3300
3500

COMPUTER SYSTEMS MASTER

REFERENCE MANUAL

CONTROL DATA
CORPORATION

3300
3500

COMPUTER SYSTEMS MASTER

REFERENCE MANUAL

CONTROL DATA
CORPORATION

Additional copies of this manual may be obtained from the nearest Control Data Corporation Sales office listed on the back cover.

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

December, 1966
Pub. No. 60176800

FOREWORD

This manual is directed to programmers using MASTER Operating System in a closed shop computer center. It discusses basic principals, features, methods and techniques available to all users.

The manual assumes the reader has a basic knowledge of the 3300/3500 COMPASS Assembly Language and is familiar with the 3300/3500 Computer.

CONTENTS

INTRODUCTION 1

MASTER is a highly versatile multiprogramming computer system for the Control Data 3300 and 3500 Computers. Its task orientation is particularly adaptable to multi-access and multi-processing applications. The basic design permits expansion to multiple on-line remote stations and to multiple central processing units (CPU's).

MASTER accepts a wide variety of real-time applications.

MASTER achieves its primary purpose -- increased efficiency of the computing system -- by minimizing the idle time of the various processors: compute modules and data channels. This it accomplishes by simultaneously considering more than one job so that activities can be found for processors as they become idle.

MASTER consists of a system executive and an operating system. The operating system accepts jobs and translates them into executable entities called tasks, which are manipulated by the system executive. The system executive administers requests made by tasks for the execution of other tasks and assigns tasks to processors whenever they become idle. Interrupts are the mechanism for entering the system executive, which operates in the monitor state. The operating system, which consists of tasks administered by the system executive, operates in program state.

1.1 FEATURES

The MASTER computing system:

- Overlaps compute and I/O operations from several jobs, surpassing the throughput obtainable from a serial batch processing system.
- Features a centralized, file-oriented, input/output control system. This system, which is used by the operating system as well as users' jobs, is easily adapted to various hardware configurations, including mass storage and non-mass storage equipment.
- Completely protects programs and files through a combination of hardware and software.

- Loads, links and executes system programs and object programs from the library or from some programmer-defined file. Programs may be loaded and linked from several files and written on permanent mass storage as an absolute file for rapid reloading.
- Segments programs into non-contiguous pages of core memory.
- Permits users to reserve one to four data channels for real-time application with priority interrupt handling for these channels.
- Transmits console typewriter statements to the operator from a job or the system operator and waits for and returns responses upon request.
- Includes a number of installation parameters enabling the system to be tuned to a particular installation's requirements. If not specified, certain values are assumed for these parameters.
- Automatically processes input and output files from jobs in such a manner that it optimizes the use of peripheral equipment and simplifies the disposition of files by the operator.
- Facilitates generating and updating the standard system library or a user library, and permits several versions of the system library on mass storage, with the desired version called by the operator at system start-up time.
- Includes debugging aids such as recovery and snapshot dumps, and octal correction cards.
- Recognizes a versatile set of user macro requests of the system.
- Includes a set of file maintenance routines that perform housekeeping operations on the mass storage files.
- Includes a routine (XFER) that facilitates transfer of information from one mass storage or record device to another.

1.2 COMPUTER

MASTER operates on either the CONTROL DATA 3300 or 3500 Computer equipped with the executive mode and relocation features (Multiprogramming Option).

Executive Mode

Executive mode, selected from the console, consists of two states:

- Monitor state which allows execution of all instructions. (Master clear initially sets monitor state when the computer is in executive mode; thereafter it is entered through any interrupt condition.)
- Program state which prohibits execution of certain instructions reserved for the monitor (such as I/O). It is entered only through execution of a boundary jump by the system executive (EXEC) operating in the monitor state.

An attempt to execute the below instructions while MASTER is in the program state causes an interrupt.

- Halt and pause
- I/O operation initiation
- I/O status interrogation
- Interrupt selection
- Inter-register transfers that might alter the contents of register file locations 00-37.

Relocation

Relocation in the 3300 and 3500 Computer System contributes two vital features to MASTER: It makes it possible to expand core memory storage to a maximum of 262,144 words, and provides protection of the many user and operating system programs currently residing in core memory.

Core memory is divided into pages of 4000_8 words.

1.2.1 CONFIGURATION

MASTER operates with the following minimum configuration:

32K Core Memory

One 3304 or 3504 Central Processor

One 3311 or 3511 Multiprogramming Option

One 405 Card Reader and buffered controller

One 501 or 505 Line Printer and buffered controller

One 415 Card Punch and buffered controller

Two 3306 or 3307 (3506 or 3507) Communications (Data) Channels

2.5 million words of mass storage. This may be obtained by:

Five 852 Disks

Three 853 Disks

Two 854 Disks

One 813 Disk

One 814 Disk

Three 863 Drums

or any combination of the above that totals 2.5 million words.

A recommended configuration (page 1-13) includes four to eight magnetic tape units (none required), additional core memory, and an additional printer.

1.2.2 PERIPHERAL EQUIPMENT

The efficiency of any time-sharing program hinges on optimum use of peripheral equipment. In the typical system, MASTER uses a disk or drum for system storage and temporary storage of user programs; for user files, it uses disk storage. In systems with only one card reader, users do not have direct access to it because one card reader is required for input file preparations. MASTER reserves printers and card punches for processing of output files, but releases them on request.

All I/O channels are pooled; each channel is assigned to a task as it becomes idle. Requests for I/O have priority although the actual I/O task does not.

1.2.3 EXPANDABILITY

The system can be expanded to include up to eight channels, new input/output equipment, Satellite® computers, additional compute modules, and core memory up to 262K.

1.2.4 RELIABILITY

Continuous operation is essential when concurrently processing a number of jobs. To attain this end, MASTER provides automatic recovery procedures for hardware errors.

1.3 SYSTEMS COMMUNICATIONS

A user communicates directly with MASTER through

- control cards
- binary object program
- typewriter responses

and indirectly through source language programs which MASTER compilers and assemblers convert to binary object decks. The COMPASS assembler, in particular, generates code for over 50 macro instructions interpreted and acted on by MASTER. A user program may communicate with the operator by typing messages on the console typewriter. A user receives a standard printer output for each job which includes:

- accounting information
- printouts of control cards
- memory maps
- diagnostic and routine messages
- recovery information

When MASTER detects a condition requiring operator action, it types a request for action on the console typewriter.

In addition, it types informative messages that do not require responses. For example, it logs the beginning and ending of each job.

1.4 LIBRARY

The MASTER library maintained on a mass storage file contains:

The system executive

Operating system tasks

Absolute records or binary card images of routines, subroutines, and tasks

Standard system tasks such as FORTRAN and COMPASS

All assemblers and compilers operating under MASTER generate binary object decks acceptable to the MASTER relocatable loader. Information about standard language systems is given in the reference manuals for each system.

A MASTER library task, GLIB, provides a means of generating a new, revised or unrevised copy of the old library. GLIB can be run in a multiprogramming batch. Several versions of MASTER may be available on different editions of the library; the one selected at system start-up is the one used for the duration of a MASTER run.

For details of library generation, refer to the MASTER Installation Manual.

1.5 REAL-TIME ROUTINES

MASTER includes a facility which permits several user-supplied, real-time routines to be linked into the system executive at library generation time. These routines operate in the Monitor state with all its privileges; they drive equipment on reserved or dedicated channels not accessible to the MASTER I/O Control System (MIOCS).

Interrupts on dedicated channels have high priority and are disabled only a minimal time within EXEC. A real-time interrupt receives early consideration by EXEC which routes control to the user-supplied routine. This routine may also be executed when a program task requests its execution of EXEC.

Through the real-time facility, a user may add special-purpose or nonstandard drivers regardless of whether or not the equipment being driven has stringent real-time requirements; but all such equipment must exist by itself on a dedicated data channel.

A real-time routine cannot perform I/O on any channel not assigned to it. Because it is not a task, it may not use any of the MASTER macros described in this manual. Instead, it has available a special set of requests through which it communicates with EXEC. One such request permits a real-time routine to call a program task to manipulate its data. The MASTER Installation Manual describes these requests as well as installation procedures required for adding a real-time routine to MASTER EXEC.

1.6 INSTALLATION PARAMETERS

Installation parameters permit a system to be adjusted to meet the particular needs of any installation. An installation parameter is assigned a nominal value when the system is generated but may be modified at any installation in one of two ways:

- Through use of the SET control card when a new library is generated. (See MASTER Installation Manual)
- Through use of the SET command from the operator whenever the system is initialized. (See MASTER Operating Guide)

Parameters mentioned in this manual are:

tmin	minimum time for special jobs
tmax	maximum time for special jobs
tl	time limit
l	line limit
class	job class
core	core requirement
scr	scratch requirement
cycle	time limit cycle
dt	device type if not specified

1.7 MACROS

MASTER recognizes a versatile set of I/O and special-purpose macro requests. These requests originate in a user program and result in transfer of control to the System Executive (EXEC) through use of coded halt instructions. When execution of the halt instruction is attempted, an interrupt occurs returning the computer to the monitor state. EXEC processes the interrupt and either performs the requested function or calls a task to perform the function. It then selects a task and executes a boundary jump to it, returning control to the program state.

All MASTER macros are in standard COMPASS language format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	macro	$(p_1,p_2,\ldots,p_n)$	

macro is the name of a MASTER macro, such as CALL, MODIFY, etc.

$p_1 - p_n$ is a parameter list composed of alphanumeric strings.

See also, 3100/3200/3300/3500 Compatible COMPASS Language Reference Manual, Pub. No. 60174000.

Alphanumeric Strings

In this manual, the term "alphanumeric strings" defines a parameter at a string of alphabetic characters and/or numeric characters excluding the comma (,), left and right parenthesis ( ), oblique (/), dash (-), and space (∧) where ∧ is a blank. A parameter consisting of the single character zero, will be treated as blank.

Legal Alphanumeric Strings	Illegal Alphanumeric Strings
156A	1 ∧ 56
ABE25	A, ∧ B
SAM*	CD1 ∧ ,

Parameter Lists

Any of the following forms in parameter lists in a call is interpreted as null or blank:

,0, , ,

(0, ,)

,0) (,

Each parameter must be in its correct relative position. If several null parameters follow a legal parameter, the correct position must be established by commas. Thus, a macro call of the form:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD                          COMMENTS
1        8  10                    20                                     41
           LIBM                  MODIFY
             :                     :
           MODIFY                (W,DEPT238,SAMJONES,01,Q32,X578,,,,,0,0513167)
```

modifies the protection of a file and the expiration date but nothing else since the intervening parameters 7-11 are null.

When the parameter list terminates before the last possible parameter, all remaining parameters are considered null or blank. Thus, a call of the form:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD                          COMMENTS
1        8  10                    20                                     41
           LIBM                  MODIFY
             :                     :
           MODIFY                (W,DEPT238,SAMJONES,01,Q32,X478,,BILLSMITH)
```

modifies only the file name. In this example, parameters 1-7, and 9 are specified; parameters 8, 10, 11, 12 are omitted. (For details of MODIFY macro, refer to section 5.4.2.)

1.8 FILE SUMMARY

In MASTER's file-oriented input/output, each unit record device such as a printer, punch, magnetic tape unit, etc., and each file definition for mass storage devices (disks and drums) has associated with it a unique file identifier called its data set identifier. The following discussion summarizes material presented in detail in Chapters 5, 6, and 7.

1.8.1 FILES

System files, all on Class A mass storage, divide into those owned by MSIO and those owned by MASTER. MASTER owns a library file, a library directory file, and a pool made up of segments from one or more standard files. This pool provides segments that make up the standard job input, output, punch, and scratch files. Of these, only the scratch files are managed by users.

The remaining mass storage on Class A and B devices, and all unit record devices are available for user files.

1.8.2 SCHEDULE UNIT OR RESERVE SPACE

Space on Class B mass storage devices, and units to be used as files must be scheduled for all user files. MSIO and MASTER reserve their own space for the system files.

A user must estimate lines of printer output and number of cards to be punched for MASTER to allocate a corresponding amount of space for the OUT and PUN files. The user must schedule mass storage scratch file requirements if the job requires more than the standard amount allocated. This scheduling is done through JOB card line and punch limits, and SCHED card SCR estimates.

1.8.3 FILE DEFINITION HANDLING

Functions that manage file definitions include allocation and release of space, modification of labels, expansion of defined file size, and opening and closing of files. MSIO and MASTER library definitions cannot be altered by users. They are managed by GLIB and *FMU (Generate Library and File Maintenance Utility Routines) and by the system initialize routine. Job INP, OUT, and PUN files cannot be directly defined by users.

Users manage their job scratch file definitions through System OCARE; the mass storage files through a set of file function routines called OCAREM; and the unit record device files only through open and close of OCAREM.

1.8.4 DATA TRANSMISSION FUNCTION

The blocker and deblocker routines are convenient for transferring data to or from files with a minimum of physical transfers. They are also available for standard blocking. Users may also directly call MIOCS macros for data transmission and related functions on their files.

1.8.5 DIRECT STANDARD FILES

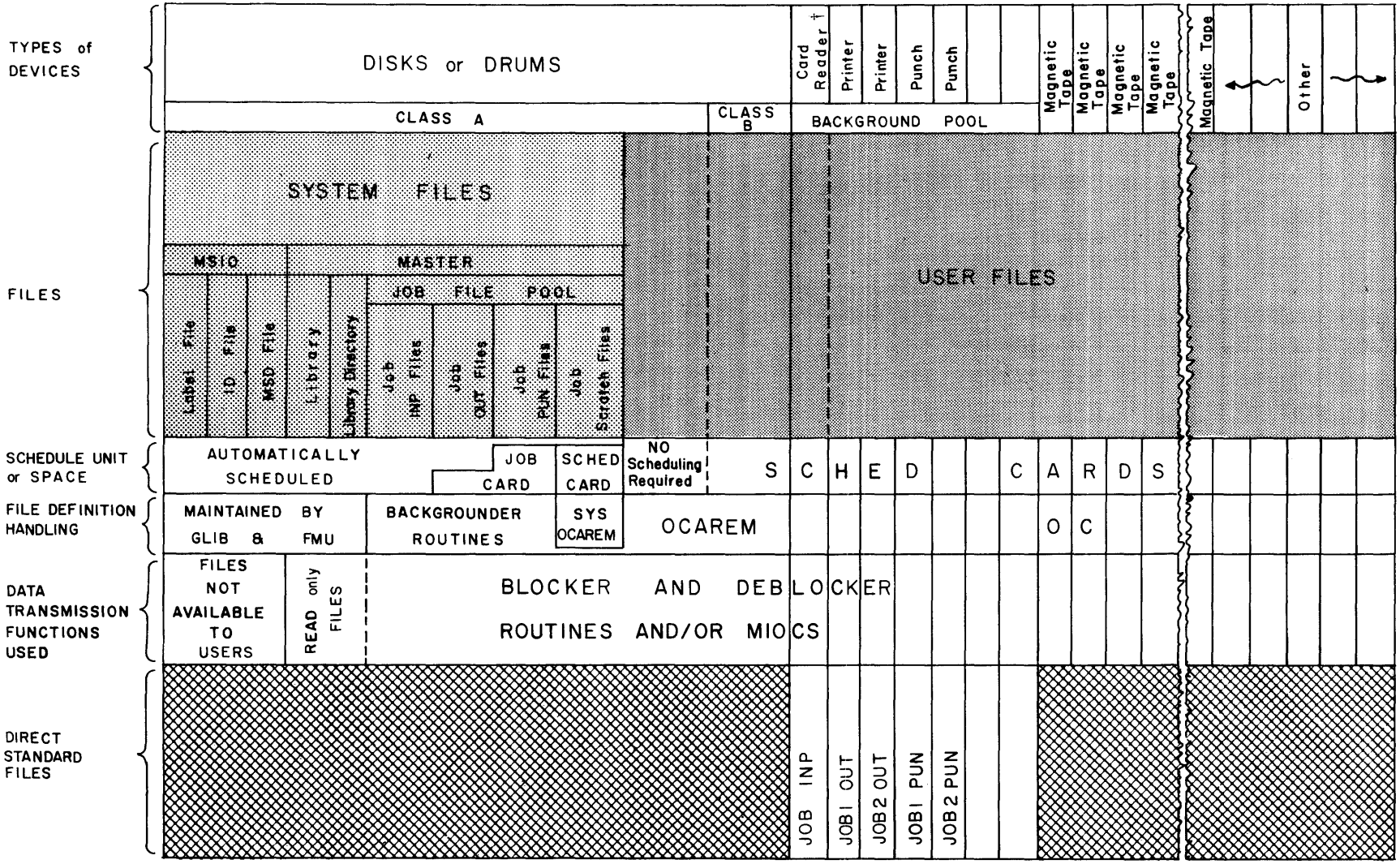
A DIRECT card substitutes a card reader, printer, or punch to be used in place of mass storage for a job's INP, OUT, or PUN file. The user refers to the file as if it were on mass storage; it is blocked in standard format.

1.9 FILE MAINTENANCE ROUTINES

A comprehensive set of file maintenance routines on the MASTER library permits installation personnel to:

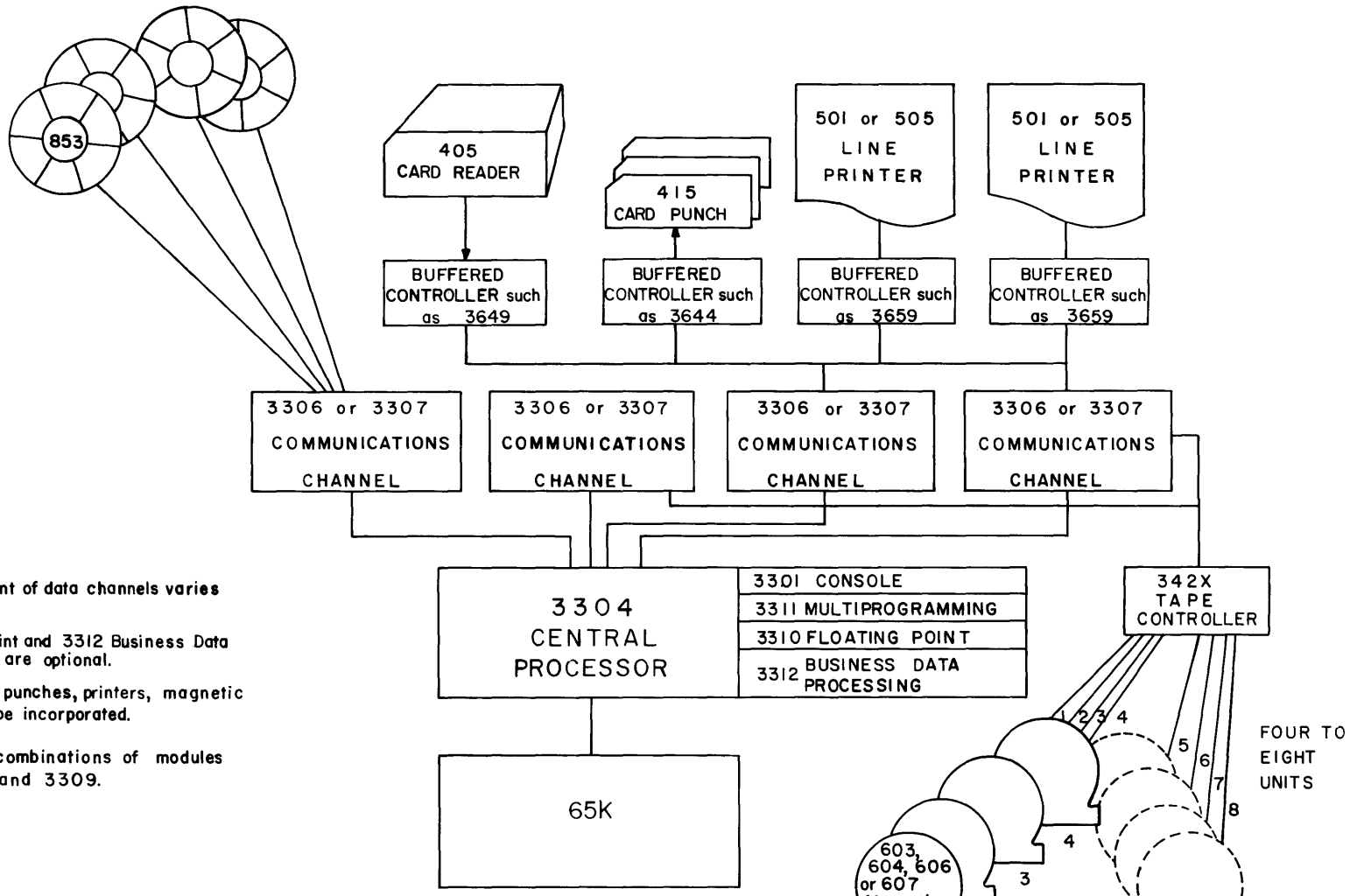
- Obtain a listing of one or all entries on the Mass Storage Directory.
- Obtain a listing of one or all entries on the File Label Directory.
- Enter a new disk pack specifying device type, number, mode, and class on the Mass Storage Directory and write a device label on the disk pack.
- Remove an entry from the Mass Storage Directory.
- Scan the File Label Directory and list any file having an expired date.
- Obtain on magnetic tape, a copy of files (Class A or B) which may then be reloaded onto mass storage.

Some file maintenance utility routines require the entire computer and must be run in a multiprogramming batch. For use of File Maintenance Utility routines, refer to the MASTER Operator's Guide or MASTER Installation Manual.



FILE SUMMARY

† Input card reader not available for use as user file



RECOMMENDED EQUIPMENT CONFIGURATION

NOTES:

- Number and assignment of data channels varies between systems.
- The 3310 Floating Point and 3312 Business Data Processing Modules are optional.
- More disks, readers, punches, printers, magnetic tape units, etc. may be incorporated.
- Memory consists of combinations of modules such as the 3302 and 3309.

JOB FLOW 2

The following discussion on job flow outlines the stages through which a job progresses in the MASTER system. Most of these stages involve calls for program tasks, some of which are permanently allocated and others which are loaded as part of the job. In all cases, the tasks executed in the progress of a job are multi-programmed with all other tasks currently active in the system; and the execution of these tasks proceeds on a priority basis.

2.1 AUTOLOAD

The operator begins a MASTER run by autoloading a version of MASTER from the library. The portions of MASTER that are autoloaded include the system initialization routine and permanently resident portions of MASTER, such as EXEC, its tables, and MIOCS.

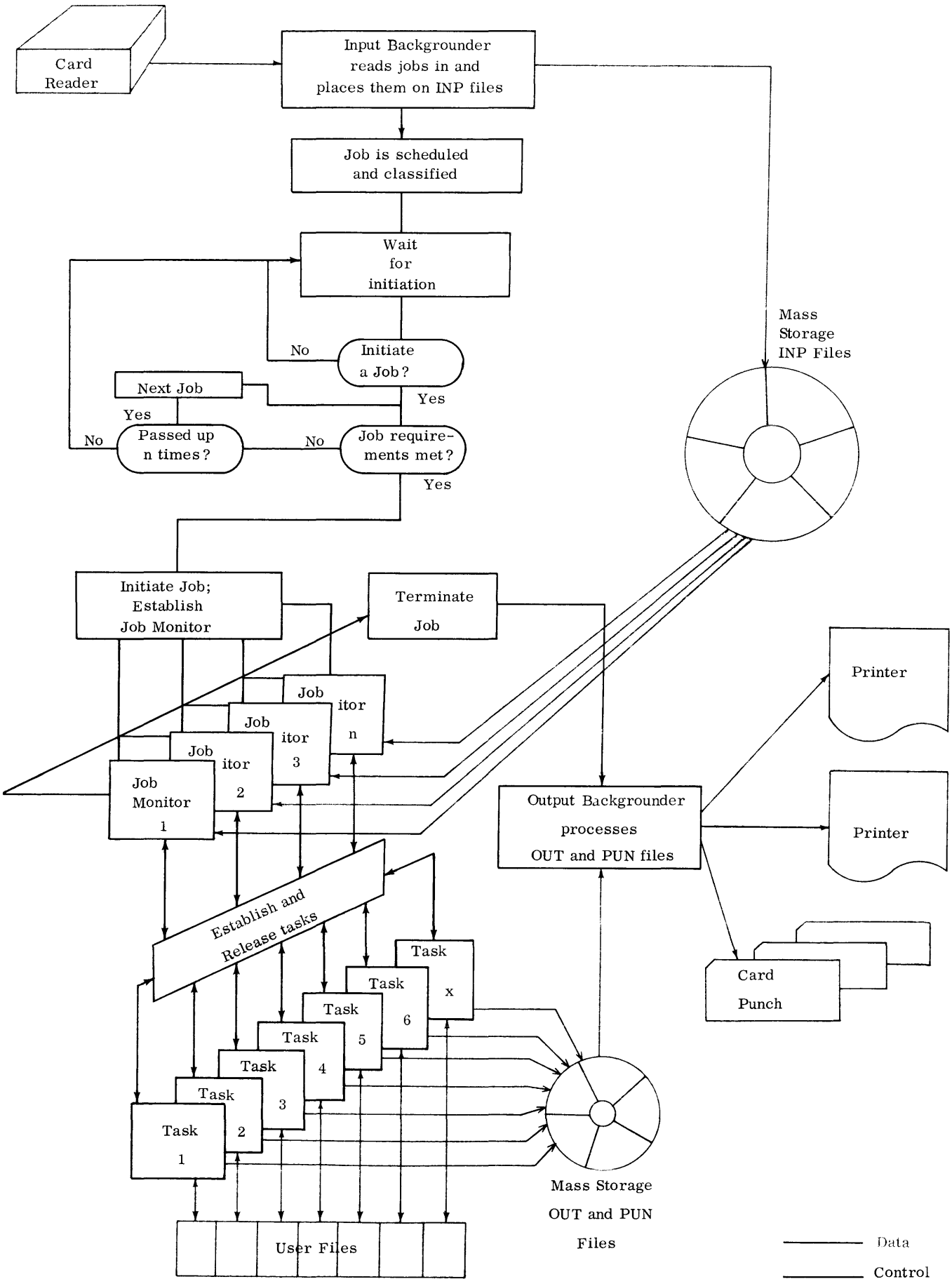
MASTER automatically begins executing the initialization routine at which time the operator enters time and date and may enter SET commands changing system installation parameters from the console typewriter or card reader. When all SET commands have been entered, the system establishes its system files, and transfers control to the input backgrounder, which prepares job files for MASTER.

2.2 JOB INPUT

All user job decks, consisting of control cards and possibly program and data decks, are presented serially to MASTER through the input card reader. The input backgrounder writes card images of a user's deck on a job INP file on the disk. This transfer is bypassed on a job basis when CR is specified on a DIRECT card. Jobs can be transferred continuously to job INP files by the input backgrounder until there are no more jobs on the input card reader or until the pool from which INP files are formed is depleted.

2.3 PREPROCESSING

After the input backgrounder transfers a job to its INP file it calls the job scheduler, an operating system task, and passes to it the information obtained from the DIRECT, JOB, and SCHED cards. The scheduler checks and assigns job classes as outlined below. Classifying the job and listing it as a candidate for initiation constitute scheduling.


Card Reader
Input Backgrounder reads jobs in and places them on INP files
Job is scheduled and classified
Wait for initiation
No
Initiate a Job?
Yes
Next Job
Yes
No
Passed up n times?
No
Job requirements met?
Yes
Mass Storage INP Files
Initiate Job; Establish Job Monitor
Terminate Job
Job itor n
Job itor 3
Job itor 2
Job Monitor 1
Printer
Printer
Output Backgrounder processes OUT and PUN files
Establish and Release tasks
Task x
Task 6
Task 5
Task 4
Task 3
Task 2
Task 1
Card Punch
Mass Storage OUT and PUN Files
User Files
Data
Control

2.3.1 JOB CLASSES

Each job submitted to MASTER is assigned a job class by the user or the job scheduler. Class determines when a job is initiated and the priorities of its tasks. The job initiator looks at job classes when seeking work. Job classes, from highest to lowest are:

Emergency

Background

Special

Input/Output

Compute

Emergency

Emergency jobs are submitted in the same way as all other jobs. A job is classified as an emergency job if (1) the user has declared it as class E on the SCHED card or (2) the job deck is preceded by a DIRECT card, and MASTER has reclassified the job as emergency.

Background

A job primarily intended to drive slow-speed peripheral equipment can be declared as class B on the SCHED card. Background jobs generally use little compute time and when ready, require attention quickly to drive their equipment at full speed.

Special Job

Upon receiving a job declared by the user to be an I/O or compute job, MASTER determines if the job qualifies as special. For a job to be reclassified as special, the user must supply on the SCHED card a time estimate (te) that lies within a range determined by installation parameters Tmin and Tmax, where

$$Tmin \leq te < Tmax.$$

The special class provides fast turnaround time to average length jobs. With this scheme, several jobs are likely to pass through the system during the processing of an I/O or compute job. The class is eliminated if the upper parameter for the range is zero.

Input/Output

A job is classified by MASTER as an input/output job if it does not qualify as special and the user declared it as I on the SCHED card or the installation parameter was used in lieu of a declaration.

Compute

A job is classified by MASTER as a compute job if it does not qualify as special and the user declared it as C on the SCHED card or the installation parameter was used in lieu of a declaration.

2.4 JOB INITIATION

Whenever possible, MASTER seeks a new job. It considers such variables as job class, equipment and core requirements, and wait time. All required core and I/O devices, such as tapes, card readers, and printers must be available before a job is initiated.

MASTER first looks for emergency jobs waiting for initiation; and if equipment requirements can be met for one, MASTER initiates it. When MASTER initiates a job, it types B i on the console typewriter where i is the job identifier taken from the JOB card.

Except when emergency jobs are active or waiting, MASTER attempts to keep active at least one job from each of the four regular job classes (background, special, input/output, and compute). Another job from the same class is initiated only if no scheduled jobs from other classes are capable of being initiated.

Within a class, jobs are initiated on a first-in-first-out basis. However, the first job in the list might not always be the first initiated if core and I/O requirements cannot be satisfied by the available equipment and storage; a job submitted later may be initiated first. On the other hand, no job can be refused initiation because of lack of equipment more than a certain number of times as determined by an installation parameter. When this limit is reached the job's class is changed to emergency and no non-emergency job is initiated until the equipment required by the waiting job is released by terminating jobs.

When its requirements can be met, the waiting job is initiated and normal job initiating resumes. The job initiator loads a copy of the blocking and deblocking routines into memory for the job and calls the job monitor, an operating system task. The job monitor is then loaded and established.

2.5 PROCESSING

After initiation, the job monitor processes control statements from the job's INP file. These control statements result directly or indirectly in the loading (when necessary) and execution of program tasks.

A task is a direct part of a job when loading and execution is directly called for by a Task Name control card or by a task of the job currently in execution. Tasks resulting indirectly from a job are those required by the operating system in processing tasks directly resulting from the job.

Once the job is initiated, its priority is set according to its class. Any task having inherited priority inherits this job priority (4.6).

Tasks which require loading, including the job monitor and relocatable loader, occupy core scheduled for the job. The loader, as do most tasks, releases core upon completion of the operation.

When a task and all of the tasks it called are completed, it returns to its caller. If the task is called by a control card, the caller is the job monitor. Processing of a job ends when its job monitor, seeking more work in the job's INP file, detects an end-of-file condition.

Processing can also be terminated by the operator or when a returning task notifies the job monitor of an abnormal condition. Upon abnormal termination if the user requested ABORT on the SCHED card, a recovery dump is written on the job's OUT file. Otherwise, the user receives only a dump of the console registers, and locations 40_8 through 77_8 of the register file, if used.

At job end, any open files are closed. All scratch files, the INP file, core, and any scheduled devices are released. The output backgrounder is requested to process the OUT and PUN files when no DIRECT processing takes place. For a DIRECT job, the direct unit file is closed and the device returned to the output backgrounder.

When MASTER closes a job it types T i, where i is the identifier taken from the JOB card.

2.6 POSTPROCESSING

The output backgrounder drives all available printers and punches at full speed as long as there are OUT and PUN files to be processed. All printers and punches are controlled by the output backgrounder which may relinquish control of an idle printer or punch upon receiving a request from a job. This request results directly from processing a user's OPENU request for a printer or punch and indirectly from processing of a DIRECT card specifying PR or PU.

```
JOB ACCOUNTING INFORMATION

  NAME=XXXXXXXX   ACCT=XXXXXXXX

  TIME USED
    COMP= HH/MM/SS.SSS
    CHAN= HH/MM/SS.SSS

  FACILITIES NOT USED
    CORE=XXX
    SCR =XXX


                               **** JOB ABORTED ****


IAC=

  TASK SCOOP
    NAME=XXXX    P=XXXXX    STATUS=XXX    CALLER=XXXX
    A=XXXXXXXX    Q=XXXXXXXX    B1=XXXXX    B2=XXXXX    B3=XXXXX    IM=XXXX    SR=X    IS=X    OS=X


                               *** REGISTER FILE ***

            XX   XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
            XX   XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
            XX   XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
            XX   XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
```

```
  TASK SCOOP
    NAME=XXXX    P=XXXXX    STATUS=XXX    CALLER=XXXX
    A=XXXXXXXX    Q=XXXXXXXX    B1=XXXXX    B2=XXXXX    B3=XXXXX    IM=XXXX    SR=X    IS=X    OS=X


                               *** MEMORY ***

         XXXXX     XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX  XXXXXXXX
         X                X  XXXXXXXX  X     XXX  XXX     X     XXXXX  XXXXX     XXXXXXX     XXXXX
```

JOB ACCOUNTING INFORMATION

NAME = Identifier taken from JOB card

COMP = Central processor time used by job in hours, minutes, seconds and milliseconds.

CHAN = Sum of time consumed on each I/O channel used by job in hours, minutes, seconds and milliseconds.

CORE = Number of quarter pages reserved in excess of those used; the difference between the core estimated on SCHED card and the maximum used at any one time.

SCR = Number of scratch area segments reserved in excess of those used; the difference between scratch-file estimate on SCHED card and maximum number of segments used at any one time.

***** JOB ABORTED *****

IAC = abort message.
or
VAC = If abort was at task's request (voluntary) IAC is replaced by VAC. IAC abort message inserted by EXEC (Appendix D)

*** REGISTER FILE ***

Contents of locations 40_8-77_8 of register file.

*** MEMORY ***

***** OUT FILE OVERFLOW *****

Abort dump exceeds scheduled space.

TASK SCOOP

NAME = Task's name.

STATUS = Task status when terminated.

CALLER = Caller of terminated task.

P, A, Q, B1, B2, B3 } Register contents at termination.

IM — Interrupt mask of internal faults selected by the task.

SR — The subcondition register indicates whether operand addresses were routed through operand or instruction state register.

IS — Instruction state last assigned to task.

OS — Operand state last assigned to task.

2.6.1 OUT FILES

When the output backgrounder is requested to process an OUT file and no printer is available for assignment, the backgrounder places the file in a file disposition list which it processes on a first-in-first-out basis. OUT files are printed on the standard form for the installation. (See XFER, chapter 11).

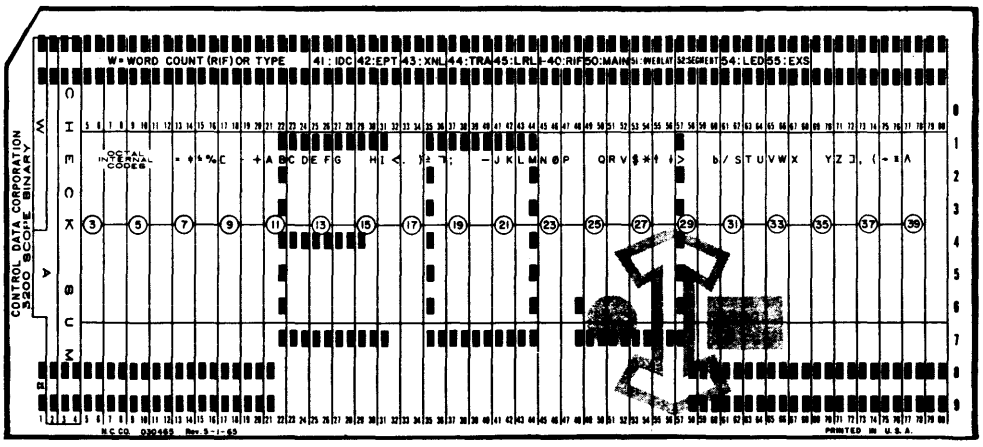
An OUT file begins with a heading --- or for a direct OUT job, ends with a trailer --- of the form shown. If the job ends normally, only the accounting information is printed. If the job terminates abnormally the information under JOB ABORTED is printed. The register file and MEMORY are printed according to SCHED card options.

The backgrounder then prints information placed on OUT by the user and the job monitor.

2.6.2 PUN FILES

When output backgrounder is requested to process a PUN file and no punch is immediately available, the background routine places the job's PUN file in a file disposition list which it processes on a first-in-first-out basis. PUN files are punched on standard cards for the installation.

When the backgrounder detects an end-of-file condition, it punches and offsets the end-of-job card. On a DIRECT job, the end-of-job card is punched and offset before the backgrounder processes the next PUN file.

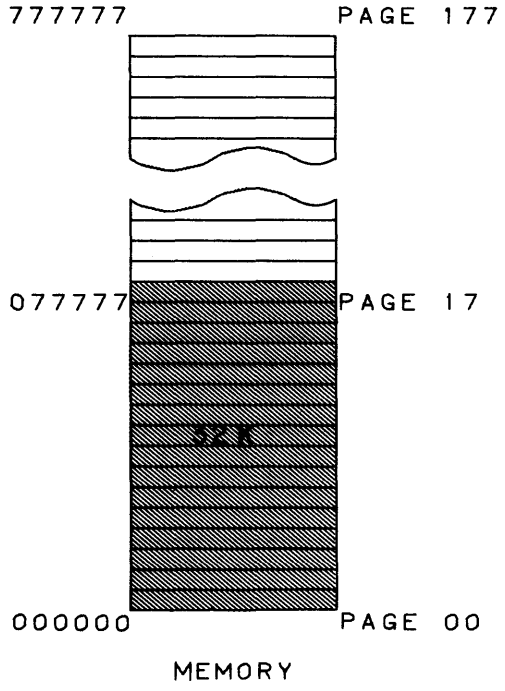


MEMORY 3

This chapter discusses addressing schemes and how loading and execution of one and two chapter tasks are implemented in MASTER in conjunction with the 3300/3500 relocation and paging features. Users should be familiar with the techniques described herein to better understand and make use of the task-oriented structure of MASTER. Paging and relocation are completely automatic.

3.1 PHYSICAL MEMORY

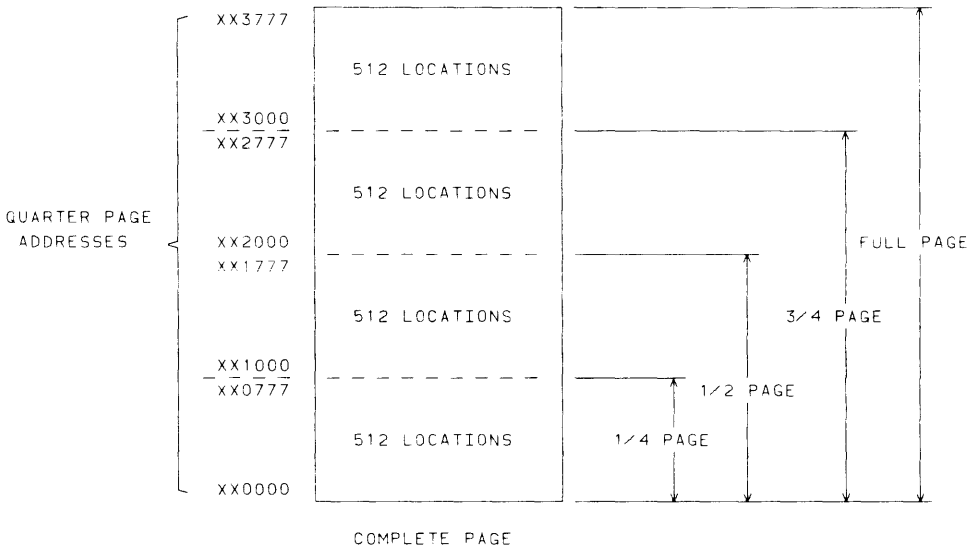
The minimum core requirement for MASTER is 32,768 twenty-four bit words. Systems may have core added up to an allowable maximum of 262,144 locations.



MEMORY

3.1.1 PAGE STRUCTURE

A page is an addressable block containing 2048 memory locations. A fully expanded system contains 128 of these pages. Individual pages may be subdivided into four partial pages of 512 address locations each. Programs may be allocated full pages, 3/4 page, 1/2 page or 1/4 page of memory.

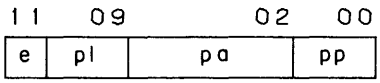


3.1.2 PAGE MAP

When EXEC or a job calls for loading of a program task, one of the MASTER loaders (relocatable or absolute) loads the task into available pages of physical memory where it resides until its memory is released by the job or by MASTER.

During loading, the loader generates either one page map for the program and common area or two separate maps for the task's program and common areas. A task that has separate page maps for program and common is also referred to as a two-chapter task (3.5). The page maps are retained by MASTER for as long as the task is in core.

A page map consists of 16 page indexes of the following form:



e Exclusion bit: 0 indicates task can read or write in page; 1 indicates task can only read page.

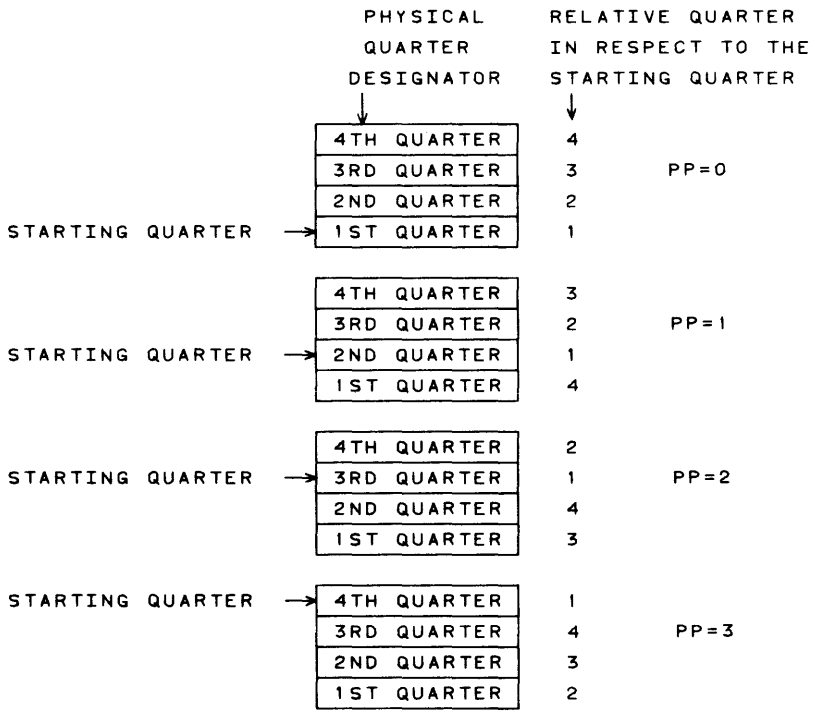
pl Page length:

- 0 task uses all of page pa
- 1 task uses 1/4 of page pa
- 2 task uses 1/2 of page pa
- 3 task uses 3/4 of page pa

pa Page address designator, the number of the physical page used by the task. pa can be 0 to the maximum number of pages in the system.

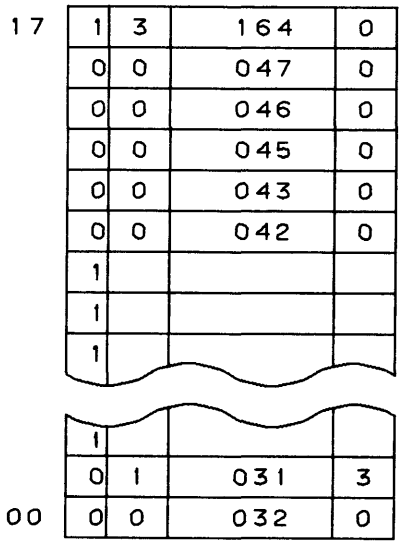
pp Partial page designator

- 0 task addressing starts at top of page; logical 0 equals physical 0 for page
- 1 task addressing starts at 1st quarter page; logical 0 starts at physical address 01000 for page
- 2 task addressing starts at 2nd quarter page; logical 0 starts at physical address 02000 for page
- 3 task addressing starts at 3rd quarter page; logical 0 starts at physical address 03000 for page.



QUARTER PAGE IN RELATION TO PP DESIGNATOR

When a task requires fewer than 16 indexes, the remaining indexes in its page map have the exclusion bit set. Any reference by the task to an excluded page index results in an interrupt causing the job associated with the task to be terminated. A message on the job's OUT file notifies the user of the system action.

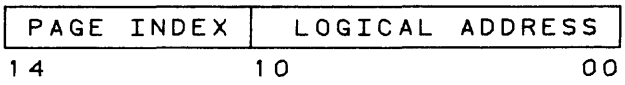


PAGE MAP

A task cannot reference any physical memory not assigned to it by EXEC; this assures complete memory protection between tasks.

3.2 LOGICAL ADDRESS

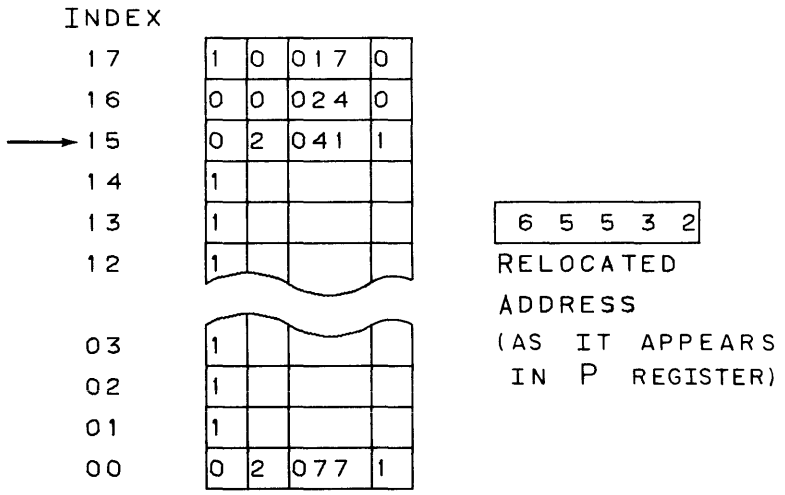
A program address of a loaded task has the following significance:



page index — Octal location in the page map of the indexed page containing the instruction or operand

location address — Address in the page relative to its logical zero. To obtain the physical address, the hardware logically adds bits 09, 10 of the logical address to the pp bits of the index.

Example:

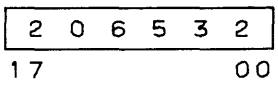


PAGE MAP FOR TSKA

The upper four bits of Address are 1101, indexing entry 15 in Page Map. Thus, Address is for page 41.

The lower 11 bits of Address refer to logical address 1532 of page 41.

pp is 1 indicating that logical 0 for the page 41 is at 01000. Adding pp to bits 09, 10 of address produces physical page address 2532. This, added to the page number produces the 18-bit physical memory address:



Logical address 00000 through 03777 index entry 00, addresses 04000-07777 index entry 01, and so on up to 74000 through 77777, which index entry 17. A complete set of logical addresses is a chapter.

When the P register is incremented past an index boundary, for example from 73777 to 74000, the hardware automatically indexes up to the next entry. This indexing does not occur on quarter page boundaries. Operand addresses cause similar indexing.

3.3 CHAPTERS

To simplify addressing for the user, MASTER memory is represented as two blocks, each with 32K consecutive logical addresses, called chapters. Binary object programs generated by compilers and assemblers, absolute records, and all program listings and program memory maps reflect chapter addressing. Addresses in a chapter range from 00000 to 77777.

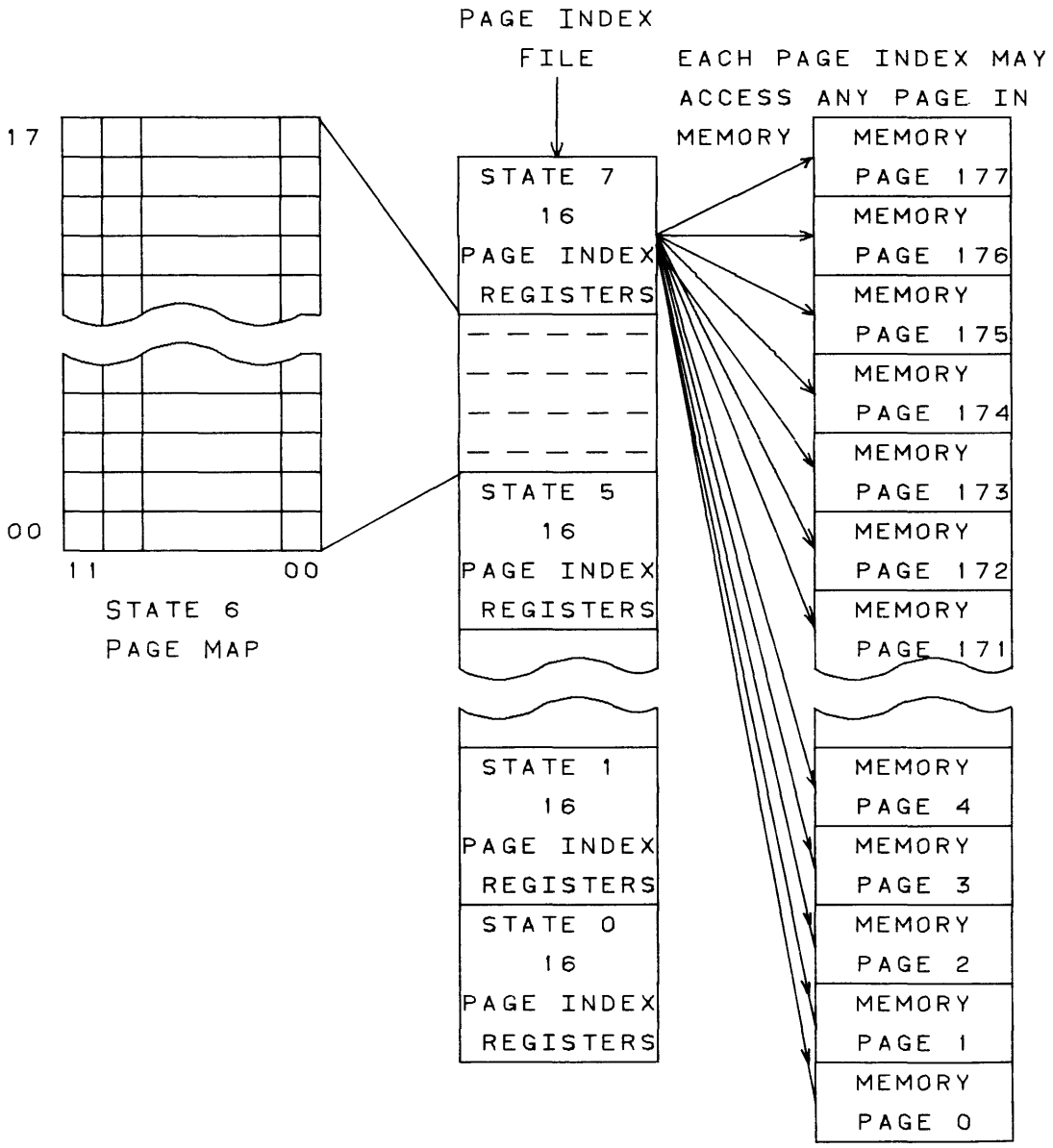
A MASTER user writing a one-chapter task can use 30K words for program, common, and data areas assuming a corresponding amount of physical memory is available in the system. For a two-chapter task, however, he can use one set of 30K for the program and data areas, and a second set of 32K for the common area. Or he can put common in the first chapter and reserve the right to use a second chapter to receive another task's common.

A task may be designated as a two-chapter task with common assigned to chapter one. In this case, the task may reference its own common in chapter one and the common passed to it in chapter two.

In this manner, programs and subprograms within a task can communicate through the task's chapter one common and tasks of a job can communicate through their chapter two common areas.

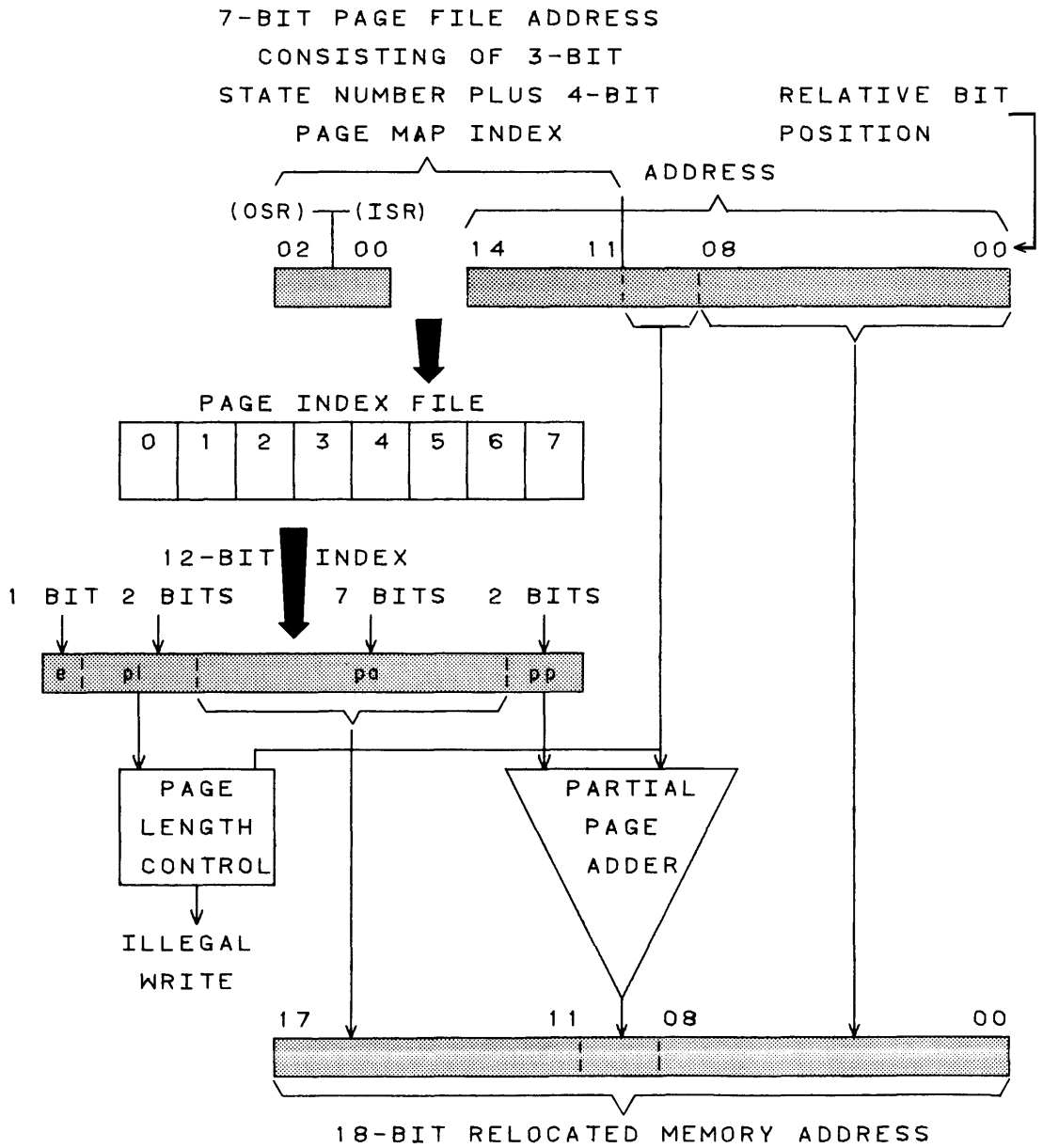
3.4 STATES

Each time a task is placed into execution, its page map must be written into one of the eight areas--called states--of the Page Index File so that the physical page assignments are available to the 3300/3500 hardware. Hardware refers to the page map through a state register containing the state number. For a two-chapter task both page maps must be written into two separate states.



When a task is to be placed into execution, MASTER assigns a state for each chapter present (two maximum). The state assigned to Chapter One is written into the Instruction State Register (ISR); the state assigned to Chapter Two is written in the Operand State Register (OSR). For a one-chapter task, the ISR and OSR are set equal. The user must execute an ROS instruction to reference Chapter Two.

A user task requires a state only when it is active, that is, in execution or has some I/O operation in progress. Hardware is then using the map and it cannot be disturbed. It is possible for all eight states to be active. When a task is inactive, its state is inactive; the page map can be written over with a new map of a task to be executed.



TASKS 4

4.1 TYPE OF TASKS

MASTER deals with basic entities known as tasks, of which there are two types: program tasks and I/O tasks. Tasks are executed on processors, of which there are two corresponding types: central processors (CPU's) for program tasks and data channels for I/O tasks.

Program Tasks

A program task consists of one or more subprograms and associated routines. To be recognized by MASTER, it must be in relocatable binary format as are programs produced by compilers and assemblers operating in conjunction with MASTER, or in absolute format having undergone loading and relocation with the results recorded in absolute format.

A program task can be loaded into available memory from the library or any file open to the job when MASTER operation begins, when needed by the operating system, or when requested in a job. Active jobs can continue to call for tasks as long as they do not exceed their scheduled storage requirements. When a task that is not permanently allocated completes its work, its core can be released and assigned to another task.

The operating system includes program tasks such as a scheduler, initiator, monitor, loader, etc. The operating system accepts jobs from users and subdivides them into a collection of program tasks, such as a compilation task and execution task.

I/O Tasks

Requests for I/O from operating system tasks or user tasks result in the execution of I/O tasks, or in data channel activity.

4.2 TASK ASSIGNMENT

All current program and I/O tasks are contained in task lists and have priorities assigned to them. MASTER assigns I/O tasks to data channels and program tasks to the CPU, on priority basis, in such a manner as to maximize the work load on the computing system. (See priorities, 4.6). In general, when EXEC is entered on an interrupt, it processes the interrupt, updates accounting information, initiates as many I/O tasks as possible, and transfers control to the highest priority program task ready to execute.

4.3 TASK CALLS

When a program task in execution requests execution of another program task or I/O task (calls a file), it calls EXEC which routes control to the called task. When a program task which has been called by another task has completed operation, it relinquishes control by issuing a return to EXEC which notifies the caller of the completion. Both the call and the return are EXEC requests and as such are accomplished by generation of an interrupt (Chapter 8).

When a job calls for execution of a program task by means of a Task Name control card (10.1.4), the job monitor issues the call to EXEC. Then, when the callee returns, EXEC routes control to the job monitor.

4.4 TASK ORIGIN

Program tasks can be on the library or any file accessible to the user. Those on the library are library tasks; those on other files are user-supplied. I/O tasks result from calls for I/O operations on a file.

4.4.1 LIBRARY TASKS

Tasks may be placed on the system library file during library generation (GLIB execution), either as absolute records or relocatable binary card images. A library task is identified and called by a task name. It may be restricted to operating system use only. Permanently allocated library tasks are loaded during MASTER initialization and remain in core throughout the MASTER run. Each library task has a priority (inherited, inherent, or both) used by MASTER to determine which task to place in execution. For generation of library tasks, refer to the MASTER Installation Manual.

A typical library task is XFER, described in chapter 11. For correct task usage of systems such as FORTRAN and COMPASS, refer to the related publication.

4.4.2 USER-SUPPLIED TASKS

Tasks on files other than the library can be called by control cards or from within the program in the same way as library tasks except that the file containing the task must be declared in the call. The called task may or may not require a TASK card (10.1.5). A user-supplied task consists of subprograms in the form of:

card decks

card images

absolute records

A user task in absolute format has a name permanently associated with it. A relocatable binary task, however, acquires a name only when called. The Task Name control card or macro call supplies a unique temporary name that EXEC uses in the monitoring of the task; no other task in the job can have that name.

4.5 TASK NAMES

Each program task has a name of 1-4 alphanumeric characters that identifies it when called. For library tasks, the names are assigned at GLIB (library generation) time, for example:

Name	Task
GLIB	Library generation program
*DEF	Operating system task for file processing (section 5.4)
RLDR	Relocatable loader
FTN	FORTRAN
CMP	COMPASS

User-supplied relocatable binary tasks have no permanently assigned names. A user assigns a task name the first time he calls a task in a job, and the name exists for that task for its lifetime in the job. As an example, when a job consists of compile and execute, execution of the compiled program is called for with a Task Name card that assigns it a name. An absolute task, on the other hand, acquires a name from an RLDR card.

I/O tasks are named by MASTER which uses the data set identifier assigned to the file when it is opened (Chapter 5).

4.5.1 COPIABLE TASKS

Any task with a name that does not begin with an * is copiable. Each job that calls a copiable task will have its own copy loaded into its available memory. If it calls the task several times, however, it will not receive a new copy if the old one is still in core. Instead, if the task is busy, additional calls must wait for it to become available. That is, they are queued on the task. XFER, described in chapter 11, is one example of a task that can be called many times by a job.

4.5.2 ONE-COPY TASKS

A task with a name that begins with * is not copiable. When it is called and a copy is in core, the call is queued on a priority basis. The operating system and all jobs calling the task must use the one task. (MASTER has no re-entrant tasks.) *DEF, described in section 5.4 is an example of a one-copy task.

4.6 TASK PRIORITY

The priority of a program task is computed as

priority = $a \cdot i+b$

- a inherited priority multiplier
- i priority of caller
- b inherent priority

If a = 1 and b = 0, the copy of the task inherits the priority of its caller; but, if a = 0 and b $\neq$ 0, the task has a fixed or inherent priority equal to b. Several other combinations of inherited and inherent priority are also possible. If the computed priority exceeds six bits ($a \cdot i+b \leq 77_8$), it is set to 77.

When EXEC seeks a task to place in execution, it looks among the ready tasks for the one with the highest priority.

Parameters a and b are set for library tasks at GLIB (library generation) time. User-supplied task parameters are set at a = 1 and b = 0 unless superseded by TASK card parameters (10.1.5). The inherited priority, i, is the priority computed for the calling task. When the caller is the job monitor, i reflects the priority of the job class (3.1).

A request for an I/O task inherits the priority of the program task with which it is associated.

4.7 INTER-TASK COMMUNICATION

Tasks within a job may communicate with each other by passing parameters, transferring common, and using the register file.

4.7.1 PARAMETER PASSING

A caller can pass parameters (511 maximum) to its callee and the callee may return parameters to its caller as a function of the CALL and RETURN macros (9.1). A caller cannot pass more parameters than a callee is prepared to receive. Nor can a callee return to its caller more parameters than it received. A task that is to receive or return parameters requires a non-standard copy of the user interrupt control routine (4.10).

A caller may pass parameters through the console registers; however, the callee cannot return parameters through console registers.

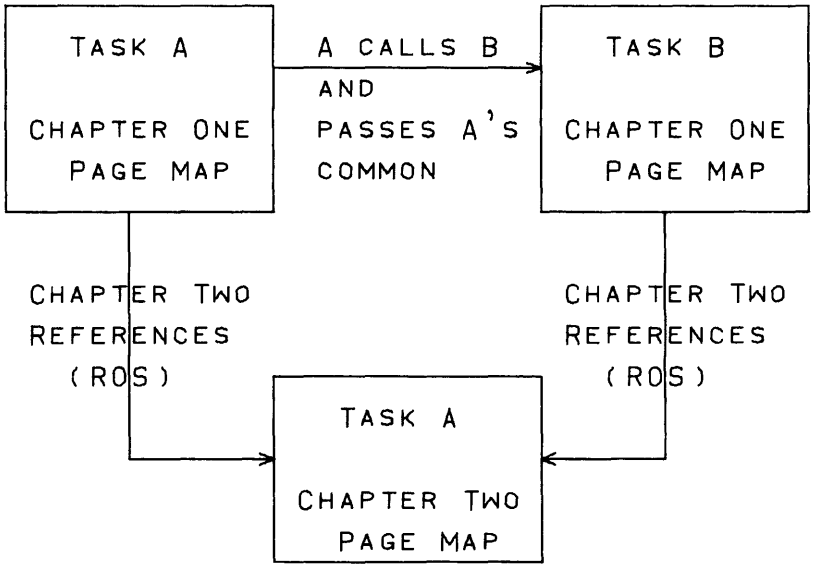
4.7.2 COMMON

Two-chapter tasks within a job can communicate with each other through Chapter Two common (3.5.2). The operating system passes Chapter Two common as a single contiguous block of logical addresses starting at address 00000. The user is responsible for linking common addresses since there is no symbolic linking at the time common is passed. For example, when Task A calls Task B, it may pass to B a beginning address as a parameter.

According to a parameter of the CALL, the callee may use common passed to it as read only or as both read and write. Another parameter of the call specifies whether the caller is passing its own common or common of the task that called it. As an example, assume three, two-chapter tasks, each having its own Chapter Two common.

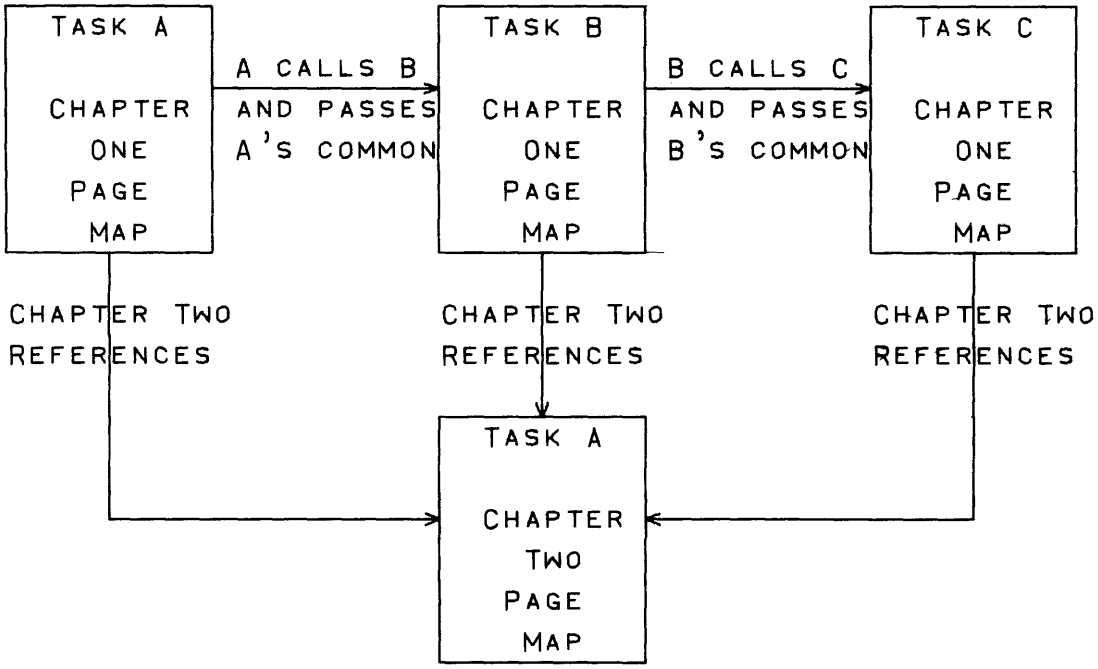
Case 1:

In calling Task B, Task A requests that its common be passed to B. Then, when B is placed in execution, the page map for A's Chapter Two is copied into the state in the page index file assigned to B's Chapter Two (3.5.2).



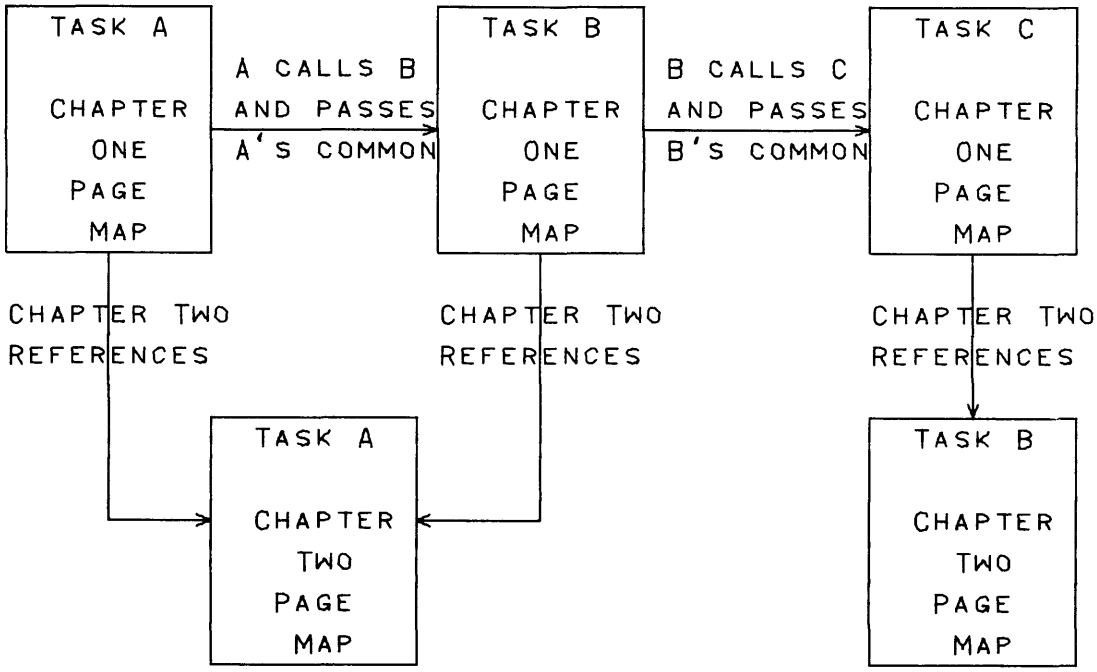
Case 2:

Task B calls Task C and passes its caller's common. Thereafter, when Task C references Chapter Two, it accesses the memory assigned to Task A's Chapter Two.



Case 3:

Task B calls Task C and passes its own common. Thereafter, when Task C references Chapter Two it accesses the memory assigned to Task B's Chapter Two. Note that, after B has been called by A, although it may not itself reference the common assigned to it at load time, B may pass its own common to its callees.



4.7.3 REGISTER FILE

All tasks of a job can communicate through locations 40_8-77_8 of the register file. When register file usage is declared on the SCHED card, the contents of the register will be maintained for the job (10.1.3).

4.8 MULTI-PROGRAMMING

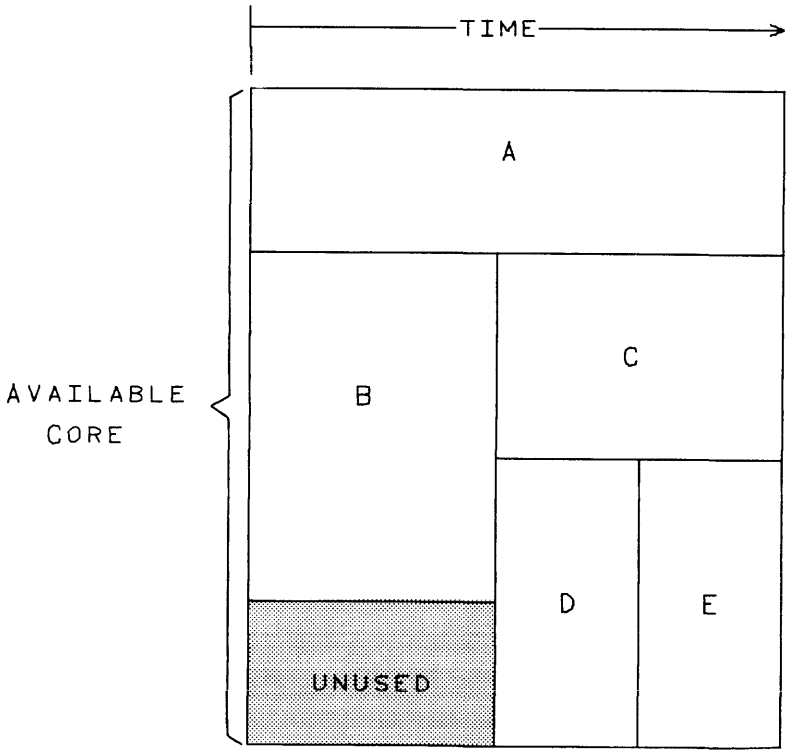
A programmer may segment a job into tasks to accomplish parallel execution among several tasks. Thus, Task A may call Task B and assume ready status. They may then time share the computer with one task executing while the other is waiting for I/O.

4.9 OVERLAY TASKS

A user with a job too large for existing core, should segment it into tasks.

For example: A user has a main task, A, and two overlay tasks, B and C. Task C in turn has two segment tasks, D and E. Task A calls Task B, which later returns and releases its core (9.1).

Task A then calls C, which can occupy the core previously occupied by B. Task C may then call D and E returning and releasing core in the same manner as Task B. Thus, E overlays D. The overlaying of subordinate tasks may be carried to any level. That is, Tasks D and E may in turn overlay subordinate tasks.



A subordinate task (B in the example) can be called several times by the main task (A) throughout the execution of a job. If Task B is to be loaded each time it is called from the file named in the CALL macro, the user may choose to expedite the processing, by creating an absolute file. (See RLDR).

4.10 TASK ENTRANCE/ EXIT

Each task must incorporate a library routine known as User Interrupt Control (UIC) that provides the task with:

- an entrance
- an exit
- a reserved area for parameter passing
- internal fault selection

A task begins at address tpep, the task's primary entry point, which consists of a jump to **.

Task

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
tpep	UJP	**	

A call to a task connects with the UIC routine which contains a return jump to tpep. The return address is inserted into ** as a normal function of a return jump execution. The return address in **, +1, is the first word address of parameters passed in the call.

A task can return to its caller through a jump to tpep or through a RETURN macro coded in the program.

A user may obtain a standard or non-standard copy of UIC. The standard copy does not include a parameter passing area and the RETURN included in UIC has no parameters. A standard copy of UIC includes a copy of the fault selection routine.

The non-standard copy may include parameters. It will not include a copy of the fault selection routine unless requested.

To obtain a standard copy of UIC, the programmer coding the task must declare UIC as an external symbol:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD    COMMENTS
1        8  10                    20               41
           EXT                   UIC
            .
            .
            .
```

The FORTRAN compiler, as a standard function of compilation, declares UIC as an external symbol; the COMPASS assembler does not. The user must declare the symbol.

To obtain a non-standard copy of UIC, the task programmer must use the following macro:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD    COMMENTS
1        8  10                    20               41
UIC         UIC                   (n,F,NR)
```

n Maximum number (0 to 511) of parameters task may receive or return to its caller.

F Task requires fault selection routine in UIC. When F is omitted the task cannot select faults (SELECT, section 9.3.1)

NR The task is not to be released from core. When NR is omitted, the task is released after its execution.

When a task requires a non-standard UIC, the task programmer must also declare UIC (as well as tpep) as an entry point.

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD    COMMENTS
1        8  10                    20               41
           ENTRY                 UIC
            .
            .
            .
```

4.11 TASK STATUS

During its life in the system, a task may repeatedly pass through many of the status conditions listed below. EXEC maintains and uses status in its administration of tasks. The code appears on the OUT file when the job terminates abnormally.

Octal Code	Status	Significance
00	Inactive	The task has completed its work and returned to its caller. Status remains inactive until the task is called again (RETURN).†
01	Wait	The task has called another task and, as a parameter of the call, requested that it (the caller) not be permitted to resume execution until its callee has returned. (CALL)†
02	Deferred Wait	The task has called other tasks, resumed execution and then requested that it not be permitted to continue execution until one of a set of callees returns (DWAIT).†
03	Wait I/O	The task has called for an I/O task and, as a parameter of the call requested that it not be permitted to resume execution until the I/O task has been completed. (Data Transmission Macros, Chapters 6 and 7).
04	Deferred Wait I/O	The task has called I/O tasks, resumed execution and then requested that it not be permitted to continue execution until one of the set of I/O tasks has been completed. (DWAITIO).†
05	Table Wait	The task is waiting for access to a table being used by another task.
06	File Wait	The caller is waiting for access to a file being used by another task (OCAREM, Chapter 5).
07	Read Lockout	The task has called the job copy of the deblocking routine while another task was using it.
10	Write Lockout	The task has called the job copy of the blocking routine while another task was using it.
11	Call	The task has called another task. Until the call can be connected, the caller remains in call status; it cannot resume execution.
12	Finis	The task has completed its work and is waiting to return to its caller (TASK has outstanding callees).
13	Select	The task has been called and is waiting to be loaded.
14	Ready	The task may begin or resume execution or is in execution.
15	Busy I/O	The task is not in execution but has I/O going on.

† Chapter 9

MASS STORAGE 5

The efficiency of MASTER's time sharing and multiprogramming depends on optimum random access of mass storage. A comprehensive set of MASTER routines efficiently manages and processes files on mass storage. This chapter describes the structure of mass storage files, their establishment, and access. Chapter 6 describes the processing of mass storage files.

5.1 FILE STRUCTURE

The MASTER system operates in an environment in which all files have an identical basic structure. All of mass storage for MASTER is subdivided in two levels. The device level is the higher; the units of allocatable storage level is the lower.

5.1.1 DEVICES

Mass storage devices are hardware entities with independent schemes of addressing. MASTER distinguishes between Class A devices which are logically or physically affixed to drives, and Class B devices which are removable.

All drives not mounted with Class A devices are available for any Class B device. Since Class A devices are drive associated, they constitute mass storage that is always on-line. Files of importance to MASTER itself are maintained on Class A devices.

5.1.2 DEVICE LABELS

MASTER uses device labels in identifying all mass storage devices. Each removable disk pack, as well as each drum and non-removable disk, has a device label written on track 0 (or the first hardware address) of its first allocatable unit. Device labels are written by MASTER utility routines prior to the use of the device.

Device labels contain information relating to mass storage devices including a device number that relates to a numer in an external label, such as a label on a disk pack.

Because files may extend over more than one mass storage hardware unit, not all of which may be on-line, MASTER checks device numbers in each call. If the call involves a device that is not on-line, MASTER issues a message requesting the operator to mount that device on a specified drive. The content and format of device labels are described in Appendix B, operator messages in Appendix C.

5.1.3 FILES

All data operated on by the MASTER system must be in entities of logical block structure. These entities are called files.

MASTER files are subdivided into groups that have identical logical block sizes. A logical block size is the number of 6-bit characters in each block. Logical block sizes may vary among files, but no file may exceed MSIO specified limits (Appendix B). Each logical block starts at the beginning of a physical hardware record, and may not exceed 131071 characters.

5.1.4 FILE LABELS

File labels are tabulated entries in system files (Appendix B) that identify and describe space on mass storage. A mass storage file exists in the system when the user defines a label (allocates). The user must provide information (file identification) that uniquely identifies and describes a file each time he makes a definition.

5.1.5 CREATING FILE LABELS

The important prerequisite to using MASTER's mass storage is that space for files must be labeled and reserved. The user makes calls to the MASTER operating system (*DEF task) to create a file label. These calls provide file identification, security codes, block size, block count, etc. MASTER, in turn, assembles this user-provided information, a mass storage map, and related information from various internal tables, into entries in system files (section 5.2).

5.1.6 FILE IDENTIFICATION

The concatenation of owner, file name, and edition number make up a file identification. Each file definition (file label) includes unique file identification, an access security code, and a modification security code. The file label retains the file identification for MASTER comparison during label handling calls(RELEASE, EXPAND and MODIFY, section 5.4.2). If identification in a call does not match identification in a label, an error results.

5.1.7 FILE SECURITY

Each file label has a provision for security codes. The user may supply an access security code, a modification security code, or both in a label definition call. The access security code protects a file from unauthorized use. If the access security code in an access call(OPEN)does not match one in a file label, the call is rejected. Similarly, mismatched modification security codes reject correspondent label handling calls RELEASE, EXPAND and MODIFY. Label access and modification codes can be overridden through use of the master access and modify codes in the call.

When space must be segmented to satisfy a file definition call, MASTER maintains a threaded map of segments and inserts it in the file label. One or more segments of a file may be on one or more physical units. *DEF allows files to be segmented up to a maximum of 63 segments per file.

5.2 FILE ENVIRONMENT

In order for MASTER to run and start job processing it needs a file environment. The user establishes the file environment by autoloading MASTER. The autoload program enters an intialize routine that generates nine mass storage files making up the initial file environment for MASTER. The table that follows outlines each of these nine system files.

COMMON NAME	MASS STORAGE DIRECTORY FILE	FILE LABEL DIRECTORY FILE		MASTER LIBRARY FILE	MASTER LIBRARY DIRECTORY FILE	JOB FILE POOL			
		FILE LABEL FILE	FILE IDENTIFIER FILE			INPUT	OUTPUT	PUNCH	SCRATCH
FILE IDENTIFICATION OWNER FILENAME EDITION NO.	MSIO MSDFILE 00	MSIO LABELFILE 00	MSIO IDFILE 00	MASTER LIBRARY 00-99	MASTER LIBRARY DIRECTORY 00-99	MASTER SYSTEM INPUT 00	MASTER SYSTEM OUTPUT 00	MASTER SYSTEM PUNCH 00	MASTER SYSTEM SCRATCH 00
SECURITY CODES ACCESS MODIFICATION	←			master security codes as specified by installation (See Summary)					→
BLOCKSIZE	←		variable (See Summary)	→		fixed length segments (See Summary) →			
DATA SET IDENTIFIER†	*MSD	*LAB	*IDF	*LIB	*DIR	INP, OUT, PUN (See Summary)			
Contents (See details in Summary)	copies of all device labels for all mass storage devices in the system; therefore, a complete mapping of all mass storage	labels of all files in the mass storage system	the file identification and security codes from all file labels in the mass storage system	complete library for MASTER See Summary	a directory of the MASTER library	a pool of permanently allocated contiguous mass storage that will be managed by EXEC for each job in the system See Summary			
Status after Initialize	opened and assigned data set identifier *MSD	opened and assigned data set identifier *LAB	opened and assigned data set identifier *IDF	opened and assigned data set identifier *LIB	opened by and assigned data set identifier *DIR	allocated by operating system task and segment maps read into EXEC tables. Managed by system OCARE on a job basis through user and operating system calls.			

† The data set identifier (dsi) is a shorthand file identification that the system and user employs in accessing files after their definition (Allocation). Thus, a file definition will be known to the system by only its dsi, eliminating the time-consuming use of the full file identification and, in the case of the job file pool, eliminating time-consuming disk accesses.

5.2.1 SYSTEM FILES SUMMARY

In general, the nine system files exist throughout the life of MASTER. They are maintained on permanently on-line (Class A) mass storage.

Among the five mass storage and library files, block sizes vary, according to the nature of their content, within the block size limitations of MSIO specifications (Appendix B). The size of each file in the job file pool is always in segments; length will vary according to each installation. Each file may be up to 63 segments, and will always be at least one.

Three system files (*MSD, *LAB, and *IDF) are involved with the management of the mass storage system. Their dsi's indicate that they may be accessed by the operating system only. The library files (*LIB and *DIR) are accessible by any job and task in the system in addition to the system itself. The files of the permanently allocated job file pool are accessible by both the system and user for jobs running in the system.

5.3 MASS STORAGE AND JOBS

To make the most efficient use of time sharing and multiprogramming capabilities, MASTER automatically file orients each job presented to it for processing.

5.3.1 JOB FILES

Each job presented to MASTER will automatically obtain a set of associated job files. MASTER operating system tasks (Job Flow, Chapter 2) make calls to a set of MASTER executive routines (System OCARE) that allocates, from the job file pool, files for each job. Each job may have the following files:

An input file, INP, which contains the job's input deck.

A printer output file, OUT, which receives the job's listable output.

A punch output file, PUN, which receives the job's punch output (providing a punch limit is specified on the JOB card.

Scratch files which the user must schedule and manage through calls to System OCARE.

Each of the above files exists only for the life of a job; all are manipulated through the MASTER operating system. The user is mainly concerned with the job's scratch. As the system completes a job, it closes and releases the input file (INP), closes and releases scratch files (if the user does not call System OCARE to do this, the system handles it automatically), and closes the printer and punch (OUT and PUN) files passing through to the operating system (Output Backgrounder) for listing and punching.

5.3.2 SYSTEM OCARE

System OCARE is a set of file management routines that manipulates the job file pool files. Since the pool is permanently allocated, and its segment maps are present in executive tables upon initialization, no new mass storage accesses are necessary to manage them. A job file exists only as a dsi (e.g., INP, OUT and PUN) relating to a segment map in MASTER executive tables.

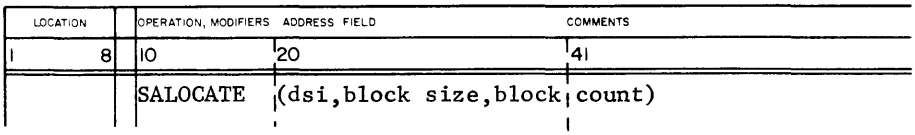
5.3.3 MANAGING JOB SCRATCH

As the user is responsible for scheduling and managing each job's scratch requirements, he makes macro calls from his own program to System OCARE. All macros must meet the specifications for macros outlined in 1.7. If a call is unsuccessful (function reject), an error code specifying the nature of the failure returns in the A register. The calls to System OCARE, and the resulting function of each call, for managing job scratch are described as follows:

SALOCATE

As a result of the system allocate call a scratch file is allocated, opened and assigned a data set identifier. The function manipulates segments from the standard file pool into a scratch file definition. SALOCATE sets the block count to one.

MACRO call format:



dsi	Data set identifier (1-4 alphanumeric characters) to be associated with the file; it must not start with *. The dsi is the only identification the file has while it exists in the current job.
block size	Number of characters (1 to 131072) per block in this file.
block count	Number of blocks (1 to 511) to be allocated to the file.

System OCARE allocates by segments. It computes the number of segments from block size and block count. The file may have from 1 to 63 segments.

Reject Codes

(A) = x0000000

x = 3	Requested more than 63 segments, or exceeded scratch segment count on schedule card (10.1.3).
1	Tried to allocate a file with dsi PUN, INP, or OUT.
5	dsi already in use, by *DEF OPEN call, for another open file in the job.
6	block size is zero.

If no room is available in the MASTER executive tables, or no segments are available, the calling task is placed in table-wait status until the condition clears. If the file has already been allocated, an attempt will be made to open it. All SALOCATE calls cause the file to be allocated and opened. It is ready for data transmission (Chapter 6).

SOPEN

The system open call reopens a file previously allocated by SALOCATE. The OPEN call identifies the file through the dsi declared in the SALOCATE request, and prepares it for data transmission. SOPEN sets the block count to one. See Chapter 6.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SOPEN	(dsi,block size)	

dsi	Data set identifier associated with the file; must be the same as in SALOCATE request.
block size	Number of characters (1 to 131071) per block in this file. (This may differ from the block size in SALOCATE request; however, the total number of characters allocated for the file does not change).

Reject Codes

(A) = x0000000

x = 4	No file with this dsi has been allocated by SALOCATE.
1	dsi is PUN, INP, or OUT.
5	File is already open in this job, by a *DEF OPEN call.
6	Block size is zero.

If the system executive tables are full, the calling task is placed in table-wait status until table space is available. If the file is already opened by SOPEN or SALOCATE, the request is a do nothing request.

SEXPAND

A system expand call increases the job's scratch file by one segment.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SEXPAND	(dsi)	

dsi	Data set identifier of the file.

Unlike *DEF expand (5.4.1) which allows expansion only on closed files, system expand can be made on either open or closed scratch files.

Reject Codes

(A) = x0000000

x = 1	Tried to expand PUN, INP, or OUT.
4	No file with this dsi has been allocated by SALOCATE.
3	Tried to expand beyond 63 segments or used up number of segments scheduled for this job.

If no segments are currently available, the calling task is put into table-wait status until the condition clears.

SCLOSE This call removes the job's scratch file definition (dsi) from the MASTER. executive tables, preventing data transmission until it is reopened by SOPEN.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SCLOSE	(dsi)	

dsi Data set identifier of file.

Reject Codes

(A) = x0000000

x = 1 Tried to close PUN, INP, or OUT file.

4 No file with this dsi has been allocated by SALOCATE.

SRELEASE The system release function releases the segments of mass storage associated with the dsi and returns them to the job file pool. The job's scratch files must be closed with SCLOSE before they can be released.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SRELEASE	(dsi)	

dsi Data set identifier of the file.

Reject Codes

(A) = x0000000

x = 1 Tried to release PUN, INP, or OUT files.

4 File with this dsi was never allocated.

2 File has not been closed by SCLOSE.

5.4 MASS STORAGE AND PERMANENT FILES

Permanent files are user files. The user has full control of creation, label manipulation, access, and life term. The manipulation of permanent files is determined by the user, and is limited only by the range of functions (*DEF) provided by the system.

5.4.1 *DEF

The MASTER operating system contains a task (*DEF) for handling permanent files that is callable by any other operating system task, library tasks, or program tasks. *DEF is a package of routines (OCAREM) that the user calls to manage mass storage.

*DEF is callable in either of two ways:

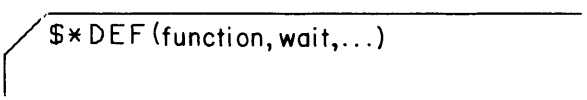
1. By call macros from within a dynamic program.
2. By Task Name control card calls (where the task name is *DEF) preceding or following a dynamic program.

General format of call macro:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	function	(wait,....)	

Macro calls are standard COMPASS library macros that reference library macros the user establishes in LIBM statements at the beginning of his program.

Task Name cards are described in section 10.1.4. The general form of the *DEF Task Name control card call is:



In both types of *DEF call formats:

function — Specifies the *DEF routine being called.

wait — Defines action to be taken when the call causes a conflict with the use of the file across jobs. The wait parameter may be W or R.

- W — If a conflict occurs between jobs *DEF places the calling task in file wait status. When the conflict is resolved, it honors the call automatically. If the conflict is within a job, the call is rejected.
- R — If a conflict occurs on a macro request, *DEF rejects the call, and returns a reject code to the caller. If the conflict occurs when the R parameter is used on a Task Name control card call, *DEF terminates the job making the call with an error message to the operator.
- Other — If wait is other than W or R, MASTER interprets it as W.

*DEF functions may be divided into two subgroups -- label handling and data transmission.

5.4.2 LABEL HANDLING FUNCTIONS

One subgroup of *DEF (OCAREM) supplies the user with a broad range of functions for manipulating the permanent file definitions. In a sense, these functions manipulate file definitions (file labels) external to MASTER. The user may call any of these functions to create a file definition to build entries in system files (* LAB, *MSD, and *IDF), reserve additional mass storage for a definition, remove the definition from all system files and tables, or change the definition itself. The permanent file label handling functions of *DEF (OCAREM) are described below:

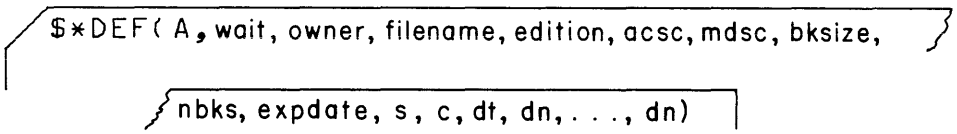
ALLOCATE

The ALLOCATE function of *DEF uses the information supplied in the call to build entries in the *LAB, *IDF (thereby creating a file definition in the system). The function also updates *MSD to reflect the units of mass storage allocated to the definition, and builds entries in executive tables.

MACRO call format:†

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ALLOCATE	(wait,owner,filename,edition,acsc,mdsc,	
	ALLOCATE	(bksize,nbks,expdate,s,c,dt,dn,...,dn)	

Task Name control card format:



The parameter groupings in both formats are described below:

File identification information

owner	1 to 8 alphanumeric characters
filename	1 to 30 alphanumeric characters
edition	1 to 2 alphanumeric characters

File security information:

acsc	Access security code 1 to 4 alphanumeric characters.
mdsc	Modification security code 1 to 4 alphanumeric characters.

File structure specification:

bksize	Number of 6-bit characters per logical block, decimal constant 1 to 131071.
	Default: On control card call, the job containing the call is terminated. On macro call, fault diagnostic code 70 returns in the status word.
nbks	Number of logical blocks in the file, decimal constant 1 to 8388607.

† Macro continuation line required if information extends beyond column 72.

Default: On control card call, the job containing the call is terminated. On a macro call, fault diagnostic code 70 returns in the status word.

File expiration date:

expdate — Expiration date of file in the form yymmdd; yy is year, mm is month, and dd is day.

Default: Current date will be used.

Hardware specification:

s

S: File is to be allocated on devices formatted in sector mode.

T: File is to be allocated on devices formatted in track mode.

Default: File will be allocated on devices formatted in sector mode, if applicable to specified device type.

c

C: File must be allocated to contiguous area (one segment).

S: File segmenting is permitted.

Default: Segmenting is permitted.

dt

852, 853, 854, 813† — Model number of disk on which file must be allocated.

863 — File is to be allocated on drum.

Default: File is allocated to models specified in an installation parameter.††

dn

Device numbers (Class B) to be used for allocation of the file numbers are considered in the order listed. The list may have a maximum of 9 device numbers; each may range from 1 to 262143.

Default: File is automatically allocated on Class A devices of type specified. Error code 58 indicates no Class A devices are available.

---

† Two 813's constitute an 814 in a single cabinet.

†† May be changed by the installation at library generation time, see MASTER Installation Manual.

Additional file label entries made on ALLOCATE request:

File protection:
: Automatically set to 0 (INPUT and/or OUTPUT) on an allocate request, but can be changed to I (INPUT only) on modify request.

Usage count:
: Automatically set to zero.

Creation date:
: Automatically set to current date.

Last access date:
: Automatically set to 00 00.

File identification must be unique or a master file identification. Allocation may not be across storage classes (part on Class B and part on Class A). If device numbers are listed, they must all be Class B mass storage with a specified hardware type and mode.

Allocation Algorithms

Class A storage:
: This algorithm minimizes the number of segments per file. *DEF checks *MSD 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 file segment followed by the next largest, etc.

Class B storage:
: This algorithm searches maps of each device in the device number list. The search proceeds serially (starting with device containing the most available storage) to minimize the number of devices per file.

Error processing:
: Error processing during an allocate call varies with the format of the call. An error prevents allocation.

Errors occurring in the course of a call using a *DEF(A,...) control card terminate the job making the call. The condition is reported through an error message. If errors occur in the course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
52	Calling sequence contains an illegal device type and/or recording mode.
54	Requested file size exceeds the MSIO specification limits.

Error Code	Condition
55	File identifier already exists.
56	*LAB and *IDF are full. Operator is informed through a diagnostic message.
57	No device of requested type and mode in system.
58	Not enough space available to allocate a file of this size.
59	Maximum segment count would have been exceeded if allocation was completed.
60	User requested contiguous space, but no contiguous block of sufficient size was available.
70	User omitted bksize or nbks parameter in call.

MODIFY

Through the MODIFY call, the user may change elements of a permanent file definition that do not concern the mapping of the definitions' structure. He may change owner's name, file name, edition, access security code, modification security code, and file protection or expiration date.

MACRO call format†:

LOCATION 1 8	OPERATION, MODIFIERS 10	ADDRESS FIELD 20	COMMENTS 41
	MODIFY	(wait, owner, filename, edition, acsc, mdsc,	
	MODIFY	new owner, new filename, new edition, new acsc,	
	MODIFY	new mdsc, new protection, new expdate)	

Task Name control card format:

```
$*DEF( M, wait, owner, filename, edition, acsc, mdsc, new owner,
       new filename, new edition, new acsc, new mdsc, new protection,
       new expdate )
```

† Macro continuation line required if information exceeds column 72.

The following parameters are mandatory in both formats.

File identification information for the file to be modified:

owner	Owner of file
filename	Name of file
edition	Edition number

File security information:

acsc	Access security code
mdsc	Modification security code

The following parameters are optional, and specify any changes to be made to the file label in *LAB and *IDF. A null or blank parameter specifies no change in that field of the label.

new owner	New owner identification; 1 to 8 alphanumeric characters.
new filename	New filename identification; 1 to 30 alphanumeric characters.
new edition	New edition identification; 1 to 2 characters.
new acsc	New access security code; 1 to 4 characters.
new mdsc	New modification security code; 1 to 4 characters.
new protection	O: file may be used for input and/or output. I: file may be used for input only. blank: the protection does not change. other: the file is input only.
new expdate	New expiration date in the form yymmdd where yy is the year, mm is the month , and dd is the day.

The file identification and the access and modification security codes (of the file to be modified) must match those in an entry in *LAB and *IDF. If wait is requested and the file is open for data transmission in another job, the call will be completed as soon as the file is closed; otherwise, the call is rejected in spite of wait request. The modified file label identification must be unique (not the same as an existing file).

Error Processing

Error processing during a MODIFY call varies with the format of the call. If errors occur, modification is not done. If errors occur when the *DEF (M,...) control card is used, the job making the call is terminated; the condition is reported through an error message. If errors occur in the

course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
55	New file identifier is not new, already exists for a file other than the one being modified.
61	Original file identification not present in *LAB and *IDF.
62	The file to be modified is open in the same job that is making the call.
63	One, or both, security codes do not match the security codes in the file. The access and modification security codes must match the corresponding fields in the label or the master access and modification security codes.

EXPAND

The user may make EXPAND calls to *DEF to enlarge the amount of mass storage space for an existing definition. The user specifies how many additional blocks are needed (optionally the device numbers).

MACRO call format:†

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	EXPAND	(wait,owner,filename,edition,acsc,mdsc,nbks,	
	EXPAND	seg,dn,...,dn)	

Task Name control card format:

```
$*DEF(E,wait,owner,filename,edition,acsc,mdsc,nbks,seg,dn,...,dn)
```

The following parameters identify the file to be expanded:

owner	Owner of file.
filename	Name of file.
edition	Edition number.

† Macro continuation line required if information exceeds column 72.

File security information:

acsc	Access security code.
mdsc	Modification security code.

Expansion specifications:

nbks	Number of blocks to be added to the file using the block size specified in the label. A decimal integer from 1 to n, where n plus the number of blocks already in the file must not exceed 8388607. Default: No expansion will be done.
seg	C: Blocks to be added to the file must be contiguous (with each other, not necessarily with the original file). S: Additional portion may be segmented. Default: Assume segmenting permitted.
dn	Device numbers to be used for the expansion; refer to Class B devices, and are considered in the order listed. The list may have a maximum of nine device numbers. Each device number may range from 1 to 262143. Default: If original definition was on Class A devices, expansion will be on Class A. If original definition was on Class B devices, same devices (device numbers) will be used for the expansion.

The number of blocks specified in the call does not actually reserve as many blocks, but it specifies an amount of mass storage that is to be added to the mass storage map of a file definition. The additions may be contiguous with each other, or segmented, but not necessarily contiguous with the original definition. If wait is requested and the file is open for data transmission in another job, the call will be completed as soon as the file is closed; otherwise, the call is rejected in spite of wait request. The file identification, access security code, and modification security code in the EXPAND call must match an entry in *LAB and *DEF, or the call is rejected.

Error Processing

Error processing in an EXPAND call varies with the format of the call. If errors occur, expansion is not done, and the original file definition is not disturbed. Errors occurring in the course of a call using a *DEF (E,...) control card terminate the job making the call. The condition is reported through an error message. If errors occur in the course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
54	File size, if expanded, would exceed the limits of MSIO specifications.
57	The call listed an illegal device number.
58	Additional mass storage is not available.
59	MSIO limitations on segment count would have been exceeded if the expansion had been completed.
60	Caller requested contiguous space, but no contiguous area big enough was available.
61	File identification in the call does not match any in *LAB or *IDF.
62	File with this definition is open for data transmission in the same job.
63	One, or both, security codes do not match the codes in the file label or master security codes.

RELEASE

The user may remove the definition of a permanent file from MASTER executive tables, *LAB, and *IDF by calling the RELEASE function. The space associated with the definition returns to the mapping of mass storage in *MSD.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	RELEASE	(wait,owner,filename,edition,acsc,mdsc,amount)	

Task Name control card format:

```
$*DEF( R, wait, owner, filename, edition, acsc, mdsc, amount)
```

File identification information:

owner	Owner.
filename	Name of file.
edition	Edition number.

Security information:

acsc	Access security code.
mdsc	Modification security code.

Release specification:

amount	ALL:	Release the definition and all associated mass storage.
	UNUSED:	Release all of the blocks above the last block written.
	1 to 7 decimal digits:	Specifies the number of trailing blocks to to be released.
	Other:	Causes error.

The file identification and the access and modification security codes, of the definition undergoing release, must match entries in *LAB and *IDF or the call is rejected. If wait is requested and the file is open for data transmission in another job, the call will be completed as soon as the file is closed; otherwise, the call is rejected in spite of wait request. A zero or a blank in the release specification field (amount) of the call is a no operation.

Error Processing

Error processing during a release call varies with the format of the call. If errors occur, release is not done. Errors occurring in the course of a call using a *DEF(R,...) control card terminate the job making the call, and inform the user of the condition through an error message. If errors occur in the course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
61	File identification in the call does not match a *LAB entry.
62	Definition to be released is open for data transmission in the job making the call.

Error Code	Condition
63	Both security codes do not match any security codes *LAB and *IDF entries.
70	Call contains an illegal control value.

5.4.3 TRANSMISSION PREPARATION FUNCTIONS

A second functional subgroup of the *DEF task (OPEN and CLOSE) is callable to establish a file definition in MASTER executive tables. It sets up the record keeping that prepares a definition for data transmission; or it removes a definition from the executive tables so that no further data transmission may be done.

OPEN

The user may call the OPEN function of *DEF to prepare existing file definitions for data transmission. The OPEN function sets up internal tables and, where necessary, requests the operator to prepare mass storage devices associated with the file definition. When open, a file may be involved in data transmission functions (refer to Chapter 6). The OPEN function opens both Class A and Class B mass storage, which can be mounted on-line together. MASTER puts all mass storage devices, associated with a file definition, on-line during an OPEN call.

MACRO call format :†

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	OPEN	(wait,dsi,owner,filename,edition,acsc,usage,S,	
	OPEN	block,bypass)	

Task Name control card format:

```
$*DEF( 0, wait, dsi, owner, filename, edition, acsc, usage, s,
                                                  {block, bypass)
```

dsi — Data set identifier, 1 to 4 alphanumeric characters, may not be blank or start with *. In lieu of total identification, *DEF uses a data set identifier in referencing file definitions to reduce the number of disk accesses which reduces the total time requirements with mass storage.

† Macro continuation line required if information exceeds column 72.

File identification:

owner	Owner of file.
filename	Name of file.
edition	Edition number.

File security information:

acsc	Access security code.
usage	O: File may be used for output, or input and output.
	I: Designates an input only file.
	Defaults: Assumed input only file.

The following two parameters specify partial opening of a definition. Only one device, associated with a definition, will be put on-line at a time. Only definitions allocated on Class B mass storage may be opened partially. Definitions on Class A mass storage are always completely on-line.

S	Definition will be partially opened. When S is omitted, definition will be opened normally.
block	This parameter is used only if S is used. It may be an unsigned decimal integer in the range 1 to 8388607. The device containing this block number is put on-line.

The user calls the OPENSEG (open segment) function when it becomes necessary to open a new segment of a definition, and the new segment is on a different device. The user must specify a new block number on the device.

bypass	B:	Bypass all input and output requests.
	Other:	Process normally (do not bypass).

The file identification and access security code in the OPEN call must match an entry in *LAB and *IDF or the call is rejected. If drive requirements for definitions on Class B storage are not scheduled in advance of OPEN calls, the calls are rejected. The usage specified must not conflict with protection specified in the original definition. If the dsi matches a dsi already open in the same job, the conflict terminates the job making the call. If the dsi matches a dsi open in another job, no conflict occurs since dsi's are further qualified by MASTER. A partial OPEN call has no meaning for definitions on Class A mass storage. It is treated like a normal OPEN request. The block number on a partial OPEN must be in range 1 to n, where n is the highest block number of the allocated file.

Error Processing

Error processing during an OPEN call varies with the format of the call. If errors occur, opening is not done. Errors occurring in the course of a call using a *DEF (O,...) control card terminate the job making the call, and inform the user of the condition through an error message. If errors occur in the course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
61	File identification specified in the call does not match any in *LAB and *IDF.
63	Access security code in the call does not match any *LAB and *IDF. This error also occurs when the operator does not respond to a MASTER request to mount a file.
64	Data set identifier is illegal, i.e., already in use in the same job, contains an * as its first character, or is blank.
67	Usage specified in call conflicts with usage specified in the original definition.
69	Scheduling error - not enough drives to put a Class B definition on-line.
71	Definition open in this same job, which is in job wait status.
72	No drives available. This system error may occur if the operator, or the system, marks a scheduled unit as down and not available after job initiation.
73	Block number out of range on partial OPEN call.

OPENSEG

The user may make an OPENSEG call to open a new segment of a definition after a partial OPEN call (an OPEN call to *DEF) on that same definition. If the user requests data transmission to or from a block, that is part of a partially open file not currently on-line, the MASTER I/O Control System rejects the call, and reports the condition with a code and the block number of the last call. In such case, the user can call the open segment function of *DEF, providing the dsi and block number in the new segment to be put on-line, and then request data transmission on that block.†

† This technique should be used only on sequentially processed files. Random accessing degrades system performance because a great deal of operator action is necessary to continually mount new devices.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ALLOCATE	(wait,dsi,block)	

Task Name call format:

```
$*DEF( S, wait, dsi, block)
```

dsi — Data set identifier of definition opened with partial OPEN.

block — Number of the block in the segment to be put on-line.

The dsi in the call must match the dsi of a definition opened with a partial OPEN request.The block number in the call must be in the range 1 to n where n is the highest block number allocated to the file. Segments of partially open files do not have blocks which cross from a segment on one device to a segment on the next device because the MASTER Input/Output Control System is not able to read such blocks.

Error Processing

Error processing during an OPENSEG call varies with the format of the call. If errors occur, the open segment is not done. Errors occurring in the course of a call using a *DEF(S,...) control card terminate the job making the call. The user is informed through an error message. If the errors occur in the course of a macro call, an error code returns in the status word of the call as follows:

Error Code	Condition
73	Block number out of range.
75	Illegal dsi - no definition with this dsi open in this job, or definition is not partially open.

CLOSE

The close call to *DEF removes a file definition from MASTER executive tables. The definition is then no longer available for data transmission under MASTER.

MACRO call format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	CLOSE	(wait,dsi)	

Task Name control card format:

```
$*DEF(C,wait,dsi)
```

dsi — Data set identifier of file to be closed. If blank, the request closes all files associated with the job making the request except INP, OUT, PUN, and job scratch.

The CLOSE call inserts information pertaining to the use of the definition in *LAB, i.e., last access date, usage count, and number of highest block written. CLOSE calls on non-existent dsi's (definitions not open) are rejected with an error code. CLOSE calls on system files or job files (INP, OUT, PUN, or job scratch) are no operations.

Error Processing

Error processing during a CLOSE call varies with the format of the call. If errors occur, closing is not done. Errors occurring in the course of a call using a *DEF(C,...) control card terminate the job making the call. The condition is reported in an error message. If an error occurs in the course of a macro call, an error code returns in the status word of call as follows:

Error Code	Condition
64	dsi in the call does not match any open in the same job.

5.4.4 CALL CONFLICTS

MASTER provides alternatives for handling calls which it cannot honor because they conflict with previous calls (generally from another job) yet outstanding. MASTER honors calls on the basis of a parameter (wait-reject option) in each call. Tasks containing calls to *DEF are automatically put in file-wait status. If a call cannot be completed at the time it is made, one of the following actions is taken (in accordance with the wait-reject parameter):

1. If wait is chosen, the calling task is left in file-wait status, and an entry is made in the file-wait table. When the conflict clears (another job closes it), the file is returned to ready status, and the call is reissued automatically.

2. If reject is chosen, the calling task is immediately put into ready status, and the operator is informed of the reject in a status parameter. It is then up to the calling task to determine whether or not the call should be reissued and when.† If a conflict occurs during a Task Name control card call and reject is chosen, the job containing the calling task is terminated.

The following conflict conditions make use the wait-reject option:

1. A RELEASE, EXPAND, or MODIFY call on an open definition for any job in the system.

2. Any OPEN call on a definition already open for output only in any job in the system.

3. An OPEN call for output only on a definition already open in any job in the system.

4. A partial OPEN call on a definition open in any job in the system.

5. Any OPEN call on a definition open under the partial or segmented option in any job in the system.

6. An OPEN call when there is no space in the MASTER executive tables to make entries to complete the call.

Any call going into wait status on a definition open in the same job will be rejected with an error code in the status area of the call. The call request is not put into wait status, regardless of the wait parameter.

Reject Code	Condition
62	Definition open when a RELEASE, EXPAND, or MODIFY call occurs.
65	No system table space is available (closing a definition may clear space).
66	The definition is busy as in conditions 2,3,4, and 5 above.

† This option requires more machine time to process the request.

MASS STORAGE I/O 6

MASTER provides comprehensive I/O facilities to complement its mass storage management. These facilities are divided into two interdependent levels; logical I/O uses the physical I/O for actual reading and writing.

6.1 LOGICAL I/O

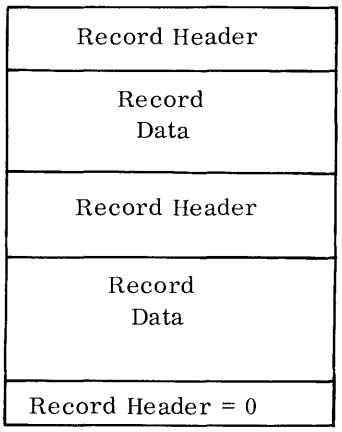
MASTER moves logical records to and from a buffer area, and automatically transmits buffers to or from an I/O medium when the buffer is full or empty.

6.2 BLOCKER AND DEBLOCKER

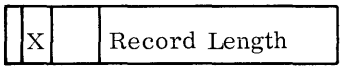
Logical I/O, called blocker and deblocker, consists of a set of MASTER library routines (four in blocker and four in deblocker) callable through a set of system macros from any task in a job. The entire set of routines and a set of buffers are loaded once for each job. A job should use blocker and deblocker for I/O operations on its standard files (INP, OUT, and PUN). Each task may use the blocker and deblocker for user files. The ability to perform I/O with logical records in a reduced number of data transfers results in more efficient use of the MASTER I/O system.

For a working understanding of the blocker and deblocker, the user should be familiar with block format, buffers, and buffer formats.

The block format is characterized by an alternate series of header records and record data areas, ending with a zero header record. The block format, applicable to both job files (INP, OUT, and PUN) and permanent (user) files, appears as follows:



A record header is a 24-bit data word containing information about the record data area that follows it. A record header appears as follows:

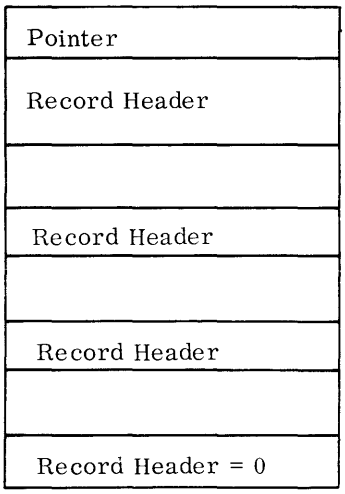


Record Length	Count of 24-bit words of record data in bits 0-14.
X	Mode of record data in bit 22:
	0 BCD
	1 Binary

A block for a job file is always 1280 characters from the first record header to the zero record header inclusive. The size of a user file block varies according to the size of the buffer area.

The blocker and deblocker uses a buffer in each I/O operation. Each job file (INP, OUT, and PUN) has a permanently assigned buffer within the job's protected core area. The user must establish buffers for non-job files (user files) within the program area of a calling task.

The format of a job file buffer is the same as a job file block (1280 characters) preceded by a pointer word. A user file buffer has the same basic format, but varies in size according to the user's specification. The basic format of a buffer appears as follows:



The pointer uses one word of the buffer to point to available space or a new logical record in the block. The file never contains the pointer.

6.2.1 BLOCKER

The blocker is a set of four routines, callable by system macros (PACKD, PACK, PACKC and PACKR) that perform blocking on job files and user files. All files to be blocked must have been previously opened.

PACK DEFINE

The user may call this function to define, within the calling task, the blocking area (buffer) to be associated with a non-standard file dsi. The blocker's file definition table has space for ten output file entries (dsi's). All tasks in a job may request definition of not more than ten output user files at one time. †

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PACKD	(dsi,reject address,bfwa,blength)	

dsi	Data set identifier for a user file, 1 to 4 BCD characters.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.
bfwa	First word address of buffer area to be used by the blocker.
blength	Length of the block area in words.

The following is an example of calling pack define on a user file named NSDF:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PRG		
	LIBM	PACKD	
BLAREA	00	*+1	
	00	0	
	BSS	499	
RCDAREA	BSS	249	
	:		
	PACKD	(NSDF,REJECT1,BLAREA,501)	Defines file.

† Exercise caution when making a PACK call on a user file because the task calling pack define is the only task defining the block area for the call. Any task other than making a PACK call on a user file may destroy the job.

If errors occur, pack define makes an error return and reports an error code in the Q register, Q = X0000000 as follows:

Error Code	Explanation
X = 2	dsi already defined.
3	Blocker's file definition table full (10 output files have been defined).
4	dsi never opened.
5	Defined block area out of bounds, and within blocker and deblocker area.

If the user does not totally define the block area within the calling task, pack may terminate the job. The user must set the first word of the block to zero, and place the address of BFWA + 1 in the buffer's pointer before making a pack define call.

PACK

The user may call the blocker pack function to move a record to the file's block area. In moving a record, pack removes trailing zeros (if the record is binary) or trailing blanks (if the record is BCD), but does not remove them from the caller's record buffer. If packing a record would cause the buffer area to overflow, pack writes the existing block on the file and then packs the record into the empty buffer.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PACK	(dsi,mode,reject address,fwa,rlength)	

dsi	Data set identifier of receiving file, 1 to 4 BCD characters.
mode	0 BCD 1 Binary
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.
fwa	First word address of record to be moved.
rlength	Number of words in the record.

Following is an example of calling PACK to block a binary record for a user file called ABLE:

```
| LOCATION | | OPERATION, MODIFIERS | ADDRESS FIELD | COMMENTS |
|----------|-|----------------------|---------------|----------|
| 1      8 | | 10                   | 20            | 41       |
|          | | LIBM                 | PACK          |          |
| ABLE1    | | BSS                  | 100           |          |
|          | |  :                   |               |          |
|          | |  :                   |               |          |
|          | | PACK                 | (ABLE,1,RERROR,ABLE1,100) |
```

If errors occur, pack makes an error return, and reports an error code in the Q register, Q = X0000000, as follows:

Error Code	Explanation
X = 1	Pack reached allocation limits on an output file. Last record passed not accepted, and block area still contains records.
2	Word count too large to fit block area (after removal of trailing blanks or zeros). For user files, depends on size of block area.
4	dsi not previously defined by a pack define call, or file not open.
6	Pack was unsuccessful in trying to mount a new segment on a file.
7	Defined user file block area out of bounds to calling task.

If the user calls PACK on a partially open file, PACK automatically requests mounting of necessary files that are not on-line.

PACK CLEAR

The user may call the blocker's pack clear function to write a block (on a user file) before its block area is full.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PACKC	(dsi,reject address)	

dsi	Data set identifier of user file, 1 to 4 BCD characters.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.

Following is an example of calling pack clear to write an incomplete block on a user file named JIM:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	LIBM	PACKC	
	.		
	.		
	PACKC	(JIM,RERROR1)	

If errors occur, pack clear makes an error return and reports an error code in the Q register, Q = X0000000, as follows:

Error Code	Explanation
X = 4	dsi not previously defined.
1	Pack clear reached allocation limits when attempting to write the current block on an output file, the buffer cannot accept the last record passed, and the block area still contains records.

PACK REMOVE

The user may call the blocker's pack remove function to remove a user file definition (dsi) from the pack table. For instance, if an I/O operation is necessary and the pack table is full (10 files defined), he may call pack remove to provide room for a new entry.

Macro format:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD              COMMENTS
1        8  10                    20                        41
           PACKR                 (dsi,reject address)
```

dsi	Data set identifier of the file.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.

If an error occurs, pack remove makes an error return and reports an error code in the Q register, Q = X0000000, where X = 2 indicating that a file with this dsi was never opened.

Following is an example of defining a file and block area, packing a binary record, clearing the block, and removing the definition from pack's table, for a user file named TAPE.

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD              COMMENTS
1        8  10                    20                        41
           PRG
           LIBM                  PACK,PACKD,PACKC,PACKR
BLOCK      00                    *+1
           00                    0
           BSS                   999
RECORD     BSS                   800
            .
            .
            .
           PACKD                 (TAPE,REJECT1,BLOCK,1001) Defines file called TAPE.
           PACK                  (TAPE,1,REJECT2,RECORD,800) Blocks a binary record.
           PACKC                 (TAPE,REJECT3)            Writes block even though it is
                                                             not full.
           PACKR                 (TAPE,REJECT4)            Removes file from the tables;
                                                             it cannot be referenced by a
                                                             PACK call until it has been
                                                             defined again.
```

6.2.2 BLOCKER SUMMARY

Blockers functions affect job output files (OUT and PUN) only.

In all functions, the blocker sets bit 23 in the first record header if it cannot write the block without errors. If a pack writes a defective block (bad spot), the blocker will automatically complete the write, clear bit 23, and issue a new write of the same information in the next block.

6.2.3 DEBLOCKER

The deblocker is a set of four routines, callable by system macros (PICKD, PICK, PICKC, and PICKR), that perform deblocking on job files and user files. All files to be deblocked must have been previously opened.

PICK DEFINE

The user may call the pick define function to: define a user file dsi, and its associated buffer area, in the calling task, into which blocks will be read for deblocking. The deblocker's file definition table has space for 10 entries for input files (dsi's). All tasks in a job may not ask for deblocking on more than 10 input files at a time. †

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PICKD	(dsi,reject address,bfwa,blength)	

dsi	Data set identifier for a user file, 1 to 4 BCD characters.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.
bfwa	First word address of area used to deblock.
blength	Length of deblocking area in words.

† Exercise caution when making a pick define call on a user file, because the task making a pick define is the only task defining the deblocking area for the call. A task making a PICK or pick clear call on a non-standard file, other than the task defining the deblocking area, may be destructive to the entire job.

Following is an example of calling pick and pick define to deblock a user file named NSDF:

```
LOCATION  OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1      8  10                   20                     41
          PRG
          LIBM                 PICK,PICKD
BLAREA    00                   *+1
          00                   0
          BSS                  499
RCDAREA   BSS                  249
           .
           .
           .
          PICKD                (NSDF,REJECT1,BLAREA,501)  Defines file.
          PICK                 (NSDF,0,REJECT2,RCDAREA,249)  Deblocks a record and
                                                             places it in RCDAREA.
```

If errors occur, pick define makes an error return and reports an error code in the Q register, Q = X0000000, as follows:

Error Code	Explanation
X = 2	dsi already defined.
3	Deblocker's file definition table full; 10 input files have been defined.
4	dsi never opened.
5	Defined block area out of bounds and within blocker and deblocker area.

Pick define uses the buffer pointer to point to the next logical record to be picked. The user must set the first word of the deblocking area to zero, and place the address of bfwa + 1 in the buffer's pointer before making a pick define call.

PICK

The user may call the pick function to move a record from the file buffer into the user's record area or pass a pointer to the record to the caller. If the caller does not want the record moved, he receives the first word address of the corresponding record header in the Q register.†

† Exercise care when the task using pick is not to receive the record. If another task is using INP, the record in the block area (not passed in the preceding task) could be destroyed.

If the caller requests passage of the record and the record is smaller than the record area, the pick function fills the remainder of the record area with blanks (BCD) or zeros (binary). Pick truncates the record if it is larger than the user's record area. If pick returns normally from handling a binary record, it sets bit 23 of the Q register. For a BCD record, bit 23 of the Q register is not set.

Macro format:

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD                    | COMMENTS
1        8 | 10                 | 20                              | 41
         |  PICK                | (dsi,key,reject address,fwa,rlength)
```

dsi	Data set identifier for a file, 1 to 4 BCD characters.
key	Record passing specification: 0 record is passed. 1 fwa of header is passed.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.
fwa	First word address of buffer to receive record.
rlength	Size of buffer in words.

Following is an example of picking and passing a record for a file named ABLE.

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD                    | COMMENTS
1        8 | 10                 | 20                              | 41
         |  LIBM                | PICK
ABLE1    |  BSS                 | 100
         |   .
         |   .
         |   .
         |  PICK                | (ABLE,0,REJECT1,ABLE1,100)
```

In the following variation of the above, the fwa of the header record is passed to the caller:

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD                    | COMMENTS
1        8 | 10                 | 20                              | 41
ABLE1    |  BSS                 | 100
         |   .
         |   .
         |   .
         |  PICK                | (ABLE,1,REJECT2)
```

If errors occur, pick makes an error return, and reports an error code in the Q register, XY00ZZZZ, as follows:

Error Code	Explanation
X = 1	Pick reached the end of an input file (block buffer is empty and no record has been passed) and tried to read a block that was never written, or tried to read beyond allocation in a user file.
4	dsi not previously defined by a pick define.
5	Defined record area within blocker and deblocker area.
6	Pick unsuccessfully tried to mount a new segment on a file.
7	Defined block area out of bounds to calling task.
Y = 4	Irrecoverable read error where records are in the block, none were passed; and as far as can be determined, their word counts are not affected.
5	Irrecoverable read error where the records are in the block, none were passed, and the word count on one or more is bad.
ZZZZ	Contains a count of the number of records in the block at the time of an error only if error involves records without bad word counts.

Error Summary

If a pick causes an irrecoverable read error, but word counts are good, the next pick will read the first record of the bad block. If a pick causes an irrecoverable read error, but one or more word count is bad, the next pick will read a new block. If the user requests a pick of partially open file, pick requests that the operator mount all needed files not on-line.

PICK CLEAR

The user may call the pick clear function to read a record from a new block before all records have been read from the last block on a user file.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PICKC	(dsi,reject address)	

dsi	Data set identifier for a user file, 1 to 4 BCD characters.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.

The following is an example of a pick clear on user file named JIM:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	LIBM	PICKC	
	.		
	.		
	.		
	PICKC	(JIM,RERROR)	

If errors occur, pick clear makes an error return, and reports an error code in the Q register, Q = XY00ZZZZ, representing the same error codes and causes as apply in the pick description.

PICK REMOVE

The user may call the pick remove function to remove a user input file buffer definition from pick's tables. If an I/O operation is necessary when the pick table is full (10 input files defined), the user may call pick remove to get room for a new entry.

Macro format:

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1        8  10                    20                    41
           PICKR                 (dsi,reject address)
```

dsi	Data set identifier of file being removed.
reject address	If errors occur, a return jump is done to this address. If blank, the program continues in normal sequence.

If an error occurs, pick remove makes an error return, and reports an error code in the Q register, Q = X0000000, where X = 4 indicating this file was never opened.

Following is an example of calling pick functions to define a file, pick a record, clear the block (read in a new block), pick a record, and remove the file from pick's table. The functions involve a user file named TAPE. The records will be passed.

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1        8  10                    20                    41
           PRG
           LIBM                  PICK,PICKD,PICKC,PICKR
BLOCK      00                    *+1
           OCT                   0
           BSS                   999
RCORD1     BSS                   800
RCORD2     BSS                   800
           PICKD                 (TAPE,REJECT1,BLOCK,1001)  Defines file called TAPE.
           PICK                  (TAPE,0,REJECT2,RCORD1,800)  Picks a record.
           PICKC                 (TAPE,REJECT3)             Reads a new block before last
                                                              one was emptied.
           PICK                  (TAPE,0,REJECT3,RCORD2,800)  Picks a record.
           PICKR                 (TAPE,REJECT4)             Removes file from the table;
                                                              it cannot be referenced by
                                                              a PICK call until it has
                                                              been defined again.
```

6.2.4 DEBLOCKER SUMMARY

The MASTER operating system employs the deblocker only on the job input file (INP). In all deblocking functions, the deblocker uses bit 23 in the first header record to check for read errors. If a pick reads a block with bit 23 set in the first header, it will ignore that block and issue a new read on the next block.

6.3 PHYSICAL I/O

The physical I/O for mass storage is a part of the system's total I/O facility called the MASTER I/O Control System (MIOCS). MIOCS, in turn, is a part of the MASTER executive that the operating system, blocker and deblocker, and user employ for I/O on all equipment in the system.

6.3.1 MASS STORAGE I/O

The user may directly control his I/O on mass storage; he may bypass the blocker and deblocker and issue his own read and write commands. Although the operating system always uses blocker and deblocker on its job files, the user may elect physical I/O on his own files.

A subset of the system macros available for I/O under MIOCS may be used in I/O on mass storage. The user can call MIOCS mass storage I/O routines, from within his own program, with a set of call macros. Each executed call establishes an I/O task. When the task is completed, a return is made to the calling task or if the call is to be buffered, the status of the I/O task is set to be terminated. All calls for physical I/O on mass storage must be on files that have been opened or the job is terminated. The following macros make up the set of physical I/O functions available for mass storage.

FORMAT

The user may call the MIOCS format function to set up various conditions under which future I/O functions will operate.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FORMAT	(dsi,comp,bninc,endblock,rec,,,,,statadr)	

dsi	Data set identifier of a file, 1 to 4 alphanumeric characters.	
comp	Compare alternative after every WRITE:	
	SCOMP	MIOCS automatically compares.
	UCOMP	User may compare with COMPARE calls.
bninc	Block number pointer incrementing alternative after I/O function (READ, WRITE, or COMPARE):	
	SBNINC	MIOCS automatically increments the block number pointer adding number of blocks processed to block pointer.
	UBNINC	User increments the block pointer through LOCATE call.
endblock	Alternative in reading and writing beyond the end of a block:	
	SENDBLK	MIOCS will not process past the end of a block. (e.g., if a WRITE of 20 words is attempted into a 15-word block, only 15 words are written. The NUM field in the status words will so indicate.)
	UENDBLK	User instructs system to process beyond end of block if required. In previous example, 20 words would be written, requiring two blocks.
rec	Error recovery handling specification:	
	SREC	System performs automatic error recovery.
	UREC	User handles own error recovery.
statadr	Address where status of the I/O operation will return:	
	non-zero	First word address where two words of status return.
	zero or blank	Status returns to tag + 2 and tag + 3, where tag is the location of the call.

READ WRITE, and COMPARE

The user may make calls to three MIOCS functions (READ, WRITE, and COMPARE) to establish correspondent I/O functions on mass storage. The call formats of these three functions are comparable.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	func	(dsi,fwca,n,char,chap,statadr,buf)	

func	READ	Read forward from a mass storage user file according to parameters.
	WRITE	Write on a mass storage user file according to parameters.
	COMPARE	Compare a record in core, with a block in a mass storage user file according to parameters.
dsi	Data set identifier of file to be read, written or compared, 1 to 4 alphanumeric characters.	
fwca	First word address† of a core buffer that MIOCS will use in the read, write, or compare.	
n	Number of words involved in the read, write, or compare.	
char	Mode of read, write, or compare. Will be blank or zero to indicate only word mode allowed on mass storage.	
chap	Chapter location of the buffer employed in read, write, or compare:	
	C2	Buffer in program task Chapter Two.
	zero or blank	Buffer in Chapter One.
statadr	Address where the status of the read, write, or compare returns:	
	non-zero	First word address where two words of status return.
	zero or blank	Status returns to tag + 2 and tag + 3, where tag is the location of the call.
buf	Status of calling task after connecting I/O call:	
	BUF	Enters ready status and continues execution.
	zero or blank	Enters wait status until I/O is complete.

† Since MASTER does not permit character I/O on mass storage, all calls involving a word/character parameter must specify word I/O only.

LOCATE

The user may call the MIOCS locate function to change the value in the block pointer; and thereby specify a new location to begin I/O on mass storage.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	LOCATE	(dsi,t,n,statadr)	

dsi — Data set identifier of mass storage file to be relocated, 1 to 4 alphanumeric characters.

t — Relocation specification that may be:

FORWARD	Locate forward n blocks.
BACKWARD	Locate backward n blocks.
BLOCK	Locate to block number n.
NEXTAVAL	Locate to block following the last block written.

n — Number of blocks of relocation, $0 \leq n \leq 8388607$

statadr — Address where the status of relocation returns:

non-zero	First word address where two words of status return.
zero or blank	Status returns to tag + 2 and tag + 3, where tag is the location of the call.

POSITION

The user may call the MIOCS position function to ascertain the current value of the position counter during an I/O operation. The value is returned in an address specified in the call.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	POSITION	(dsi,fwa,chap,statadr)	

dsi — Data set identifier of the mass storage file in question, 1 to 4 alphanumeric characters.

fwa — Address where value of block pointer returns.

chap — Chapter location fwa:

- C2 — Chapter Two.
- zero or blank — Chapter One.

statadr — Address where status of the position I/O operation returns:

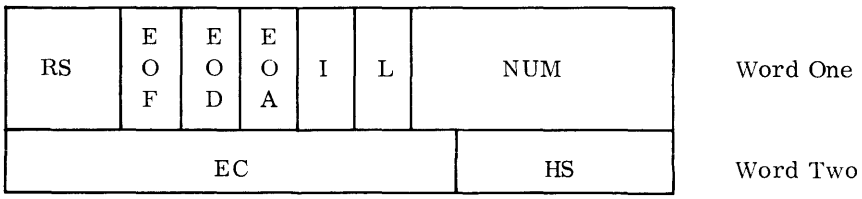
- non-zero — First word address where two words of status return.
- zero or blank — Status returns to tag + 2 and tag + 3, where tag is the location of the call.

6.3.2 MIOCS CALL SUMMARY

In all mass storage calls to MIOCS, the user must specify the address for the two status words. A parameter in the statadr field of each call tells MIOCS where to return a general status description (two computer words). It can be declared in either of two ways:

A non-zero value in this field of the call is the first word address where the status words return. If this field is zero or blank, status returns to tag + 2 and tag + 3, where tag is the location of the call.

The format of these two status words, regardless of where they are located, is as follows:



RS — Status of the call in bits 22-23 are represented as:

- 00 Call terminated normally.
- 01 Call terminated abnormally; the other fields (EOF, EOD, EOA, I, L, EC) will clarify the status.
- 10 Call is in active status.

EOF End-of-written file indicator in bit 21.

0 No end-of-written file.

1 End-of-written file condition detected.

An end-of-written-file may occur on mass storage in a READ or COMPARE call on the highest +1 data block written in the file. For example, if blocks 1, 2, and 6 have been written during random processing, and the user called a read or compare function on block 7, MIOCS would set EOF to 1.

EOD End-of-mass-storage-device indicator in bit 20.

0 No end-of-device

1 End-of-device condition detected.

An end-of-device may occur in a READ, WRITE, or COMPARE call, on a record block of a disk pack not mounted on a drive. For example, if record blocks 1-100 are on pack 1, blocks 101-200 are on pack 2, not mounted on a drive, a READ call on block 105, will set EOD to 1.

EOA End-of-allocated-area indicator in bit 19.

0 No end-of-allocation-area.

1 End-of-allocated-area detected.

An end-of-allocated-area may occur when a LOCATE call block number is greater than the number originally allocated to the file.

I Irrecoverable-error indicator in bit 18.

0 No error.

1 One of the following irrecoverable errors was detected:

lost data
parity error
non-operable condition
compare error on automatic compare after a write

If any of the preceding errors occur even though system error recovery was not selected (see FORMAT call) I will be set equal to 1. (See Error Processing.)

L File accessibility indicator in bit 17.

0 File accessible.

1 File inaccessible because of hardware malfunction such as:

Reject on connect instruction (CON).
Reject on select instruction (SEL).
Reject on I/O instruction (INPC OUTC INPW OUTW).
Not ready.

NUM Number of words involved in the called function in bits 16-00.

EC Contains error codes in bits 23-12. (See Error Processing.)

HS Status of the mass storage hardware during I/O operation, unedited.

The user may specify status of his calling task in three (READ, WRITE, and COMPARE) MIOCS calls on mass storage. This is specified by the buf parameter, in each call, as follows:

zero or blank Calling task enters wait status during the called function. As soon as the I/O operation is completed, the calling task enters ready status, and continues executing.

BUF Calling task enters wait status, but only until the call is completed; at which point, the calling task resumes execution and continues while I/O continues. If the calling task executes another I/O call on the same file, it (the calling task) automatically queues on the file until the previous I/O operation is complete.

6.3.3 ERROR PROCESSING

Two alternatives are available for processing errors that may occur during MIOCS calls on mass storage. The user may initiate his own error processing by interpreting the EC field of the status word and call his own recovery procedures, or he may allow MASTER to interpret errors and initiate automatic recovery through the mass storage driver exec.

Two general categories of errors may occur in MIOCS calls on mass storage; program errors, and hardware errors.

Program Errors

All possible program errors, during I/O functions on mass storage, may be categorized in three groups.

Errors involving illegal function requests:

Code	Explanation
1	Illegal function code (may be for non-mass storage).
2	Memory addresses for data transfer (fwa to fwa + 1) not in allowable range.
3	Illegal value in type (t) field of LOCATE call.
4	LOCATE call on a new block number of less than 1.
5	A WRITE called on a read only file.

Errors involving invalid parameters in the call:

Code	Explanation
30	Illegal error recovery selection (REC) in FORMAT call.
31	Illegal selection of automatic COMPARE after WRITE in FORMAT call.
32	Illegal selection of automatic update current block number after READ, WRITE, or COMPARE, in FORMAT call.
33	Illegal selection of transfer data only up to end of record block in FORMAT call.
34	Illegal selection of character I/O in POSITION call.

Errors involving file references:

Code	Explanation
40	READ or COMPARE call referenced a block greater than, or equal to, the next available block (nab), where nab is the number of the block one greater than the highest block written.
41	LOCATE call on a new block number that is greater than the number of blocks allocated.
42	READ, WRITE, or COMPARE call referenced a block outside the area allocated to the file.
43	Call on a file segment on an unmounted disk pack.

Hardware Errors

The mass storage driver exec will interpret, and attempt recovery for, various hardware errors that may occur during MIOCS calls on mass storage. If automatic recovery is unsuccessful, the EC and I field of the status words are set as follows:

Code	Explanation
50	Transmission parity error on READ, WRITE, or COMPARE call.
51	Hardware reject error such as reject on select for a READ, WRITE, or COMPARE call.
52	Lost data during a READ, WRITE, or COMPARE.
53	Parity error during a READ or COMPARE.
54	Non-operable device containing a file with READ, WRITE, or COMPARE call.
55	Hardware compare error during an automatic COMPARE after WRITE.
56	Drive busy when READ, WRITE, or COMPARE call is issued.
57	Drive not ready when READ, WRITE, or COMPARE call is issued.
58	No COMPARE requested but hardware compare occurred.

Program and Hardware Error Summary

Some of the previously described errors set fields in the status words as follows: For programming errors resulting from illegally referencing files, error 40 sets EOF, errors 41 and 42 set EOA, and error 43 sets EOD. Hardware error 52 will set I after unsuccessful automatic recovery.

Error Recovery

In the rec field in each FORMAT call, the user may select one of two methods of MIOCS recovery procedures for hardware errors that occur during MIOCS functions on mass storage. The user of the mass storage driver exec may attempt recovery for any of the previously described hardware errors.

User

If a hardware error occurs when UREC is specified in the rec field of a FORMAT call, the user must interpret EC of the statadr, and call the necessary recovery procedure from the General Automatic Error Recovery Procedures, Appendix B.

Automatic

If a hardware error occurs when SREC is specified in rec of a FORMAT call, control passes to the mass storage driver exec, which determines the nature of the error, and automatically initiates recovery procedures. Recovery procedures vary according to the error, but they do not deviate from the recovery procedures described in Appendix B. The sequence of automatic error recovery for hardware errors during MIOCS operation is given below:

1. Hardware error occurs.
2. Control transfers to mass storage driver exec.
3. Driver exec determines exact nature of error, and outputs a diagnostic message (System to Operator Messages, Appendix C).
4. Driver exec calls and initiates applicable error recovery procedures given in Appendix B.
5. If recovery procedures are successful, control returns to the user, and execution continues.
6. If recovery procedures are not successful, the driver exec outputs a diagnostic message, declares the error irrecoverable, sets the I field in statadr of the call, and returns control to the user.

UNIT RECORD DEVICES 7

Effective use of unit record devices such as the card reader, printers, punches, and magnetic tape units contribute greatly to the efficiency of MASTER's multiprogramming. The backgrounders automatically use record devices (card reader, printer, and punch) to process job files (INP, OUT, and PUN). The backgrounder's pool consists of all devices of types specified by the installation. Upon an open request for a backgrounder unit, the backgrounder will relinquish a unit after completing the current file and before beginning the next. Normally, when a user is opening a unit of a backgrounder type, he sets the wait parameter to W. Units not reserved by the backgrounder are assigned to a job when the job is initiated.

A user specifies a unit record device on a SCHED card. He defines the device as a file according to the device types reserved for the job. He may then request the *DEF task of the MASTER operating system to make the devices available to him, perform operations on them, and return them to the system.

7.1 UNIT RECORD DEFINITION

The user defines his scheduled unit record devices by making calls to the *DEF task of the MASTER operating system. There are two methods of calling *DEF:

*DEF control cards inserted in input decks

Call macros in a program

MASTER's *DEF task is called for two functions (open and close) on unit record devices.

7.1.1 OPEN UNIT

The *DEF open unit function associates a data set identifier with a unit record device (card reader, punch, printer, or tapes) making the unit available for I/O.

Macro format:†

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	OPENU OPENU	(wait,dsi,dt,reel id,file name,edition,reel no, usage,bypass)	

Control card format:

```
$*DEF(U,wait,dsi,dt,reel,id,file,name,edition,reel no,usage,bypass)
```

wait defines the action to be taken when a call is on a device in the background pool. This parameter may have either of the following values:

W: If the backgrounder is using all of the devices of the type requested, *DEF places the calling task in file wait status until a device becomes available, and then honors the request automatically. It means the same in both call formats.

R: If this condition occurs on a macro request, *DEF rejects the request, and sends a reject code to the caller. If it occurs when the R parameter is in a control card call, *DEF terminates the job with an error message.

other: If wait is other than W or R, MASTER interprets it as W.

dsi Data set identifier to be associated with unit device (card reader, magnetic tape punch or printer); 1 to 4 alphanumeric characters, first character cannot be *.

dt type of unit record device

405 Card Reader (must be more than one card reader in configuration)

415 Card Punch

501 } Line Printer
505 }

603 }
604 }
606 } Magnetic Tape Units
607 }

† Macro continuation line required if information extends beyond column 72.

The following parameters appear in an open unit call only if dt is a magnetic tape unit (see Summary), and should be null for other unit record devices.

reel id	Reel identifier of tape for mounting on magnetic tape dt; 1 to 5 alphanumeric characters.
file name	Tape label association; 1 to 4 alphanumeric characters corresponding to a file name field of a tape label.
edition	Tape label association; 1 or 2 alphanumeric characters corresponding to an edition field of a tape label.
reel no	Tape label association; 1 or 2 alphanumeric characters corresponding to a reel number field in a tape label.

If the above four parameters are blank the tape is assumed to be a scratch tape.

The following is examined only for magnetic tapes having one or more of the four preceding parameters. It is assumed O for scratch tapes:

usage — Unit record usage may be either of the following:

- O: Input and/or output
- I: Input only
- other: Interpreted as input only; usage is set on I

This parameter applies to all unit record devices:

bypass — Specifies how device is to handle I/O calls:

- B: Bypass all I/O calls
- other: Do not bypass calls

Open Unit Summary

If the user specified the device as magnetic tape (dt is 603, 604, 606 or 607) MASTER immediately checks magnetic tape parameters and reacts as follows:

1. If reel id, file name, edition, and reel no parameters are blank, *DEF assumes that the tape is a scratch tape and has no label; it automatically sets usage to O (input and/or output).

2. If the file name, edition, and reel number are not blank MASTER checks them against those in the standard magnetic tape label for validity. If file name, edition, reel number are all blank, *DEF assumes that the tape is not labeled and sets usage according to the usage parameter.

3. If the tape is unlabeled, it is left at load point. A labeled tape is left at the first record following the label.

MASTER provides no other handling of standard magnetic tape headers and trailer labels.

When MASTER processes a valid open unit call, it logs the request by job identifier, data set identifier, hardware type, and connect code on the console typewriter (See System to Operator Messages, Appendix C). If device type (dt) is magnetic tape and there is a reel id in the request, the message includes the reel id. When a write ring is to be inserted WR is appended to the message.

After the operator mounts a tape, MASTER

1. Rewinds it.

2. Confirms that the operator mounted a tape on the specified unit.

3. Confirms write ring status as follows:

   If a write ring is on the tape, and usage is I, MASTER repeats logging.

   If the tape has no write ring, and usage is O, MASTER repeats logging.

4. Verifies the label for a tape having one, and sets tape unit to proper density.

If MASTER notes an error in any of the preceding, it repeats logging until:

The error condition is corrected,

The operator indicates he did not honor the request,

The request has been repeated five times, then returns an error indication.

After an open unit request is processed the physical unit assignment cannot be changed by attempting to dial in another unit.

Error Processing

Error processing during an open unit call depends on the format of the call. If an error occurs, the unit is not opened. Errors occurring in a call made by a *DEF control card terminate the job, and a standard job termination message is typed to the operator. If an error occurs in a macro call, one of the following error codes returns in the status word of the call:

Error Code	Explanation
61	Operator would or could not comply with logging requests.
64	Illegal data set identifier, i.e., already in use in this job, too long, or contains a * as its first character.
69	Scheduling error, not enough drives (magnetic tape) to mount requested devices.
72	No drives available, system error occurring when operator or system marks a scheduled unit as down and unavailable, after the job is initiated.

7.1.2 CLOSE UNIT

The close function of *DEF removes the definition of a unit record device from MASTER executive system tables making the device unavailable for I/O.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	CLOSE	(wait,dsi)	

Control card format:

```
$*DEF(C,wait,dsi)
```

wait — See wait parameter explanation in OPEN UNIT discussion.

dsi — Data set identifier of unit record device to be closed. If blank, *DEF closes all unit record devices opened in the job.

*DEF ignores all requests for closing system files or job files (INP, OUT, and PUN).

Error Processing

Error processing during a CLOSE call varies with the format of the call. An error prevents closing. An error occurring during a call using *DEF control card, terminates the job; a message is typed to the operator. If the following error occurs in the course of a macro call, its code returns in the status word of the call:

Code	Condition
64	dsi is not legal (i.e., no such dsi currently exists for this job).

7.2 MIOCS

The MASTER I/O Control System (MIOCS) makes available a comprehensive I/O facility for all equipment (unit record devices and mass storage). MIOCS does the following:

- Processes I/O calls from program tasks
- Assigns equipment and channels to I/O tasks
- Initiates physical I/O requests
- Processes all non-real-time interrupts
- Processes all operating system I/O requests

It is available to library tasks, operating system tasks, and user tasks.

7.2.1 UNIT RECORD I/O

To perform I/O on unit record devices,a user makes macro calls to MIOCS functions (or the blocker or deblocker). MIOCS establishes the called I/O tasks on respective unit record devices. Each device on which I/O is being called must have been scheduled and opened, or the I/O call is rejected.

The following describes the various I/O macros for:

Magnetic Tape Units

Card Readers

Card Punch

Line Printer

Magnetic Tape Units

MIOCS performs the following I/O functions on magnetic tape units open to the job: FORMAT, READ, WRITE, LOCATE, UNLOAD, ERASE, and WEOF.

FORMAT

The magnetic tape format function sets conditions under which I/O will be performed.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FORMAT	(dsi,,,,rec,den,mode	,,,statadr)

dsi — Data set identifier of schedule and opened magnetic tape unit.

rec — Error recovery procedure to be employed with each I/O call:

- SREC — Automatic system recovery.
- UREC — User will handle own recovery.

den — Magnetic tape density (bits per inch)

- 800
- 556
- 200

} See footnote†

mode — Specifies recording mode on magnetic tape:

- BCD — Binary coded decimal
- BIN — Binary

} See footnote†

statadr — Address to which word one of status of the I/O returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the macro.

† Mode and density specifications may be changed from those in initial format requests, by specifying them differently in new format requests on the same dsi. Once specified, their omission in new format requests means that their original specifications hold true. If never specified, MIOCS assumes 200 bpi density, and normal magnetic tape mode which is binary.

READ, READB, WRITE

The user has three MIOCS I/O call macros (READ, READB, and WRITE) for MIOCS functions on magnetic tape units.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	func	(dsi,fwca,n,CH,C2,statadr,BUF)	

func	READ	Read forward on an open magnetic tape according to parameters.
	READB	Read backward on an open magnetic tape according to parameters.
	WRITE	Write on an open magnetic tape according to parameters.
dsi		Data set identifier of a magnetic tape file, 1 to 4 alphanumeric characters.
fwca		First word or character address of a buffer, in core, that will be employed in reads or writes.
n		The number of words, or characters, to read or write.
CH		The mode of the read or write will be character. When CH is omitted (blank, zero, or other), MIOCS assumes word mode.
C2		The buffer to be employed in the read or write is in Chapter Two. When C2 is blank or zero, the buffer is in Chapter One.
statadr		Address to which word one of the status of the read or write returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the macro request.
BUF		Specifies that the status of the calling task be ready upon completing connection of the READ or WRITE call: When the I/O call connects, the calling task enters ready status and may continue execution. When BUF is omitted (zero or blank), the calling task enters wait status until the reads or writes are completed.

LOCATE

The magnetic tape locate function moves (drives) a tape to a new position.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	LOCATE	(dsi,t,n,statadr)	

dsi — Data set identifier of the magnetic tape unit

t — Change position:

FORWARD	Forward n blocks
BACKWARD	Backward n blocks
BLOCK	Locate to block number n. (If n = 1, the tape is rewound; any other value results in an illegal request reject).
FMF	Forward n file marks
FMB	Backward n file marks
zero	Rewind

n — Number of blocks to position,0 to 8388607

statadr — Address to which word one of the status of the read or write returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the call.

UNLOAD, ERASE, WEOF

Three MIOCS call macros (UNLOAD, ERASE, and WEOF) request comparable functions on magnetic tape.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	func	(dsi,statadr,BUF)	

func.

UNLOAD	Unloads the magnetic tape specified by the dsi.
ERASE	Erases six inches of the magnetic tape specified by the dsi.
WEOF	Writes an end-of-file mark on the magnetic tape specified by the dsi.

dsi — Data set identifier of the magnetic tape unit, 1 to 4 alphanumeric characters.

statadr — Address to which word one of the status returns. When no address is given (zero or blank) status returns to tag + 2 and tag + 3, where tag is the location of the call.

BUF — Specifies that calling task enter ready status upon completing connection of an UNLOAD, ERASE or WEOF call. Upon call connection, the calling task enters ready status and continues execution.

When BUF is omitted (zero or blank) the calling task enters wait status until the I/O is complete.

7.2.2 CARD READER

In computer configurations having two or more card readers, the user may call two MIOCS functions (format and read) on an open card reader. Do not use these functions when card reader is an INP file for a DIRECT job.

FORMAT

The user may call the MIOCS card reader format function to set conditions under which I/O will operate.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FORMAT	(dsi,,,,,,mode,blk,,statadr)	

dsi — Data set identifier of the card reader file to be formatted, 1 to 4 alphanumeric characters.

mode — Specifies mode of I/O on the card reader:

BCD	Binary coded decimal	See footnote†
BIN	Binary	

blk — Blocking specification:

NONBLK — MIOCS is to read one card on each read call, and pass it to the caller.

BLK — MIOCS is to read several cards on each read call, and block them in standard format. Blocking will continue until the buffer specified in a read call is full.

statadr — Address to which word one of the status of the called I/O returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the request.

READ

The read function reads cards from a card reader open to the job.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	READ	(dsi,fwca,n,CH,C2,statadr,BUF)	

dsi — Data set identifier of card reader file to be read.

fwca — First word or character address of a buffer in core into which data will be read.

n — Number of words or characters to read.

† Mode specification may be changed from that in an initial format request by specifying it differently in new format requests. Once specified, its omission in new format requests means that its original specification holds true. If never specified, MIOCS assumes the normal mode of a card reader which is BCD.

CH — The mode of the read will be character. When CH is omitted, blank, zero, or other, MIOCS assumes word mode.

C2 — The read buffer is in Chapter Two. When C2 is omitted (blank or zero), the buffer is in Chapter One.

statadr — Address to which word one of status of the read returns. When no address is given, (statadr is zero or blank) status returns to tag + 2 and tag + 3, where tag is the location of the call.

BUF — Specifies that the calling task enter ready status upon completing connection of read call. When the READ call is connected, the calling task enters ready status and may continue execution. When BUF is omitted (zero or blank), the calling task enters wait status until the read is completed.

7.2.3 CARD PUNCH

The user may call two MIOCS functions (format and write) on an open card punch. Do not use these functions when card punch is a PUN file for a DIRECT job.

FORMAT

The format macro sets I/O conditions:

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FORMAT	(dsi,,,,,,mode,blk,,statadr)	

dsi — Data set identifier of the card punch, 1 to 4 alphanumeric characters.

mode — Specifies the mode of I/O on the card punch:

BCD	Binary coded decimal	See footnote†
BIN	Binary	

† Mode specification may be changed from that in an initial format request by specifying it differently in new format requests. Once specified, its omission in new format requests means that its original specification holds true. If never specified, MIOCS assumes the normal mode of a card punch which is BCD.

blk — A blocking specifier:

- NONBLK — MIOCS accepts one record on each write request, and punches it.
- BLK — MIOCS unblocks a standard buffer from the user, and punches until the buffer is empty.

statadr — Address to which word one of the status of the format call returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the call.

WRITE

The card punch write function punches cards on a card punch open to the job.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	WRITE	(dsi,fwa,n,CH,C2,statadr,BUF)	

dsi — Data set identifier of an open card punch file.

fwca — First word or character address of a buffer in core that will be employed in the write.

n — The number of words or characters to be punched.

CH — Mode of punching will be character. When CH is omitted, (blank or zero) MIOCS assumes word mode.

C2 — Specifies the write buffer is in Chapter Two. When C2 is omitted, blank or zero, the buffer is in Chapter One.

statadr — Address to which word one of the status of the write returns. When no address is given (statadr is zero or blank) status returns to tag + 2 and tag + 3, where tag is the location of request.

BUF — Specifies that the calling task enter ready status upon completing connection of the write call. The calling task enters ready status and may continue execution. When BUF is omitted (zero or blank) the calling task enters wait status until the write is completed.

7.2.4 LINE PRINTER

MIOCS may be called to format and write on a line printer open to the job. Do not use these functions when line printer is an OUT file for a DIRECT job.

FORMAT

The line printer format function sets a series of conditions for I/O.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FORMAT	(dsi,,,,,,,blk,cc,statadr)	

dsi — Data set identifier of a line printer file

blk — A blocking specifier:

- NONBLK — System accepts one record on each write request, and prints it.
- BLK — System unblocks a standard buffer from the user, and prints until the buffer is empty.

cc — A carriage control code specifying paper handling during printing on 501 or 505 printers coded as follows:

Octal Control Code	Action
zero blank or omitted	Program carriage control
01	Select single space
02	Select double space
03	Select advance to last line of form
04	Select page eject
05	Select automatic page eject
06	Select suppress space
10	Clear format selections
11	Select format level 1 for postprint spacing
12	Select format level 2 for postprint spacing
13	Select format level 3 for postprint spacing
14	Select format level 4 for postprint spacing

Octal Control Code	Action
15	Select format level 5 for postprint spacing
16	Select format level 6 for postprint spacing
20	Select preprint spacing
21	Select format level 1 for preprint spacing
22	Select format level 2 for preprint spacing
23	Select format level 3 for preprint spacing
24	Select format level 4 for preprint spacing
25	Select format level 5 for preprint spacing
26	Select format level 6 for preprint spacing

statadr — Address to which word one of the write status returns. When no address is given (statadr is zero or blank), status returns to tag + 2 and tag + 3, where tag is the location of the call.

WRITE — The user may specify the MIOCS line printer write function on a line printer open to the job.

Macro format:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	WRITE	(dsi,fwca,n,CH,C2,statadr,BUF)	

dsi — Data set identifier of the printer file.

fwca — First word or character address of a buffer in core that is to be printed.

n — Number of words or characters to print.

CH — Mode of printing will be character. When CH is omitted (blank or zero), MIOCS assumes word mode.

C2 — Specifies the location of the buffer is Chapter Two. When C2 is omitted (blank or zero), the buffer is in Chapter One.

statadr Address where word one of the printing status returns. If zero or blank, status returns to tag + 2 and tag + 3, where tag is the location of the request.

BUF Specifies the calling task is to enter ready status upon completing connection of a WRITE call. The calling task enters ready status and continues execution. When BUF is omitted (zero or blank) the calling task enters wait status until I/O is completed.

In the FORMAT call, the user may specify control of paper movement during printing (cc). If he specifies program control (zero or blank in the cc field), the printer driver interprets but does not print the first character of a print line. It interprets the character as follows:

Control Character	Action Before Print	Action After Print
A	Space 1	Page Eject
B	Space 1	Skip to Last Line
C	Space 1	Skip to Level 6
D	Space 1	Skip to Level 5
E	Space 1	Skip to Level 4
F	Space 1	Skip to Level 3
G	Space 1	Skip to Level 2
H	Space 1	Skip to Level 1
1	Page Eject	No Space
2	Skip to Last Line	No Space
3	Skip to Level 6	No Space
4	Skip to Level 5	No Space
5	Skip to Level 4	No Space
6	Skip to Level 3	No Space
7	Skip to Level 2	No Space
8	Skip to Level 1	No Space
0 (zero)	Space 2	No Space
+	No Space	No Space
-	Space 3	No Space
(blank)	Space 1	No Space
Q	Clear Auto Page Eject	No Print
R	Select Auto Page Eject	No Print

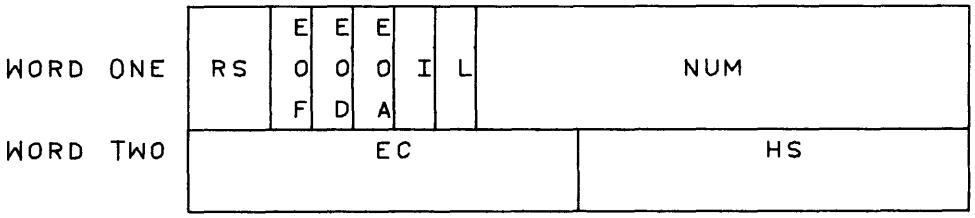
7.2.5 MIOCS CALL SUMMARY

In all MIOCS calls on unit record devices, the user may specify an address for the first of two status words. A parameter in the statadr field of each call tells MIOCS where to return a general status description (in two computer words) of a called function. It can be declared in either of two ways:

A non-zero value in this field of the call is the first word address where the status returns.

If this field is zero or blank, status returns to tag + 2 and tag + 3, where tag is the location of the macro call issue.

The general format of these two status words, regardless of where they are returned, is as follows:



RS Status of the call (bits 22-23)

00 Call terminated normally.
01 Call terminated abnormally.
10 Call is in active status.

If RS=01, the other fields (EOF, EOD, I, L, EC) will clarify the status.

EOF End-of-file in bit 21:

0 No end-of-file
1 End-of-file condition detected.

An end-of-file condition occurs on a card reader or magnetic tape unit when a READ call encounters an end-of-file card or an end-of-file mark.

EOD End-of-device indicator in bit 20:

0 No end-of-device.
1 End-of-device condition detected.

An end of device condition occurs on a magnetic tape unit when a READ or WRITE call encounters an end-of-tape reflector.

EOA End-of-allocated-area indicator applies to mass storage only (bit 19). (See Chapter 6.)

I Irrecoverable error indicator in bit 18.

0 No error.
1 An irrecoverable error was detected on the device. (See Error Processing.)

Occurrence of a hardware error, even though system error recovery was not selected in the FORMAT function, sets I to 1. (See Error Processing.)

L File accessibility indicator in bit 17.

0 Unit record device accessible.
1 Unit record device inaccessible because of hardware malfunctions such as:

Reject on connect instruction (CON)
Reject on select instruction (SEL)
Reject on I/O instruction (INPC OUTC INPW OUTW)
Not ready

NUM Number of words involved in the called function in bits 16-00.

EC Contains error codes in bits 23-12. (See Error Processing.)

HS Unedited status of unit record hardware during I/O operation.

7.2.6 ERROR PROCESSING

The processing of errors occurring during I/O on unit record devices, either by the user or the corresponding driver relates to two categories, program errors, and hardware errors. All error codes return (in binary coded decimal) in the EC field of the status words for the requested function.

Program Errors

Program errors occurring during I/O functions on unit record devices are grouped in two areas; program errors applicable to all unit record devices, and program errors applicable to magnetic tape only.

Program errors involving illegal function requests on all devices:

Error Code	Explanation
1	Illegal function code, request not possible on this unit record device†
2	Memory addresses for data transfer (fwca to fwca + n-1) not in the range assigned to the program.
3	Illegal value in type (t) field on
4	Locate attempted a backward motion at load point on magnetic tape.
5	Attempted writing end-of-file on a read only file.
6	Logical record within a block exceeds the maximum size allowable for unit record devices (40 words for punch) (34 words for printer).†
7	Data transmission request had word count of zero.

Program errors involving illegal function calls on magnetic tape:

Error Code	Explanation
20	Illegal reverse read call on 603 or 606 tape unit.
21	Illegal density selection in FORMAT request on 603 or 606 tape unit.
22	Requested fewer than six characters output on a write.

Unit Record Hardware Errors

With the exception of user/system recovery options on magnetic tape units, drivers and driver EXEC handle hardware errors and recovery on all unit record devices. The following describes the possible hardware errors and corresponding recovery procedures for the unit record devices:

---

†Applicable device driver sets I in status word.

Magnetic Tape

If the user specifies SREC in a magnetic tape FORMAT call as he normally does, the magnetic tape driver and driver EXEC handle recovery for the following hardware errors:

Parity errors

Lost data conditions

Hardware rejects

The driver employs the automatic recovery procedures described in Appendix B, and considers a record of fewer than six characters a noise record.

If the magnetic tape driver's automatic error recovery is unsuccessful, the error is declared irrecoverable. The calling task is informed of an irrecoverable error condition in the "I" field in its status word. The operator is also informed of an irrecoverable error condition on magnetic tape through a message that declares the particular channel, equipment, and unit (System to Operator Messages, Appendix C).

The user may select his own error recovery (UREC in rec field of FORMAT request) on magnetic tape only. Hardware errors and subsequent recovery procedure for the card reader, punch, and line printer are the joint responsibility of the driver and driver EXEC with corrective action on the part of the operator. If the user selects his own error recovery (UREC) on magnetic tape, and the driver EXEC detects a parity error, error code 50 returns on the EC field of the status. The user must interpret this code in conjunction with the HS and I fields of the status word, and initiate his own recovery.

If the user selects automatic error recovery (SREC) on magnetic tape, the magnetic tape driver and driver EXEC perform the automatic recovery procedures described in Appendix B.

Card Reader

The card reader driver and EXEC can detect the following hardware errors:

Compare

Preread

Fail-to-feed

Stacker-full

Hopper-empty

Recovery varies with the nature of the error. In general, compare and preread error conditions are followed with a compare error diagnostic message on the console typewriter. The operator takes corrective action by re-inserting the last card at the head of the input stack. If the operator does not take corrective action the card is lost.

A fail-to-feed, stacker-full, or hopper-empty condition produces a READY CARD READER message on the console typewriter. The operator takes the necessary corrective action, and then readies the card reader as requested. Reading continues when the card reader is ready.

Card Punch

The card punch driver and driver EXEC can detect the following hardware errors:

Compare

Fail-to-feed

Stacker-full

Hopper-empty

For a compare error, the card reader driver attempts automatic recovery through the offset card option. Because the error condition occurs after the card (following the mispunched card) is punched, both cards are offset. Both cards are then repunched, and the system informs the operator (System to Operator Messages Appendix C) of the error condition. The operator may remove the offset cards at any time. Punching is not interrupted.

In the event of a fail-to-feed, stacker-full, or hopper-empty error condition, the operator is informed through a READY PUNCH message on the console typewriter. When MIOCS receives a ready interrupt, it takes the proper corrective action, with no further operator intervention.

Printer

The printer driver and EXEC can detect only an out-of-paper condition. The condition produces a console typewriter message requesting the operator to ready the printer. The message specifies the channel, equipment and unit number. Normal operation continues when the operator corrects the condition and readies the printer.

INTERRUPTS 8

All interrupts cause the computer to enter Monitor State and execute the System Executive, EXEC. After MASTER operation has begun, interrupts provide the only entrance to EXEC.

An interrupt is processed by a portion of EXEC known as External Interrupt Control (EIC), which saves register contents, decodes the interrupt, and depending upon the interrupt, routes control to a section of EXEC.

After EXEC processes an interrupt, it transfers control to Assign a Processor which places a program task into execution.

EXEC recognizes eight types of interrupts:

- Internal Fault Interrupts
- Trapped Instruction (Floating Point Instruction)
- Request Interrupts (Halts)
- Clock Interrupt
- I/O Interrupts (Other than real-time)
- Dedicated Channel Interrupts (Real-time)
- Manual Interrupt
- Other:
  - Associated Processor
  - Illegal

8.1 INTERNAL FAULTS

User Interrupt Control (UIC), a routine that resides as part of a task (section 4.10), processes all user fault selections such as arithmetic overflow. When a selected fault occurs, EXEC routes control to the portion of UIC that interprets the interrupt and performs a return jump to the task's interrupt routine. Because UIC is a portion of the task, there is no EXEC intervention upon exit from an internal fault routine. UIC routes control to the task at the interrupted location.

For fault selection, refer to section 9.3.1.

8.2 FLOATING POINT

The floating point instructions may be simulated at the option of the installation. When floating point hardware is not in the system, the instructions become trapped and are processed by the simulation package, FDPBOXS. When required, FDPBOXS is a resident part of EXEC. If arithmetic faults are to be processed, they must be selected in the same manner as if the floating point hardware were present.

Simulated instructions include:

ELQ	AEU	FAD
EUA	AQE	FSB
EAQ	MUAQ	FMU
QEL	DVAQ	FDV

8.3 REQUESTS

All requests of EXEC from a program task result in an illegal instruction, which causes EXEC to perform the function requested or route control to the appropriate task.

8.4 CLOCK

When EXEC places a task in execution, it sets a time limit which may cause a clock interrupt. The time set is the smallest of:

Time remaining in job time limit (10.2)

Time remaining in LIMIT request (9.6)

Time of execution cycle (maximum time a task is allowed to be executed without interruption)

When a clock interrupt occurs, it is processed to determine which time expired. If it is the job or LIMIT time, the job is terminated. If it is the execution cycle time, EXEC places the next highest priority task in execution.

Cycle time is an installation parameter that prevents any one task from occupying the compute module indefinitely.

8.5 I/O

Upon detection of an I/O interrupt, EIC routes control to the appropriate MIOCS routine (Chapters 6 and 7). The three types of I/O interrupts are:

Channel Equipment I/O Call

8.6 DEDICATED CHANNEL

Whenever an interrupt occurs on a dedicated channel, EIC routes control to a user-supplied real-time routine, which is, in fact, an extension to EXEC. See MASTER Installation Manual.

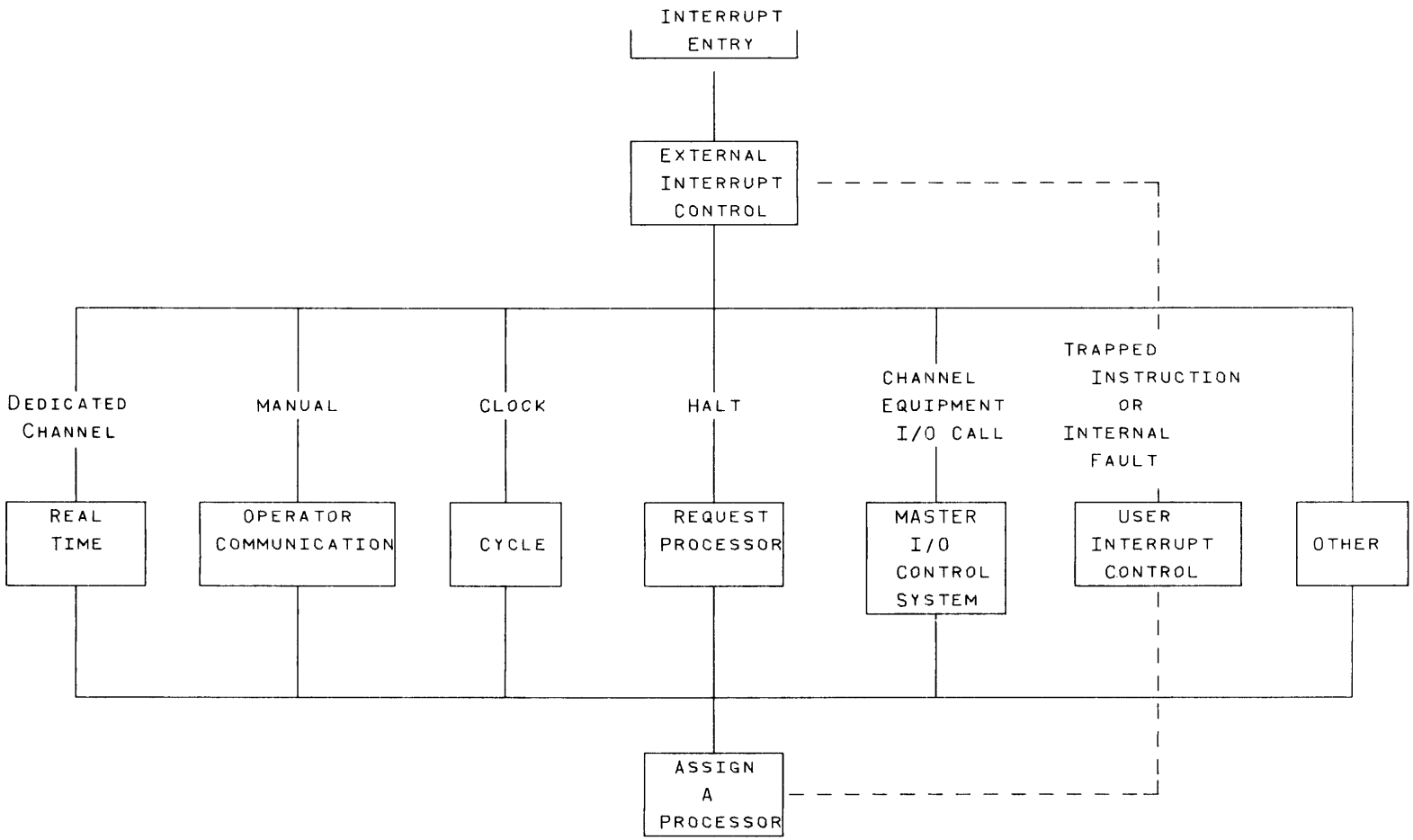
8.7 MANUAL

When a manual interrupt occurs, EIC routes control to the operator-communication routine that processes requests from the operator. See section 9.4 and MASTER Operator's Guide.

8.8 OTHER

The associated processor interrupt is not used. An illegal instruction interrupt (other than defined requests) terminates the job that caused it.

A block diagram of interrupt processing follows.


INTERRUPT ENTRY
EXTERNAL INTERRUPT CONTROL
DEDICATED CHANNEL
MANUAL
CLOCK
HALT
CHANNEL EQUIPMENT I/O CALL
TRAPPED INSTRUCTION OR INTERNAL FAULT
REAL TIME
OPERATOR COMMUNICATION
CYCLE
REQUEST PROCESSOR
MASTER I/O CONTROL SYSTEM
USER INTERRUPT CONTROL
OTHER
ASSIGN A PROCESSOR

EXECUTIVE REQUEST MACROS 9

An executive request is a request that a task makes of the MASTER EXEC routine either because it cannot itself perform the requested operation in the program state or because the request involves linkage with another task.

MASTER EXEC recognizes two types of executive requests: those directly coded as illegal instructions and those coded as macros which generate a coded halt. All macros must be coded to meet the specifications for macros outlined in 1.7.

Executive requests include the following types:

Task Linkage	Abort Job or Suppress Task
Deferred Wait	Copy Common
Interrupt	Bypass File
Typewriter Output	Reserve File
Time and Date	Copy Directory Entry
Time Limit	Ascertain Device Type

9.1 TASK LINKAGE

Two macros establish or remove linkage between tasks. With the CALL macro, one task requests execution of another, and with the RETURN macro, a called task indicates to its caller that its work is completed.

9.1.1 CALL

A task that requires execution of some other task issues a call for that task, optionally passing parameters and common to it. The caller may issue many calls. If the callee is already active (has not completed execution of an earlier call), the caller is queued on a priority basis. The call is connected when the callee becomes inactive. The caller remains in call status until the call is connected. Then, according to a parameter of the call, the caller may be multiprogrammed with its callee, or may wait for the callee to complete its work before resuming execution.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	CALL	(name,dsi,num,fwa,com,R,A)	

name Task name of callee

dsi File containing callee. When the callee already resides in core the dsi is ignored; the task is not reloaded. When dsi is omitted callee is assumed to be on library.

num Number of 24-bit words of parameters (0 to 511) to be passed to callee.

fwa Address containing first word of parameter list.

com Chapter Two common passage (4.7).

- blank No common passed.
- OC Pass own common as read and write.
- OCR Pass own common as read only.
- CC Pass caller's common as read and write.
- CCR Pass caller's common as read only.

R Indicates that the task containing the call is to be placed in ready status upon call connection and may continue processing multi-programmed with its callee. When R is omitted, the caller assumes wait status.

A Abandon parameter used only by an operating system task. It indicates that the caller does not require notification of callee completion. (See MASTER Installation Manual).

A task that has requested ready status may check the status bit of the call (bit 0 at the address containing the CALL request) to determine if the callee has returned. If this bit is not set, the callee has not returned; if the status bit is set, the callee has returned. A task should not loop and test the status bit continuously, however, since the time consumed could be used by another task. By issuing a DWAIT (9.2), the CPU becomes available for reassignment to another task, possibly the task for which the caller is waiting.

Rules:

1. Only two-chapter tasks may pass or receive common.
2. A callee may not call its caller nor may a caller call itself (circular calls).
3. When parameters are to be passed, the callee must include a non-standard copy of UIC (4.10).

Caller

```
LOCATION        OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1          8    10                    20                    41
                CALL                  (KEN,,18,PLIST)
```

Callee

```
LOCATION        OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1          8    10                    20                    41
                LIBM                  UIC
UIC             UIC                   (18)
```

4. If the caller is to receive parameters on a return from its callee, it must pass parameters to the callee, since parameters are returned to the same area from which they are passed. Referring to the example under Rule 3, KEN can return no more than 18 parameters to the caller.

9.1.2 RETURN

Every task must contain a RETURN request through which it directly or indirectly notifies its caller† of completion.

A task may directly return to its caller or may request that a callee of its own handle its return. That is, if Task B, the completed task had previously called Task C which is still active, it can request that Task C return to the original caller (Task A) rather than to B. This is called transfer of call end. One advantage of transferring call end is to free Task B for a call by a queued caller.

† When the user requested the task by a Task Name control card, the job monitor generated the call for the task and is notified of the return.

When the returning task has active callees, its return is delayed until all calless (other than the one to which call end is transferred) have returned.

When a task returns, EXEC releases any files the task may have reserved (9.10).

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	RETURN	(num,fwa,ca,NR,R)	

num — Number of 24-bit words of parameters to be returned to caller. It is limited by number of parameters in CALL since parameters are returned to the area within the caller from which they are passed to the callee.

fwa — Address containing first word of parameter list.

ca — Address at which this task made a call for a task to which call end is to be transferred. When a callee transfers call end, it cannot return parameters to its caller.

NR — Indicates that this task is not to be released from core. When NR is omitted, the task is to be released.

R — Used only by operating system tasks; R indicates that this task will continue executing although it has notified its caller of completion. (See MASTER Installation Manual.)

The UIC routine which must be a part of each task contains a RETURN request (4.10).

9.2 DEFERRED WAITS

A task that has called one or more subordinate program or I/O tasks and is being multiprogrammed with them may reach a point beyond which it should not continue until one of its callees has completed. It can issue a deferred wait request which places the caller in deferred wait status until one of the callees designated in the request returns. If any of the named tasks has already returned, or was never called, the caller immediately assumes ready status. Otherwise, it must wait for one of the designated callees to return.

For Program tasks, the macro is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	DWAIT	(adr$_1$,...,adr$_7$)	

adr Addresses (1-7) at which task issued calls.

For I/O tasks, the macro is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	DWAITIO	(dsi$_1$,...,dsi$_7$)	

dsi Files (1-7) identifying I/O calls for which the program task is to wait.

A task may check the status of an I/O call at any time by testing the RS bits in Status Word One (Chapter 6).

If more than one task or I/O call is mentioned in the wait request, the caller must determine which callees are completed by testing the status bits of the task CALL (9.1.1) or I/O call (Chapter 6).

9.3 INTERRUPT CONTROL

Four interrupt macros give users limited control of interrupts:

SELECT
CANCEL
DINTS
EINTS

9.3.1 SELECT AND CANCEL

The SELECT request directs EXEC to route control to the user-specified address upon occurrence of the designated internal fault condition. CANCEL removes the directive. SELECT is in effect only for the requesting task.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SELECT	(fault,adr)	

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	CANCEL	(fault)	

fault Interrupt condition

AROV	Arithmetic overflow
DIVIDE	Divide fault
EXOV	Exponent overflow

adr Address to which control is to be transferred upon the occurrence of fault.

9.3.2 DINT AND EINT

DINT and EINT are COMPASS instructions, not macros. An attempt to execute either DINT or EINT in the program state causes an interrupt returning control to EXEC which then executes a DINT or EINT for the task.

The disable interrupts (DINT) instruction causes a logical lockout of all user SELECT requests. That is, all internal fault selections from this job made previous to DINT or while the DINT is in effect, are temporarily cancelled. Any internal faults occurring while the DINT is in effect are lost. A DINT request is ignored if one is already in effect when it occurs.

The DINT is in effect only for the task making the request.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	DINT		

The enable interrupts request (EINT) cancels a DINT request. EINT restores any uncancelled SELECT requests made prior to or while a DINT request is in effect.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	EINT		

9.4 CONSOLE TYPEWRITER OUTPUT

A task uses the TYPE macro to send a message to the operator and receive one in return.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	TYPE	(mesnum,mesfca,resnum,resfca)	

mesnum	Number of characters (1-127) in message to be typed.
mesfca	First character address of message.
resnum	Number of characters (0-127) in message to be returned by operator.
resfca	First character address of buffer receiving response.

When a response is designated, MASTER assigns one of the response codes (R0-R9) to the message, returns the carriage, and types

Rr, <message at mesfca>

The task cannot resume execution until the operator types the requested response message

Rr, <message to be transferred to resfca>

The response parameters are optional. For example:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	TYPE	(18,MESSLOCA)	

MASTER returns the typewriter carriage and immediately types the 18-character message beginning at MESSLOCA without assigning a response code. The task may resume execution as soon as the message is sent.

9.5 TIME AND DATE

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	TIME		

Upon receiving the TIME request EXEC presents the task with the time of day in the A and Q registers in the format:

(AQ) = hr/mn/sc

hr	BCD codes or the hour of the day (0-24)
/	BCD code for a slash
mn	BCD codes for the minute of the hour (0-60)
sc	BCD codes for the second of the minute (0-60)

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	DATE		

Upon receiving a DATE request EXEC presents the task with the date in the A and Q registers in the format:

(AQ) = mo/dy/yr

mo	BCD codes for month (1-12)
/	BCD code for a slash
dy	BCD codes for day (1-31)
yr	BCD codes for year (00-99)

9.6 LIMIT TASK TIME

A user may set a time limit for execution of a task or a portion of a task by using the LIMIT macro.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	LIMIT	(hr/mn/sc)	

hr	Hours (0-24)
mn	Minutes (0-60)
sc	Seconds (0-60)
/	Slash

Only one limit per job can be in effect at any one time; a later LIMIT request takes precedence over an earlier request. A job is terminated when LIMIT time expires (Chapter 8).

A FREE request removes the time restriction set by LIMIT. It does not alter the job limit.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FREE		

9.7 ABORT JOB OR SUPPRESS TASK

When a failure occurs, the user can stipulate that subsequent tasks of the job not be executed or that the job be terminated.

The ABORT request may be inserted into a task which may encounter a condition requiring job termination.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ABORT	(fwa,nc,STO,ND)	

fwa — First-word address of diagnostic message (0-56 characters) to be printed on job's OUT file (2.6.1).

nc — Number of characters to be output (56 maximum).

STO — Diagnostic message is to be output on console typewriter as well as on the user's OUT file. When STO is omitted, the message is written on OUT file only.

ND — Indicates core dump is to be suppressed regardless of ABORT parameter on the SCHED card. When ND is omitted, taking of a core dump depends on the ABORT parameter.

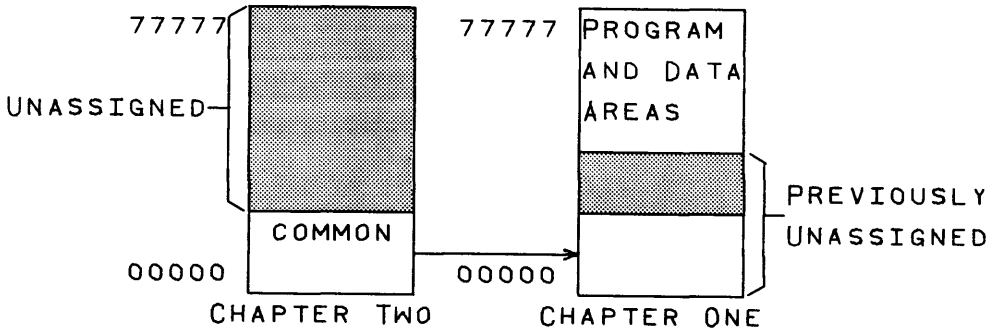
The SUPPRESS request may be made by an assembler or compiler task when it detects a condition that should prevent execution of the task. It causes the Job Monitor to ignore any additional Task Name cards except those which call a task contained in a table of task names maintained by the Job Monitor. This table contains names of assemblers and compilers, such that assemblies and compilations may proceed for the job although object tasks may not execute.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SUPPRESS		

9.8 COPY COMMON

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	COPYCOM		

A COPYCOM request permits a two-chapter task to transfer its Chapter Two common to Chapter One. Chapter One initially contains program and data areas.



Following a COPYCOM request, Chapter Two still contains its common. The Chapter One page map contains the page indexes for Chapter Two common as well as the program and data areas. COPYCOM permits the task to reference Chapter Two common without relocating to operand state, and to pass and receive common in Chapter Two.

If logical common addresses overlap the program and data addresses or if the task already has a Chapter One common area, the job is terminated.

9.9 BYPASS FILE

A user may direct MIOCS to ignore all I/O operations for a specific file requested by any task of the job.

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	BYPASS	(dsi)	

dsi File other than system file

When MIOCS detects a reference to a bypassed file, it ignores the request and sets RS in Status Word One as completed (Chapter 6).

A bypass is in effect for all tasks of the job containing the request.

9.10 RESERVE FILE

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	RESERVE	(dsi)	
	:	:	
	RELRESV	(dsi)	

dsi File reserved to task

A task may reserve an I/O file for I/O operations.

No other task may perform I/O on a reserved file until the task that reserved it issues a RELRESV remove file reservation request. If a task attempts to perform I/O on a file reserved for another task, the call is queued until the reserved file is released.

In this discussion:

A global file may be referenced by system or user tasks of any job.

A system file may be referenced by system tasks only.

A job file may be referenced only by tasks associated with the job.

A sequential file is processed sequentially; no locates need be performed on sequential files.

A random file is processed randomly. Users must locate to position file before I/O operation.

If a task is to perform I/O on a global random file, the order of events should be as follows:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
. . .	. . .	. . .	
	RESERVE	(dsi)	
	(I/O Operations)		
	RELRESV	(dsi)	
. . .	. . .	. . .	

Rules:

1. Do not leave a global file in RESERVE status when the I/O operations are completed. If a task never releases the reserve, the file remains reserved until the task returns to its caller.

   Careless reserving of a user file decreases job efficiency but has no effect on the remainder of the system.

2. A job is terminated if one of its tasks requests wait status or deferred wait status while it has a file reserved.

   If a random file is not reserved, there is no way to guarantee that its position has not changed between a task's LOCATE and the next I/O operation. This is true whether processing is random or sequential.

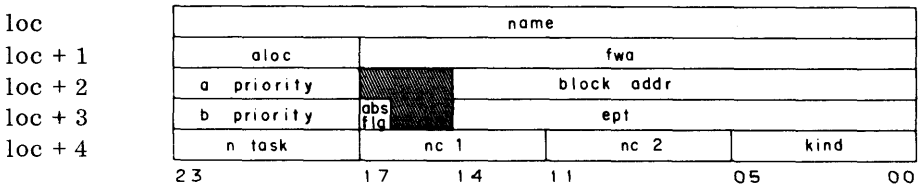
9.11 COPY DIRECTORY ENTRY

The COPYDIR request permits a user to obtain a copy of a library directory entry for a specified task in order to locate the task on the library. This information may be required by a task such as FORTRAN or COMPASS so that it can read in its own overlays from the library. The macro format for the request is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	COPYDIR	(name,loc)	

name Name of task for which directory entry is requested.

loc First word address of five locations in the requesting task to which the directory entry is to be copied.



Word	Bits	Field	Significance
loc	23-00	name	4-character BCD name of task, record, or routine.
loc + 1	14-00	fwa	Logical first word address of absolute record in core; fwa applies only to absolute records (lda on LTASK card).
	23-15	aloc	Used only for absolute tasks, aloc contains the number of unused locations in the last quarter page assigned to the task.
loc + 2	14-00	block addr	Task's block address on library.
	17-15		Unused; zero.
	23-18	a priority	Inherited priority multiplier of library task (a on LTASK card).

Word	Bits	Field	Significance
loc + 3	14-00	ept	Entry point address for task or routine.
	16-15		Unused; zero.
	17	abs flg	If abs flg is 1, task is an absolute record on the library; if abs flg is 0, task is in relocatable format.
	23-18	b priority	Inherited priority (b on LTASK card).
loc + 4	05-00	kind	6-bit field flagging kinds of restrictions to which task may be subjected. bit = 1 00 real-time task (RT on LTASK card) 01 linked to resident tables (TL on LTASK card) 02 requires two chapters (2CH on LTASK card) 03 permanently allocated (PA on LTASK card) 04 operating system task (OS on LTASK card) 05 may not be aborted during execution (ABT on LTASK card)
	11-06	nc 2	Number of quarter pages required for Chapter Two common.
	17-12	nc 1	Number of quarter pages required for Chapter One common.
	23-18	ntask	Number of quarter pages required for task itself.

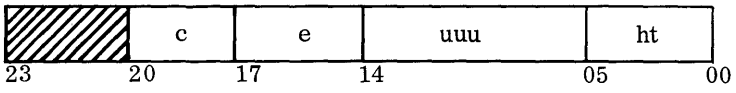
For library entry and library task generation and use of LTASK card, see MASTER Installation Manual.

9.12 ASCERTAIN DEVICE TYPE

The TYPEIO macro ascertains the device type to which a file is assigned. The macro format is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	TYPEIO		

When the macro is executed, the Q register must contain the dsi of the file in question, left justified with blank fill (60's). The macro returns the hardware connect code (c e uuu) in bits 20-06 and the hardware type code in bits 05-00 of the A register. After this request is executed, the A register contains the following information right justified.



c 3-bit channel number. First channel on which the unit associated with the file is connected.

e 3-bit equipment number. First equipment to which the unit associated with the file is connected.

uuu 9-bit unit number. Unit currently assigned to the file.

ht 6-bit hardware type code. The hardware type codes associated with the different device types are listed below.

Code	Device	Code	Device
01	405	33	603
05	415	40	1311
11	501	41	852
13	505	50	853
30	607	51	854
31	606	60	813 or 814
32	604	70	863

When the named dsi has been equated to another dsi (10.1.6), the primary dsi (used to open the file) is returned in the Q register in place of the named dsi. Otherwise, the Q register contains the named dsi.

TYPEIO sets bit 23 of the A register when the named dsi is undefined (never been opened or equated to an opened dsi).

Example:

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD | COMMENTS
1      8 | 10                  | 20            | 41
         |          :          |               |
         | LDQ                 | =ø47644560    | (Q)=PUN
         | TYPEIO              |               |
         |          :          |               |
```

9.13 CHECK FILE STATUS

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD | COMMENTS
1      8 | 10                  | 20            | 41
         | STATUS              | (dsi,adr)     |
```

This macro returns to adr and adr + 1 the status of the last operation on the named file. Status words are interpreted in 7.2.3.

9.14 DUMP REQUEST

With the PROGDUMP macro, a user may request a periodic printout by the SNAPSHOT dump routine of contents of selected sections of memory.

During COMPASS assembly, the PROGDUMP macro results in a call to SNAPSHOT being generated into the binary object deck. At execution time the SNAPSHOT routine is automatically loaded and linked to the subprogram. Because instructions are inserted in the object code, PROGDUMP should not be used immediately following conditional skips or tests involving more than one word of memory.

The requested dump may be in octal, character, or decimal floating point.

Each time it is called, the routine writes on the job OUT file a line containing the 1-4 character BCD dump identifier, the location of the call to SNAPSHOT, and contents of the A and Q registers and the three index registers. When the register file option is selected, SNAPSHOT prints the subheading REGISTER FILE, followed by the contents of the last 32 highspeed registers. SNAPSHOT then dumps the selected portions of memory.

The memory dump consists of 8-word lines of data printed in the designated mode and preceded by the 1-4 character identifier, the absolute octal address of the first words of the line. When SHAPSHOT detects a line that contains words all identical to the last word of the preceding line, the line is suppressed. Suppressed lines are noted with the word GAP on the listing.

The macro format is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	PROGDUMP	(fwa,lwa,mode,id,chap)	

fwa, lwa — Beginning address and ending addresses (specified as a symbol ± a constant). If they are the same or fwa is greater than lwa, the dump produces only the contents of console registers. If fwa is greater than lwa, a message is written on the job OUT file.

mode — Digit that specifies the format of the memory dump:

- 1 Octal
- 2 Character (6-bit BCD)
- 3 Floating Point
- 4 Register file
- 5 Register file and octal dump
- 6 Register file and character dump
- 7 Register file and floating point dump
- other Message ILLEGAL MODE written on the OUT file. The request is ignored.

id — 1-4 BCD characters printed on each line of the output to identify the dump.

chap — Optional parameter

- 2 Area to be dumped is in Chapter Two.
- 0, 1 or blank Area to be dumped is in Chapter One.
- other Message ILLEGAL CHAPTER chap written on OUT file

To insert temporary calls to SNAPSHOT into a loaded program see SNAP card, 10.1.9.

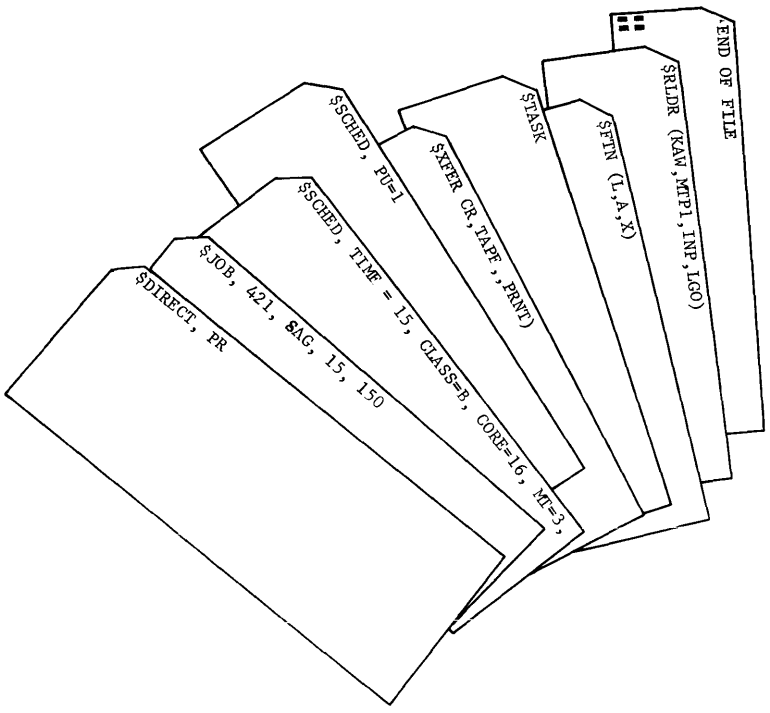
A user submits a job to MASTER as a set of control cards which may be accompanied by source language decks, binary object decks, and data. The control cards are described below. Source language decks are described in the compiler and assembler reference manuals, and binary object decks (generated by compilers and assemblers for execution under MASTER) are described in section 10.2.

10.1 CONTROL CARDS

Through the control cards, programmers provide the information MASTER needs to allocate storage and equipment, schedule a job and initiate its tasks, assign priorities, and perform other job-monitoring functions. A MASTER control card, with the exception of an end-of-file card, is identified by a $ in column 1. When a control card is transferred to a JOB's INP file, any extra spaces are removed.

When a control card is interpreted, it is copied on the job's standard output file. If a control card is out of sequence or contains an error, the job is terminated and a message is written on the job's OUT file.

Sample MASTER Control Cards



10.1.1 DIRECT

The DIRECT control card is optional; it permits a job to use directly the card reader for input, a printer or a punch for standard output without first transferring the data to files on mass storage. When used, the DIRECT card precedes the JOB card; it is the first card in the job deck.

```
$DIRECT,CR,PR,PU
```

CR When the job is to be initiated, MASTER does not transfer the job deck to an INP file on disk, but makes the input card reader the JOB's INP file. No other job can be transferred to mass storage while the card reader is so occupied. When CR is omitted, normal job input and transfer to mass storage takes place.

PR Standard printer output for the job goes directly to a printer without first being transferred to the OUT file for the job. The job will not be initiated until a printer is available. When PR is omitted, normal transfer of standard output to the OUT file takes place.

PU Standard punch output for the job goes directly to a card punch without first being transferred to the PUN file for the job. The job will not be initiated until a punch is available. When PU is omitted, normal transfer of standard punch output to the PUN file takes place. If PU appears, a p limit must also appear on the JOB card.

The parameters are free field and may appear in any order. A parameter and its comma may be omitted. Equipment reserved for a job by a DIRECT card is released when the job terminates.

If a parameter on a DIRECT card is not in correct form, the input backgrounder skips the erroneous parameter and types the following message on the console typewriter:

```
I JOBi *BKI 01
```

i identifies the last job read in. Normal job processing continues.

DIRECT makes possible the running of a job with input or output that exceeds available mass storage. A printer or a punch acquired with a DIRECT card must be scheduled on a SCHED card (10.1.3).

10.1.2 JOB

A JOB card must appear in a job deck either as the first card or, if DIRECT is used, as the second card. It can be followed by a SCHED, Task Name, TASK, or RLDR control card. The system ignores any additional JOB cards detected before an end-of-file card.

```
$JOB,c,i,tl,l,p
```

- c — 1 to 8 BCD characters indicating account to be charged; may not be omitted.
- i — 1 to 8 BCD characters identifying originator of the job; may not be omitted.
- tl — Limit ≤ 1440 minutes, for job execution beyond which MASTER terminates the job. The limit does not include background processing of files. If tl is omitted but one of the other limits is specified, its comma must appear.
- l — A line printer limit ≤ 99999 for the OUT file, beyond which MASTER terminates the job. If line limit is omitted but punch limit is specified, its comma must appear.
- p — A punched card limit for the PUN file (0 to 99999) beyond which MASTER terminates the job. Comments may follow the p field when it is terminated with a comma.

All limits are optional. When they are not specified, MASTER uses installation parameters for time and line limits. When no punch limit is specified, the job will not have a PUN file.

Example:

```
$JOB,421,BT2,15,150,100,COMMENTS APPEAR HERE
```

10.1.3 SCHED CONTROL CARD

SCHED cards, of which there may be two per job, immediately follow the JOB card in the job deck. Correctly used, the SCHED cards promote efficient MASTER operation. Chapter 2 describes how the MASTER job scheduler uses information obtained from the cards. SCHED information can be continued on second SCHED card. SCHED cards are followed by a Task Name, XFER, FILE, TASK, or an RLDR card.

```
$SCHED,TIME=te ,CLASS=cl ,CORE=qp ,SCR=seg ,
ABORT=dl ,RF=flg, peq1=u1,...,peqn=un
```

All fields are optional; they may appear in any order on either card. If a field other than peq is inadvertently repeated, (two TIME fields for instance) the second entry has precedence. When a peq field is repeated (two entries for 501=u), the first entry has precedence. A field cannot be continued from one card to the other.

DO NOT SCHEDULE MORE THAN A JOB NEEDS. A job that requests unneeded core, mass storage, and peripheral equipment may needlessly wait for these facilities, which must be available before the job can be initiated. Also, when the job is processed, the idle facilities are withheld from other jobs requesting them.

TIME=te — Estimated running time (0 to 99999 minutes). MASTER uses it to determine if the job is special (section 2.3.1). When this field is omitted, the job cannot qualify as a special job. te is not a limit; for time limit, see JOB card.

CLASS=cl

cl	Class
E	Emergency
B	Background
I	Input/Output
C	Compute

When both TIME and CLASS fields are specified and te lies in the special class range, the job acquires the higher class as determined by both fields. If the job is specified as E or B which are higher than special, it will maintain its specified class, but if it is specified as I or C, which are lower than special, it is reclassified as special.

When the CLASS field is omitted and the job does not qualify as special, it becomes I or C depending upon an installation parameter.

CORE=qp — Estimate of the maximum amount of core in quarter pages used by tasks residing in core simultaneously. An estimate includes requirements of library tasks such as compilers and assemblers. Estimates for object decks must allow for expansions of pseudo instructions, macros, library routines, etc.

A job cannot exceed its estimated core. When the loader determines that a requested task exceeds the estimate, it terminates the job and writes a message on the OUT file.

When the CORE field is omitted, qp is set by an installation parameter.

SCR=seg — Number of segments (250 maximum) of mass storage scratch area required. Each segment is 10,000 words.

If the sum of the mass storage requirements as indicated by the line and punch limits (JOB card) and the SCR and ABORT requests (SCHED card) exceed the storage reserved for these files, the job is not initiated. A diagnostic message is typed on the console typewriter and the INP file containing the job is released.

When the SCR field is omitted, seg is set by an installation parameter.

ABORT=dl — Requests a recovery dump of task. dl specifies number of lines to be dumped if the job is abnormally terminated. MASTER reserves mass storage for dl lines of dump. When dl is 0 or the field is omitted, the user obtains only the dump described in section 2.6.1 and not a dump of his task.

RF=flg — Non-zero flg indicates that the job will use the register file. When any task of the job is interrupted, the contents of file registers 40-77 will be saved. When a task of the job is again placed in execution, the contents will be restored.

When flg is zero, or the field is omitted, the register file contents are not saved.

$peq_i=u_i$ — All peripheral equipment required by a job must be scheduled. The parameter peq identifies the hardware type; u designates the number of units or drives required.

peq	Hardware Type
405	card reader
415	punch
501	printer
607	607 tape unit
606	606 tape unit
604	604 tape unit
603	603 tape unit
852	852 disk drive
853	853 disk drive
854	854 disk drive
813	813 disk drive
814	814 disk drive
863	863 drum

Examples of peq fields:

```
$SCHED,501=1,607=3,852=2
```

This card will reserve one 501 printer, three 607 tape units, and two 852 disk drives.

Mass storage drives need be scheduled only for files on removable Class B devices

A printer or punch required by a DIRECT job must be scheduled.

If peq requirements exceed system capacity, the job cannot be scheduled. The message D JOB i *SCH 07 is typed, the job is abnormally terminated,and its INP file is released.

A job can be initiated when all peq requirements are satisfied.

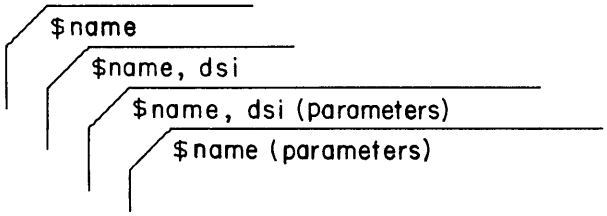
Any attempt by a job to use a device not scheduled causes job termination with a message on the job's OUT file.

To summarize, when no SCHED card is used, the job cannot be classified as special, installation parameters are used for class, core, and scratch, no recovery dump will be given on abnormal termination, and the job cannot use the register file.

10.1.4 TASK CALLS

A Task Name control card directs MASTER to call and load the named program task from the specified file (or from the library if no file is specified), to pass the parameter string if one is given on the control card, and to begin execution of the task. Usually at least one Task Name card will follow the SCHED cards for a job.

A Task Name card has one of the following forms:



name — 1-4 alphanumeric characters identifying the task to be called; name is the only required parameter.

dsi — The dsi of an opened file from which the named task is to be loaded. When the dsi is 0 or the field is omitted, MASTER will look for the task on the system library.

parameter string — Parameters used by the called task, for example the *DEF parameters specified in 5.4.1. MASTER removes any spaces in the parameter string before passing the string, with parentheses, to the called task.

When dsi is INP, the binary object deck for the task, which may or may not begin with a TASK card, (10.1.5) immediately follows the Task Name card. When dsi is other than INP or *LIB, the loader relocates to block 1 for mass storage or rewinds the tape.

A source or data deck required by the task may follow the Task Name card on the INP file.

Examples of Task Name cards:

```
$JOB1,INP
$GLIB(,LIB2,DIR2)
$*DEF(0,W,10,,,I)
$FTN(L,A,X)
$XFER(,,PGMA)
```

10.1.5
TASK

A user may precede the first IDC card (10.2.2) of a task with a TASK card.

```
$TASK,a,b,2CH
```

a Inherited priority multiplier

b Inherent priority

The priority of the task is computed as priority=a*I+b where I is the priority of the caller of the task. a and b are integer quantities such that priority≤ 63. When a is I and b is 0, the task inherits the priority of its caller. When a is 0 and b is non-zero, the task priority is inherently fixed at b.

2CH Two-chapter task; omitted indicates a one-chapter task.

When a binary object deck follows a Task Name card on the INP file, and no TASK card precedes the IDC card of the deck, the task is a one-chapter task and has inherited priority (a=1, b=0).

The first TASK card through the first ELD card (10.2.8) or end-of-file card (10.1.11) are processed as the called task. Any additional TASK cards are ignored.

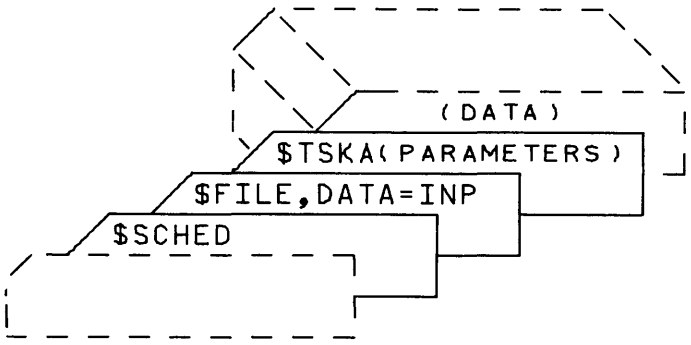
When a TASK card follows a Task Name card on the INP file, it overrides a TASK card on the specified dsi.

```
TASK CARD ──→ $TASK,1,0,2CH
TASK NAME ──→ $EXMP,LGO
   CARD
```

10.1.6
FILE

```
$FILE, dsi1= dsi2
```

A FILE control card indicates to MASTER that dsi_2, defined earlier in the job, can also be referred to as dsi_1. They are the same physical file; thus, two tasks of the same job can refer to the same file by different names. When dsi_2 is INP, OUT, or PUN, FILE can immediately follow SCHED. Otherwise, it must come after the *DEF control card, FILE card, or task that defines dsi_2 (5.4). For example, TSKA, below, refers to file DATA for its input. The data can be entered on the INP file by using:



10.1.7
RLDR

With the RLDR control card, a user can obtain an absolute record of his program task so that it can be called and loaded by the absolute rather than the relocatable loader.

```
$RLDR(name, odsi, idsi1, idsi2, idsi3)
```

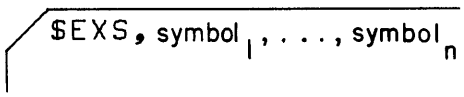
When MASTER detects the RLDR card, it calls the relocatable loader which loads and links programs from input files idsi with object routines on the library that satisfy external symbols. The loaded task, preceded by a header record that includes the task name, is written as an absolute record on odsi. Thereafter, when a programmer calls the task from odsi, it will be loaded by the absolute rather than the relocatable loader.

10.1.8 EXTERNAL SYMBOL

External symbol cards provide the programmer with a means of declaring or equating external symbols after a task has been assembled or compiled.

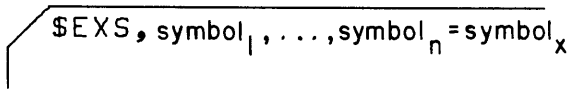
A card contains one of two types of declarations:

External symbol declaration



$Symbol_1$ through $symbol_n$ are external symbols not defined in the program decks. The loader seeks them as entry points to library subprograms causing the subprograms in which they appear to be loaded.

External symbol equivalence



$Symbol_1$ through $symbol_n$ are equated to $symbol_x$. $Symbol_x$ must be defined somewhere in the program or appear on the library file as an entry point to a library routine. It need not be defined prior to processing of the EXS card.

The external symbols may appear anywhere in the program. $Symbol_x$ must not appear on the left of = on another EXS card. If it does, the equivalence string forms a closed loop.

Symbols are separated with commas. EXS cards are placed between the IDC and ELD cards in a binary deck. Their quantity is limited only by the size of the loader symbol table.

10.1.9 SNAP

Through use of a SNAP control card, a user may request a periodic printout by the SNAPSHOT dump routine of contents of selected sections of memory. The requested dump may be in octal, character, or decimal floating point.

Each time it is called, the routine writes, on the job OUT file, a line containing the 1-4 character BCD dump identifier, the location of the call to SNAPSHOT, and contents of the A and Q registers and the three index registers. When the register file option is elected, SNAPSHOT prints the sub-heading REGISTER FILE, followed by the contents of the last 32 highspeed registers.

The memory dump consists of 8-word lines of data printed in the designated mode and preceded by a 4-character identifier (provided by the user), the absolute octal address of the first word of the line, and its relocatable word address. When SNAPSHOT detects a line that contains words all identical to the last word of the preceding line, the line is suppressed. The suppressed line or lines are noted with the word GAP on the listing.

SNAP control cards may be inserted in a job input deck after a Task Name control card if the named task is not on the INP file, after an ELD card if the task named is on the INP file, or after an OCC control card. They cannot follow binary card images on a file other than the INP file. For each SNAP card, a calling sequence to SNAPSHOT is created and stored in available memory. An instruction in the task specified on the SNAP card is replaced with a return jump to the calling sequence. The instruction is saved within the calling sequence and executed after the dump is taken. Because this instruction is not executed in its normal location, it must not be modified by the program and it must not involve more than one word such as skips, indirectly addressed instructions, and searches.

```
$SNAP,(subp)n, fwa, lwa, mode, id, chap
```

(subp)n — Location of the instruction to be replaced. The name (1-8 BCD characters) of the subprogram containing the instruction appears within the parentheses. n is an address of not more than 5 octal digits to be added to subp to obtain the relative address of the instruction.

fwa, lwa — Beginning and ending addresses of the area to be dumped. lwa must exceed or be the same as fwa, otherwise the SNAP statement is ignored, and an error message is written on the OUT file. If they are the same, the dump produces only the contents of console registers. fwa and lwa must both assume one of the following forms:

D/id/n Dump begins or ends with octal location n (0 to 77777) in the data area specified by 1-8 character id.

Cn Dump begins or ends with octal location n (0 to 77777) in the common area.

(subp)n Dump begins or ends with octal location n (0 to 77777) in subprogram subp.

The area to be dumped must reside entirely within data area D/id, the common area (C), or the subprogram area (subp) of subprogram memory.

mode Format of the dump; if mode is illegal the SNAP card is bypassed, and an error message is written on the OUT file.

O	Octal
C	Character (6-bit BCD)
F	Floating point
R	Register file
OR or RO	Octal; register file
CR or RC	Character; register file
FR or RF	Floating point; register file

id 1-4 BCD characters identifying the dump. These four characters precede each line of output on the dump.

chap 1 Blank, or omitted; dump area is in Chapter One.

2 Dump area is in Chapter Two.

Rules:

1. To prevent excessive printout, avoid calling SNAPSHOT within a loop.
2. The location at which the SNAP occurs must not be altered during execution or by OCC (octal correction) cards (10.1.10).
3. Do not replace the following types of instructions:

   Instructions involving more than one word such as searches and skips

   Indirectly addressed instructions

   Instructions modified by program execution

4. Before a SNAP card is read, the loader must have processed an XNL card (10.2.6) or an EXS card (10.1.8) declaring SNAPSHOT as an external symbol. Otherwise, the SNAPSHOT routine will not be loaded with the task containing the calling sequence.

Example:

```
$SNAP,(NTEST)23,C01000,C02215,0,COMM,2
```

Locations 01000 through 02215, subprogram NTEST Chapter Two common will be dumped in octal mode each time instruction 23_8 of program NTEST is executed.

10.1.10 OCC

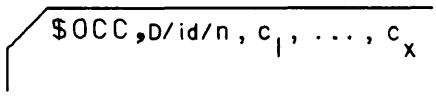
With an octal correction card, a user may change a loaded task by altering contents of locations or adding octal instructions or data. Corrections may apply to any of the subprograms making up the task; but they apply only to Chapter One of two-chapter tasks. OCC cards may be inserted in a job input deck after a Task Name control card if the task is not on the INP file, after an ELD card if the task is on the INP file, or after a SNAP control card. They cannot follow binary card images on a file other than INP. When SNAP and OCC cards are intermixed, an OCC statement must not destroy the jump to the SNAP calling sequence. An error on an OCC card produces a message on the job OUT file. Although all the OCC cards are processed, an error prevents task execution.

The location field (first parameter on OCC card) indicates whether the OCC card:

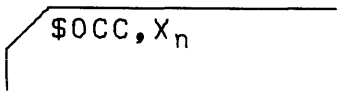
1. Sequentially corrects program instructions beginning at word location n (0 to 77777) in named subprogram (1-8 BCD characters)

```
$OCC,(subp)n, c1, ..., cx
```

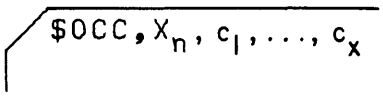
2. Sequentially corrects data words starting at word location n (0 to 77777) in the specified task data area



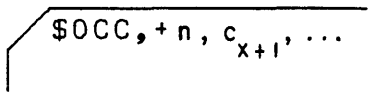
3. Extends program execution area by n word locations (0 to 77777). If n exceeds available memory for task, program area is extended only by the amount of memory available; a message is written on the OUT file specifying actual extension length.



4. Places information in extension area starting at word location n. This OCC card must be preceded by the card described in 3. If n is larger than the previously defined area, the corrections are not loaded.



5. Continues corrections to program or data area begun on card described in 4 starting at n addresses (0 to 77777) beyond last correction on preceding OCC card. When only + appears, no addresses are skipped.



$c_1,\ldots,c_x$ Octal correction fields separated by commas. A field may be blank or omitted (two contiguous commas) or may contain a 1- to 8-digit octal value which may be accompanied by a positive or negative relocation factor.

Each octal value is stored right justified in a computer word. When the value occupies fewer than 8 digits (leading zeros omitted), the loader zero-fills the remainder of the word. Blanks in a correction field are ignored; when a field is blank or omitted, the location represented by the field is unchanged.

The loader stores a correction as it appears, or with word or character relocation of the address, into the memory word determined by the location field and the position of the correction field on the card.

Fields may be combined:

correction	Stores correction by word address as it appears in field without address relocation.
correction reloc factor	Stores correction by word address relocating address field positively relative to relocation factor.
correction -reloc factor	Stores correction by word address relocating address field negatively relative to relocation factor.
correction reloc factor C	Stores correction by word address positively relocating address field as character address relative to relocation factor.
correction -reloc factor C	Stores correction by word address negatively relocating address field as character address relative to relocation factor.

Relocation factors:

(subp)	1-8 character subprogram name enclosed in parentheses. The loader relocates the address field relative to the first word address of the named subprogram (Example 2).
*	The loader relocates the address field relative to the first word address of the last subprogram named in this series of OCC cards (Example 1).
C	The loader relocates the address field relative to absolute address 00000 (Example 6).

D/id/ Loader relocates the address field relative to the first word address of a data block identified by 1 to 8 BCD character id (Example 6).

X Loader relocates the address field relative to the first word address of the program extension area (Example 6).

Examples:

1. Enter octal correction 200xxxxx at address 00070 relative to subprogram PROG1. The * tells MASTER to relocate address 00100 relative to subprogram PROG1. MASTER loaded PROG1 at address 73355.

```
$0CC,(PROG1)70,20000100*
```

Result: (73445) = 20073455

2. Enter octal correction 200xxxxx in location SUB1+77 of subprogram SUB1; relocate 00100 relative to subprogram SUB1. Enter correction 400xxxxx in location SUB1+100 of subprogram SUB1; relocate 00101 relative to subprogram SUB2. MASTER loaded SUB1 beginning at address 63652 and SUB2 beginning at 44711.

```
$0CC,(SUB1)77,20000100*,40000101(SUB2)
```

Result: (63751) = 20063752
(63752) = 40045012

3. Enter the octal value 00000036 into locations 20,21,22,23,24,25 of subprogram SUB1; because values are right justified, all the octal corrections 00000036, 0000036,... are stored as 00000036. MASTER loaded SUB1 beginning at address 63652.

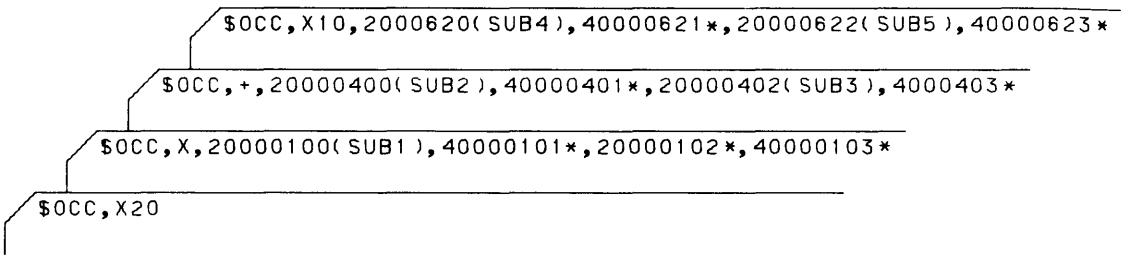
```
$0CC(SUB1)20,00000036,000036,00036,0036,036,36
```

Result: (63672)
(63373)
(63674)
(63675) = 00000036
(63676)
(63677)

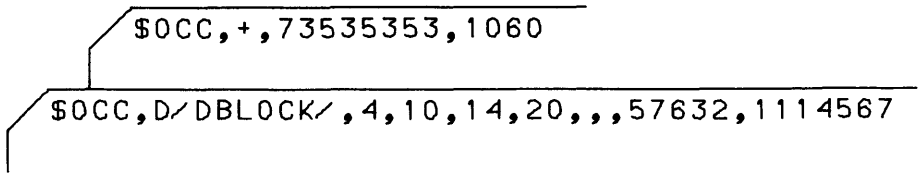
4. Assign 20_8 locations to the program extension area. Enter information into the first four locations of the extension area relocated relative to subprogram B1 which MASTER loaded at address 56345.

   Continue loading the program extension area; relocating the next two corrections relative to subprogram SUB2, the last two relative to subprogram SUB3.

   Load the corrections pertaining to subprogram SUB4 and subprogram SUB5 into program extension location 10.



5. Enter four octal values in successive data area locations starting at first word. Skip 2 addresses. Enter four more values. MASTER assigned the data area to address 30000.



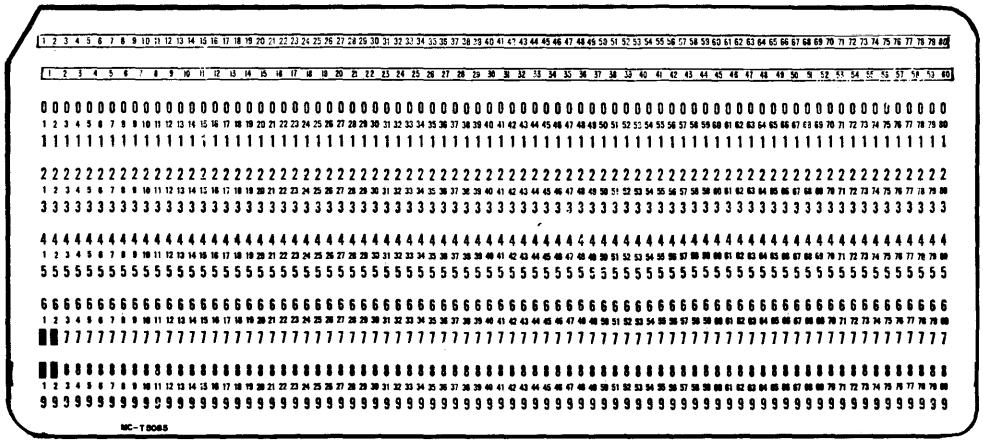
Result: (30000) = 00000004
(30001) = 00000010
(30002) = 00000014
(30003) = 00000020
(30004) = unchanged
(30005) = unchanged
(30006) = 00057632
(30007) = 01114567
(30010) = 73535353
(30011) = 00001060

6. Enter correction 010xxxxx into location 70 of B1; xxxxx is 00001 modified relative to the address of the program extension area. Enter 200xxxxx into B1+71; xxxxx is modified relative to the fifth address in the common area. Put 400xxxxx into B1+72; xxxxx is 00007 relocated relative to the seventh address in data area. Skip 2 addresses; put 200xxxxx into B1+75. xxxxx is 00007 character-relocated relative to B1.

```
$0CC,+2,20000007(B1)C
$0CC,(B1)70,01000001X,20000005C,40000007D/BLOCK/
```

10.1.11
END-OF-FILE

A job is terminated with an end-of-file card characterized by 7,8 punches in columns one and two. This card has the same format as an end-of-file card that terminates PUN output (2.6.2). Columns 3-80 may contain comments.



10.2 BINARY CARDS

The MASTER relocatable loader (section 3.5) accepts relocatable binary information in the form of decks or card images. Compilers and assemblers operating in conjunction with MASTER generate binary decks that meet the specifications outlined below.

10.2.1 GENERAL SPECIFICATIONS

Binary card types, and the order in which the loader seeks them:

Subprogram Identification Card (IDC)

Block Common Table Cards (BCT)

Subprogram Entry Point Cards (EPT)

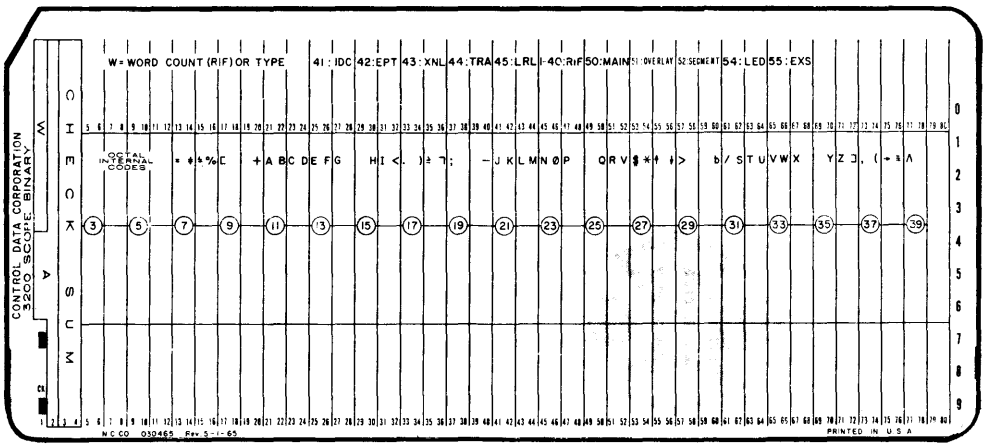
Relocatable Information Cards (RIF)

External Name and Linkage Cards (XNL)

Transfer Card (TRA)

End Loading Card (ELD)

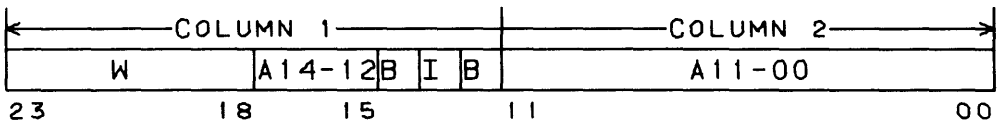
An IDC card through TRA card and the cards between, make up a subprogram deck.



The fields in columns 1-4 of all but the ELD card meet the following specifications.

Columns 1, 2 form one 24-bit word with the following format:

Word One:



W = 2-octal digits identifying card type; on RIF card it also indicates word count

A = Bits 17-15, 11-00, form a 15-bit value the significance of which is card dependent

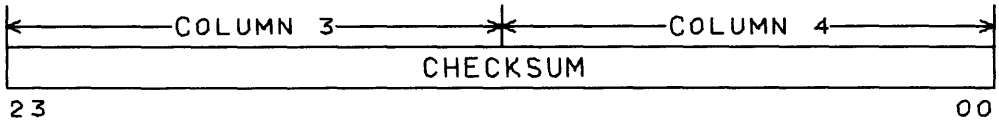
B = Bits 14 and 12 (7,9 punch in column 1) are always punched; they identify card as part of binary deck

I = Bit 13 (8 punch in column 1)

blank	checksum used by loader
punched	checksum ignored by loader

Columns 3,4 form one 24-bit word with the following format:

Word Two:

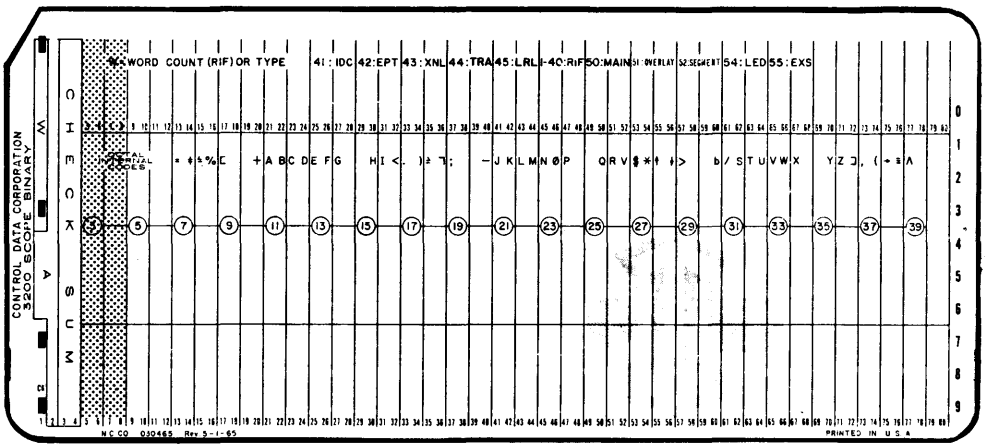


Checksum 24-bit sum with end-around carry of all other columns on card (1, 2, 5-80)

Columns 5-80 contain information relevant to the card type.

10.2.2
IDC

The Subprogram Identification card gives the length and name of the subprogram to be loaded.

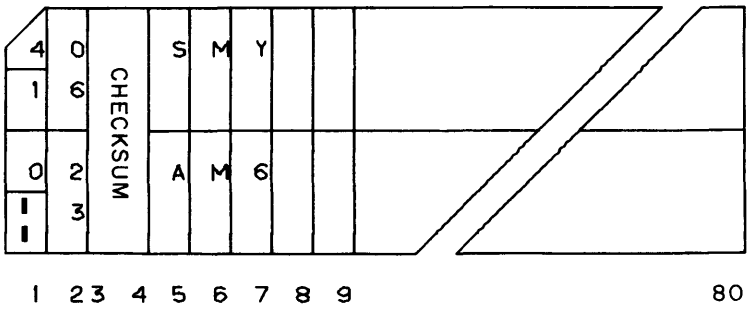


$W = 41_8$

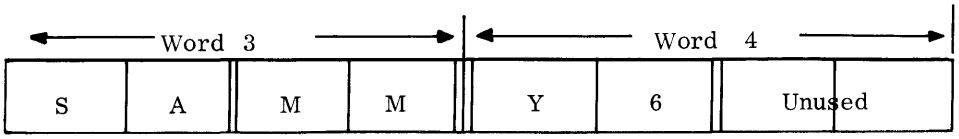
A = Number of words, in octal, in the subprogram (0-77777)

Card columns 5-8 form computer words three and four. Each 1/2 column contains one 6-bit BCD character of the subprogram name of 1-8 alphanumeric characters. Card columns 9-80 are unused.

Example:

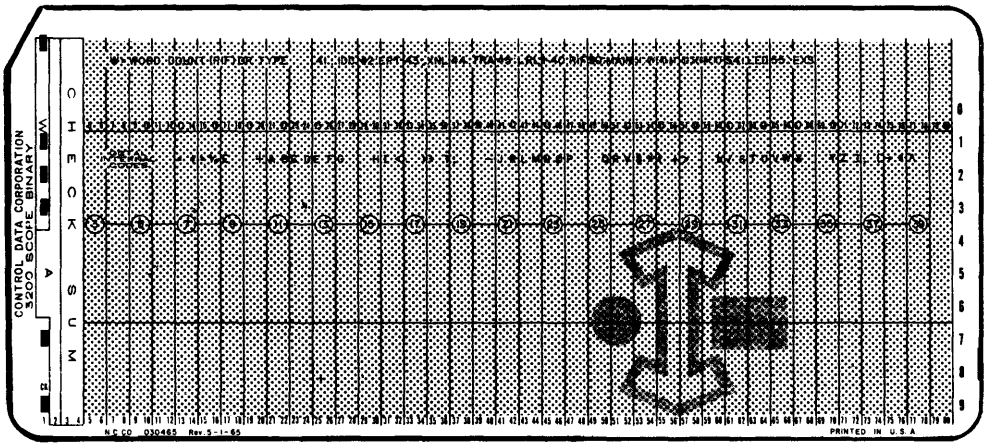


The subprogram, named SAMMY6, is 623_8 words long.



10.2.3
BCT

The Block Common Table card gives the length of each common and data block area declared in the subprogram.



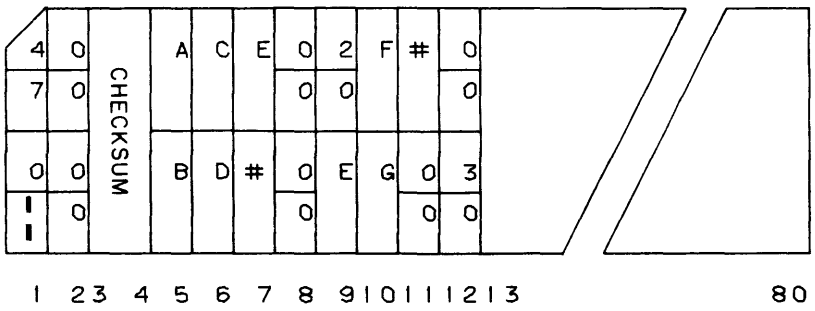
W = 47_8

A = Sequence number, right justified in field, when deck contains more than one BCT card. First BCT card has sequence number 0. BCT cards must be sequential in the deck.

Columns 5-80 contain as many fields as there are common and data blocks. A field is not continued to the next BCT card. Each field contains the name in BCD code of the block of common and data, and their length in octal. A name that begins with a number identifies numbered common; when it begins with a letter, it identifies labeled common, or data. It may be 1 to 8 characters long. When it is fewer than 8 characters it is separated from length by #(72_8).

Length is an 18-bit value. The least significant 15 bits give the length of common. When length occupies fewer than 15 bits it is right justified in the field and zero filled. The most significant bit indicates chapter one common (0) or chapter two common (1).

Example:

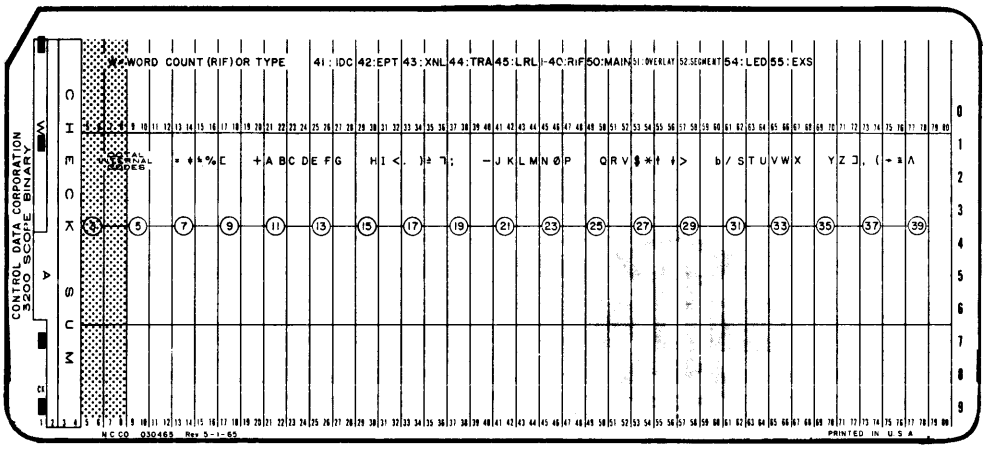


Sequence number is 0

Labeled common block named ABCDE is 16 words (20_8); EFG is 24 words. Both are in Chapter One. Columns 13-80 are unused.

10.2.4 EPT

An Entry Point Name card may appear in any position in the deck. It defines entry point names of the subprogram.



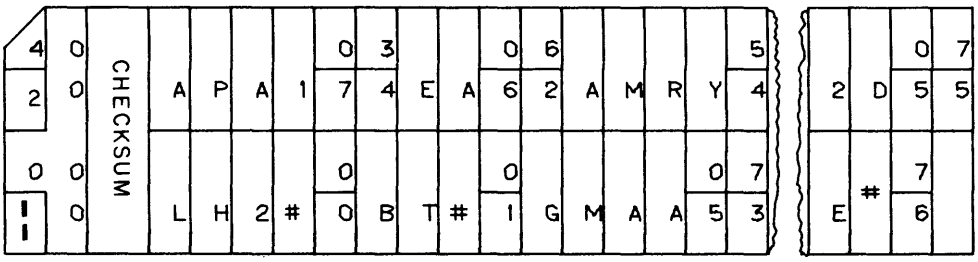
W = 42_8

A = Sequence number right justified in field when deck contains more than one EPT card. First EPT card has sequence number 0. Cards must be sequential in deck.

Each field in columns 5-80 consists of an entry point name in BCD and its address. A field is not continued to the next EPT card. A name may be 1-8 alphanumeric characters; when it is fewer than 8 characters, it is separated from address by #(72_8). Address is right justified in an 18-bit field. If the most significant bits are non-zero, the loader converts the address to a character address; if zero, the address remains a word address. The remainder of the field is zero-filled.

When the loader reads a name of fewer than 8 characters, it left justifies it, removes spaces and fills the remaining character positions with blanks (60_8). It then places the 8-character name in a symbol table.

Example:

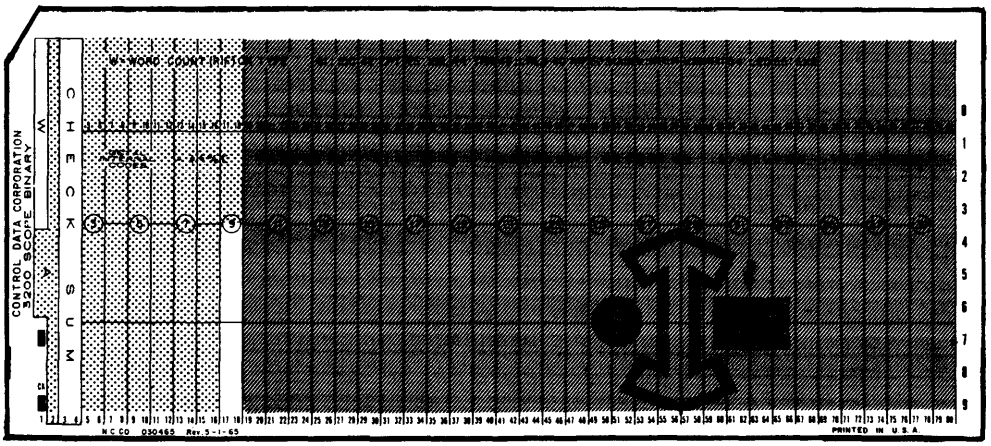


Sequence number is 0

Entry Point Name	Address
ALPHA 21	70034
BETA	60162
GAMMARAY	55473
⋮	⋮
ZED	57675

10.2.5 RIF

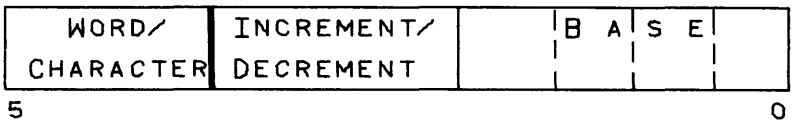
A Relocatable Information card contains program information in the form of relocation bytes and data words. RIF cards are not sequenced.



W = $1\text{-}36_8$; number of words on card

A = load address of first word

Columns 5-20 contain up to 31 6-bit relocation bytes. The first applies to the load address in A. Each of the others applies to the address field of a machine word in columns 21-80. A byte has the following format:



Word/Character

0 relocation factor applies to 15-bit word address

1 relocation factor applies to 17-bit character address

Increment/Decrement

0 increment relocation counter (numbered common)

1 decrement relocation counter (program or labeled common)

Base Address to be relocated relative to subprogram or one of 14 possible common blocks (numbered or labeled)

0 no relocation

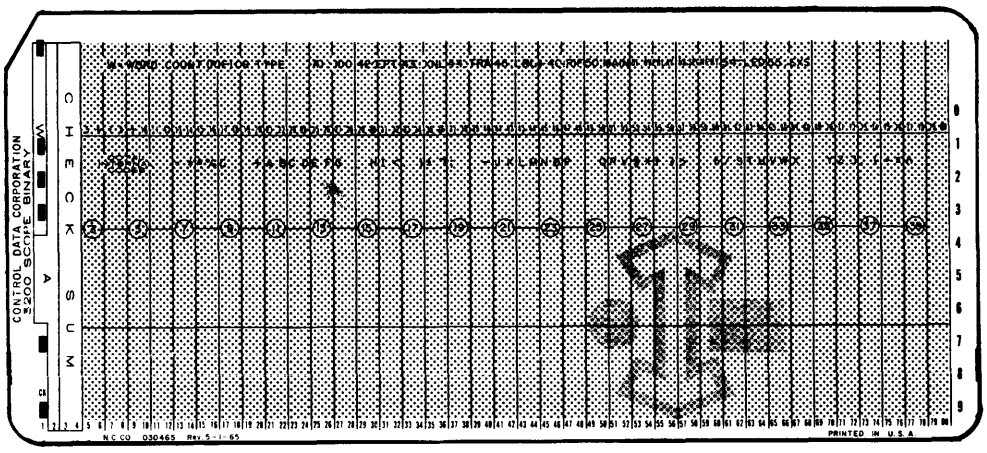
1 Subprogram

2 to 17_8 Common block 1-14

Columns 21-80 contain up to 30 words of data (labeled common) or machine instructions to be loaded after the corresponding relocation factor is applied to the address portion of each word.

10.2.6
XNL

The external name and linkage card indicates external references made by the subprogram.



W = 43_8

A = Sequence number right justified in field when deck contains more than one XNL card. First XNL card has sequence number 0; cards must be sequential in deck.

The field in columns 5-80 each consist of an external name in BCD and its address. A field is not continued to the next XNL card.

A name may be 1-8 alphanumeric characters; when it is fewer than 8 characters, it is separated from address by #(72_8).

Address (low order 15 bits of 18-bit field) indicates location containing a reference to the external name.

The address field of the word at that location may specify another location within the subprogram at which a reference is made to the external name, and so on. Such a series of addresses is called an external string. The low order 15 bits of the last entry in the string contains 77777_8. This string may run in any order through the subprogram. If no reference to the external name is made in the subprogram, the address on the XNL card is 77777_8.

When the loader reads a name of fewer than 8 characters, it left justifies it, removes any spaces it contains and fills the remaining character positions with blanks (60_8). It then places the 8-character name in a symbol table.

All external references are to subprogram relocatable word addresses. However, the external reference may be made either from a word or character type instruction or string. All entries in a string correspond to references from word type instructions, or all are from character type instructions.

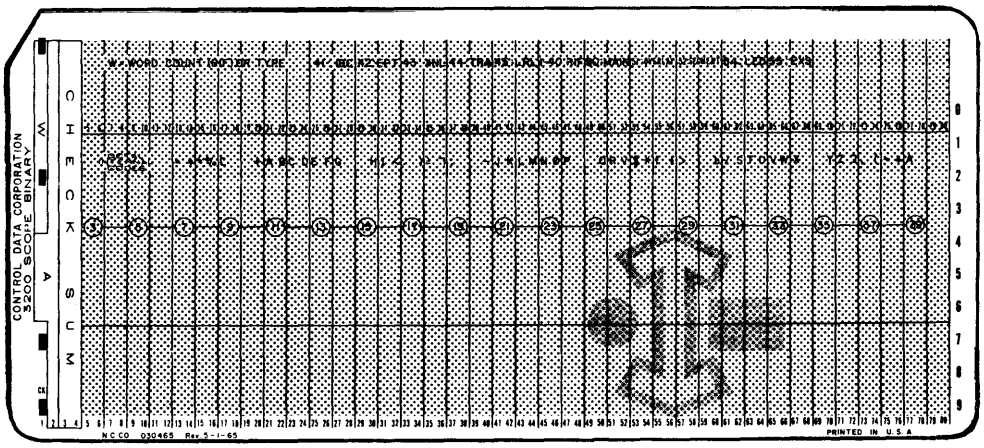
The upper 3 bits of the address field following the external name flag the type of string. If the string is word-oriented, the upper three bits are zero; if the string is character-oriented, they are non-zero.

An external name may be specified more than once in a subprogram deck; each occurrence has its own string. Multiple strings for a single name of a type are tied together by the loader.

The XNL cards may be in any but the first position in the subprogram deck. An external string may refer to previously encountered external symbols only after the relocatable information has been loaded for them.

10.2.7
TRA

The transfer card indicating the end of the subprogram appears at the end of the subprogram deck.

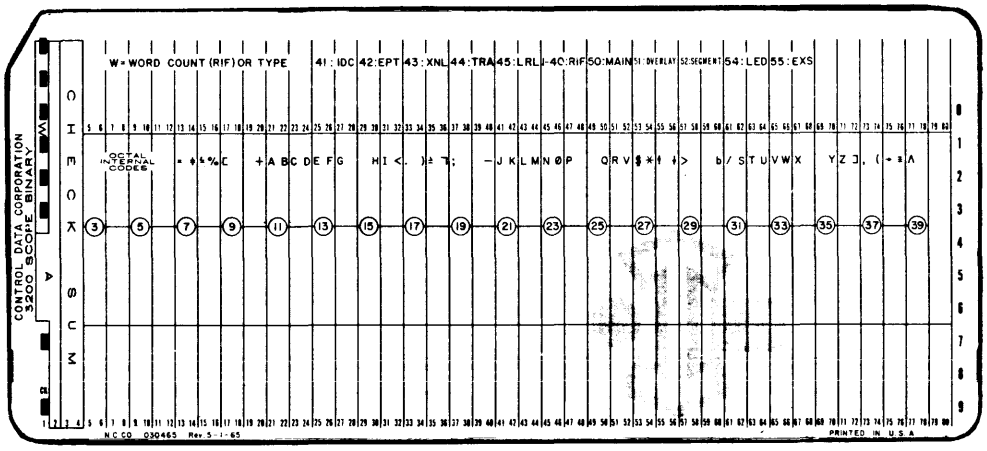


$W = 44_8$

Columns 5-8 give the transfer point symbol in column binary (internal BCD). Columns 9-80 are unused.

The transfer point symbol must appear as an entry point name in the loaded program deck or in a library subprogram. After loading, MASTER transfers control to the last encountered transfer point. If no transfer point symbol is given, or if more than one appears, the loader indicates an error; the first TRA card terminates loading of the subprogram from the library.

10.2.8
ELD



W = 77

Columns 2-80 are unused.

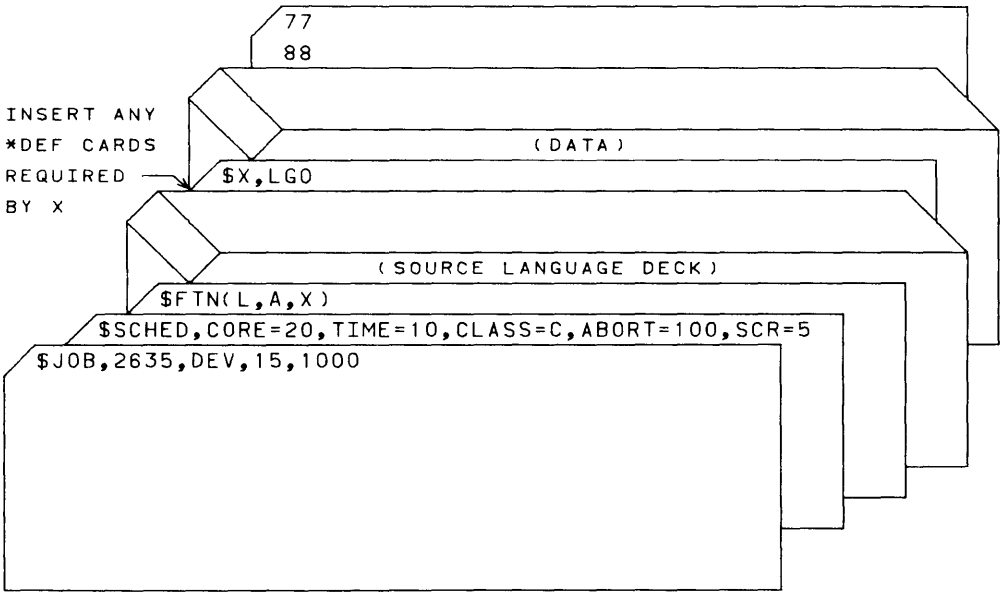
An ELD card signals the end of the program deck. When assembling several subprograms into a task, remove any extra ELD cards generated by the compilers or assemblers. Leave only one at the end of the deck.

10.3 DECK PREPARATION

Decks portrayed in this section represent general job applications of MASTER. The operator submits decks in sequence on the input card reader.

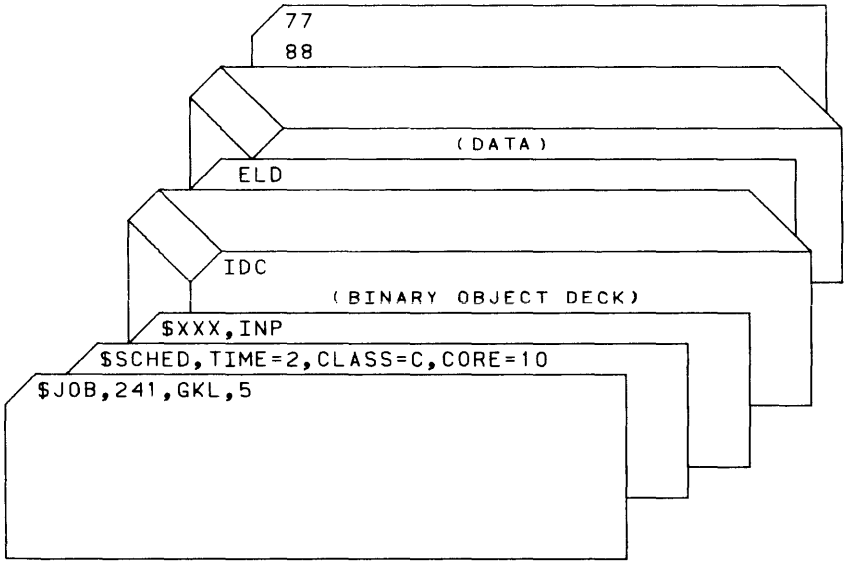
10.3.1 COMPILATION AND EXECUTION

The user with a source language program, such as a set of FORTRAN statements, to be compiled and executed prepares his deck as shown below. In the example, LGO is a scratch file allocated and opened by the first compiler or assembler of a job using System OCARE. It is not closed until the end of the job.



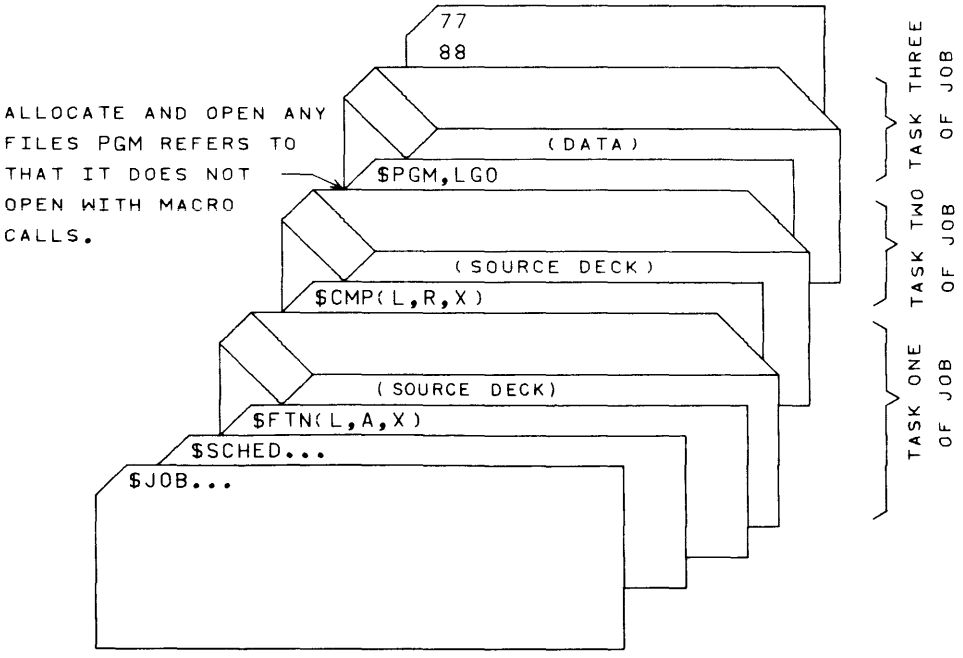
10.3.2 OBJECT DECK EXECUTION

To load and run a binary object deck generated by a MASTER compiler or assembler, organize the deck similar to the example below. Schedule all peripheral equipment required and, if the program itself does not do so, insert control cards to allocate and open files used by the program.



10.3.3 MULTIPROGRAMMING A JOB

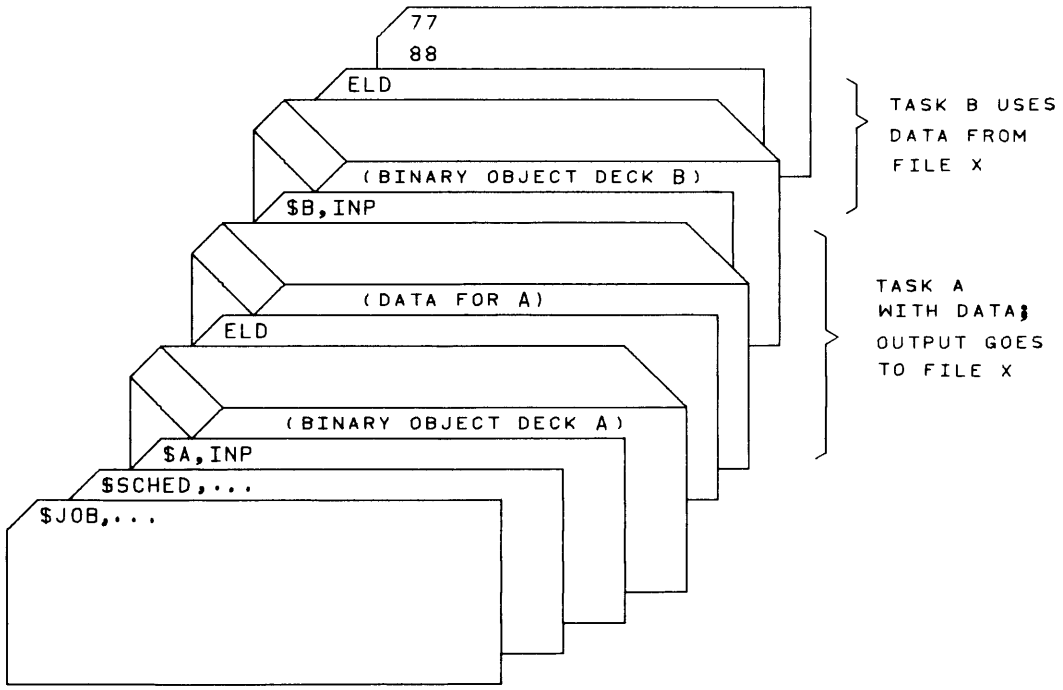
When a job consists of several Task Name cards, the named tasks will not be multiprogrammed within the job.



In this job, the FORTRAN compiler will not be multiprogrammed with the COMPASS assembler. Subtasks of any one task, in this case FORTRAN, COMPASS, or PGM, can be multiprogrammed on an internal basis. Each may consist of several tasks to be multiprogrammed or processed sequentially. The user may be unaware of the existence of such tasks. Internal structuring of tasks within a task is accomplished through the COMPASS CALL macro described in section 9.1.1. In the above example, the COMPASS source deck being assembled into PGM may contain calls for other tasks.

10.3.4 SERIALLY DEPENDENT JOBS

If the output from one job is required before a second job can proceed, the second job must not be submitted to the system before the first has reached completion. There is no way in which the two jobs† can be linked so that one will wait for the other. Some serially dependent jobs can be made serial tasks of the same job so that the system will not time share them. In the example, task B which uses the output from task A, will not be initiated until task A has been completed.

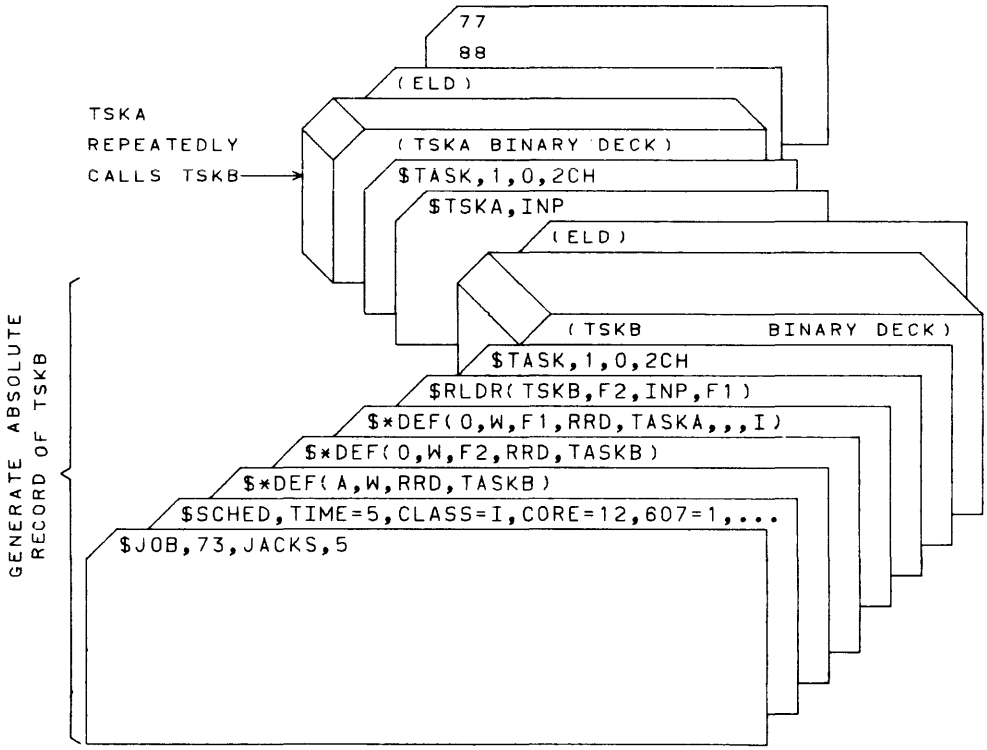


†A job begins with a JOB or DIRECT card and terminates with an end-of-file card.

10.3.5 GENERATE AND CALL ABSOLUTE TASK

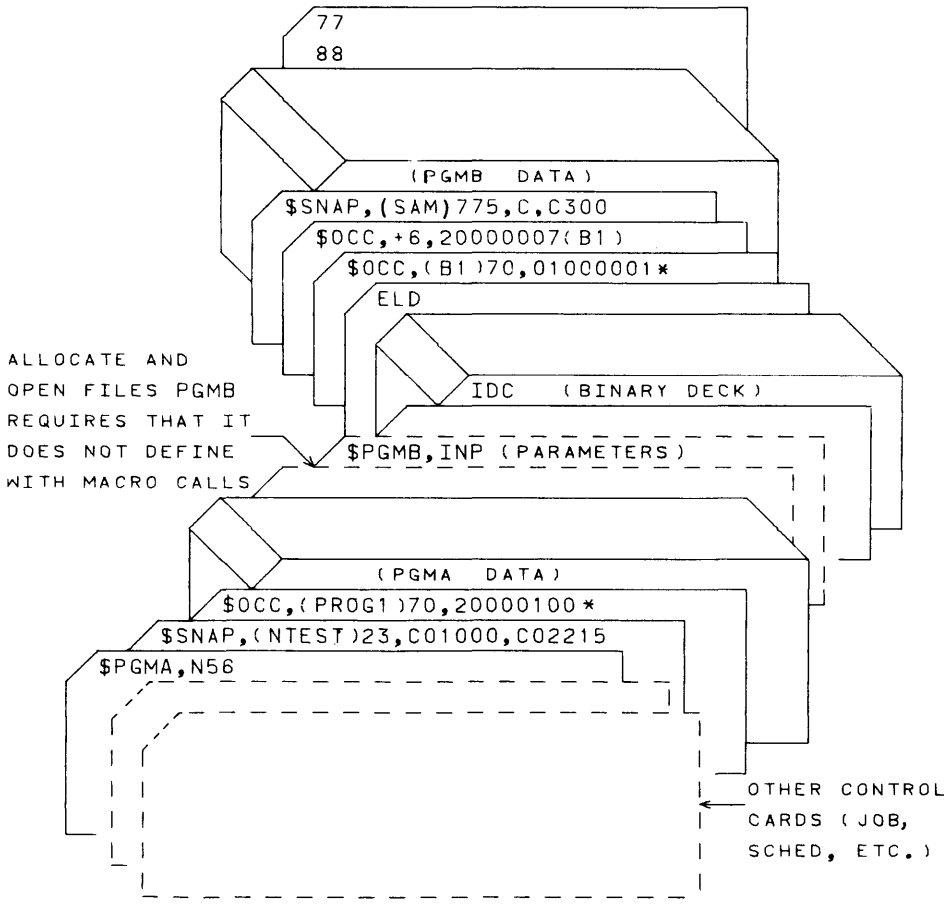
In this example, TSKB is loaded and linked from a binary deck on the INP file and binary card images on F1. It is then written in absolute on F2.

The second part of the job calls for the loading from the INP file and execution of TSKA. TSKA, in turn, repeatedly calls TSKB using a CALL macro.



10.3.6 INSERT SNAP AND OCC CARDS

In this example, PGMA is loaded from file N56. Before it is executed, PGMA is corrected and has a SNAPSHOT call inserted. After PGMA is executed, control returns to the job monitor which interprets the PGMB task name control card. PGMB is loaded from the INP file and has corrections and a SNAPSHOT call inserted. Data for PGMB follow the OCC and SNAP cards.



10.4 LOADER ERRORS

The loader audits and evaluates input during loading. It diagnoses errors and writes messages on a job's standard output file and also on the console typewriter---if the job terminates because of the error.

The format of the error depends on the error type and card format.

checksum

format

symbolic address and linkage

deck and subprogram sequence

I/O errors caused by faulty information, hardware failures or improper input formats

10.4.1 ERROR MESSAGE FORMATS

A loader error causes a message to be written on the OUT file in one of the following formats:

Binary Card

Hollerith Card

Symbol

Miscellaneous

Error codes for the following formats are given in 10.4.2.

Binary Card Error

```
pppppppp    cc    ww    aaaaa
```

p Subprogram Name

c Error Code

w Word Count

a Card Address Field

Example:

```
PROGRAM1   RL    05    06421
```

Hollerith Card Error

```
ppppppp     cc      ww      hhhhhhhh
```

p	Subprogram Name
c	Error Code
w	Word Count
h	Hollerith (columns 2-9)

Example:

```
PROGRAM1   CS      66      SPEC,4,2
```

Symbol Error

```
ppppppp     cc      ssssssss
```

p	Subprogram Name
c	Error Code
s	Symbol, when applicable

Example:

```
PROGRAM1   DS      TAG4A
```

Miscellaneous Errors

```
ppppppp     ccc     ct      xx
```

p	Subprogram Name
c	Error Code
ct	Card Type
x	Information pertinent to card type

Examples:

```
PROGRAM1   TR      03
PROGRAM1   I/O     60
```

10.4.2 ERROR CODES

Formats: B Binary; H Hollerith; S Symbol; M Miscellaneous

Error Code (c)	Meaning	Format	w, ct or s	Card in Error	Error
CB	Common Block	S	Block name	Undetermined	Named common block exceeds previously defined block. (Only the last common block may be expanded by subsequent programs.)
CC	Control Card	M	RLDR	$RLDR	Format error indicated by xx: xx = 01 No task name xx = 02 No input dsi xx = 03 No output dsi
			TASK	$TASK	Format error indicated by xx: xx = 01 Computed priority exceeds 77_8.
CF	Card Format	B	44	TRA	Only one transfer card allowed.
CK	Card Checksum	B	44_8	TRA	Subprogram checksum is in error.
			blank or other	Undetermined	Loader checksum disagrees with card checksum.
CS	Card Sequence	B	01-30 42, 43_8	Determined from word count	No IDC card follows last TRA card.
			Undefined	Undetermined	Card type unrecognizable; card is copied on job's OUT file.
			41_8	IDC	Two IDC cards read with no intervening TRA card. Loader processes second IDC card as if it followed TRA card.
			47_8	BCT	BCT cards out of sequence; must follow IDC card and precede RIF card.
		M	TASK	$TASK	TASK card must precede IDC card.
DB	Data Block	B	$01-36_8$	RIF or BCT	Named data block is larger than previously defined block.

Error Code (c)	Meaning	Format	w, ct or s	Card in Error	Error
DS	Duplicate Symbol	B	42_8	EPT	Named entry point has been previously defined as an entry point.
ELD	End of Load	M		ELD	End of load card was encountered at illegal point during loading of this subprogram.
EOF	End of File	M		Undetermined	End of allocated file condition is encountered during data transmission on dsi named.
LX	External Symbol Loop	S	External Symbol	$EXS	Named external symbol is equated so that it forms a symbol loop.
MA	Memory Allocation Error	S		Undetermined	Program task exceeds available memory; system error.
OV	Memory Overflow	B	41_8 47_8	$SCHED	Subprogram exceeds memory scheduled for job. Common exceeds memory scheduled for job.
RE	Read Error	M			Irrecoverable error during loading of task from named dsi.
RL	Relocation	M	$01\text{-}30_8$	RIF	Load address relocation byte not defined.
SL	String Loop	S	43_8	XNL	String of addresses for symbol given causes a loop.
TR	Transfer Symbol	S	44_8	TRA	The count of transfer symbols in subprograms is not 1. x = number of symbols encountered.
WE	Write Error	S		Undetermined	Irrecoverable error detected while writing task in absolute on dsi named on RLDR card.
UD	Undefined Symbol	S	43_8 44_8	XNL TRA $EXS	External or transfer symbol was not an entry point or equated to defined entry point in any subprogram. Symbol does not exist as entry point in non-system directory.

Users of MASTER can call from the library a general-purpose copy routine (XFER) that enables them to create, maintain, and dispose of standard and nonstandard files. XFER transfers files from one medium to another. It is useful for conversion of nonstandard files to MASTER standard blocked format, transfer of tasks to files, preparation of large volumes of data, transfer of a file to special forms on the printer or punch, etc.

11.1 TASK CALLS

A call to XFER can be made with a Task Name control card or with an XFER macro coded into a user's program task.

Control Card

```
$XFER(idsi,n,odsi,m,f,SS,mode,N
```

Macro

```
LOCATION  | OPERATION, MODIFIERS  ADDRESS FIELD          COMMENTS
1       8 | 10                    20                     41
          | XFER                  (R,idsi,n,odsi,m,f,S$,BN,N)
```

R — (Macro only) Caller assumes ready status and can resume execution without waiting for XFER completion. However, R should not be used if the caller uses either file in the transfer, as they are not available until XFER is complete. When R is omitted, the caller assumes wait status; it will not resume execution until the XFER is completed.

idsi — Data set identifier of the input file. When idsi is not specified, the input file is INP.

n — Number of words per input block (1 to standard block size). When n is not specified, block size is standard.

odsi — Data set identifier of output file. When odsi is not specified, the output file is OUT.

m — Number of words per output block (1 to standard block size). When n is not specified, block size is standard.

f — A form number, 1 to 8 alphanumeric characters, defined at an installation that identifies a punched card form, printer form, or printer tape format to be mounted for odsi.

SS — Appears only when odsi is a printer and indicates that printer carriage control is single space. When SS is omitted, carriage control is program-controlled.

mode — Specifies mode of output; overrides mode parity of odsi.

BN Binary
BD BCD

When mode parameter is omitted, output is in mode of input.

N — Appears for a mass storage file when idsi was prepared by other than a standard blocking routine (PACK) or the input backgrounder. Otherwise, N is omitted. N specifies standard error recovery if input is mass storage, since special recovery is performed for records prepared by PACK or the input backgrounder.

Parameters must appear in the order shown. A comma must appear for an unspecified field if any fields appear to its right. Otherwise, the right parenthesis suffices. For example:

```
$XFER ( TAPE,,PRNT ) or
```

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD
1 8	10	20
	LIBM	XFER
:	:	:
	XFER	(R,TAPE,,PRNT)
:	:	:

Using limited blocking and deblocking, XFER copies file idsi until it reads an end-of-file. Both files must have been opened prior to the transfer. Also any unit devices required must have been scheduled on the SCHED card (10.1.3); for example, a tape drive and printer must be scheduled for a tape-to-print operation. To transfer directly from the card reader, the job must be declared a DIRECT job (10.1.1).

A single user cannot have several XFER's occurring simultaneously. A single non-re-entrant copy of the XFER task is loaded on a call to XFER. When several calls to XFER occur in a program, the calls are queued on this single copy for the job. XFER is copiable to the extent that each job can have its own copy of the routine.

When XFER is called by control cards and the transfer is unsuccessful, the job containing the card is terminated; a message typed on the console typewriter and written on the job's OUT file. If idsi or odsi is a system scratch file, it is released.

For the macro, upon return from XFER, the contents of macro tag + 5 are zero if the transfer was successfully completed and non-zero if the XFER was abnormally terminated.

11.2 SPECIAL FORMS

The printing or punching of files requiring special forms such as checks represents one of the most useful applications of XFER. Instead of putting the print file on OUT for automatic printing by the output backgrounder, which makes no allowance for special forms, the user may create a special output file and call XFER to print it.

When the form parameter is used, the operator is directed to change to the specified form with the message:

```
Rr XFER 01 (MOUNT FORM f ON htCcEeUuuu)
```

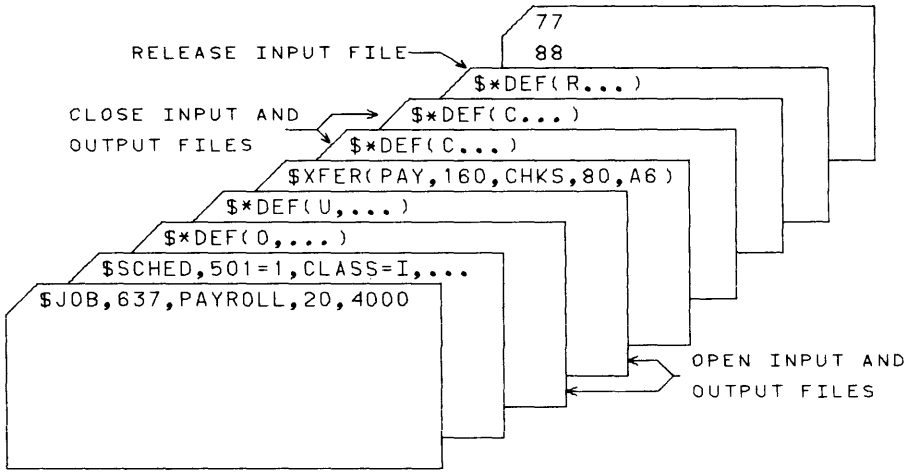
The operator can indicate compliance or refusal. If he complies, the files are processed. If he refuses, the files are not processed; the contents of macro tag + 5 are set non-zero (macro request) or the job is terminated (control card request).

When the transfer is complete, the operator is directed to return to the standard form for the installation with the message:

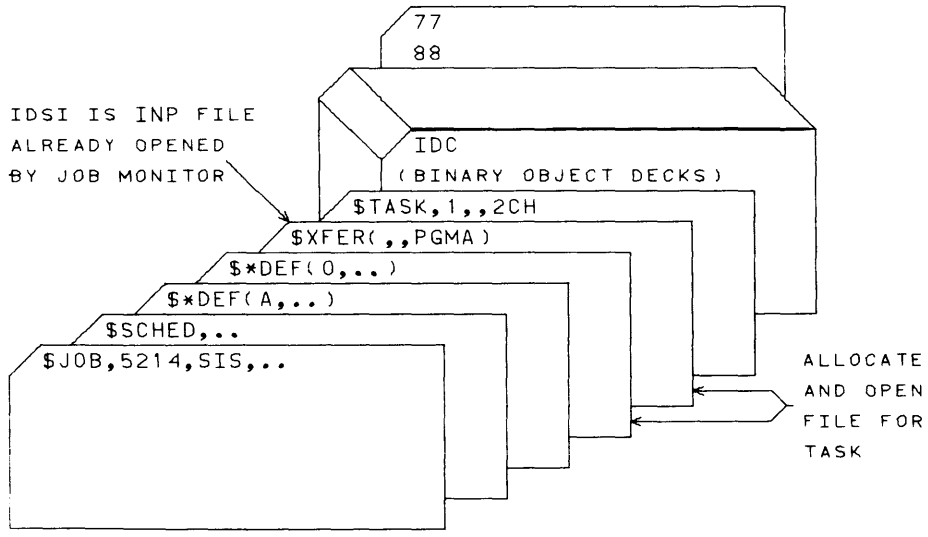
```
Rr XFER 02 (REMOVE FORM f FROM htCcEeUuuu)
```

XFER execution cannot resume until the operator responds.

PRINT FILE ON SPECIAL FORMS



GENERATE A TASK



Example:

```
LOCATION | OPERATION, MODIFIERS | ADDRESS FIELD | COMMENTS
1        8 | 10                 | 20           | 41
         | LIBM                 | XFER
         |  .                   |  .
         |  .                   |  .
         |  .                   |  .
         | XFER                 | (R,TEMP,,CHKS,,18161532,SS,,N)
```

An installation can also designate form numbers that specify changes of the format tape for the printer.

11.3 BLOCKING/ DEBLOCKING CONVENTIONS

When input and output block sizes differ, some form of blocking takes place on an XFER. Deblocking is restricted to files being transferred to the printer or to the punch when the input file is assumed to be in MASTER standard blocked format (Chapter 6).

Deblocking takes place if the specified input block size is larger than the specified output block size (n>m).

Blocking takes place if the specified input block size is smaller than the specified output block size as shown below:

1) Card reader

   The specified input block size indicates that blocking is required; 40-word reads are given, and the hardware determines whether the card is binary or BCD. The count is adjusted accordingly. The block is filled until the remaining area defined by input block size cannot hold another 40-word image.

2) Magnetic tape

   The input block size specifies the largest record appearing on the tape. XFER uses this value in the request itself, although it uses the word count of the read to determine the actual record length. A block is filled until the remaining area cannot hold another record of the specified input size.

3) Disk

   The input block size specifies the actual size to be read. The block is filled until the remaining area cannot hold another input block.

XFER blocking is useful for converting nonstandard files to standard MASTER format.

11.4 XFER ERROR CONDITIONS

Conditions detected by XFER that result in termination of the task produce the following message on the console typewriter and the codes on the job's OUT file:

D JOBi XFER 03 (SEC=xx XEC=yy)

xx System error codes (7.2)

yy XFER error codes

- 01 disk format error
- 02 XFER reached end of allocated file
- 03 file lockout condition
- 04 XFER unable to define abnormal condition
- 05 write attempted on read-only file
- 06 end-of-tape condition
- 07 irrecoverable error

3.5 RELOCATABLE LOADER

Input to the relocatable loader is binary object decks generated by compilers and assemblers according to specifications outlined in Section 10.2. This loader is called when a job requires loading of task in relocatable format. The MASTER relocatable loader automatically performs the following services:

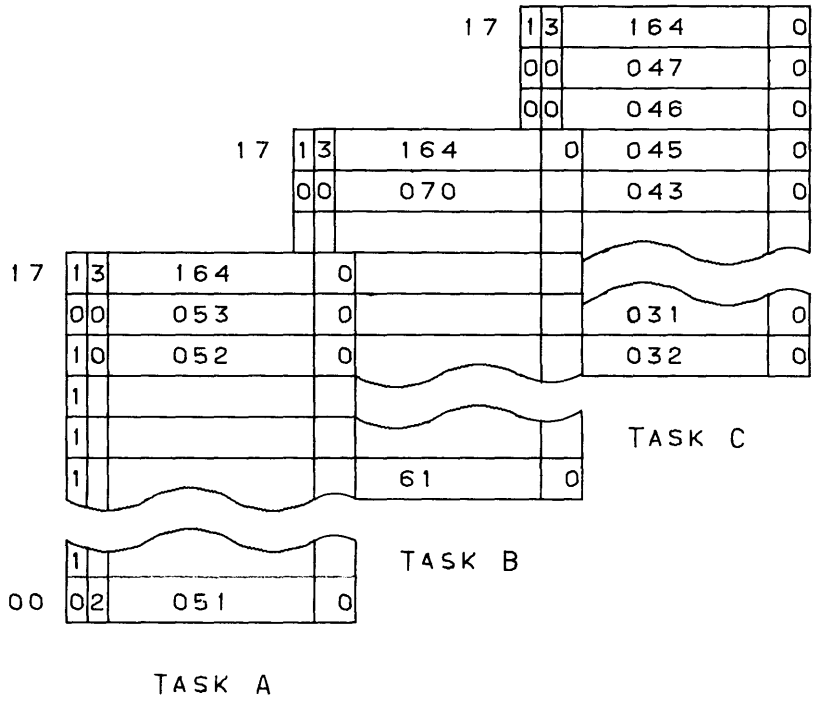
1. Loads relocatable binary information into memory from a file named in the call.
2. Links independently compiled or assembled subprograms that reference each other through symbolically named entry points. Several symbols can be equated to the same entry point.
3. Loads and links into a task any externally referenced library routines.
4. Detects and records format errors or violations of loading procedures.
5. Assigns the task's common area to chapter 2, upon request.
6. Prepares a memory map.

Upon option, the relocatable loader can be called by an RLDR control card (10.2.7). Then, the relocatable loader generates an absolute record of a task after loading and linking its relocatable subprograms, and stores the task on a user specified file.

During program loading, the loader generates a table of declared external symbols and entry points. When the last card of a task is sensed, any external symbols not matched to entry point symbols defined in the task are sought as entry points to library subprograms. The loader then loads and links the required subprograms.

The memory allocator makes physical memory available on a page basis to the program and data area as the task is being loaded and on a quarter-page basis to the common area after the entire task has been loaded. At the time, the common area is assigned to physical memory, and the memory assigned to the loader has been released to the system and may be used for common. Allocated core is zeroed when assigned.

Each task of a job will have as its last entry in its chapter-one page map the page containing the job's copy of the blocker/deblocker routines. In the example, a copy of the blocker/deblocker for JOB1 is in page 164.



Chapter One Page Maps for Tasks A, B, and C of JOB1

3.5.1 SUBPROGRAM ELEMENTS

The subprogram elements recognized by the loader are entry points, external names, and data and common blocks.

Entry Point

An entry point consists of a name and an address. The name, one to eight alphanumeric characters the first of which is alphabetic, specifies a location in a subprogram which may be referenced by another subprogram. The entry point address specifies the computer location assigned to the name. One subprogram may have any number of entry points (10.2.4).

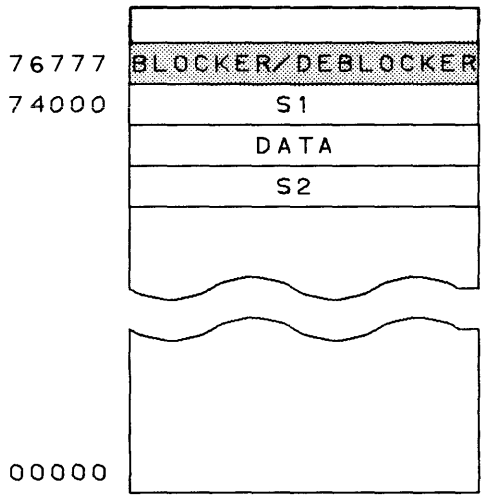
External Names

An external name, one to eight alphanumeric characters, specifies a location symbol which is external to a particular subprogram. That is, it is not a defined symbol in the subprogram in which it is declared as external, but appears as an entry point name for another subprogram. The external linkage information given with the name specifies locations referencing an external name (10.2.6).

Data Block

The data block is an area of memory that may be shared between subprograms of one task, but not between tasks. It may be preloaded with data at load time. It is assigned logical memory addresses in the same manner as a subprogram upon being encountered the first time. Thereafter, its declared length must not exceed the assigned length. Data blocks are sometimes referred to as labeled common (10.2.3).

When a subprogram does not refer to the data block, the length of its data block is zero. For a task consisting of two subprograms, S1 and S2, and a data block, logical memory assignment appears as:



Common Block

The common block is an area of memory that may be shared between subprograms of one task. It may also be shared between two-chapter tasks of a job. The common block is always assigned logical memory beginning at address zero. This block may not be preloaded with data, because it is not assigned physical memory until loading is complete. The declared length of the common block may vary from one subprogram to another; it is restricted by the amount of logical memory addressing assigned to it and the physical memory available. A common block is sometimes referred to as numbered common (10.2.3).

3.5.2 MEMORY MAP

When a task is relocated, the programmer obtains on his standard output file (OUT) a memory map of logical (chapter) memory allocated to a loaded program. The information for the map is obtained from the loader symbol table. Map headings have the following significance:

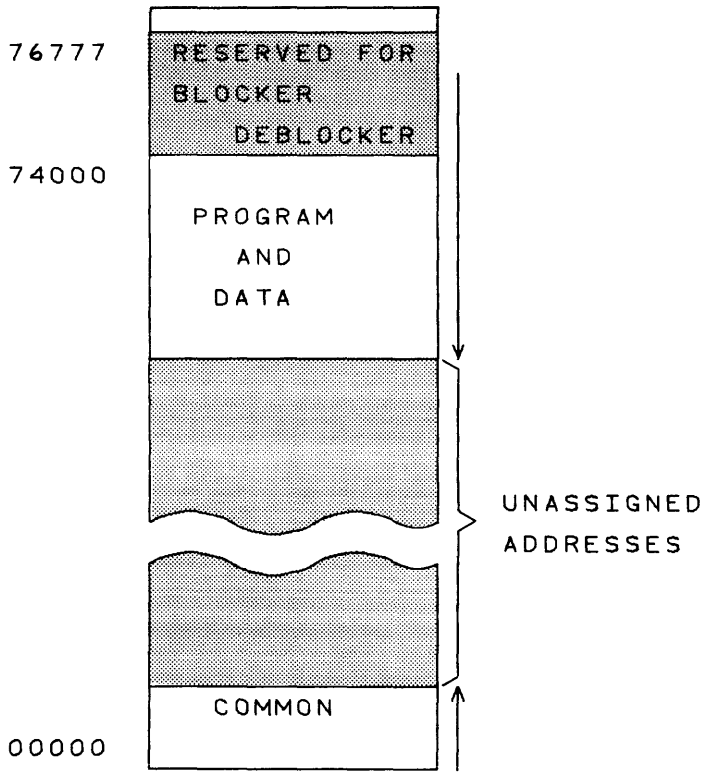
Heading	Significance
SUBP	Below SUBP are listed absolute addresses, and names of subprograms as stated on the IDC card. It does not show addresses reserved for the blocker/deblocker.
ENTR	Below ENTR are listed the entry point symbols as taken from EPT cards and the absolute address of each entry point declared in any subprogram loaded for the run. It should always show an entry point for UIC.
COMM	Under COMM is given the name of the common area and its length. The common area derives its name from the chapter to which it is assigned. That is, name can be 1 or 2. NONE indicates the task has no common area of its own.
DATA	Under DATA is given the absolute addresses and names of the data areas. NONE indicates that the task has no data areas.

Example:

```
MEMORY MAP
   SUBP
   73025    NONSDTST    72666    SDTST

   ENTR
   73030    TEST1       72727    TEST2    72666    UIC

   COMM
        NONE

   DATA
        NONE
```

3.5.3 ONE-CHAPTER TASK

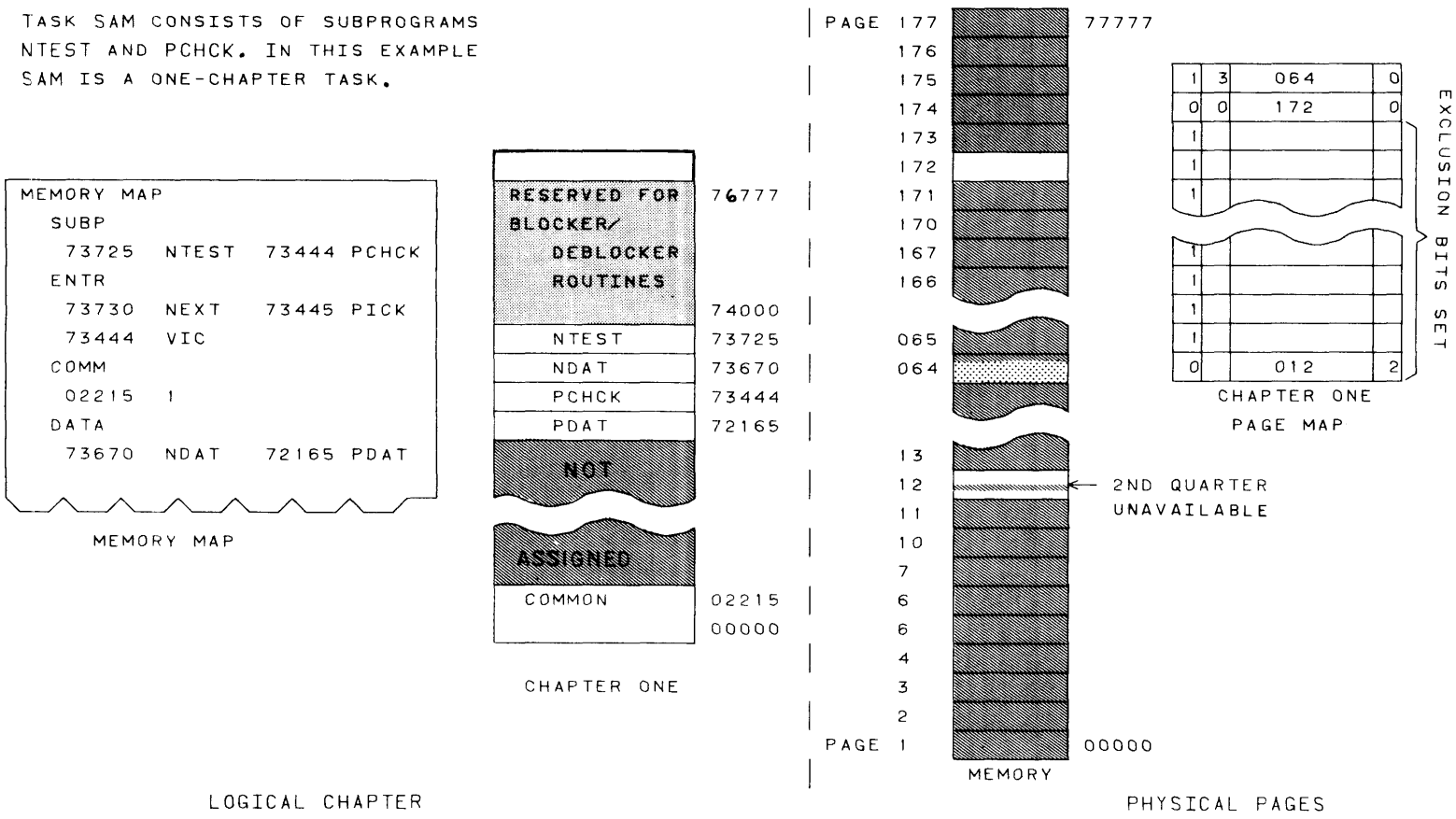
For a one-chapter task, the loader reserves addresses 76777 down to 74000 for the job's copy of the blocker/deblocker. It assigns addresses from 73777 downward to subprogram and data blocks as they are loaded and linked according to requirements indicated on Identification cards (IDC) and Block Common Table cards (BCT). Common addresses it assigns upward from 00000. At the end of loading, an address gap will usually occur between the top of common and the bottom of the program and data addresses. This gap is <u>not</u> assigned physical memory and the core it represents should not be scheduled by the user when he estimates core requirements for a job.



ADDRESSING OF ONE-CHAPTER TASK

EXAMPLE

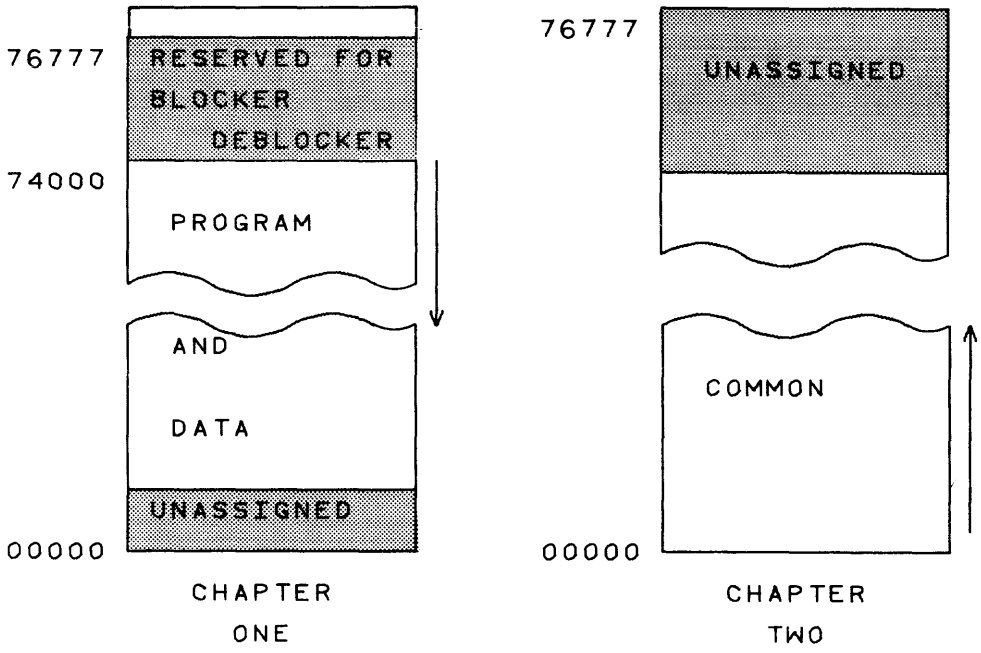
TASK SAM CONSISTS OF SUBPROGRAMS NTEST AND PCHCK. IN THIS EXAMPLE SAM IS A ONE-CHAPTER TASK.



ONE CHAPTER TASK

3.5.4 TWO-CHAPTER TASK

For a two-chapter task with its common in Chapter Two, the relocatable loader reserves addresses 76777 down to 74000 for the job's copy of the blocker/deblocker and assigns addresses 73777 downward to program and data as it does for a one-chapter task. Program and data can extend down to 00000, however, in which case the entire 32K chapter is assigned and will require 16 physical pages. Common addresses the loader maintains separately; it assigns addresses upward from 00000 to a maximum of 77776 in Chapter Two. Usually, parts of both chapters remain unassigned and the core represented will not have to be included in the core estimate for the job.



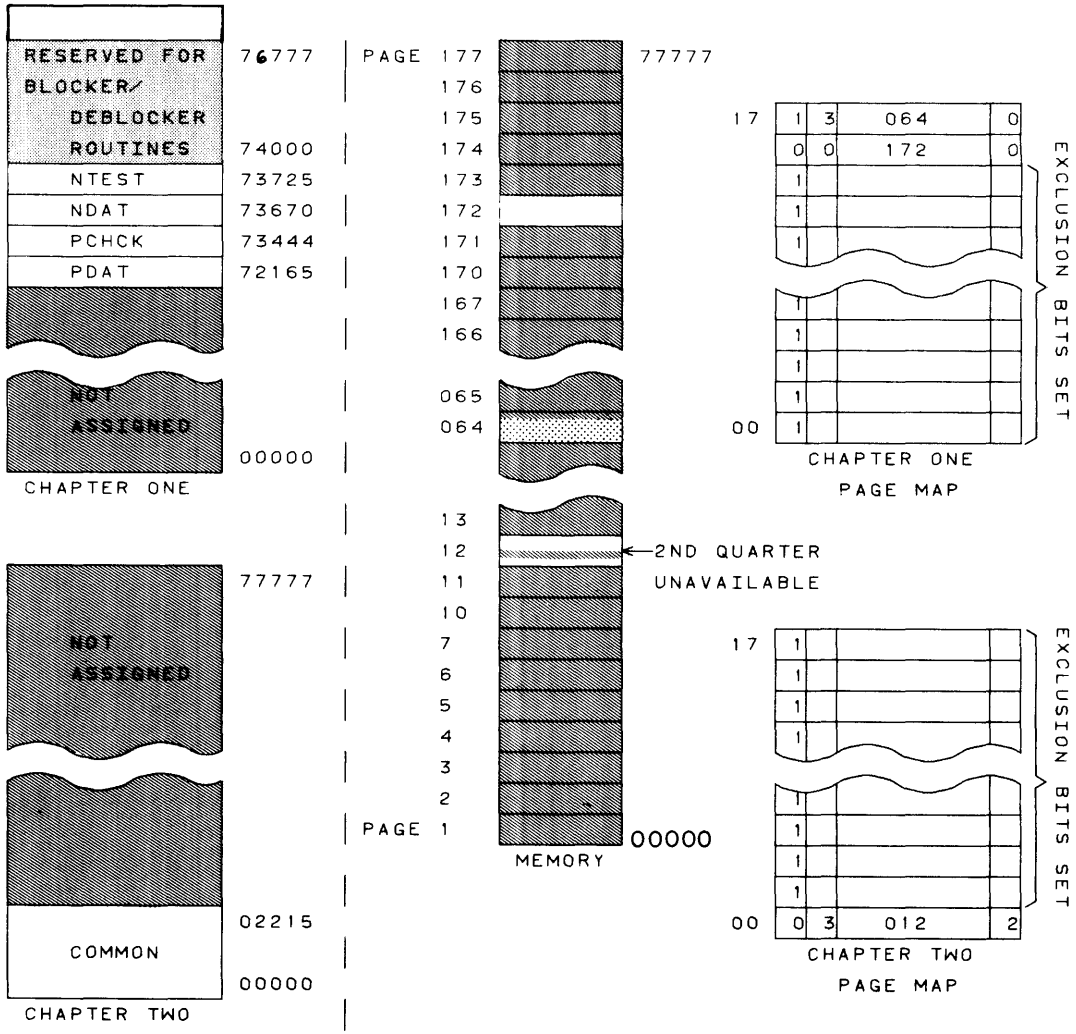
When a two-chapter task has its own common in Chapter One. Loading is the same as for a one-chapter task.

EXAMPLE

TASK SAM CONSISTS OF SUBPROGRAMS NTEST AND PCHCK. IN THIS EXAMPLE SAM IS A TWO-CHAPTER TASK.

```
MEMORY MAP
  SUBP
   73725  NTEST   73444 PCHCK
  ENTR
   73730  NEXT    73445 PICK
   73444  UIC
  COMM
   02215  2
  DATA
   73670  NDAT    72165 PDAT
```

MEMORY MAP



TWO-CHAPTER TASK

APPENDIX SECTION

HINTS AND CAUTIONS A

1. Schedule only core and equipment required by the job.

2. When using the XFER routine remember scratch files are released upon job termination.

3. Whenever possible, avoid using DIRECT cards.

4. Remember the console typewriter is shared; over-using it could radically slow down a job.

5. Use XFER for special forms.

6. The printer driver uses pre-print spacing rather than post-print spacing used by previous 3200 systems. That is, the paper is spaced before a line is printed rather than after the line is printed. This allows overprinting of the previous line.

7. The offset function of the card punch is used by the background output routine to signal job end and compare errors. Writing on the PUN file is implemented with mass storage functions - not punch functions.

8. Character I/O is not allowed on any mass storage devices.

9. Program overlays are prepared and processed differently by MASTER than by other CONTROL DATA operating systems (Chapter 4).

10. Using blocking and deblocking routines for data transmission is the easiest and most efficient method of I/O for mass storage.

11. A task that is to be repeatedly called within a job should be self-initializing because, even though it may request release on return, a queued caller may be connected to it before it is released.

12. When programming the parameter receiving area for a task to be called by a control card, allow for the parentheses enclosing the parameter string on the control card because they are passed with the string.

13. A Class B mass storage file opened partially and in segments should not have blocks that cross from a segment on one device to a segment on the next device.

MASS STORAGE

The following is general information concerning mass storage files.

MASS STORAGE GENEOLOGY

CLASS A DEVICES

Permanent On-Line

User Files (A life expectancy of longer than one job)

1. ALLOCATE/OPEN/CLOSE/EXPAND/ MODIFY/RELEASE by *DEF only
2. Files on Class A store may be opened in:
   a. Normal mode only
3. Drives needed for Class A devices (packs) are not scheduled for the job. The device label contains a bit set to indicate that these devices are not to be removed.
4. The allocation algorithm assigns Class A storage to a file by finding the largest area in which the file will fit; or it works from the largest available area to the smallest in building segments of the file (minimizes number of segments needed for file). (See Note 1.)

CLASS B DEVICES

Non-Permanent On-Line

User Files (A life expectancy of longer than one job)

1. ALLOCATE/OPEN/CLOSE/EXPAND/ MODIFY/RELEASE by *DEF only
2. Files on Class B store may be opened in:
   a. Normal mode
   b. Segmented mode
3. Drives needed for Class B devices (packs) must be scheduled with the job using them.
4. The allocation algorithm assigns as much of the file to one device as possible before continuing on the next device (minimizes number of drives needed for file). (See Note 1.)

Note 1: A file cannot be allocated on different classes of devices.

System Files

MSIO Files:

*MSD (Mass Storage Directory)
*FLD (File Label Directory)
*FID (File Identifier Directory)
} (See Note 2)

MASTER Files:

*LIB (System Library File)
*DIR (Library Directory File)
} (See Note 2)

Standard Files

INP, OUT, PUN } created from a pool of segments, and managed by the operating system through system OCARE

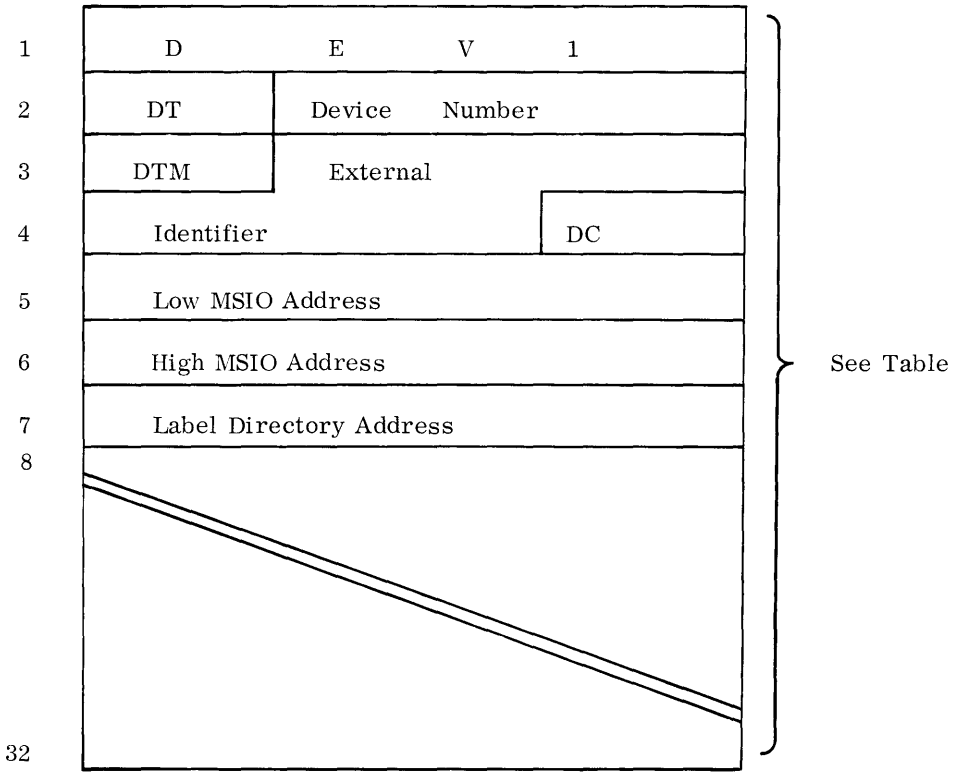
SCR } created from a pool of segments, and managed by the user through system OCARE

MASS STORAGE LIMITS

Maximum Block Length (Characters)	131,071
Maximum Number of Blocks Per File	8,388,607
Maximum Number of Physical Records (i.e., tracks or sectors) per Block	4,095
Maximum Number of Physical Records (i.e., tracks or sectors) per File	8,388,607

Note 2: All system files are opened at autoload time, and remain open while MASTER is in operation. System files may be referenced by operating system tasks only.

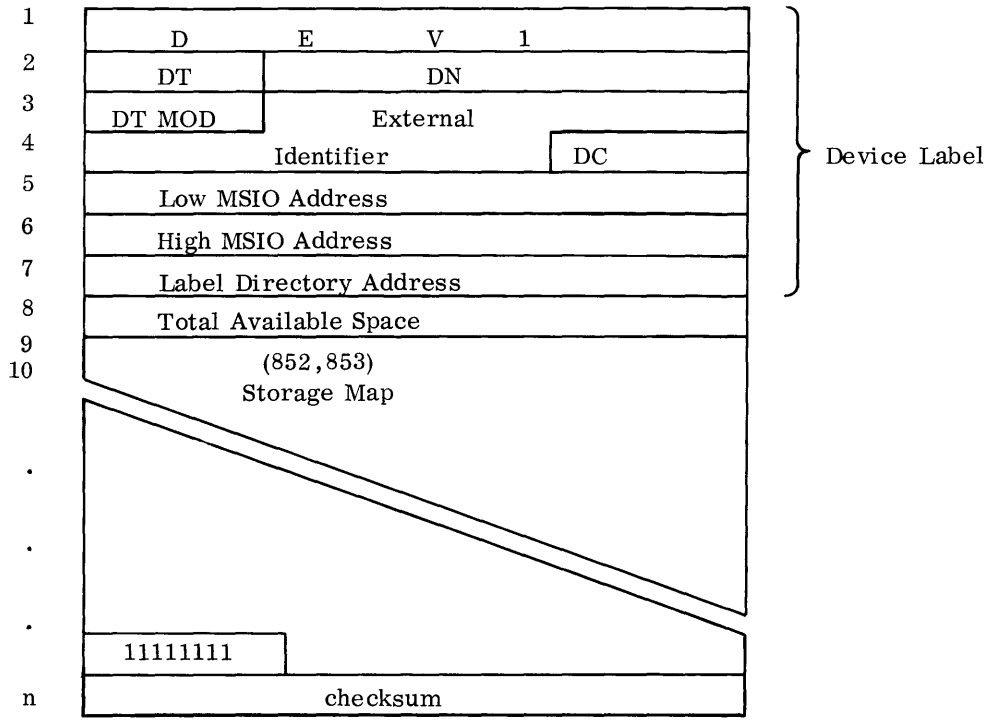
MASS STORAGE DEVICE LABEL



DEVICE LABEL FIELDS

Field Name	Number of Characters	Description
DEV1	4	A standard 4-character identifier which is prefixed to device labels.
DT	1	A 6-bit code to represent device type. DT = octal 40 for 852 disk packs.
DN	3	An 18-bit device number that matches an external number on each device.
DTM	1	A 6-bit device type modifier. The only values defined for 852 disk packs are: XXXXX0 this device is recorded in track mode. XXXXX1 this device is recorded in sector mode.
External Identifier	6	Any alphanumeric characters. This field corresponds to an external identifier on each device.
DC	1	A 6-bit device class identifier. XXXXX0 this is a Class B device. XXXXX1 this is a Class A device.
Low MSIO Address	4	The lowest hardware address (binary) that can be accessed by MSIO.
High MSIO Address	4	The highest hardware address (binary) that can be accessed by MSIO.
Directory Address	4	The binary hardware address at which the file label directory is stored. This is the low address of the LABELFILE and is present only on the device which contains the label directory.

ENTRY IN MSD

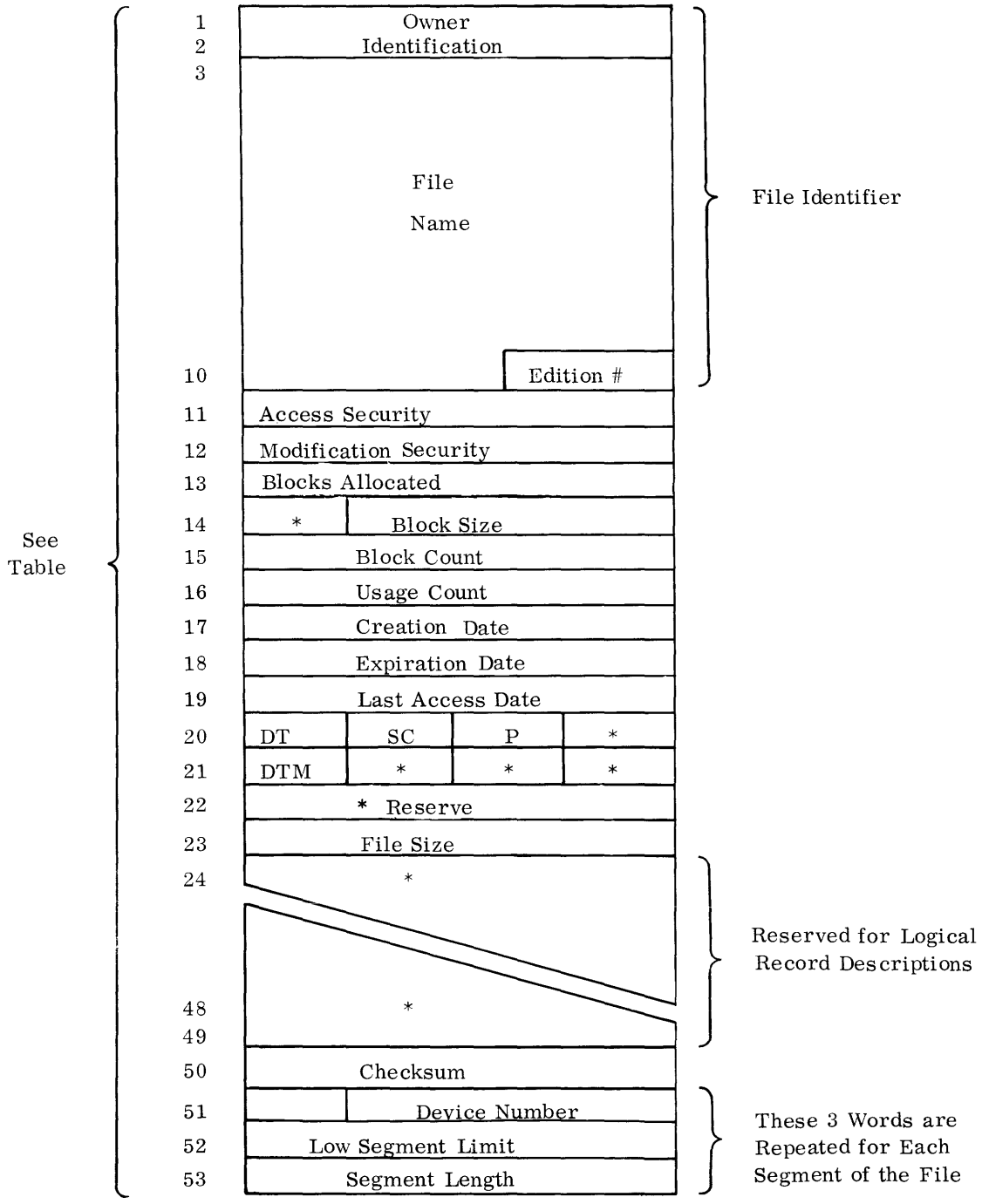


Word 8 contains the number of unassigned tracks. Words 9-n -1 contain a bit mapping of the tracks on the device and represent tracks zero through 999 of the disk. A bit set to one indicates the corresponding track is assigned. A bit set to zero indicates the corresponding track is available. The correspondence between bits and tracks is:

Bit Number	Word Number	Track Number
0	9	0
1	9	1
- - -	- - -	- - -
23	9	23
0	10	24
1	10	25
- - -	- - -	- - -
i	j	24(j-9)+i
- - -	- - -	- - -
14	50	998
15	50	999

Bits 16 through 23 of word 50 are set to 1's.

MASS STORAGE FILE LABEL



* Denotes Reserve

FILE LABEL FIELDS

Field Name	Number of Characters	Description
File Identifier	40	Uniquely identifies a file in the label directory. The standard identifier consists of: Owner Identification - 8 characters File Name - 30 characters Edition Number - 2 characters The 40-character field may be divided otherwise at installation option.
Access Security	4	This field is supplied when the file is allocated; it must be supplied for each succeeding OPEN request.
Modification Security	4	This field is supplied when the file is allocated; it must be supplied for each RELEASE, EXPAND, and MODIFY request.
Block Size	3	This field contains, as a binary integer, the number of 6-bit characters in each record block ($0 < \text{Block Size} < 131072$).
Block Count	4	This field contains, as a binary integer, the number of the highest block written. If the file is processed sequentially, this corresponds to the number of blocks written into the file ($0 \leq \text{Block Count} < 2^{23}$).
Usage Count	4	Binary count of the number of times the file has been opened.
Creation Date	4	This date is supplied by the I/O system when the file is allocated; it is stored as a binary integer in the form yymmdd.
Expiration Date	4	This date is supplied by the user when the file is allocated; it is stored as a binary integer in the form yymmdd. This field determines when a file may be deleted.
Last Access Date	4	This date is supplied by the I/O system each time the file is opened or changed; it is stored as a binary integer in the form yymmdd.

FILE LABEL FIELDS

Field Name	Number of Characters	Description
DT (Device Type)	1	6-bit code to indicate the type of mass storage device containing the file. Octal 40 indicates an 852 disk pack.
SC (Segment Count)	1	Contains, as a binary integer, the number of segments in the file ($0 < SC < 64$).
P (Protection)	1	Contains protection flags for use by the I/O system. The only values currently defined are: 0 file may be read or written. 1 file may not be written.
DTM (Device Type Modifier)	1	6-bit code which provides further device information. The only values currently defined are: XXXXX0 Track Mode XXXXX1 Sector Mode
Blocks Allocated	4	This field contains as a binary integer the number of blocks allocated to the file.
File Size	4	Contains, as a binary integer, the number of allocatable units (tracks) assigned to the file ($0 < \text{File Size} < 2^{23}$).
Checksum	4	24-bit binary checksum of the entire label. This field is checked by the I/O system to detect accidental modification of the label.
Device Number	3	Number of the device on which this file segment is stored. This field is checked against the device label to assure that proper packs are mounted.
Low Segment Limit	4	Binary hardware address at which this file segment begins.
Segment Length	4	Number of allocatable units (tracks) in this segment.
* (Reserved)	117	These fields are reserved for future use by the I/O system.

MAGNETIC TAPE

GENERAL HEADER LABELS

All header label records are 80 characters (480 bits) long, and are unblocked. They are recorded in even parity at the same density as the remainder of the data file. Header records are separated from succeeding data records by an inter-record gap only. Header label record fields are defined below; they are positioned as shown within the physical record. Values that may be used within these fields are also specified.

Field Name	Starting Character Position	Length in Characters	(If used) Defined Values BCD Characters Only	Function
Density†	1	1	2, 5, 8	Specifies density of recording file
Header Label Identifier	2	2	( )	Identifies record as header label record
Logical Unit No.	4	2	As applicable	Specifies logical unit to which file is assigned
Retention Code	6	3	000-999	Specifies, in days, the retention period of the file
File Name†	9	14	Any combination of legal BCD characters	Identifies the file
Reel Number†	23	2	01-99	Identifies sequence of reels for multi-reel files
Date Written	25	6	Any legal numeric date, expressed as mmddyy	Identifies date written; used with retention period to determine release date of file
Edition Number†	31	2	00-99 or blank	Identifies a single file set
User Supplied Information	33	48	Any combination of legal BCD characters	User comments field

† Used by *DEF open unit function (7.1).

ERROR RECOVERY PROCEDURES

The following tabulates standard hardware errors and error recovery procedures used by MASTER EXEC on mass storage devices and magnetic tape units.

Device	Error Function	Recovery Procedure
852 Disk	Read/Compare Parity Error Lost data Non-operable status; COMPARE after WRITE errors, transmission parity errors, hardware rejects	1. Execute a direct seek. 2. Execute function request and follow with an error detect. 3. If error persists, repeat steps 1 and 2 four times. 4. Execute a normal seek. 5. Execute a direct seek. 6. Execute function request and follow with an error detect. 7. If error persists, repeat steps 5 and 6 four times. 8. If error persists, it is declared irrecoverable.
853, 854, 813, 814 Disk, 863 Drum	Read/Compare Parity Error Lost data Non-operable status; COMPARE after WRITE errors, transmission parity errors, hardware rejects.	1. Execute a load address. 2. Execute function request and follow with an error detect. 3. If error persists, repeat steps 1 and 2 nine more times. 4. If error persists it is declared irrecoverable.
603, 604, 606, or 607 Magnetic Tapes	Parity errors, lost data, non-operable status, transmission parity error and hardware rejects during a forward read function	1. Execute a backspace. 2. Execute a read followed by an error detect. 3. If the error persists, repeat steps 1 and 2 three more times. 4. If error persists backspace three records (to insure that error record passes under the tape cleaner) check for load point after each record. 5. Execute up to three reads (depending on load point) non-stop if possible, transmitting the record contents only during the last read. 6. Check for a read error of the last transmission. 7. If error persists, repeat steps 4 through 6 up to four more times. 8. If the error persists, it is declared irrecoverable.

Device	Error Function	Recovery Procedure
603, 604, 606, or 607 Magnetic Tapes	Any of the preceding errors during a reverse read function	1. Execute a clear reverse read. 2. Execute a forward read followed by an error detect. 3. Execute a backspace and a set reverse read. 4. Insert the order of the words that make up the record in core. 5. If error persists, repeat steps 1-4 up to four more times. 6. If error persists it is declared irrecoverable.
	Any of the preceding errors during a write function	1. Execute the sequence of commands backspace, write, and detect two times. 2. If error persists, execute a backspace and SKIP BAD SPOT command. 3. Execute a write, followed by an error detect. 4. If error persists, repeat steps 1 through 3. 5. If error persists, repeat step 1. 6. If error persists, it is declared irrecoverable.

SYSTEM-TO-OPERATOR MESSAGES C

MASTER communicates with the operator by means of the console typewriter, and most system messages are in standard format:

xy JOB i ssss nnn (des)

- xy Message type:
  - D Destructive
  - I Informative
  - A Operator action required, no response
  - Rr Operator decision and response required; r (0-9) is the message number assigned by the system (appears optionally)
- i Job identifier taken from JOB card
- ssss Name of task; 4 characters maximum
- nnn Message number relative to the task
- (des) Optional description enclosed in parentheses; maximum 65 characters

The System Executive and all tasks associated with MASTER may type non-destructive messages. MASTER types a job abort message when a system error or malfunction occurs external to the user's task, and job termination results.

File Logging

When *DEF opens a file, it logs hardware assignments in the following format:

Rr JOB i *DEF LOGGING

dsi = hhhh, yyyyyy, CcEeUuuu, WR
yyyyyy, CcEeUuuu,
etc. (One line for each device not currently on line required for the file)

END LOG i

- i Job identifier taken from JOB card
- dsi Data set identifier
- hhhh Hardware type

yyyyyy	Device number or device identifier (blank for scratch tapes and unit record)
CcEeUuuu	Channel, equipment, and unit
WR	Insert write wring of tape for which output is requested; otherwise omitted from message.

The operator must type a response confirming that he has readied the file as requested. This response may be Rr, OK if the requested devices are ready, or Rr, NO if the request cannot be honored. The latter response causes a reject of the user's request.

Begin and Terminate Messages

B i	is typed when job i is initiated
T i	is typed when job i is terminated

System Messages

The following table lists all job abort and non-destructive messages typed on the console typewriter. Symbols in the messages are defined as follows:

c	channel number
e	equipment number
ht	hardware type
uuu	unit number

MESSAGE TO OPERATOR						
Type	Job	Task	nnn	Optional Message	Cause	Operator action
O		EXEC	00	(MEMORY PARITY ERROR)	System incurred a memory parity error.	Restart the computer.
I		EXEC	71	(ILL R-T INT xxxx)	Interrupt within EXEC was not real-time, MANUAL, or associated processor.	
I		EXEC	72	(ASSOC PROC INT)	Associated processor interrupt when no processor was present.	
A		EXEC	101	(ht CcEeUuuu CR ESxxxx ISxxxx)	Connect reject ESxxxx - 12 bit external status ISxxxx - 12 bit internal status	Dial in the correct equipment and unit. If this cannot be done or is already done, only the real-time program can continue. System must be restarted.
I		EXEC	102	(ht CcEeUuuu SRyyyy ESxxxx ISxxxx)	Select reject SRyyyy - 12 bit select code ESxxxx - 12 bit external status ISxxxx - 12 bit internal status	
A		EXEC	103	(ht CcEeUuuu RDY)	Hardware is not in ready status.	Place the unit in ready status.
I		EXEC	104	(ht CcEeUuuu DWN)	Hardware is inoperable. System or operator has removed the unit from the available list.	

MESSAGE TO OPERATOR					Cause	Operator action
Type	Job	Task	nnn	Optional Message		
I		EXEC	105	(ht CcEeUuuu FLO FI = dsi JI = i)	Reference was made to a file on an inoperable unit. JI i - identifier from JOB card of job making request. FI dsi- data set identifier of file to which request was made.	
I		EXEC	111	(UNDEF INT xxxx)	Interrupt was not expected by the I/O system. xxxx - interrupt code. 00LCh† External interrupt 010Ch I/O channel interrupt 0110 Real-time clock interrupt 0111 Arithmetic overflow fault 0112 Divide fault 0113 Exponent overflow fault 0114 BCD fault 0115 Search/move interrupt 0116 Manual interrupt 0117 Associated processor interrupt	
I		EXEC	112	(PAR ERR CHAN c)	Number of consecutive parity errors on channel c exceeded the installation defined maximum.	

† L = line 0-7 and Ch = channel designator, 0-7

MESSAGE TO OPERATOR						
Type	Job	Task	nnn	Optional Message	Cause	Operator Action
I		EXEC	151	(MT CcEeUuuu F/I ESxxxx FI = dsi JI = i)	System recovery procedure cannot recover from error on magnetic tape. F - Error occurred on first try I - System has not re-recovered	
A		EXEC	161	(CR CcEeUuuu CMP ERR)	Read compare error on last card read from card reader.	Remove the last card from the output stacker and place it as the first card in the input stacker; press the READY button on the card reader to re-read card.
I		EXEC	162	(CP CcEeUuuu CMP ERR)	Punch compare error on card punch. Punch background routine automatically offsets the card in error and the following card and repunches both.	
I		EXEC	171	(DP CcEeUuuu F/I A=xxxxx ESxxxx ECxx DRx FI=dsi JI=i)	Error on mass storage device. F - Error occurred on first try I - System has not recovered A=xxxxx-Address at which error occurred ESxxxx-12-bit external status of unit (Control Data 3000 Series Peripheral Equipment Reference Manual)	

171 Continued on next page

MESSAGE TO OPERATOR					Cause	Operator action
Type	Job	Task	nnn	Optional Message		
I		EXEC	171 (Cont'd)	(DP CcEeUuuu $^{F}_{I}$ A=xxxxx ESxxxx ECxx DRx FI=dsi JI=i)	ECxx-Error code (See 7.2.) DRx - Driver code (MASTER Operator's Guide) FI=dsi - File Identifier JI=i - JOB identifier	
I	JOB i	*BKI	01		A parameter on DIRECT card following job i is not in correct form. Parameter is ignored and normal processing continues.	
D	JOB i	*BKI	02		Source deck of job i is too big for mass storage. The job is not run; it may be resubmitted as a DIRECT job.	
I	JOB i	*BKI	03		Card following source deck of job i was not a JOB or DIRECT. Cards are passed up to next JOB or DIRECT card.	
A	JOB i	*BKO	01		Irrecoverable error when reading JOB i punch file from mass storage. An error card is inserted in job's punch deck.	Mark deck as having an error card in it.

MESSAGE TO OPERATOR					Cause	Operator action
Type	Job	Task	nnn	Optional Message		
D	JOB i	*DEF	56	(MAXIMUM FILE COUNT EXCEEDED)	JOB i was terminated because it requested that a file be allocated when the system had no more room in the file label directory.	
I	JOB i	*DEF	63	(FILE SECURITY CODE ERROR)	A task in JOB i made request to *DEF giving incorrect security code.	
D	JOB i	*DEF	77	(I/O ERROR ON *LAB/ *MSD/*IDF)	JOB i was terminated because *DEF encountered irrecoverable errors on label directory file, mass storage directory file, or label id file while processing a request.	
D	JOB i	*EST	000	(*LIB, NAME)	JOB i was terminated because of irrecoverable read errors on library file during loading task name.	
D	JOB i	*EST	001	(*LIB, NAME)	JOB i was terminated because it attempted to call task name from library when the task was not defined in the library directory.	

MESSAGE TO OPERATOR					Cause	Operator action
Type	Job	Task	nnn	Optional Message		
D	JOB i	*EST	005	(*LIB, CODE)	JOB i was terminated because a locate error occurred when trying to load from library. CODE is the error code returned to *EST from the I/O system. (See MS status codes)	
D	JOB x	*SCH	00		Job card is missing or unreadable; x is meaningless.	
D	JOB i	*SCH	01		No account number on job card. JOB i will not be run.	
D	JOB i	*SCH	02		Illegal separator on the job card. JOB i will not be run.	
D	JOB x	*SCH	03		No job identifier on job card; x is meaningless. Job will not be run.	
D	JOB i	*SCH	04		Time field on job card is not in correct form. JOB i will not be run.	
D	JOB i	*SCH	05		Segments requested by SCR parameter, for mass storage scratch, exceed the system capacity. JOB i will not be run.	

MESSAGE TO OPERATOR					Cause	Operator action
Type	Job	Task	nnn	Optional Message		
D	JOB i	*SCH	06		Space requested by CORE parameter for the job exceeds system capacity. JOB i will not be run.	
D	JOB i	*SCH	07		Peripheral requirements on the SCHED card exceed the system capacity. JOB i will not be run.	
D	JOB i	*SCH	08		The JOB card OUT file limit exceeds the maximum mass storage segment limit for a file.	
D	JOB i	*SCH	09		The JOB card PUN file limit exceeds the maximum mass storage segment limit for a file.	
Rr		XFER	01	(MOUNT FORM f ON ht CcEeUuuu)		1. Mount requested form. 2. Press MANUAL INTERRUPT. 3. Type Rr, OK on console. 4. Press FINISH, or if operator does not mount form, 1. Press MANUAL INTERRUPT. 2. Type Rr, NO. 3. Press FINISH.
Rr		XFER	02	(REMOVE FORM f FROM ht CcEeUuuu)		1. Remove requested form. 2. Press MANUAL INTERRUPT. 3. Type Rr, OK. 4. Press FINISH.
D	JOB i	XFER	03	(SEC=xx XEC=y)	XFER requested termination xx - System code (7.2) y - XFER code (11.4)	

MESSAGE TO OPERATOR						
Type	Job	Task	nnn	Optional Message	Cause	Operator action
R		SINT	007	(DATE)	MASTER date initialization	1. Type mmddyy mm = month 2. Press FINISH dd = day yy = year
R		SINT	008	(TIME)	MASTER time initialization	1. Type hhmmss hh = hour 2. Press FINISH mm = minute
R		SINT	009	(SET)	Enable operator to change system parameters	To change parameters see MASTER Operators Guide; otherwise press FINISH when no parameters are to be altered.
R		SINT	015	(EDITN)	Enable operator to select edition of MASTER library	1. Type ee ee = edition number 2. Press FINISH of MASTER library

FATAL CONDITIONS DETECTED BY EXEC D

When the System Executive encounters a fatal condition, it includes an involuntary job abort code (IAC) in the standard output listing. These codes are listed below:

Code	Description
E00	Memory parity error.
E01	Task has attempted to perform a request restricted to an operating system task.
E02	Job contains an illegal function code.
E03	Job contains an illegal request code.
E04	Illegal write into protected memory has been requested or reference has been made to excluded memory.
E05	Operator has terminated the job.
E09	Illegal blocking or deblocking request.
E10	Task not in the operating system has attempted a call with abandon specified.
E11	Parameter receiving area of callee is out of bounds.
E12	Parameter receiving area of callee is too small to receive the caller's parameter string.
E13	One-chapter task has attempted to pass common.
E14	Two-chapter task has attempted to pass common to a one-chapter task.
E15	Both passing of parameters and transfer of call end exist on a return request.
E16	Parameter receiving area of caller is too small to receive callee's parameter string.
E17	DWAIT request did not include parameters.
E18	Wait status was requested while files were reserved for a task.
E19	Backgrounder flag table is full.
E20	LIMIT time for this job has elapsed.
E21	JOB time limit has been exceeded.
E22	Interrupt address on the SELECT request is illegal.
E23	Error in COPYCOM request; the task already has Chapter One common.
E24	Error in COPYCOM request; the task program plus Chapter Two exceeds 30K.
E25	Buffer is out of bounds on COPYDIR request.

Code	Description
E50	Dsi in DWAITIO request is not defined.
E51	Dsi in I/O request is not defined.
E52	Dsi in I/O request is restricted to the system.
E53	Illegal file disposition request (BYPASS, RESERVE, or RELRESV)
E54	There has been a request to a file which is in the process of being closed.
E55	Bounds error on the status area of I/O call.

GLOSSARY

TERMS

Definitions preceded by an asterisk are from the proposed "Vocabulary for Information Processing" of the American Standards Association, which has granted permission to reprint them. These definitions have not been adopted as standard and are subject to change, modification, and withdrawal in part or whole by the ASA.

ABNORMAL DUMP
A dump occurring immediately following abnormal termination of a program.

ABORT
To terminate a program when a condition (hardware or software) exists from which the program or computer cannot recover.

ABSOLUTE BINARY PROGRAM
A program that must be loaded according to specific logical addresses.

ABSOLUTE CODE
A code using absolute operators and addresses, i.e., a code using machine language.

ALLOCATE
To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

ALPHAMERIC
See alphanumeric.

ALPHANUMERIC
Pertaining to the character set that contains alphabetic letters, numerical digits, and special characters which are usually machine processable.

ASSEMBLE
To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic addresses.

ASSEMBLER
A computer program that generates machine instructions from symbolic input data through translating symbolic-operation coding into computer operating instructions, assigning locations in storage for successive instructions, or computing absolute addresses from symbolic addresses. An assembler generates machine instructions from symbolic codes and produces as output nearly the same number of instructions or constants as were defined in the input.

BINARY
A characteristic property, or condition having two alternatives; a numbering system based on 2 rather than 10 and using only 0 and 1.

BLOCK
Consecutive machine words or characters considered or transferred as a unit, particularly applicable to input and output.

BLOCKING
Combining of two or more numbers (records) into one block to reduce the number of physical operations.

BLOCK LENGTH
Number of records, words, or characters in one block.

BOOTSTRAP
A technique or device that brings itself into a desired state through its own action. A loading routine the first few instructions of which bring the rest of the routine into core memory. This usually involves using the AUTOLOAD key on the console.

BUFFER
A magnetic core buffer external to core memory to compensate for speed differences between peripheral devices and the processor. Operations can then be overlapped with all devices operating simultaneously at rated speeds. Buffering eliminates the need for more expensive, multiple I/O channels, and reduces programming having complex I/O timing considerations.

BUFFERING
Overlapping execution of one or more I/O tasks with the program task that called them.

*CALL
To transfer control to a specified closed subroutine or program task.

CALLEE
The task called by a caller.

CALLER
A task that calls another.

CARD COLUMN
A vertical line of punching positions on a punch card.

CARD IMAGE
A representation in storage of the holes punched in a card such that holes are represented by one's and unpunched spaces are represented by zeros. In machine language, a duplication of the data on a punched card.

CARD ROW
A horizontal line of punching positions on a punched card.

CHAPTER

32K consecutive logical addresses; the addresses that can be referenced through one state in the page file.

COMMON AREA

An area of memory that may be shared between subprograms of one task or between two-chapter tasks of the same job. A two-chapter task may have its own Chapter One common through which its subprograms communicate, and a Chapter Two common through which two or more tasks communicate.

COMPILER

A program which translates a programming language such as FORTRAN or COBOL into an assembly language and, often, into machine language. A compiler may generate many machine instructions for a single symbolic statement.

COPY

To transfer data to a new location within a computer system without altering the original data.

COPIABLE TASK

A program task of which more than one copy may exist in memory concurrently, each copy receiving requests from a separate job.

DATA AREA

An area of memory that may be prestored with data at load time and shared between subprograms of one task, but not between tasks.

DATA SET IDENTIFIER

One to four characters used to identify a file.

DECK

A collection of punched cards that have a definite service or purpose.

DRIVER

A program that controls the use of a peripheral device.

DUMP

To copy the contents of all or part of a storage device, usually from internal storage into external storage; the process of performing the above; or the resulting document.

END-OF-FILE

Termination or point of completion of data.

END-OF-FILE INDICATOR

A signal supplied by an input or output unit that makes an end-of-file condition known to the routine or operator controlling the device.

END-OF-FILE MARK OR CONDITION
A code or condition which signals that the last record of a file has been read.

EQUIPMENT
An interface between a data channel and a unit.

ERROR
Any deviation of a computed or a measured quantity from the theoretically correct value.

ESTABLISHMENT
The process of locating, loading, and preparing a program task for execution.

EXECUTE
To carry out an instruction or perform a routine.

EXTERNAL INTERRUPT
An interrupt occurring as a result of conditions within peripheral devices or their immediate interfaces.

Note: Interrupts occurring as a result of conditions within a data channel are classified external or internal in keeping with specifications set forth in individual hardware system reference manuals.

FAULT
1. A physical condition that causes a device, a component, or an element to fail to perform in a required manner, e.g., a short circuit, a broken wire, an intermittent connection, synonymous with malfunction.
2. An operation whose results exceed the capacity of one or more registers and which is detected by the hardware.

FIELD
In a record, a specified area used for a particular category of data, e.g., a group of card columns used to represent a wage rate or a set of bit locations in a computer word used to express the address of the operand.

FILE
1. A collection of related records treated as a unit, e.g., in inventory control, one line of an invoice forms an item, a complete invoice forms a record, and the complete set of such records forms a file.
2. A peripheral device uniquely identified by a data set identifier used by a computing system for the purpose of storing data.

FLAG
1. Any of various types of indicators used for identification, e.g., a wordmark.
2. A character or bit that signals the occurrence of some condition, such as the end of a word.
3. An indicator (program or hardware initiated) used frequently to tell some later part of a program that some condition occurred earlier.
4. To generate a flag (1,2,3).

FLOW
A general term to indicate a sequence of events.

INITIALIZE
To set counters, switches, and addresses to zero or some other starting value at the beginning of or at prescribed points in a program.

INITIATION
The process of activating a previously scheduled job when its requirements can be met.

INPUT
Information or data transferred from an external storage device into computer memory.

INPUT/OUTPUT
The bidirectional transmission of information to or from computer memory to or from peripheral devices.

INTERLEAVE
A technique in multiprogramming whereby segments of one program are inserted into another program so that the two programs can, in effect, be processed simultaneously.

INTERNAL INTERRUPT
An interrupt occurring as a result of conditions within computer mainframe or immediate interfaces.

INTERRUPT
1. A break in the normal flow of a system or routine such that the flow can be resumed from that point at a later time. An interrupt is usually caused by a hardware-generated signal.
2. To cause an interrupt.

JOB
A deck consisting of control cards and possibly program and data decks presented serially to MASTER through the input card reader and recognized by MASTER as a piece of work. A job can consist of several tasks.

JOB CLASS
A classification assigned to a job by the job originator or by MASTER so that MASTER can more efficiently multiprogram its work load.

JOB MONITOR
A system program task that reads and interprets job control cards.

LIBRARY
An organized collection of standard, checked-out programs, routines, and subroutines which can be used to solve many types of problems and parts of problems.

LINKAGE

The interconnections between a main routine and a closed routine, i.e., the entry into the closed routine and the exit back to the main routine.

LOCATION

A position in storage where one computer word can be stored and which is usually identified by an address.

LOGICAL ADDRESS

An address in a chapter. The address that appears in the program address register when in Executive Mode.

MACRO INSTRUCTION

An instruction in a source language that is equivalent to a specified sequence of machine instructions. Usually, a symbolic mnemonic type instruction that a programmer can write in a source program to call for library or special routines.

*MAP

To establish a correspondence between the elements of one set and the elements of another set. In MASTER, a listing that correlates symbolic relocatable addresses and logical addresses for a run.

MASS-STORAGE CAPABILITY

The executive and operating system is designed to provide effective and efficient use of available mass storage devices. The result is a lightening of operator duties thus eliminating many of the errors believed inherent in large-scale software systems. The system provides for the maintenance of permanet data and program files on mass storage devices with full facilities for modification and manipulation of these files. Security access codes prevent unauthorized use.

MASS STORAGE (ON-LINE)

High-capacity data storage accessible to the central processing unit.

MEMORY PROTECT

Hardware that protects EXEC and all other programs.

MULTI-ACCESS

The capability of a computing system to collect and distribute data through several terminals.

MULTIPROCESSING

The use of two or more computers to logically or functionally divide jobs or processes and to simultaneously execute various programs or segments of programs asynchronously.

MULTIPROGRAMMING

A technique for processing numerous routines or programs simultaneously by overlapping or interleaving their execution; multiprogramming permits more than one program to time-share a machine component.

OBJECT LANGUAGE

The language that is the output of a given translation process, i.e., the language into which an assembler or compiler translates a source language.

OFFSET STACKER

A card stacker that can stack cards selectively under computer control so that they protrude from the balance of the deck to give physical identification.

OPERATING SYSTEM

An organized collection of programmed techniques and procedures for operating a computer.

ORDINAL

The location of an entry in a table.

ORIGIN

1. The absolute address of the beginning of a program or block.
2. In relative coding, the absolute address to which addresses in a region are referenced.

OUTPUT

Information transferred from memory to secondary or external storage; information transferred to any device exterior to the computer.

OVERLAY

A technique for bringing routines into high-speed storage from some other form of storage during processing so that several routines will occupy the same storage locations at different times. An overlay is used when the total storage requirements for instructions exceed available physical or logical memory.

PAGE

One of a number of blocks of arbitrarily predetermined uniform size into which core memory is divided. Paging is a technique in which pages are used to facilitate the dynamic allocation of data in storage.

PAGE FILE

On the 3300 computer, a set of 128 12-bit registers divided into 8 states of 16 registers. Each register indexes all or part of a page.

PAGE MAP

A set of 16 page indexes each reflecting 2K of a 32K chapter. The page map correlates logical addresses with physical page assignments.

PARAMETER

1. A variable that is given a constant value for a specific purpose or process.
2. A quantity in a routine which specifies a machine configuration, subroutines to be called, or other operating conditions.

PHYSICAL MEMORY
Actual memory that can be referenced only through the page file.

PRIORITY
A scheme for determining that one task can be executed before another.

PROCESSOR
A device capable of receiving data, manipulating it, and supplying results.

PROGRAM
1. The precise sequence of coded instructions necessary to solve a problem.
2. To plan the procedures for solving a problem. This may involve, among other things, analyzing the problem, preparing a flow diagram, providing details, developing and testing subroutines, allocating storage, specifying input and output formats, and incorporating a computer run into a complete data processing system.

READ
To transfer information, usually from an input device, to internal storage.

REAL TIME
Pertaining to a program for which time requirements are particularly stringent, that is, the data processing must keep up with a physical process within a time period of seconds or less.

RECORD
1. A collection of related items of data treated as a unit. Contrast with file.
2. To put data into a storage device.

RE-ENTRANT
Capable of being called into use while in use.

RELOCATABLE BINARY SUBPROGRAM
A program that can be contiguously loaded with the aid of a loader program into available logical memory.

RESIDENT, CORE MEMORY
That part of the system residing in core memory at all times.

RETURN
To transfer control back to a point in a program or program task from which a call was issued.

*ROUTINE
A set of instructions arranged in proper sequence to cause a computer to perform a desired task.

SCHEDULE

The acceptance of jobs from the input card reader and the inclusion of them into a list of scheduled jobs. EXEC refers to the jobs scheduled to determine the next job to be initiated.

PAGING

A technique permitting numerous jobs in memory at one time. Memory is divided into pages which can be subdivided into halves and quarters. A job need not be loaded into contiguous physical pages but can be loaded into any available (unused) pages where it is referenced through logical addresses.

*SNAPSHOT DUMP

A selective dynamic dump performed at various points in a machine run.

*SOURCE LANGUAGE

A language that is an input to a given translation process.

STATUS

A state or condition of hardware or task; e.g., busy or not busy.

SUBPROGRAM

A part of a larger program which can be converted into machine language independently.

SUBROUTINE

1. A portion of a routine that causes a computer to carry out a well-defined mathematical or logical operation.
2. A routine arranged so that control may be transferred to it from a master routine and so that, at the conclusion of the subroutine, control reverts to the master routine. Such a subroutine is usually called a closed subroutine.

TASK, I/O

Work to be performed on a file. Execution of an I/O task may result in transmission of data over a channel; e.g., READ, WRITE, or in preparation of that file by the System Executive for a subsequent data transmission; e.g., FORMAT, LOCATE.

TASK, PROGRAM

A program and any number of subprograms requiring the intervention of the executive system.

TIME-SHARING

The capability of a computing system to accommodate more than one user during the same interval of time without apparent restriction by the existence of other users. In time-sharing, a given device is used in rapid succession by a number of other devices or various units of a system are used by different users or programs.

TRAPPED INSTRUCTION

1. An instruction that is executed by a software routine if the necessary hardware is lacking or if the central processor is not in the required state.
2. An instruction whose execution is blocked.

UNIT

A peripheral device capable of storing, receiving, transmitting, or interpreting data.

UNLOAD

To remove a tape from ready status by rewinding beyond the load point; the tape is then no longer under control of the computer.

UPDATE

1. To modify a file with current information according to a specified procedure.
2. To modify an instruction so that the addresses it contains are changed by a stated amount each time the instruction is performed.

UTILITY ROUTINE

A routine in general support of the operation of a computer, e.g., an input/output, diagnostic, tracing, or monitoring routine.

WRITE

To transfer information, usually from internal storage, to an output device.

SYMBOLOGY †

ABORT	Abort executive request macro
ALGOL	Algorithmic language compiler
ALLOCATE	Allocate file macro
BCD	Binary coded decimal
BCT	Block common table loader cards
*BKO	Output backgrounder
*BKI	Input backgrounder
BYPASS	Bypass file I/O requests macro request
CALL	Call task macro
CANCEL	Cancel interrupt selection macro
CLOSE	Close file macro
COBOL	Common Business Oriented Language Compiler
COMPARE	Compare I/O macro
COMPASS	Comprehensive Assembly System
COPYCOM	Copy common macro
CP	Card punch
CR	Card reader
DATE	Date macro request
*DEF	File definition functions routine and control cards
DINT	Disable interrupt request
*DIR	Library directory file
DWAIT and DWAITIO	Deferred wait macro request
EINT	Enable interrupt request
ELD	End loading card
EOF	End-of-file mark or condition
EOJ	End-of-job card
EOT	End-of-tape mark
EPT	Loader subprogram entry point cards

† This glossary does not include COMPASS pseudo instructions, machine language instructions, programmer control cards, operator control statements, or diagnostics.

ERASE	Erase I/O macro
EXEC	MASTER executive routine
*EST	Establisher operating system task
FCA	First character address
FDPBOXS	Floating point simulation package
*FMU	File maintenance utility requests
FORMAT	Format I/O macro
FORTRAN	Formula Translation compiler
FREE	Remove time limit macro request
FWA	First word address
FWCA	First word or character address
GLIB	Library generation task
IA	Interrupt address
IC	Interrupt condition or code
IDC	Subprogram identification card
*IDF	Label ID file
IM	Interrupt mask
INP	Standard input file
*INT	Job initiator operating system task
I/O	Input/Output
ISR	Instruction state register
JOB	JOB control card
JMTR	Job monitor operating system task
LCA	Last character address
*LAB	Label directory file
LGO	Load-and-go unit
*LIB	Library file
LIMIT	Set time limit macro request
LOCATE	Locate I/O macro
LWA	Last word address
MAP	Memory Allocation Print
MIOCS	MASTER input/output control system
MODIFY	Modify file label macro

*MSD	Mass storage directory file
MT	Magnetic tape
ND	No dump
OCC	Octal correction control card
OPEN, OPENSEG, OPENU	Open file macro
OSR	Operand state register
OUT	Standard output file
PACK, PACKC, PACKD, PACKR	Blocking macros
PICK, PICKC, PICKD, PICKR	Deblocking macros
PR	Printer
PU	Punch
PUN	Standard punch file
READ	Read file I/O macro
READB	Read file backward I/O macro
RELEASE	Release file macro
RELRESV	Remove file reservation macro
RESERVE	Reserve file macro request
RETURN	Return to caller macro request
RIF	Relocatable information loader cards
RLDR	Absolute task generation control card or routine
SALOCATE	System file allocate macro
SCLOSE	System file close macro
*SCH	Job schedular operating system task
SCHED	Schedule equipment control card
SELECT	Select interrupt macro request
SET	Set control card
SEXPAND	System expand macro
SNAP	Snapshot dump control card
SOPEN	System open macro
SRELEASE	System file release macro

SUPPRESS	Suppress object deck execution macro request
TASK	TASK priority control card
TRA	Transfer address loader card
*TRM	Job terminator
UIC	User interrupt control routine and macro
TIME	Time macro request
TYPE	Console typewriter input/output macro request
UNLOAD	Unload tape I/O macro
WEOF	Write end of file I/O macro
XFER	Interdevice transfer routine and macro
XNL	External name and linkage loader cards

INDEX

† IM designates Installation Manual; OG designates Operator's Guide.

† IM designates Installation Manual; OG designates Operator's Guide.

† IM designates Installation Manual; OG designates Operator's Guide.

† IM designates Installation Manual; OG designates Operator's Guide.

† IM designates Installation Manual; OG designates Operator's Guide.

†IM designates Installation Manual; OG designates Operator's Guide.

† IM designates Installation Manual; OG designates Operator's Guide.

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

3300/3500 Computer Systems
MASTER Reference Manual

Pub. No. 60176800 December, 1966

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 REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

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

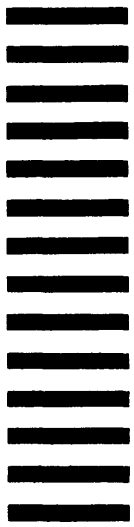
FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

LD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.



BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

)LD FOLD

STAPLE STAPLE

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60176800

Litho in U.S.A.