no further records are written to the file. The file should be dumped at the end of the day and/or before reautoload. The file is initially allocated at 100 blocks. Figures I-2 through I-4 illustrate expanded status for various equipments.

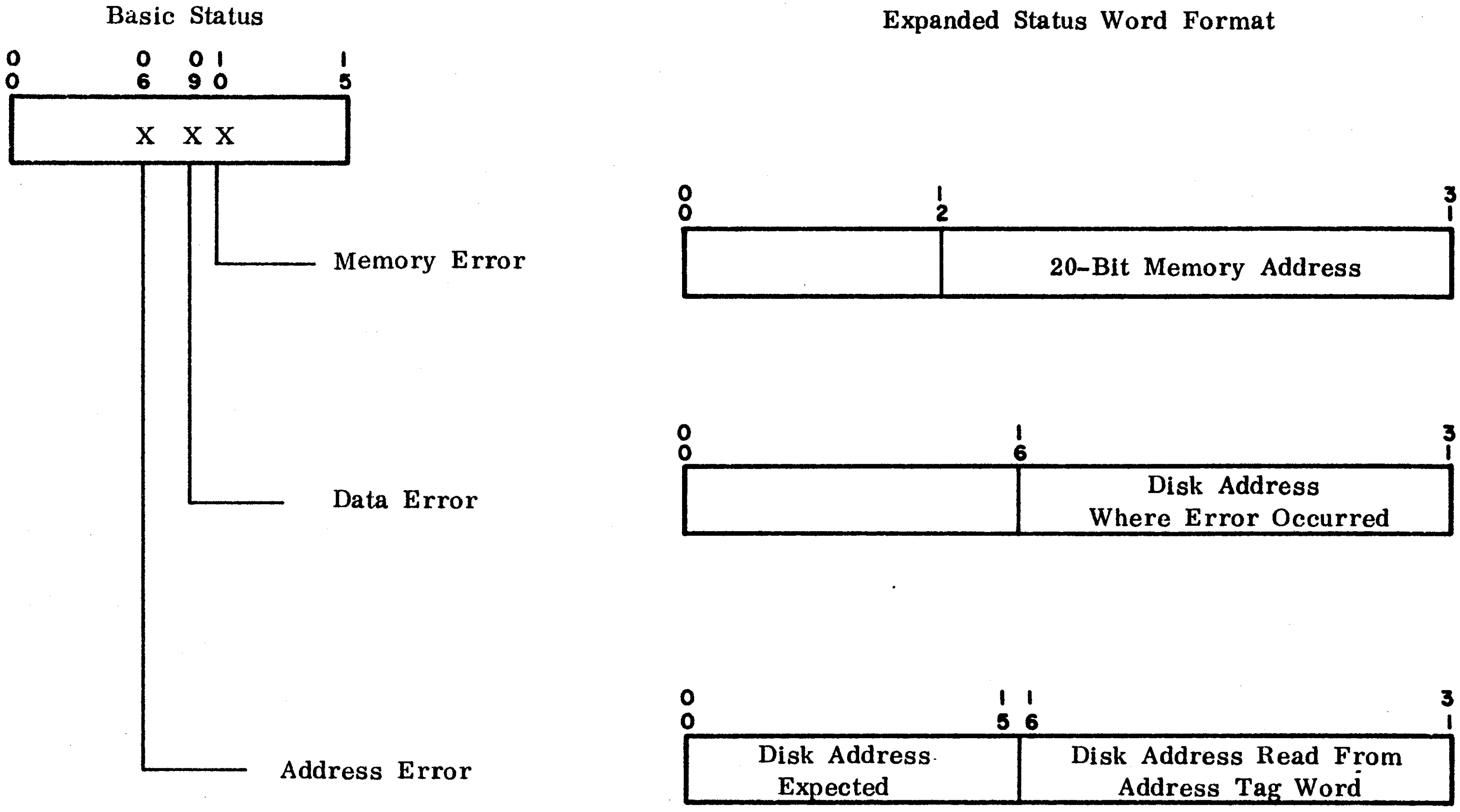
<table>
<tr><th>Bit No.</th><th></th><th colspan="2">0 7</th><th colspan="2">8 15</th><th>16 23</th><th>24 31</th></tr>
<tr><td>Word</td><td>1</td><td colspan="4"></td><td colspan="2">N</td></tr>
<tr><td></td><td>2</td><td>DT</td><td>HT</td><td>U</td><td>▨</td><td colspan="2">Status Bits</td></tr>
<tr><td></td><td>3</td><td colspan="6">Expanded Status</td></tr>
<tr><td></td><td>4</td><td colspan="2">M</td><td colspan="2">M</td><td>D</td><td>D</td></tr>
<tr><td></td><td>5</td><td colspan="6">BT</td></tr>
<tr><td></td><td>6</td><td colspan="6">AET or MST Entry<br>Not applicable for memory</td></tr>
<tr><td></td><td>7</td><td colspan="6" rowspan="2">Job Iden (ASCII Codes)</td></tr>
<tr><td></td><td>8</td></tr>
</table>

N Number of consecutive errors of the same type
DT Device type for the system devices (bits 0 through 2)
HT Hardware device type code (bits 3 through 7)
U Unit number of device (bits 8 through 11)
MM ASCII codes for month (0 through 12)
DD ASCII codes for day (0 through 31)
BT Binary value of time in milliseconds

Engineering File Record Format



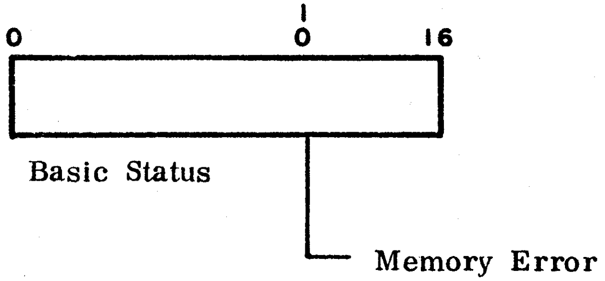
6X Expanded Status

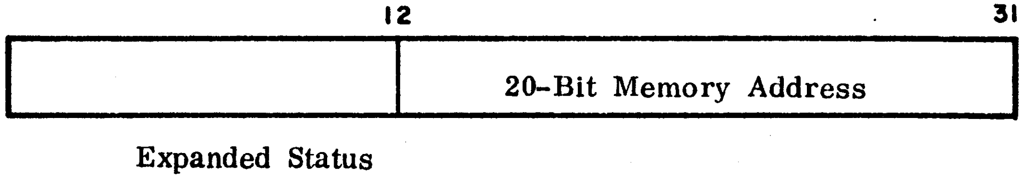


Basic Status

Expanded Status Word Format

Disk Expanded Status





Applicable Equipments:

- Cartridge Tapes
- High Density Tape Unit
- Card Reader
- Line Printer
- Calcomp Plotter
- 714-123 CRT
- Plasma Display

Expanded Status

# ENGINEERING FILE REPORT GENERATOR J

The engineering file report generator on the MPX/OS library is used to format the engineering file into a report. The user may selectively receive a report on a hardware type by unit number or obtain a report on the entire file. Optionally, the file may be cleared after generating the report. The report generator is executed as a job from the ENGRPT control statement. Figure J-1 describes the control card options and Figure J-2 describes the format of the engineering report.

The form of an engineering report generator job is as follows:

```
*JOB(ID=RPT)
*SCHED
*ENGRPT(M, U, C)
*EOJ
*ENGRPT(H, U, C)
```

H = Hardware type to be reported. If this parameter is null, all errors for all devices will be reported.

U = Unit number of specified hardware type to be reported. If this parameter is null, all errors for the specified hardware type will be reported. If H is null, this parameter is meaningless.

C = This parameter specifies whether to clear or not clear the engineering file at the completion of the report generation. This parameter is mandatory.

The parameters must appear in the indicated order with null parameters specified by adjacent commas. The acceptable parameters for the MPX/OS system devices will be as follows:

H = MEM<br>
M667 (667 magnetic tape)<br>
M669 (669 magnetic tape)<br>
MCAR (cartridge magnetic tape)<br>
MIEC (IEC high density tape unit)<br>
D927 (9427 disk drive)<br>
CARD (card reader)<br>
LINE (line printer)<br>
DIGV (DIGIVUE plasma display)<br>
DISP (714-123 display)<br>
PLOT (plotter)

U = 0 through 7

C = 1 clear engineering file after report generation

0 do not clear engineering file after report generation

Examples:

1) *ENGRPT(,,1) report errors for all devices and then clear file.

2) *(ENGRPT(MCAR,,0) report errors for all cartridge tapes; do not clear file.

3) *(ENGRPT(MCAR,6,0) report all errors for cartridge tape, unit 6; do not clear file.

ENGRPT Control Card

```
 EQUIPMENT NAME      HT    UNIT  STATUS    EX STATUS   MM/DD    HH/MM/SS        NO. ØF ERRORS    JOB IDENT.

 XXXXXXXXXXXXXXXX    XX    X      XXXX     XXXXXXXX    XX/XX    XX/XX/XX             XXXX        XXXXXXXX
```

| | | | |
|---|---|---|---|
| CONTROL DATA | TITLE Figure H-2. Engineering File Report Format | | |
| | DESCR. | | |
| PROGRAM | SYSTEM | | |
| SUB PR/RT | PROGRAMMER/ANALYST | | |
| CLIENT | CONTROL | DATE | PAGE 1 of 1 |

AA 1405

Engineering File Report Format

# COMMENT SHEET

MANUAL TITLE MP-60 MPX/OS Reference Manual

PUBLICATION NO. 17329110 REVISION A

**FROM:** NAME:

BUSINESS ADDRESS:

## COMMENTS:

This form is not intended to be used as an order blank. Your evaluation of this manual will be welcomed by Control Data Corporation. Any errors, suggested additions or deletions, or general comments may be made below. Please include page number references and fill in publication revision level as shown by the last entry on the Record of Revision page at the front of the manual. Customer engineers are urged to use the TAR.

CUT ALONG LINE

PRINTED IN U.S.A

NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

FOLD ON DOTTED LINES AND STAPLE

CORPORATE HEADQUARTERS' P.O. BOX O, MINNEAPOLIS, MINNESOTA 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD.

LITHO IN U.S.A.

CONTROL DATA CORPORATION