CONTROL DATA
CORPORATION

CONTROL DATA®
3100/3150/3170
3200/3300/3500
COMPUTER SYSTEMS

MSOS VERSION 5
OPERATING SYSTEM
REFERENCE MANUAL

CONTROL DATA
CORPORATION

CONTROL DATA®
3100/3150/3170
3200/3300/3500
COMPUTER SYSTEMS

MSOS VERSION 5
OPERATING SYSTEM
REFERENCE MANUAL

REVISION RECORD	
REVISION	**DESCRIPTION**
A	Manual released.
2-74	
B	Manual revised to correct miscellaneous errors and omissions.
8-74	
C	Manual revised to incorporate changes for 580 printer and class-R mass storage interchange-
1-75	ability, plus miscellaneous corrections and additions. This revision applies to Version 5.1.
Publication No. 60410600	

Address comments concerning this manual to:

Control Data Corporation
Technical Publications Department
4201 North Lexington Avenue
Arden Hills, Minnesota 55112

or use Comment Sheet in the back of this manual.

PREFACE

This manual describes the use of MSOS Version 5. In general, the manual is oriented toward the COMPASS assembly language user. However, the first five sections apply equally to ALGOL, COMPASS, COBOL, and FORTRAN users. These sections describe the methods of allocating and using files, and the use of the system job control statements. In addition, section 7 describes the use of overlays in FORTRAN and COMPASS programs, and section 22 describes auxiliary libraries that can be used with ALGOL, COMPASS, COBOL, and FORTRAN programs.

In addition to this manual, the user will need a copy of the MSOS Version 5 Diagnostic Handbook to interpret the system diagnostic messages and an MSOS Version 5 Operator's Guide. The following is a list of the manuals available for MSOS Version 5 and its product set:

Control Data Publications	Pub. No.
Operating System V5	
MSOS Reference Manual	60410600
MSOS Operator's Guide	60410700
MSOS Diagnostic Handbook	60410900
MSOS Installation Handbook	60410800
Product Set	
MSOS ANSI COBOL	60417900
MSOS MS COBOL	60191100
MSOS COSY	60207300
MSOS SORT MERGE (TAPE SORT, MS SORT)	60281500
MSOS SAINT	60213700
MSOS ADAPT	60173400
MASTER/MSOS ANSI FORTRAN	60281400
MASTER/MSOS MS FORTRAN	60057600
MASTER/MSOS ALGOL	60371800
MASTER/MSOS COMPASS	60236800
MASTER/MSOS LISA	60236900
MASTER/MSOS PERT TIME	60131100
MASTER/MSOS PERT COST	60132500

The MSOS V5 operating system is intended to be used only as described in this manual. Control Data cannot be responsible for the proper functioning of any features or parameters that are not described or not used as described in this manual.

CONTENTS

FIGURES

TABLES

INTRODUCTION 1

MASS STORAGE OPERATING SYSTEM

The Mass Storage Operating System (MSOS) loads and executes programs on CDC lower 3000 series computer systems. The major functions provided by the operating system are as follows:

- System control with control statements from console typewriter, the system input unit (card reader), magnetic tape, or a mass storage file.
- Program loader which loads relocatable binary object programs from the card reader, tape files, mass storage files, or the system library.
- Absolute loader which loads absolutized object programs from mass storage files.
- FORTRAN, ALGOL, and COBOL compilers, and a COMPASS assembler program.
- Use of independent auxiliary libraries which can be generated, used, and released without affecting the system library or the system EXEC operation.
- Use of priority programs (user supplied) which reside in core and are periodically initiated by preselected interrupts.
- Overlay functions for very large programs.
- Memory protection which prevents untested programs from writing in resident executive core area or core areas reserved for other programs.
- Priority interrupt processing routines which initiate priority and real-time programs, sense I/O completions and system faults, and allow limited multiprogramming.
- I/O processing routines to simplify driving the system's I/O units.
- Automatic output record blocking and automatic input record deblocking during I/O operations.
- Automatic system I/O error recovery.
- BCD to ASCII code conversion for ASCII I/O devices.
- Utility and XFER routines for copying, transferring, and listing of tape, mass storage, and card files.
- Mass storage spooling of batch job input files, list output files, and punch output files.
- Test patterns for special forms alignment on the printer.

HARDWARE DESCRIPTION

Version 5 of MSOS runs on any of the following CDC computers.

3100 Computer System	3200 Computer System
3150 Computer System	3300 Computer System
3170 Computer System	3500 Computer System

The following are the hardware requirements for using MSOS on any of these systems.

1 Central processor (CPU module)

16K Core memory†

1 Control console and operator's console typewriter

2 I/O data channels

1 Card reader (system input unit)

1 Card punch (system punch unit)††

1 Line printer (system list output unit)†††

2 Mass storage units (library files and system scratch files)

2 Magnetic tape drives (recommended)

Figure 1-1 is a diagram of a typical 3200 computer system that uses MSOS.

All CDC lower 3000 series computer systems have fully buffered data transfer between core memory and I/O devices. Under program direction, the CPU selects an I/O unit by connecting to its data channel and controller. Then the CPU sends an I/O function code to the data channel and controller. The channel and controller perform the I/O (read or write a block of data) leaving the CPU free to process other data or initiate an I/O function on another channel. When the I/O function at a unit is completed, the controller returns an end-of-operation interrupt to signal completion of the I/O.

MSOS VARIANTS

MSOS is available in three variants: standard, memory protection, and dynamic memory protection.

STANDARD MSOS

The standard variant of MSOS contains no memory protection for the resident executive routines and resident priority programs. Only the autoload/autodump routines are protected. Caution should be used when running untested programs during normal job processing periods since the untested programs may destroy the resident executive or resident priority programs. The standard variant is available with 16K or 32K of memory and may be used for both batch and priority programs.

---

† Allows limited batch or priority programming. 32K normally required for both batch and priority programming.

†† Recommended, but not essential for system operation. The system punch unit can be assigned to a tape unit or not used for a punchless system.

††† A tape drive may be substituted for this function.

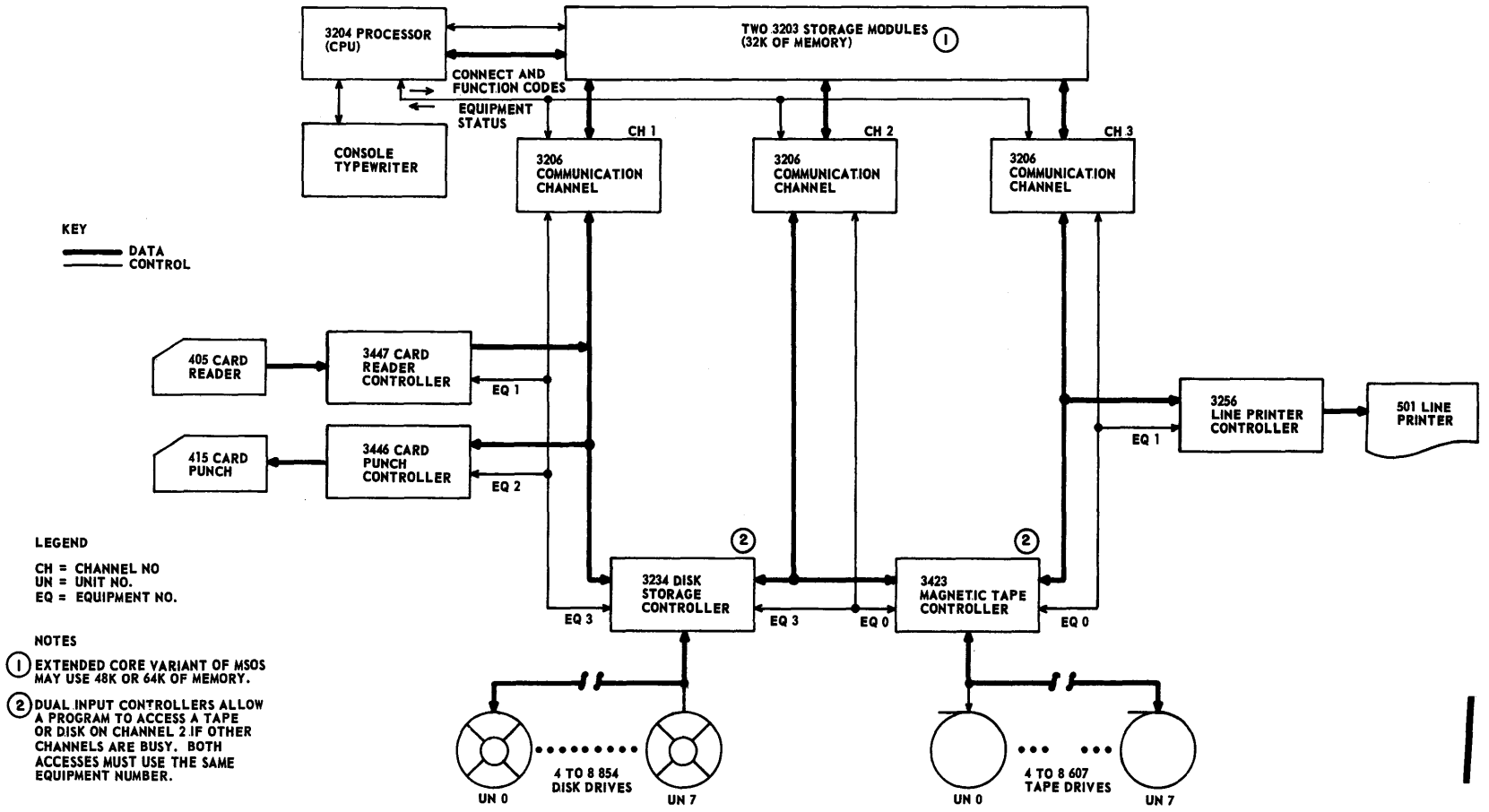


Figure 1-1. Typical 3200 Computer Equipment Configuration

MEMORY PROTECTION VARIANT †

The memory protect variant of MSOS prevents batch and priority level 3 and 4 programs from writing in the areas reserved for the system executive and priority level 1 and 2 programs. This allows untested batch and priority level 3 and 4 programs to be run without danger of destroying the operating system resident or priority level 1 and 2 programs.

Memory protect is provided by memory bounds switches, or optionally by memory bounds registers for dynamic memory protection. The memory protection switches are set by the operator. MSOS sends a message to the operator when system conditions require changing the switches. The new switch settings are included in the message.

The dynamic memory protect option uses bounds registers to perform the function of the memory protect switches. †† MSOS automatically sets and clears the flip-flops, thereby relieving the operator of the task of setting memory protect switches. The bounds registers restrict batch and priority level 3 and 4 programs to their assigned program area. In addition to protecting the operating system and priority level 1 and 2 programs, dynamic protection protects resident batch and priority 3 and 4 programs from each other. The dynamic memory protect variant of MSOS is available with 32K, 48K, or 65K of memory.

EXTENDED CORE VARIANT

The extended core version of MSOS (refer to Figure 1-2) allows the use of more memory for larger batch and priority programs. This variant of MSOS is available with 48K or 65K of memory and uses standard or dynamic memory protect.

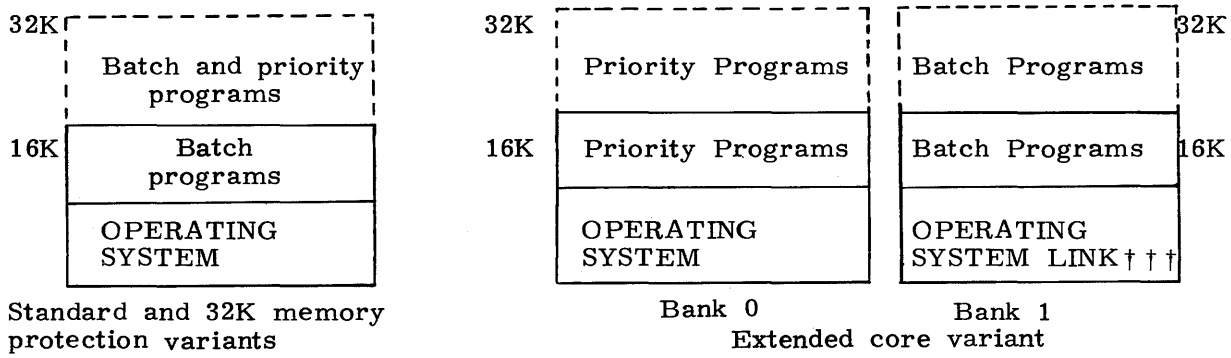


Figure 1-2. MSOS Variants

†Memory protection does not restrict the higher level (level 1 and 2) priority programs from writing anywhere in core.
††Available only with 3100, 3150, and 3200 computer systems.
†††Less than 64 words.

BATCH JOBS

MSOS processes batch jobs sequentially from the system input unit, normally a card reader. The operator may assign and reassign the system input unit to any available card reader or magnetic tape unit. In addition, priority programs may submit a batch job from mass storage files.

The operator places batch job decks (job stacks) in the system card reader and MSOS sequentially reads and executes the jobs. Unless a batch job is terminated by the operator, each batch job is processed and completed before the next batch job is started.

When priority programs are present in core, they periodically interrupt the execution of batch programs, do the priority processing, and then return control to the batch program at the point of interruption. If a batch job is submitted by a priority program, the batch job is initiated immediately after the interrupted batch job has finished processing. After the priority submitted batch job has finished, batch-program input reverts to the system input unit.

PRIORITY PROGRAMS

Priority programs reside in core and are brought into execution by either a system or an external interrupt. Priority programs normally remain in core after completion of their interrupt processing and are reexecuted each time the system interrupt occurs. Each time a priority program completes processing, control returns to the interrupted program at the point it was interrupted.

Priority programs may be terminated (released from core) by the operator or by self termination. Self termination occurs only if a self termination routine was coded into the program to terminate the program after a certain number of executions, after a certain time interval passed, or when certain system or programming conditions exist.

Priority programs must be assigned a priority level (1 through 4) when they are loaded. A higher level priority program interrupts all lower level priority and batch programs. The highest priority level (priority 1) is designed for use in real-time applications.

JOB ACCOUNTING

An accounting routine in MSOS accumulates the elapsed time for each job and prints the elapsed time at the console typewriter and at the system output unit with the job's list output.

A system accounts table is maintained which batch programs may reference for the date, time, job time, print line limits, cards punched limits, library editions in use, etc.

A provision for adding a system accounts file is included for those sites that choose to add their own special accounting routines and a system accounting file.

OPERATOR CONTROL OF JOB PROCESSING

The operator can control job processing in the following ways.

1. Use the SELECT JUMP 6 switch to stop processing between batch jobs after the next end-of-job card is read.† This allows the operator to switch the system input unit, punch output unit, or list output unit to a different unit (that is, from card punch to a tape drive, etc.). At this time, the operator may also type control statements for job sequencing and I/O equipment control and mount tapes or disk packs if required.
2. Use the SELECT JUMP 5 switch to stop processing immediately after starting the next batch job (that is, immediately after the next JOB card is read). This allows the operator to inspect the JOB card, enter job control statements such as special equipment assignments with EQUIP cards, and to terminate or initiate the job.††
3. Press the MANUAL INTERRUPT switch to interrupt the program in execution. This allows the operator to terminate any program in core (batch or priority), or to use the typewriter to send commands or data to any program in core that contains an operator message processor routine.
4. Use a SEQUENCE statement to search the input unit for a particular SEQUENCE card. This would be useful in cases where SEQUENCE cards were used to index job stacks on magnetic tapes.

† The SELECT JUMP 6 switch cannot be used while the system is running under control of APC (section 20).

†† The SELECT JUMP 5 switch cannot be used to change JOB statements while the system is running under control of APC.

I/O EQUIPMENT ASSIGNMENT 2

UNIT RECORD DEVICES

Unit record devices consist of all I/O units except mass storage units (that is, disk packs, disk files, and drums). Under MSOS, the user must use an EQUIP statement to assign a logical file number to each unit record device he intends to use. The file numbers are used in the program to designate units for I/O.

The file number may be any number between 1 and 53. It may be assigned to a specific type of hardware (any available unit of the type specified), or to a specific unit by physical location (that is, by channel, unit, and equipment number).

Example:

```
$EQUIP 21 = C1E2U4        C = channel number, E = equipment number
                          U = unit number
$EQUIP 20 = MT            MT is code for a tape drive
```

The above example assigns file number 21 to the I/O unit at channel 1, equipment 2, unit 4. It also assigns file number 20 to a tape drive. Any tape drive that has not been assigned may be assigned as number 20 by MSOS. These file numbers remain assigned until the job they were assigned in is completed. All file numbers are automatically released at the end of a job. Refer to section 4 for a detailed description of the EQUIP statement.

Once assigned, the file numbers are used to designate units for I/O functions within a program. For example, to write on the tape unit that was assigned file number 20, the following statements could be used.

```
COMPASS     RTJ  CIO        (refer to section 11 for description of CIO)
             02   20,0

COBOL       SELECT PAY-FILE ASSIGN TO TAPE 20
            OPEN I/O PAY-FILE
            WRITE PAY-FILE FROM NAME

FORTRAN     DIMENSION A(9260), B(4)
            WRITE (20) A, B
```

The file number assignments are automatically cleared at the end of each job; they must be reassigned at the beginning of the next job.

MASS STORAGE FILES

Mass storage files are files on disk packs, disk files, and drums. FORTRAN and COMPASS users must use logical file numbers to reference exiting mass storage files for I/O, the same as for unit record devices. New mass storage files must be allocated before they can be used for I/O (refer to section 3).

To assign a logical file number, the user must use a FET statement to identify the file (by owner name, file name, and block size), and an OPEN statement to assign a file number (1 through 53) to the file.

Example:

```
$FET, OWNER, FILENAME, 512          512 indicates 512-character blocks
$OPEN, 22
```

This example opens a file called FILENAME which is owned by OWNER. The OPEN statement assigns number 22 to the file. This number is used to reference the file for I/O in a program. The following examples are program statements that write on file number 22.

```
COMPASS     RTJ  CIO        (refer to section 11 for description of CIO)
             02  22,0
FORTRAN     DIMENSION A(100), B(40)
            WRITE (22) A, B
```

If a mass storage file is on a disk pack (device) that is not mounted when the file is opened, MSOS stops job processing and requests the operator to mount the device. When the device is mounted, job processing resumes.

COBOL users cannot open mass storage files outside of the COBOL program. Existing mass storage files must be opened in the COBOL program. New files must be allocated outside the COBOL program before they can be opened in the COBOL program.

Example:

```
SELECT PAY-FILE ASSIGN TO DISK
OPEN I/O PAY-FILE
WRITE PAY-FILE FROM NAME
```

The file identification must be supplied for each file that is opened in the LABEL RECORD VALUE statement of the file description (FD) entry.

Example:

```
FD PAY-FILE
    .
    .
    .
  LABEL RECORD IS STANDARD
  VALUE OF;
    ID IS FILENAME
    OWNER IS OWNERNAM
    ACCESS-PRIVACY IS PRIV
```

The FILENAME, OWNERNAME, and PRIV codes are assigned with the $FET card when the file is initially allocated.

LOGICAL UNIT AND FILE NUMBER ASSIGNMENT

MSOS uses file numbers for its system files and units. The MSOS system file numbers are reserved and cannot be assigned by the user. However, some system files may be referenced for input or output in user programs. Table 2-1 lists the system file numbers and the file numbers that the user can assign and use.

SYSTEM UNIT PROTECTION

MSOS contains a subroutine (PROTECT) which monitors all I/O functions on system files 58 through 63. This subroutine protects the system files from I/O functions which would interfere with normal job processing of these files.

Input (file 60) - The following I/O functions are prohibited on file 60. If attempted, PROTECT rejects them with an error code.

1. If file 60 is a tape instead of a card reader, PROTECT rejects I/O functions that: write on the tape, position the tape forward, position the tape backwards beyond the limits of the job file being read, rewind the tape, unload the tape, or change the read/record density.
2. After an end-of-file condition is sensed, PROTECT allows only the following I/O functions:

If Previous Function Was:	If Next Function Is:		
	Read	Read Backward	Backspace
Read	Reject	Accept	Accept
Read Backward	Accept	Reject	Accept
Backspace	Accept	Reject	Reject

Output (files 61 and 62) - I/O functions on tape units are rejected if they reposition the file over a previous job output, write over or erase any previous job outputs, or advance the file beyond the position that the current output is being written on. The following functions are always rejected: REWIND, UNLOAD, READ BACKWARD, WRITE EOF mark, and change of recording density. BACKSPACE is rejected unless the previous function was a WRITE or an ERASE.

CTO and CFO (files 58 and 59) - PROTECT allows only a read function on file 58 and only a write function on file 59.

Library (file 63) - PROTECT allows only a read and locate function on file 63.

SYSTEM I/O FILE SPOOLING (APC)

MSOS has an automatic peripheral control routine (APC) that buffers (spools) the system input, list output, and punch output on mass storage files before processing them. Spooling I/O files provides more efficient I/O and thereby increases the system throughput.

APC operates as a priority program that can be set up for initiation by operator commands.

TABLE 2-1. SYSTEM FILE NUMBER ASSIGNMENTS

Number	Description	Batch Program Usage	Priority Program Use
1-52	User files. Assignments are made in batch jobs or during initial loading of priority programs. File assignments are automatically released at the end of each batch job or priority program.	May be assigned and used for unit record devices or mass storage files.	Logical units can be referenced for I/O if they are EQUIPPED before the priority program is initialized. Mass storage files can be opened for use before or after the program is initialized.
53	Used by L-MSIO for an overlay file. The overlay file is allocated at install time and opened by L-MSIO as required.	Can be assigned only if ANSI COBOL, MS COBOL, LISA, or L-MSIO (macros) are not used.	Same as 1-52.
54†	Hollerith scratch file. Used by MSOS to store Hollerith code which is to be passed to a language processor (that is, assemblers, compilers, etc.). May also be used for batch program scratch. Block size is 512 characters per block.	Use as mass storage scratch files. The files are opened by the system. The user needs only to reference the file numbers to read or write on them. The block pointers are reset to 0 at the end of a job (after an EOJ card).	Cannot be referenced in a priority program.
55†	Intermediate scratch file. Used to store first pass output from assemblers and compilers, and used by the loader to assemble absolutized code before loading in core. Is also used by the recovery dump routine to temporarily store the contents of memory in which RDUMP will be overlayed. May also be used for batch program scratch. Block size is 512 characters per block.		
56†	Load and go scratch file (LGO). Used by MSOS to store relocatable binary output code from assemblers and compilers. May also be used by batch programs. Block size is 960 characters.		
57†	Reserved for system account file. Assignment of this file and its block size is a system installation option. If this number is assigned, this file must be processed with a user supplied accounting routine which must reside in the variable resident area of core when it executes.	Optional. Used to record system accounting information according to site requirements.	

†Number of blocks assigned to these files is an installation option.

TABLE 2-1. SYSTEM FILE NUMBER ASSIGNMENTS (Cont'd)

Number	Description	Batch Program Usage	Priority Program Use
58	CFO (computer from operator) unit. Input from console typewriter. Referencing file 58 causes the TYPE LOAD indicator to light at the console typewriter. Maximum block size is 80 BCD characters.	Read messages from operator.	Read messages from operator.
59	CTO (computer to operator) unit. Output to console typewriter. Maximum block size is 80 BCD characters.	Send messages to operator.	Send messages to operator.
60††	Standard system input unit. This file may be either a tape unit or a card reader. Programs reading from this unit should be coded to read from either type of unit. Block size is 80 BCD characters (160 octal characters).	Read new batch jobs, priority programs, and input data.	These files cannot be referenced in priority programs. However, the file numbers can be used to open permanent files (user units) the same as file numbers 1 through 53.
61	Standard system output (list) unit. This file may be either a line printer or a tape unit. Programs should be coded to write on either type of unit. Maximum block size is 136 BCD characters.	Print a job's output. †	
62††	Standard system punch unit. This file may be either a card punch or a magnetic tape. Programs with punched output should be coded to write on either type of unit. Maximum block size is 80 BCD characters (160 octal characters).	Punch output cards.	
63-68	System mass storage files: 63 Library file 64 MSD file 65 IDF file 66 Label file 67 ABS file 68 Library directory file	Cannot directly reference any file or unit number above 63.	Cannot directly reference any file or unit number above 62.

† Print control character plus 135 data characters. Refer to appendix B for a list of print control characters.
†† Only word I/O can be used on these units.

SUBMITTING BATCH JOBS FROM MASS STORAGE

MSOS has an alternate processor routine (AP) which accepts batch jobs submitted for execution from mass storage files. The batch jobs are coded for the standard system input unit but submitted to AP for execution from mass storage files by a priority program. Batch jobs submitted by priority programs are given priority in the order of processing over batch jobs submitted from the standard system input unit.

REASSIGNMENT OF SYSTEM SCRATCH FILES

For special jobs, one or more of the system scratch files (files 54, 55, and 56) may be closed and reassigned to a larger and permanent mass storage file. At the end of the job, the system automatically reopens the files, as originally allocated and assigned, for the next job. Note that the system uses file 55 extensively during job loading and termination. User information written on file 55 is not saved after the job.

MASS STORAGE FILE MANAGEMENT 3

OCAREM

Mass storage files are managed by OCAREM. OCAREM is a set of mass storage file control routines that are used to allocate, open, close, expand, modify, and release mass storage files. OCAREM does not read or write on a file. All reading and writing must be done internally within a user's program. In COMPASS programs, reads and writes may be done with CIO statements. (Refer to section 11 or L-MSIO statements in sections 14 and 15.) In FORTRAN and COBOL programs, reads and writes must be done with I/O statements in the program.

The OCAREM routines are part of the system executive. These routines can be called and used in a job with MSOS control cards. In addition, COMPASS users may call and use these routines by using mass storage file control macros (refer to section 13) within a COMPASS program.

ENTERING A MASS STORAGE DEVICE

Before a mass storage file or disk pack (device) can be used, it must be entered in the system MSIO files with an ENTER statement. This is normally an operator function and the procedure is described in the MSOS Operator's Manual. The ENTER statement is described in section 4 as one of the mass storage utility routines.

MSOS uses the information on the ENTER statement to write a device label on track 0 of the device, and write an MSD label in the mass storage device label file (MSD file, refer to appendix H). A track map is part of the MSD label. The map contains a record of each track on the device. The map indicates which tracks are available for new files or expanding old files, and which tracks are reserved for existing files.

ALLOCATING A FILE

Before doing I/O on a new mass storage device, the user must allocate space for a file on the device. A file need only be allocated once. After allocation, the file needs only to be opened before using it and closed when it is not in use.

Allocating a file reserves space on a mass storage device for the file and creates a label which defines the file. The file label contains a file name, an owner's name, an edition number, privacy codes, the files block size and length, its starting address (that is, device number and first sector address), etc. OCAREM writes all file labels on a label file (LABELFILE, appendix H). Then OCAREM references the file label every time a file is opened for input or output.

FET and ALLOCATE statements are used to allocate a file. These statements are described in section 4. The information in these statements is used to construct the file label. The label entries defined in these cards are initially built by OCAREM and updated each time the file is used.

Example:

```
$FET,OWNER,FILENAME,512
$ALLOCATE,B10
```

In the previous example, the FET statement specifies the owner, the file name, and the block size† to be used. The ALLOCATE card specifies that 10 blocks are to be reserved for the file. This is the minimum number of parameters that can be used. MSOS assigns values for the remaining parameters (privacy codes, edition number, etc). These parameters and their default values are described in section 4.

SELECTING A FILE BLOCK SIZE

When selecting the block size for a file, the size should be equal to or an even multiple of the sector size of the device being used. Since OCAREM starts each new block at the beginning of a new sector, failure to equate sector size with block size results in wasted mass storage space.

Examples:††

1. A block size of 512 characters fills two sectors on an 853 device. If the first block of a file starts at sector 00, the second block starts at sector 02. The same block size on an 841 device fills only 512 of the 640 characters in each sector resulting in 128 characters of wasted space in each sector.
2. A block size of 640 characters fills 2.5 sectors on an 853 device. If the first block of a file starts on sector 00, the second block starts at sector 03. Half of the space on every third sector is wasted.

OPENING A FILE

After a file is allocated, it must be opened before using it. Since only a limited number of files can be open at the same time,††† files should be closed when they are not in use. All files opened in a job will automatically be closed when the job terminates.

A FET and an OPEN statement may be used to open a file. The OPEN statement assigns the file number used to reference the file for I/O. The FET statement is used to locate the file label in the label file. These statements are described in section 4.

The modification privacy parameter on the FET statement may be omitted, and any nonzero numeric value may be used as the block size parameter. These parameters are not checked by OCAREM when opening a file.

---

†In number of characters.

††Refer to Table 4-1 for device sector sizes.

†††The number of files that can be open concurrently is determined by the size of the FDT table which is an installation assembly option. Each mass storage file that is open has an entry in the FDT table that uses 8 + (three times the number of segments) words. A nonmass storage file (that is, unit record device entered with an EQUIP statement) requires 9 words.

When an OPEN statement is read, OCAREM checks the file label and the labels of all mounted devices to ensure the file is on-line. If the device that contains the file is not mounted, OCAREM sends a message to the operator requesting the device be mounted. Then OCAREM waits until the operator mounts the device before continuing the job. If the file is segmented across more than one device, OCAREM requests the operator to mount each device containing a segment of the file.

When the device containing the file is mounted, OCAREM enters the file label in a file description table (FDT) and assigns the file number given on the OPEN statement to the table. Then, when I/O requests are made on the file, OCAREM uses the file number to locate the file label in the FDT table.

Example:

```
$FET,OWNER,FILENAME,512,ED,ACCS
$OPEN,21
```

These statements open edition ED of the FILENAME file and assign logical number 21 to the file. ACCS is the privacy code needed to open the file, and the block size is 512 characters.

CLOSING A FILE

A CLOSE statement applies to mass storage and unit record device file numbers. It closes the file specified on the CLOSE statement. Closing a file clears that file entry in the file ordinal table. For mass storage files, closing updates the file label in the LABELFILE with the number of blocks written, last date accessed, etc. After a file is closed, it cannot be used for I/O until it is reopened or reequipped. The file number may be reassigned to another mass storage file or a unit rccord device. The CLOSE statement is described in section 4.

Example:

```
$CLOSE,21
```

This CLOSE statement closes file number 21.

EXPANDING A FILE

A FET statement and an EXPAND statement may be used to expand a file. Expanding a file increases the number of tracks assigned to the file. The file may be expanded to additional devices. The EXPAND statement is described in section 4.

The EXPAND statement causes OCAREM to update the MSD file to reflect the additional space reserved for the file, and to update the label in the LABELFILE with the location and number of the new tracks.

The file must be closed before it can be expanded and the FET statement must contain an access and a modification privacy code.

Example:

```
$EXPAND,B100
```

This EXPAND card adds 100 blocks of new space to the file defined by the preceding FET card.

MODIFYING A FILE

A FET statement and a MODIFY statement may be used to change the owner name, file name, block size, edition number, privacy codes, expiration date, and protection (that is, read only or read and write) in a file label.

The expiration date and protection may be changed with a MODIFY statement. For other changes, a FET statement with new values must be input following the MODIFY statement.

The MODIFY statement is described in section 4. A file must be closed before it can be modified and the FET statement must contain the files modification privacy code.

Example:

```
$MODIFY,I,760704
```

This MODIFY card changes the protection to input only and changes the file expiration date to July 4th, 1976.

Example:

```
$MODIFY,,,N
$FET,BLT,HAM,512,AB,ACE,DUCE
```

These cards replace the old FET used to identify the file for modification with a new FET statement. All new FET parameters replace the old FET parameters in the file label in the LABELFILE.

RELEASING A FILE

A FET statement and a RELEASE statement may be used to release part or all of the mass storage space reserved for a file. All released space will be made available for new files or for expansion of existing files. The RELEASE statement is described in section 4.

Releasing all of a file removes the file label from the LABELFILE, and releases all space that was reserved for the file in the device label (MSD file). Since the file label was deleted, the file can no longer be referenced for any purpose, and all space reserved for the file is set as available in the mass storage device label (MSD).

Releasing part of a file releases a specified number of tracks starting with the highest track number allocated for the file and working downward. The number of tracks to be released is specified on the RELEASE card.

If a file has been loaded with data and the amount of unused space is unknown, all unused space may be released by specifying UNUSED on the RELEASE statement. All tracks which have not been written on are removed from the LABELFILE and set as available in the MSD file. If the file has not been written on, all blocks are released except the first track (or block if a block is larger).

A file must be closed before it can be released and the modification privacy code must be included on the FET statement.

Examples:

$RELEASE,ALL	All of a file
$RELEASE,10	The upper ten tracks of a file
$RELEASE,UNUSED	All the unused tracks of a file

CLASS-R DEVICES

Class-R devices and the files written on them can be moved to and used with any MSOS or MASTER system containing class-R code. Nonclass-R devices and their files can be used only at the system which the device was initially entered in. The operator selects a device's class (R or non-R) with the R parameter when he enters the device in a system with the ENTER statement.

MSOS writes an RLABEL on class-R device when the device is entered in the system. The RLABEL contains space for file labels and the device's MSD information.

When the user allocates a new file on a class-R device, OCAREM writes the new file's label in both the system's LABEL file and in the device's RLABEL. MSOS uses the system's LABEL file for mass storage file references and file modifications. However, MSOS always updates the RLABEL entries so that they are always current with their corresponding entries in the MSD and LABEL files.

When a class-R device is taken off-line, OCAREM removes the device's labels from the system's MSD and LABEL files; and when a class-R device is brought on-line, OCAREM copies the information from the device's RLABEL into the system's MSD and LABEL files. As a result, class-R devices with files from other systems can be brought on-line and used the same as a class-R device that was initially entered in the system. However, the system has no record of files on class-R devices that are off-line. A job aborts if an attempt is made to open a file residing on a class-R device that is off-line.

Labels for nonclass-R devices and files are permanently retained in the system's LABEL and MSD files (that is, until the file or device is released). When the user opens a nonclass-R file which is on an off-line device, OCAREM locates the number of the device containing the file from the LABEL and MSD files and requests the operator to mount the device before continuing with the job.

MSOS CONTROL STATEMENTS 4

CONTROL STATEMENTS

The control statements described in this section may be input to MSOS as punched cards from the card reader, typed by the operator at the console typewriter, or read as card images from tape or mass storage files. † These control statements allow the user to load and execute programs, select I/O units for I/O functions, allocate and use mass storage files, call programs from the system library, and send messages to the system operator. In addition, a series of I/O utility routines and error detection and correction functions may be selected with these control statements.

Each control statement contains an MSOS control character (either 7/9 or $ may be used interchangeably) in column 1 and a control name starting in column 2. Following the control name, a parameter string occurs on most cards. The parameters are separated from the control name and from each other by commas.

When a parameter is omitted, the system substitutes a default value. The trailing comma must be retained for omitted parameters, and the comma is omitted after the last parameter on the statement.

Example:

```
$NAME,P1,P2,,,P5,P6
```

P Parameter

In the above example, parameters P3 and P4 were omitted (defaulted). If the complete parameter string is eight parameters long, parameters P7 and P8 are also defaulted. All blanks in the parameter string are ignored by the system.

All control statements will be printed on the jobs output listing.

JOB PROCESSING STATEMENTS

Job processing statements are control statements used to initiate and terminate a job, communicate with the operator, and load and execute user programs.

SEQUENCE

The SEQUENCE statement is an optional job or job stack identification statement that precedes JOB statement. At any time, the operator may request MSOS to skip to a specific sequence statement in a job stack and start processing the jobs that follow it. If the input unit is a card reader, MSOS will skip to the SEQUENCE card and process jobs until the input hopper is empty. If the input unit is a magnetic tape drive, MSOS searches tape for the SEQUENCE statement and then processes jobs in the job stack until an ENDSCOPE or ENDREEL statement is sensed.

```
SEQUENCE,j
```

j Any one to three digit decimal number (0-999)

The j parameter is required.

†APC or AP must be used when submitting control statements from mass storage (refer to sections 20 and 21).

The SEQUENCE statements need not occur in sequential order. For example, the SEQUENCE statements for a job stack on tape to be run at ten o'clock (SEQUENCE, 10) could precede the SEQUENCE statement for a five o'clock job stack (SEQUENCE, 5).

When a SEQUENCE statement is read, MSOS spaces the output unit one page, copies the SEQUENCE card on the output unit, and writes the j parameter in the system accounts table (refer to section 16).

JOB

The JOB statement must be the first statement in each batch job. Before starting a new job, MSOS ejects a page on the output unit † and prints the new JOB statement. MSOS also prints the information from the JOB statement and a system generated sequence number at the console typewriter. If a SEQUENCE statement was used, MSOS replaces the word JOB with the three-digit sequence statement at the typewriter.

$JOB, c, i,t, nl, nc, comments b

c Job account number. One to eight alphanumeric characters.

i Job name. Any number of alphanumeric characters may be used. Only the first four characters of the i parameter are used. They are written in the accounts file, listed with the job accounting output, and punched with the jobs punch card output.

t The maximum job run time in minutes. Time may range from 1 to 999. † If omitted, a default value that was set in the system at installation time is used. If the job running time exceeds t, the job is aborted.

The t parameter applies only to elapsed clock time and operator intervention time. It does not apply to time used by priority programs.

nl Maximum number of lines that can be printed on the output file. If omitted, the default value set in the system at installation time is used. If the number of lines to be printed on the system output unit exceeds nl, the job aborts. † †

nc, Maximum number of cards that can be punched by the job. It omitted, the default value set at installation time is used. If the number of cards to be punched exceeds nc, the job aborts. † †

comments Comments field follows after the nc parameter. Blank separators between the two fields are not necessary, since nc is followed by a comma.

b Hollerith/ASCII parameter. Selects Hollerith or ASCII to BCD conversion for card reader, card punch, and system output printer. The b parameter occupies columns 78, 79, and 80.

---

†If a SEQUENCE statement was used, MSOS ejects the page for the SEQUENCE card, but not for the JOB card.

† †A value of * specifies unlimited number.

Column	Value	Description
78	A	Selects ASCII to BCD conversion of card reader input. †
	H	Selects Hollerith to BCD conversion at the card reader input.
	Omitted	Uses the conversion selected at system installation time.
79	A	Selects printed output in ASCII characters (BCD to ASCII conversion) ††
	H	Selects printed output in Hollerith characters (BCD to Hollerith conversion)
	Omitted	Uses conversion type selected at system installation time.
80	A	Selects ASCII punched card output (BCD to ASCII conversion) †††
	H	Selects Hollerith punched card output (BCD to Hollerith conversion).
	Omitted	Uses conversion type selected at system installation time.

The job statement will be printed on the standard OUT unit with the following additional information.

1. Job sequence number
2. MSOS version number
3. Library edition number
4. Date
5. Job initiation time

Example:

```
$JOB,54ANJ60,TYPE,21,950,1260
```

---

†3447-2 controller permits 405 card reader to read cards punched in ASCII or Hollerith and convert them to internal BCD. If A parameter is specified without the 3447-2 available, the cards are read as Hollerith.

††A 512 line printer with an ASCII subset train must be used if the A parameter is selected (refer to TRAIN statement).

†††3446-2 card punch controller is required to accept internal BCD and convert to ASCII or Hollerith punch codes. If the A parameter is specified without the 3446-2 available, the cards are punched with Hollerith code.

PRIORITY

The PRIORITY statement is used in place of the JOB statement for priority programs.

```
$PRIORITY,p
```

p Priority level of the program to be loaded. The value of p may be P1, P2, P3, or P4. This parameter is required.

Only programs in relocatable binary format may be input following the PRIORITY statement. Source language programs cannot be input or compiled following a PRIORITY statement.

The priority program must contain one or more of the following statement groups.

1. LOAD and RUN if priority program is on a tape or mass storage file.
2. Binary IDC card and a RUN card if the priority program is to be input as binary deck from the card reader.
3. Library program call statement if the priority program is to be input from the library.
4. An ABSTSK statement if the priority program is on a mass storage file and in absolutized format.

The priority program may contain any of the following statements.

1. EQUIP
2. DUMP
3. Any mass storage file control cards

Before a priority program is loaded, a message is sent to inform the operator of the program priority level. Since only one priority program of each level can be in core at the same time, the operator must terminate any priority program with the same priority level before MSOS will load a new priority program.

Example:

```
PRIORITY,P3
```

CTL

The CTL statement prints messages on the jobs output listing.

```
$CTL,message.
```

The message consists of a series of 1 to 75 alphanumeric characters starting in column 6 of the statement. MSOS inserts a blank space in the first column of the printer and then prints the message.

Example:

```
$CTL,PLEASE SAVE THIS OUTPUT FOR JOES FILES
```

CTO

The CTO statement sends messages to the operator at the console typewriter.

```
$CTO,comments to operator
```

The comments consist of a series of 1 to 68 alphanumeric characters (including spaces) starting in column 6. MSOS does a carriage return at the typewriter before typing the message. MSOS also prints the message on the output unit.

Example:

```
$CTO,PLEASE MOUNT JOES TAPE ON CH1 EQUIP2 UNIT3
```

PAUS

The PAUS statement stops all processing of the job and sends a message... READY?... to the operator. Processing continues when the operator presses FINISH. The PAUS statement can be used only with batch jobs. It causes a control statement error diagnostic if used in a priority program. All comments on the PAUS card are typed after the PAUS READY message to the operator.

```
$PAUS,comments
```

TRAIN

The TRAIN statement sends a message to the operator requesting a specific print train to be mounted on the 512 or 580 line printer. MSOS waits for the print train to be mounted before continuing with the job.

```
$TRAIN,n,lu
```

n	Train number. Values are 1 through 4.
	1 595-1 (501 compatible train) 2 595-2 (AN compatible train) 3 595-3 (HN compatible train) 4 595-4 (ASCII subset train) Other Illegal, causes a diagnostic
lu	Logical unit number of the 512 or 580 line printer (refer to EQUIP card). Values are 1 through 53 or 61. Default value is unit 61.

Example:

```
$TRAIN,1,32
```

LOAD

The LOAD statement loads relocatable binary programs into core memory and links all subprograms together so they are ready for execution. † The relocatable binary programs may be loaded as card decks at the system card reader, or from a tape or mass storage file.

$LOAD,f1,f2,f3,M or $LOAD,M

f Mass storage or tape file containing the relocatable binary program. One to three files may be specified. The values for each may range from 1 through 53 or 56 (load and go scratch file). If no files are specified, the loader assumes all decks to be loaded are on the system input unit following the LOAD statement. The trailing commas are omitted for all f parameters not included.

M Request for memory map. M is used to request a memory map on the standard output unit. If the parameter is omitted, no map is produced. M may appear anywhere in the parameter list.

Examples:

$LOAD

$LOAD,21

$LOAD,21,M,22

The mass storage files specified on the LOAD statement must be opened and the file block size must be 960 characters per block. Tape files must be in card image format with 160 characters per block. All files are automatically positioned to the first block by the loader.

The loader absolutizes and links all subprograms from file f1, f2, f3, and from any decks following the LOAD statement on the input unit. Only one LOAD statement can be used per task. Each file is loaded until an EOF is read.

The absolutized program produced by the loader is assembled on scratch file 55 before loading into core. If a MAIN (overlay) card is placed in front of a relocatable binary deck, the loader assembles the absolutized program on the file number specified on the overlay statement. In this manner, absolutized code is saved on a permanent file for quick loading with the ABSTSK statement.

ABSTSK

The ABSTSK statement loads absolutized binary programs into core from a mass storage file and starts execution of the program.

$ABSTSK,f,p1,...,$p_n$

f Mass storage file number containing the absolutized program.

p1-p_n Parameters to be passed to the program. Values depend upon the program. Parameters are optional.

The ABSTSK statement loads absolutized binary programs from file f into core. The memory locations used by the absolutized program must be available or the job aborts with a diagnostic.

†Refer to section 6 for a description of the relocatable binary format.

When MSOS enters the program, register A, Q, and B1 contain the following information.

Register	Description
Register A, bits 16-00	First character address of the $ABSTASK statement in core.†
Register A, bits 23-17	Relative position of first character of first parameter on the card. The control character $\frac{7}{9}$ or $ is position 1.
Register Q, bits 14-00	Second transfer address from TRA cards (if more than one entry is used).
Register B1	Mode of program. 1 Batch 2 Priority 4 3 Priority 3 5 Priority 2 6 Priority 1

BINARY IDC CARD

Relocatable binary subprogram decks are loaded from the system input unit without a LOAD statement. Whenever MSOS reads a binary card or card image on the system input unit, it assumes the binary card is the first card (refer to IDC card in section 6) in a subprogram deck. MSOS calls the loader which loads and links the subprograms the same as if a $LOAD card were used with no f parameter specified.

RUN

The RUN statement starts execution of the program which has been loaded with a LOAD statement or a binary IDC card.

```
$RUN
```

Data cards or statements may follow the RUN statement at the system input unit.

LIBRARY PROGRAM NAME

The library program name statement loads programs from the system library or an AUX library. A RUN statement is not used with the library program name statement; execution is automatic. Data cards or statements may follow the library program name statement on the system input unit.

```
$name,parameters
```

name	A library program name (main entry point symbol)
parameters	Parameter string to be passed to the program

† On 65K memory systems, the address will be a bank 0 address.

Examples:

```
$FORTRAN,I,L
```

This statement calls MS FORTRAN into execution from the library. The I parameter indicates the FORTRAN input source statements follow on the system input unit. The L parameter specifies list output on the system list output unit.

```
$COMPASS,I=21,X=56
```

This card calls COMPASS into execution from the library. The I parameter indicates that COMPASS input source statements are on file 21 and COMPASS executable output is written on file 56, the load and go file.

Use of the standard MSOS library programs and their control parameters is described in the applicable program reference manual.

When a program is called off a system or AUX library with a program name statement, the loader enters the following information in the A, B1, and Q registers and then enters the program at the main transfer address (from TRA card).

Register	Bits	Contents
A	14-00	First word address of a block in core containing a copy (BCD) of the library program name statement used to load the program. In extended core systems, the address is a bank 0 address.
A	16-15	Zero
A	23-17	Position of first character of the parameter string (relative to 1 which is 7/9 or $) on the library program name statement. Zero if there were no parameters in the statement.
B1	2-0	Mode of the program 1 Batch 2 Priority 4 3 Priority 3 5 Priority 2 6 Priority 1
Q	14-0	Second transfer address from the TRA card. Zero if there was no second transfer address.

AUX

The AUX statement specifies that an auxiliary library is to be searched for library programs and routines before referencing the main system library. The system library is searched for all programs and routines not found on the auxiliary library or libraries.

```
$AUX,f1,f2,f3
```

f File number of a mass storage file containing the auxiliary library or directory. One to three files (different auxiliary libraries) can be specified on each AUX card.

The AUX statement may be used with the following loader statements.

IDC card	Library routines referenced in the binary decks will be searched for on the auxiliary library first.
LOAD statement	Library routines referenced in binary decks will be searched for on the auxiliary library first.
Library program name statement	The auxiliary library will be searched for the program first.

The AUX statement remains in effect throughout the job, or until another AUX statement is read. All subsequent AUX statements replace previous AUX statements. An AUX statement with omitted f parameters will clear the previous AUX statement.

When more than one auxiliary library is specified, they are searched in the order listed on the AUX card (that is, f1 is searched first, then f2, etc.).

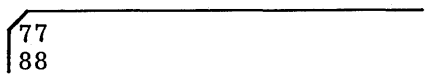
All files specified on the AUX statement must be opened with an OPEN statement before the AUX statement will be accepted.

Examples:

```
$AUX,21
$AUX,21,22,23
$AUX
```

END-OF-FILE

The end of file statement (EOF) is a $\begin{matrix}7\\8\end{matrix}$ punch in columns 1 and 2 of a card, or a $\begin{matrix}7\\8\end{matrix}$ card image on a tape or mass storage file. MSOS will not read past an EOF card on the input unit until the preceding task has been executed. † If an abort condition occurs, MSOS aborts only the task in execution (that is, all control statements and programs up to the EOF statement). Then MSOS reads and executes the remaining tasks in the job. An EOF statement clears all interrupts selected by programs in the task and is required at the end of each job.



Columns 3 through 80 are ignored by MSOS. The end of file card spaces the printed output to the top of the next page.

MSOS assumes the card following an EOF card is the first card of the next task in the same job, unless one of the following cards is read next.

EOJ	Indicates the end of a job and initiates end-of-job processing. ††

†A task is a group of MSOS control cards and/or user programs followed by an EOF statement. Several tasks may be grouped into a single job to provide common accounting, or a job may consist of only one task.

††Refer to EOJ card for a description of end-of-job processing.

JOB statement	When following an EOF card, a job card indicates the end of the current batch job and the start of a new batch job. The JOB card initiates end-of-job processing (the same as an EOJ statement) only when it follows an EOF statement.
SEQUENCE statement	Indicates the end of the current batch job and the start of a new batch job stack. The SEQUENCE statement initiates end-of-job processing (the same as an EOJ statement) only when it follows an EOF statement.
PRIORITY statement	Indicates the end of the current job and the start of a priority program. The PRIORITY statement initiates end-of-job processing (the same as an EOJ statement) only when it follows an EOF statement.
ENDSCOPE statement	Indicates the end of a batch job stack. The ENDSCOPE statement initiates end-of-job processing the same as an EOJ statement.
ENDREEL statement	When the INPUT unit is magnetic tape, the ENDREEL statement indicates the end of the tape reel. When following an EOF statement, the ENDREEL statement initiates end-of-job processing, the same as an EOJ statement.

EOJ

The EOJ statement indicates the end of a batch or priority job. It must be preceded by an end-of-file statement.

```
$EOJ
```

When an EOJ statement is read or an end-of-job condition is sensed (refer to end-of-file statement) MSOS performs the following end-of-job processing.

1. Prints the jobs output and the jobs accounting information on the standard system output unit.
2. Releases all equipment assigned with an EQUIP statement, and closes all mass storage files opened in the job.
3. Resets block pointers for scratch files (files 54, 55, and 56) to block 1. If the preceding job closed one or more of the scratch files for possible reassignment, MSOS releases all system scratch files and reopens them as originally allocated and assigned.
4. Initiates any batch job that has been submitted from a priority program.
5. Senses SELECTIVE JUMP 6 switch. If the switch is set, MSOS requests operator to reassign the system input, output, or punch units before initiating the next batch job from the input unit. The operator may assign new units, continue with the same units, or type control statements.
6. Initiates the next batch job from the input unit.

ENDSCOPE

The ENDSCOPE statement indicates the end of a batch job or batch job stack. It initiates end-of-job processing, the same as an EOJ statement. In batch jobs, the ENDSCOPE statement stops all job processing and gives control to the operator so the operator can switch any of the standard system units (input, list output, or punch output) to a different unit. The operator must restart batch job processing. In batch jobs submitted from a priority program, ENDSCOPE is treated as an EOJ statement. Normal batch processing continues.

$ENDSCOPE,a

a — Action to be taken for all standard system units that are on magnetic tape. This parameter is ignored for all standard system units not on magnetic tape.

- R — Write end-of-file marks and rewind the tape units.
- N — Write end-of-file marks, but do not reposition the tape units.

Omitted — Write end of file marks and unload the tape units.

ENDREEL

When the system input unit is a magnetic tape, the ENDREEL statement indicates the end of the tape reel. The ENDREEL statement stops the processing and unloads the reel. The operator must mount a new tape before processing resumes.

$ENDREEL

ENDREEL may be used only for batch jobs and must occur between jobs. If it is not preceded by an EOJ statement, the ENDREEL statement causes end-of-job processing to occur.

I/O UNIT CONTROL STATEMENTS

I/O unit control statements assign file numbers to I/O units, rewind and unload tape drives, and specify the type of I/O error recovery to be used with a specific I/O unit. In addition to the following statements, the CLOSE statement described with the mass storage file control statements may be used to release unit file number assignments.

EQUIP

Each I/O unit (except mass storage devices) to be used in a job must be assigned a file number in order to reference the unit for I/O functions in a program. MSOS clears all user file assignments after each EOJ statement.

$EQUIP,$x_1=u_1,x_2=u_2,\ldots,x_n=u_n$

x — A two-digit file number assigned by the user for reference purposes. The numbers may range from 01 through 53. The file number cannot have the same value as any other file number assigned or used in the same job.

u Definition of the I/O unit to which the file number is being assigned. The I/O unit may be defined in one of four ways.

1. By hardware type. u is equated to one of the hardware types as follows:

MT	Magnetic tape drive
CR	Card reader
PR	Line printer
CP	Card punch
TP	Paper tape punch
TR	Paper tape reader
DS	Display station
TY	Typewriter
TS	Typewriter station
PL	Plotter
OR	Optical character reader
SL	Satellite controller
SP	Seismic processor

Example:

```
--, 06=MT,--
```

In the example, file number 06 is assigned to any available unassigned tape drive.

2. By physical equipment location. u is equated to a specific I/O unit by channel, equipment, and unit number. The format is CnEnUnn

Example:

```
--, 07=C0E1U03,--
```

In the example, file number 7 is assigned to unit number 3 on equipment number 1 and channel 0.

If the unit number is omitted, the logical unit number is assigned to any available unit attached to the specified channel and equipment.

Example:

```
--, 08=C1E2,--
```

3. By both hardware type and equipment location.

Example:

```
--, 09=TRC2E3,--
```

In the example, file number 9 is assigned to the paper tape reader that is attached to equipment number 3 and channel 2.

When specifying both hardware type and equipment location, the equipment number and/or unit number may optionally be omitted.

Examples:

```
--, 10=MTC3,--,11=MTC4E1,--
```

In the examples, file number 10 is assigned to any available tape drive connected to channel 3 and file number 11 is assigned to any available tape drive connected to channel 4, equipment number 1.

4. By equating one file number to another file number that was previously defined (assigned to a hardware unit).

   Example:

   12=11

   In the example, file number 12 is equated to the same I/O unit that file number 11 is currently assigned to. Then, either 11 or 12 may be used to reference the unit for I/O.

Any user or scratch file (files 01 through 53) can be equated to any other program file (files 01 through 53). Also, a user file can be equated to the standard input, list output, punch output, CTO, or CFO files (that is, 21=60). However, the system files cannot be equated or reassigned with an equip statement. Only the operator can reassign system units.

The following is an example of an EQUIP statement that assigns file numbers 06 through 12. Note that spaces may be used to separate the u parameter definitions.

Example:

```
$EQUIP, 06=MT, 07=C0E1U3, 08=C1E2, 09=MTC3E3U4, 10=TPC3,11=TRC3E1, 12=11
```

REWIND

The REWIND statement rewinds magnetic tapes and positions mass storage files to block 1.

$\$REWIND,u_1,u_2,\ldots,u_n$

u — File number of a tape or mass storage file. The values for u may range from 1 through 56.

The file number is ignored if any of the file numbers are unassigned, or the file is not on mass storage or tape. No diagnostic is printed on the output unit.

Example:

```
$REWIND,01,03,27,53
```

UNLOAD

The UNLOAD statement unloads magnetic tape reels.

$\$UNLOAD,u_1,u_2,\ldots,u_n$

u — Number of the file to be unloaded. The value may range from 01 through 53.

If any file number is unassigned or not a tape unit, the number is ignored. No diagnostic is printed.

Example:

```
$UNLOAD,02,04,28,52
```

FMT

The format statement (FMT), specifies the type of error recovery to be used on tape and mass storage files if the system error recovery routines are used. †

The FMT statement may be used in both batch and priority programs. Additional FMT statements may be added to a job to reset error recovery for different programs in the same job.

1. For tape files:

   $FMT,u,$t_1$,$t_2$,...

   u Tape file number

   t Type of recovery to be selected. One or more of the following values may be used:

   SNR Four-character system noise records (SNRs) are written to bracket bad spots on tape. The SNRs are discarded on reads. The noise threshold for input files is 18 frames (characters) for a data block. Seventeen or less characters are rejected as noise. Eighteen or more characters are accepted as a data block.

   NSNR SNRs are not used to bracket bad spots on a tape. The noise record size is 17 characters (same as for SNR).

   NSN=n n is the maximum number of characters that are discarded as noise on an input file. N+1 characters are accepted as a valid data block. Maximum number of noise characters is 63 decimal. Setting n larger than 63 defaults to 63. If the NSN option is selected, SNRs are not used to bracket bad spots on the tape.

   R1 Opposite-direction READ recovery is suppressed.

   R2 The data from opposite direction read recovery is returned to the user when the parity is correct.

   R3 The data from an opposite direction READ during read recovery is returned whenever parity is correct and the record read is equal to or less than the size specified on the read request. That is, the data is returned if it is certain that truncation has not occurred.

   The t parameters may appear in any order. If conflicting parameters are used, the last parameter in the series takes precedence.

If an FMT statement is not used, the default values will be SNR and R3 for each tape drive. The default values may be changed by an assembly option when the system is installed.

Example:

```
$FMT,21,NSNR,R2,R3
```

† Refer to section 23 for a description of noise records and other system I/O error recovery methods.

2. For mass storage files:

$FMT, msopt

MS1 Do a write check (read back and compare) whenever SCARV50 or SCAR is called for write error recovery or whenever write with error recovery (CIO function code 42) is used to write a block.

MS2 Suppress the write check when SCARV50 or SCAR is called and when write with error recovery is used.

The FMT statement applies only to user mass storage files opened in the same job. The MS1 option is always in effect for files 54 through 68. Default is MS1.

Increased job throughput can be obtained when the MS2 option is used in conjunction with APC. The MS2 option also provides a significant decrease in the time required for MS SORT jobs.

Example:

$FMT, MS2

MASS STORAGE FILE CONTROL STATEMENTS

The mass storage file control statements call OCAREM routines into core to define, open, and close mass storage files. Control statements may also call OCAREM routines to expand, redefine, and release mass storage files. In addition, the REWIND statement may be used to reset the block pointer, for opened mass storage files, to block 1. All reading and writing on mass storage files must be done within the user's program, or with the mass storage utility routines.

RAT

The RAT statement is the first of a series of three statements used to allocate or expand a mass storage file. The RAT statement is an optional statement that specifies nonclass-R mass storage device or devices on which the file is to be allocated or expanded. A device need not be on-line to allocate or expand a file on it.

If the RAT statement is omitted, OCAREM allocates or expands the file on any nonclass-R device or devices that are mounted (on-line) when the job is in execution. OCAREM searches the on-line devices for the smallest available space that the file fits in. † If none of the on-line devices have a block of space large enough for the file, OCAREM may segment the file across more than one device.

$RAT, $dt_1/dn_1, \ldots, dt_n/dn_n$

dt The mass storage drive hardware type. It may be any of the following. ††

853, 854, 813, 841, 863

dn The device number specified when the device was entered in the system MSD file (refer to description of ENTER card). The values may range from 1 to 362144 (decimal). If a class-R device is specified, the device number is ignored with no diagnostic.

† If two or more devices are equal, the device is selected according to the order the devices were entered in the system with an ENTER statement (that is, first entered, first selected).

†† An 814 is considered the same as two 813 units.

If two or more devices are specified on the RAT statement, the file is allocated or expanded on the first device that contains a block of space large enough for the file. If none of the devices have sufficient space, the file may be segmented across two or more of the devices specified on the RAT statement.

The parameters on the RAT statement remain in effect until they are cleared. A new RAT statement or an EOJ statement clears the previous RAT statement. Therefore, if more than one file is to be allocated or expanded on the same device or devices, only one RAT statement is required. If the files are to be allocated on different devices, a new RAT statement must precede the ALLOCATE statement. A RAT statement without parameters clears the previous RAT statement without assigning new devices (that is, any nonclass-R device which is on-line may be used).

Example:

```
$RAT,853/21,853/22          $RAT,854/23
```

RRAT

The RRAT statement is the same as the RAT statement except that RRAT specifies only class-R devices for file allocation or expansion. Any nonclass-R devices specified on a RRAT statement is ignored by the system. The class-R requirement for file allocation and expansion remains in effect until a RAT statement is used. Allocation or expansion across class-R and nonclass-R devices is not possible.

Files cannot be expanded or allocated on a class-R device unless the device is on-line.

A RRAT statement with no parameters specifies file allocation or expansion on any class-R device or devices currently on-line. A RRAT statement with parameter limits the allocation or expansion to devices that are on-line and that are specified on the RRAT statement.

Example:

```
$RRAT,853/23,853/24
```

FET

The FET statement defines file block size, privacy codes, and identification.

```
$FET,owner,name,blksize,ed,acpr,mdpr
```

owner	File owner's name. One to eight alphanumeric characters (required).
name	File name. One to thirty alphanumeric characters (required).
blksize	File block size in characters. One to six decimal digits. Block size may range from 1 to 131071 characters (required).
ed	File edition number. Must be two alphanumeric characters or one numeric character; default is 00 (optional).
acpr	Access privacy code. One to four alphanumeric characters; default is four blanks (optional).
mdpr	Modification privacy code. One to four alphanumeric characters; default is four blanks (optional).

The FET statement is used with other mass storage control statements to perform the following functions.

RAT FET ALLOCATE	Allocate space for a file. RAT statement is optional.
FET OPEN	Open a file for input or output. The mdpr parameter may be omitted on the FET statement. Any nonzero numeric character may be used for block size.
RAT FET EXPAND	Expand the size of an existing file. The RAT card is optional. Any nonzero numeric character may be used for block size on the FET card.
FET MODIFY FET	Modify the file label. The second FET card is optional. If used, it must contain all parameters (changed and unchanged) and follow the MODIFY card. Any nonzero numeric character may be used for block size on the first FET card.
FET RELEASE	To release all or part of the space allocated for a file. The block size on the FET card can be any nonzero numeric character.

The FET parameters remain in effect for all mass storage file control statements that follow it until:

1. OCAREM reads a new FET statement for a different file
2. A MODIFY statement followed by a new FET statement is encountered
3. MSOS reads an end-of-file statement

If only one mass storage file is used in a job, only one FET statement would be needed for all file control functions.

The block size parameter is checked by OCAREM only when allocating a file or when a file is modified with a new FET statement. When allocating a file, the block size should be selected in accordance with the device sector size (refer to Table 4-1).

TABLE 4-1. MASS STORAGE DEVICE CHARACTERISTICS

Device	Characters Per Sector	Words Per Sector	Sectors Per Track	Sectors Per Device	Words Per Track	Tracks Per Device	Sectors Per Cylinder
853	256	64	16	16,000	1024	1000	160
854	256	64	16	32,480	1024	2030	160
841	640	160	14	56,840	2240	4060	280
813	256	64	32	524,288	2048	16,384	4096
814	256	64	32	1,048,576	2048	32,768	4096
863	256	64	16	16,384	1024	1024	0

ALLOCATE

The ALLOCATE statement reserves (allocates) mass storage space for the file described in the FET statement that preceded the ALLOCATE statement. On new devices, OCAREM allocates space for files upwards from the lowest available track, until the device is full. On other devices, OCAREM searches the available space map in the MSD file for the smallest contiguous area that the file fits into.

Only the devices listed on the RAT statement are considered for the file. If a RAT statement is not included, only those devices currently mounted (on-line) are considered. If a large enough space cannot be located in one of the devices, the file is segmented. The largest space available is selected for the first segment. Then a search is made for an area large enough for the remainder of the file.

OCAREM allocates all segments of a segmented file on one device whenever possible, and divides a file into as few segments as possible.

$ALLOCATE, ntrks, exp, , seg, dt †

ntrks — Number of tracks or blocks to be allocated. The B prefix indicates blocks. No prefix indicates tracks. The value may range from 1 to 8388607. (required.)

exp — File expiration date. Value is in the form

yymmdd

where yy Year
mm Month
dd Day

If exp is omitted, OCAREM inserts the current date for the expiration date.

seg — Segmentation parameter. Use either NOSEG or omit the parameter.

NOSEG — No segmentation. The file must reside on one contiguous area in mass storage. The job aborts if a large enough area cannot be located on one of the devices.

Omitted — The file may be segmented on one or more devices.

dt — Type of mass storage device the file is to be allocated on. If a RAT statement is used, the dt on this card must be included and must match the dt on the RAT statement. If a RAT statement is not used and dt is omitted, OCAREM uses the device type defined in the first MST entry.

† The mode parameter is no longer applicable and is not checked. An S or a double comma may be used between exp and seg.

OCAREM allocates space by tracks. When the space is specified in blocks, OCAREM allocates the number of tracks required to hold the number of blocks specified on the ALLOCATE statement. Depending upon block size and number of blocks, extra space may be allocated. This space can be written on without expanding the file.

Examples:

```
$ALLOCATE,24
$ALLOCATE,60,,,,853
$ALLOCATE,B1598,991231,,NOSEG,841
```

The file size limits are as follows:

Max block size	131,071 characters
Max number of blocks per file	8,388,607 blocks
Max number of sectors per block	4,095 sectors
Max number of tracks per file	8,388,607 tracks
Max number of devices for segmented files	8 devices
Max number of segments per file	64 segments†

OPEN

The OPEN statement assigns a number to a mass storage file. The file number is used to identify the file for input or output within a user program. A mass storage file cannot be referenced for input or output until it is opened.

```
$OPEN,fo,use
```

fo — File number assigned for the file by the user. The number may range from 1 to 53 (decimal).

use — File protection parameter.

I	Read only file
Omitted or other	Read/write file

When a file is initially allocated, its protection is set at read/write. Either value of the use parameter may be used when opening the file. The MODIFY statement may be used to protect a file by setting its usage to read only. In such cases, the I value must always be used on the OPEN statement. Any other value causes the job to abort.

If the I parameter is used, the file may also be opened for input at the same time by other batch or priority programs which are in core (that is, the file can be shared). If the I parameter is not used, only one program at a time can open and use the file.

Any attempt to write on a file that has been opened as a read only file causes the job to abort.

The OPEN statement must be preceded by an FET statement to identify the file to be opened.

Examples:

```
$OPEN,24,I
$OPEN,21
```

†An installation option. May be less on some systems.

EXPAND

The EXPAND statement may be used to expand the amount of mass storage space assigned to an existing file. A file must be closed to expand it.

```
$EXPAND, n, seg
```

n — Number of tracks or blocks to be added to the file. The value may range from 1 to 262142 (decimal). If B precedes the n value, n is the number of blocks to be added.

seg — Segmentation parameter.

NOSEG	No segmentation. The new block of mass storage resides on one continuous area in mass storage. If contiguous unallocated space is unavailable, the new space is added as a single segment.
Omitted or other	Segmentation allowed. The new space may be added in segments and may be on other devices.

The EXPAND statement must be preceded by a FET statement to identify the file being expanded. The FET statement may optionally be preceded by a RAT statement to specify which devices the file may be expanded on.

OCAREM expands mass storage files by tracks. If blocks are specified, OCAREM expands the file by the number of tracks required to hold the specified number of blocks. Depending upon the block size and number of blocks, extra space may be added to the file that is used without another file expansion.

Examples:

```
$EXPAND, 839
$EXPAND, 8, NOSEG
$EXPAND, B80, NOSEG
```

MODIFY

The MODIFY statement may be used to change file protection, block size, or its identification and privacy codes. The MODIFY statement must be preceded by a FET statement to identify the file being modified, and the file must be closed before it can be modified.

```
$MODIFY, prot, exp, newfet
```

prot — Protection code that restricts file usage.

I	Read only file
O	Read and write file
Omitted	No change

exp — New file expiration date in the form **yy**mmdd

yy	Year
mm	Month
dd	Day

(Optional. Default is no change.)

newfet New FET statement. Any character indicates that a new FET statement follows the MODIFY statement. All of the parameters on the new FET may be changed.

Omitting the newfet parameter indicates no new FET statement follows the MODIFY statement.

CAUTION

If file blocksize is expanded beyond the mass storage devices sector boundary, data is lost. In addition, the buffer sizes must be expanded in programs currently using the file.

Examples:

```
$MODIFY,I
$MODIFY,,,F
```

CLOSE

The CLOSE statement closes a file or releases a unit record device. The file number is available for other assignments in the same job. The CLOSE statement also updates the block count (highest block written) in the file label (refer to appendix G).

```
$CLOSE,fo
```

fo File number

All user files and unit record assignments are automatically closed when an EOJ statement is read, and all mass storage file labels are updated in the LABELFILE.

Example:

```
$CLOSE,21
```

RELEASE

The RELEASE statement releases all or some of the mass storage space assigned to a file with the ALLOCATE statement. The space is made available for assignment to other files. A file must be closed before it can be released.

```
$RELEASE,amount
```

amount Amount of the allocated space to be released. The values are as follows:

ALL	Release all space and all records of the file.
UNUSED	Release all tracks that have not been used. If no tracks have been used, all but the first track are released.
nnnn	A decimal number indicating the number of tracks to be released. Only the highest unused tracks are released. If the number is preceded by a B, nnnn is the number of blocks released.
Other	Illegal

The RELEASE statement must be preceded by a FET statement to identify the file that is to have space released.

Examples:

```
$RELEASE,ALL
$RELEASE,UNUSED
$RELEASE,27
$RELEASE,B5
```

RONL

The label file has no entries for files on class-R devices that are not on-line. The RONL card should be used to ensure that the required class-R devices are on-line before opening any file that is written on a class-R device. The RONL task checks to see if the specified class-R device is on-line. If the device is on-line, RONL allows the job to continue. If the device is not on-line, RONL sends a request to the operator to mount the device and waits until the operator responds. If operator response indicates that mounting is not possible, the job is aborted.

```
$RONL,dt/dn/dt/dn,...
```

dt Device type that the file was allocated on:

853
854
841

dn Device number written on the device label when the device was entered.

When a file is segmented over more than one class-R device, each device must be on-line. Class-R files cannot be partially opened.

UTILITY FUNCTION

MSOS has the following utility routines that may be used to control I/O equipment and mass storage devices, and to transfer data between I/O units.

XFER and DUMP Statements

UTILITY Routines

MSUTIL Routines

The XFER statement transfers binary decks from the system input unit (card reader or tape) to a mass storage or tape file. The DUMP statement dumps core whenever a job aborts itself or is aborted by MSOS.

The UTILITY routines are intended for use mainly with magnetic tape units. These routines perform the following functions.

Rewind a tape

Unload a tape

Space forward

Space backward

Erase

Write EOF mark

Check tape density

Verify a new tape

Copy from one tape to another tape

Copy from card reader to tape

List a tape

List a card deck (from card reader)

Punch a card deck (from card reader or tape)

The mass storage utility routines perform the following mass storage functions.

Release expired files

Dump for backup

Dump and reload to reorganize space on a disk

Enter a new mass storage device in the system

Release a mass storage device

List a file label from the LABELFILE

List a device label from the MSD Label File

XFER STATEMENT

An XFER statement will transfer relocatable binary decks and cards from the system input file to a tape or mass storage file, or to the card punch. If a mass storage file is used, the file must be allocated with a block size of 960 characters and opened before using XFER. Output tapes will be blocked at 160 characters. XFER can be used only with batch jobs, and will transfer only binary decks (except for file 62).

```
$XFER,lu
```

lu File number. May be 1 through 53, 56, and 62.

XFER transfers relocatable binary cards following the XFER card into file lu. When an end-of-file statement or the next MSOS control statement is read (7/9 or $ in column 1), XFER writes an end-of-file mark (EOF) on file lu and backspaces over the EOF mark. † If lu is not a tape, punch, or mass storage file, the job aborts.

Example:

```
$XFER,21
```

The XFER statement may also be used to transfer binary and Hollerith cards to the system punch file (file 62). When file 62 is specified, XFER transfers all cards (MSOS control cards, source decks, binary cards, etc.) until XFER reads an end-of-file-card. If file 62 is a tape file, XFER writes an EOF on the file and backspaces, the same as for all other files.

DUMP STATEMENT

The DUMP statement causes a memory dump to be taken if a job containing the dump statement aborts. The DUMP statement must follow the JOB statement.

```
$DUMP,FD
```

FD — Full dump parameter. Dump the aborted program and the operating system.

Omitted — Take a partial dump. Dump only the aborted program.

The FD parameter applies only to batch jobs. It is ignored if the DUMP card is inserted in a priority program. The following information is provided in the dump.

1. Full dump, batch program††
   - All registers
   - All register files
   - All memory locations from address zero to the priority program area

†On mass storage, XFER writes the EOF mark on a new block. The block pointer is left positioned at the beginning of the block containing the EOF mark.

††In batch programs run with the extended core variant of MSOS, the loader converts return-jump instructions that reference executive routines to HLT instructions that reference the routines (refer to executive interrupts in section 18). These HLT instructions appear in the core dump in place of the jump instructions.

2. Partial dump, batch program †

   - All registers
   - All register files
   - All memory locations from the beginning of the common area to the priority program area

3. Priority program dump

   - All registers
   - All register files
   - All memory locations between the priority program's upper and lower limits

If the operator terminates a program at the control console, he may select no dump, a partial dump, or a full dump.

TAPE UTILITY ROUTINES

The tape utility routines must be called with a $UTILITY statement. After UTILITY has been called, individual utility functions may be selected by name with control card statements or operator statements at the console typewriter.

MSOS control statements ($ or 7/9 in column 1) and binary decks cannot be intermixed with utility statements. Utility must be terminated with an END card before any new MSOS control cards or binary decks can be input to MSOS.

```
$UTILITY,u,m
function,parameters
   .
   .
   .
function,parameters
END
```

u — Control unit. File number of the input unit containing utility function statements. The unit can be the card reader, a tape unit, or the operator's console typewriter. Default value is unit 60 (standard system input file) if the statement is from card or tape. Default is 58 if the statement is from the console typewriter.

If the console typewriter is selected, the input functions and END statement must be typed by the operator. The format of the typewriter input is described in the operator's guide.

m — Message unit. File number that the output messages are printed on. Only logical units 59 (CTO) or 61 (system output unit) may be selected. Default is 59 if the statement was typed at the console typewriter; otherwise, default is 61.

function — Name of function to be performed.

---

†In batch programs run with the extended core variant of MSOS, the loader converts return jump instructions that reference executive routines to HLT instructions that reference the routines (refer to executive interrupts in section 18). These HLT instructions appear in the core dump in place of the jump instructions.

TAPE CONTROL FUNCTIONS

The following utility functions apply to tape drives only.

Function, parameters	Description
REWIND, $u_1, \ldots, u_n$	Rewind file $u_1 \ldots u_n$ to loadpoint
UNLOAD, $u_1, \ldots, u_n$	Rewind and unload file $u_1 \ldots u_n$
FORWSPCE, u	Space one block forward on file u
FORWSPCE, u, n	Space n blocks forward on logical file u
BACKSPCE, u	Backspace one block on file u
BACKSPCE, u, n	Backspace n blocks on file u
SKFF, u	Skip forward past one end-of-file mark on file u
SKFF, u, n	Skip forward past n end-of-file marks on file u
SKFB, u	Skip backward past one end-of-file mark on file u
SKFB, u, n	Skip backward past n end-of-file marks on file u
WREOF, $u_1, u_2, \ldots, u_n$	Write an end-of-file mark on files $u_1 \ldots u_n$†
ERASE, u	Erase bad spot (6 inches of tape) on file u
ERASE, u, n	Erase n times in a row on file u
CHKDNS, u	Check the density of file u and print one of the following statements at the message unit:

Statement	7-track drive	9-track drive
IUTIL 010 LOW	200 bpi	800 cpi
IUTIL 010 MEDIUM	556 bpi	800 cpi
IUTIL 010 HIGH	800 bpi	1600 cpi

SETDNS, u, d — Sets the density on file u

d	7-track density	9-track density
L	200 bpi	800 cpi
M	556 bpi	800 cpi
H	800 bpi	1600 cpi

COPY FUNCTION

The copy function copies files or file blocks from an input tape or card reader onto an output tape, printer, or card punch. In addition, the input data may be listed on a printer.

COPY, u_1, u_2, u_3, r, op, . . . , op

u_1 File number of the input unit. May be a tape file or a card reader. This parameter is required.

u_2 File number of the output unit. May be a tape file, printer, or card punch. No carriage control characters are provided for the printer and the data is unformatted (carriage control and data is printed exactly the way it is read from the tape or card reader). If only a listing is needed, this parameter is omitted. However, its trailing comma is required. This parameter is required if u_3 is not supplied.

† An end-of-file mark is a 17, ([character) bracketed with interrecord gaps.

u_3 File number of the list unit. May be a tape file or a printer. Printer carriage control characters are added before printing or writing and the data is formatted for easy reading. If no listing is needed, this parameter may be omitted. However, its trailing comma is required if other parameters follow.

r The number of blocks or files to be read from the input unit. If an F is affixed to the r, r indicates the number of files to be read. If the F is omitted, r indicates the number of blocks to be read. R is a one- to four-digit decimal number. Default is one file if the parameter is omitted.

op Optional parameter string. These parameters and their trailing commas are optional and may be punched in any order. When one is omitted, both the parameter and its trailing comma must be omitted. The parameters are as follows:

C Specifies I/O is in character mode at tape files u_1 and u_2. Omitted specifies I/O is normal word mode at tape files u_1 and u_2. The C is ignored for 9-track tape units, card readers, printers, and card punches.

N Allows writing on u_2 beyond the end-of-tape mark. If omitted, the COPY function will terminate upon detection of an end-of-tape condition at u_2.

H or O H specifies u_3 output is formatted as Hollerith code. O specifies u_3 output formatted as octal code. Omitting this parameter specifies the records on u_3 will be formatted the same as the data read from the input file u_1.

M Select BCD read mode for input file u_1. Omitted specifies binary mode. The M value is required only for 9-track tape drives that use code conversion. For 7-track tapes, the correct mode is automatically selected by the software.

On the list file u_3, BCD (or Hollerith) characters are printed 119 characters per line. Octal digits are printed as 8 words (64 digits) per line with an octal word count at the beginning of each line. An end-of-file mark is not written on the list file after the last record on file is listed. Each line printed on u_3 is preceded by the record (block) number and length.

If the list file is a tape drive, carriage control characters are added as part of the input written on the tape. An end-of-tape mark on the list unit terminates the copy function.

The block size copied from the input unit may vary from block to block. However, the maximum size of a block is limited by the amount of core available for use by the copy function. Before reading the first block, the copy function prints the maximum block size that can be read.

I UTIL 100 MAX REC SIZE n B

n Maximum block size in words (octal)

If a block exceeds the maximum size during copying, the following occurs.

1. The oversize block is copied with the excess characters truncated.
2. An informative diagnostic is printed on the message file.
3. Copying will continue normally.

After each file is copied and when the copy function terminates, the number of blocks or files copied are written on the message unit as follows:

```
I UTIL 110 n BIN RECORDS COPIED
                or
I UTIL 110 n BCD RECORDS COPIED
```

n Number of blocks copied (decimal)

Examples:

1. Copy 96 blocks from unit 1 to unit 2 with no list output.

```
COPY,1,2,,96
     or
COPY,1,2,0,96
```

2. List 9 files on unit 3.

```
COPY,1,,3,9F
     or
COPY,1,0,3,9F
```

3. Copy 1 file from unit 1. Use character I/O and list the file in octal format.

```
COPY,1,2,3,1F,C,O
```

VERIFY FUNCTION

The verify function compares the data on two tapes or a tape and card reader. All no compare blocks are written on a third unit which may be a tape or printer.

VERIFY,u_1,u_2,u_3,r,op,...,op

u_1 and u_2 — The file numbers of the units containing the data to be compared. May be tape or card reader. These parameters are required.

u_3 — The file on which are differences are listed. May be any tape or printer. If omitted, there is no difference listed.

r — The number of blocks or files to be compared. If an F is affixed to the r, r indicates the number of files to be read. If F is omitted, r indicates the number of blocks to be read. R is a one- to four-digit decimal number which must be greater than zero. This parameter is required.

op — Optional parameter string. These parameters and their trailing commas are optional and may be in any order. When omitted, both the parameters and the trailing comma are omitted.

- C — Character I/O for files u_1 and u_2. Selects word I/O if omitted. Ignored for 9-track tapes, card readers, card punches, and printers.

R Selects reverse reads (in word mode) at files u_1 and u_2 for the comparison. The loadpoint is counted as a file mark. Omitted selects forward reads.

M Selects BCD mode for files u_1 and u_2. Omitted selects binary mode. The M value is required only for 9-track tape drives that use code conversion. For 7-track tapes, the correct mode is automatically selected by the software.

During processing, a descriptive diagnostic is written on file u_3 (message unit) if any of the following comparison errors occur.

Error	Message
Recording mode comparison	I UTIL 102 MODE ERROR RECORD n
Word or character compare error	I UTIL 103 CONTENT ERROR RECORD n WORDx
Block length compare error	I UTIL 104 LENGTH ERROR RECORD n

n Block number (decimal)
x Word number (octal)

Only the first six compare error messages per block are printed on OUT. Then verification of the next block starts. This process continues until all blocks are verified.

When the verify function terminates, or after each file is verified, the number of blocks verified is printed on the message file.

If an end-of-file is encountered on file u_1 or u_2, the following is sent to the message unit.

I UTIL 105 EOF ABSENT u RECORD n

u u_1 or u_2
n Block number

Processing continues as if it were a matching end-of-file. If an end-of-file is encountered on both units, the end-of-file message and the block count are sent to the message file and to u_3.

An end-of-tape condition on file u_3 causes termination of the list, but the verification continues. An end-of-file mark is written on the file only if the list is terminated by an end-of-tape condition.

Variable block sizes will be compared by the verify function. However, the maximum block size that can be compared is limited by the amount of core that is available for the verify function to use.

At the beginning of each verify run, the verify function prints the maximum block size that can be compared on the message file.

I UTIL 100 MAX REC SIZEn

n Maximum block size in number of words (octal).

If the block size is exceeded, the block is truncated and the truncated portion is not verified. An informative diagnostic is printed and normal processing continues.

Examples:

1. To compare a file on unit 1 with the file on unit 2 and list the file on the system output file.

```
VERIFY,1,2,61,1F
```

2. To compare the first five blocks of a file on unit 27 with the first five blocks on unit 36, and with no list output:

```
VERIFY,27,36,,5
```

MASS STORAGE UTILITIES

The mass storage utility routines must be called with an MSUTIL statement. After MSUTIL has been called, individual utility functions may be selected by name with control card statements or with operator statements at the console typewriter.

MSOS control statements ($ or 7/9 in column 1) and binary decks cannot be intermixed with utility statements. MSUTIL must be terminated with an END, SCOPE, or STOP statement before any new MSOS control statements or binary decks can be input to MSOS.

```
$MSUTIL,u
function,parameters
   .
   .
   .
function,parameters
END
```

u — Control unit. File number of input file for utility function statements. The unit may be a card reader, a tape file, or the operator's console typewriter. Default value is file 60 (standard system input file) if the statement was read from file 60, and 58 if the statement was typed at the console typewriter.

If the console typewriter is selected as the control unit, the function and end statements must be typed by the operator. The format of the typewritten input is described in the operator's manual.

function — Name of function to be performed.

PURGE FUNCTION

The purge statement scans the file label directory and releases all user files that have reached their expiration date or a specified edition of a system library.

PURGE, ed, code

ed — Edition number. All system or auxiliary library files having edition number ed are released. If this parameter is omitted, all user files with an expiration date that is one day past the current date are released.

code — Combined MSOS system access and privacy code that was set at system installation time. The access code must appear first and the two codes must not be separated by a space or comma.

Example:

PURGE, A2, MSOSPRIV

PURGE, , ACE/DUCE

ENTER FUNCTION

The enter statement enters a new device in the system by writing a mass storage device label in the MSD file, a device label on track 0 for the device (refer to section 3), and an R-label on class-R devices. † This makes the device available for use by the system.

To enter a device nonclass-R:

ENTER, , dt/dn, , , , lta, hta, exid

To enter a device as class-R:

ENTER, R, dt/dn, lra, ntr, pass-code, lta, hta, exid

dt — Device type. It may be any of the following hardware type numbers.

813
814
841
853
854
863

dn — Device number may be any decimal identification number between 1 and 262143 that the user wishes to assign and which is unique to this device.

lra — First track number reserved for the R-label on class-R devices. A 1- to 5-digit decimal number. Must be equal to or greater than the lta parameter. ††

ntr — Number of tracks to be reserved for the R-label on class-R devices.†† A 1- to 3-digit decimal number with range of 1 to 511. The number of tracks needed for the R-label can be calculated as follows:

Device	
841	ntr = (3+2NF) ÷ 14
853	ntr = (2+4NF) ÷ 16
854	ntr = (3+4NF) ÷ 16
813	ntr = (12+4NF) ÷ 32
863	ntr = (2+4NF) ÷ 16

NF is the maximum number of files to be allocated on the device.

† R-label may be placed any where on the device above track zero.
†† lra + ntr ≤ hta +1.

pass-code A 1- to 8- character alphanumeric security code which is written into the device label on class-R devices.† Blanks are written for the default case.

lta Lowest track address that may be assigned to user file. Default is track 1.

hta Highest track address that may be assigned to a user file. If omitted, the default value is the last track on the device. Refer to the description of the allocate card for a list of the number of tracks on the different device types.

exid A 1- to 6-character alphanumeric external identification code. The code is written on the outside of the pack for identification. If the code is omitted in the statement, the operator is requested to supply a code at the console typewriter.

Examples:

```
ENTER, R,841 /32, 1, 3,, 1,, RIVETS
ENTER,,841/33,,,,,,FILE6
ENTER,,841/34
```

When MSUTIL reads an ENTER statement, it searches for a device without a label that is on the type of drive specified on the ENTER statement. MSUTIL asks operator permission to enter the first such device found. Searching continues if the operator rejects the first request.

If a device cannot be located, the operator is requested to specify the drive that contains the device to be entered or to mount a new device.

DELETE FUNCTION

The delete function removes the device label from the mass storage device label file (MSD file). The device is no longer accessible by MSOS.

DELETE, dn/dt

dn/dt The device number and device type used to enter the device.

All files on a device must be released before a device can be deleted. The files on the device that need to be saved can be dumped, released, reallocated, and loaded on a different device.

DUMP FUNCTION

The dump statement dumps a mass storage file onto a tape. This function may be used for backup purposes, or when used in conjunction with the load function, to reorganize space on a device so that all available space is in one contiguous area. A file must be opened before it can be dumped. The dump function dumps all segments of a segmented file.

DUMP, lu, fo

lu Tape file that the mass storage file is to be dumped on.

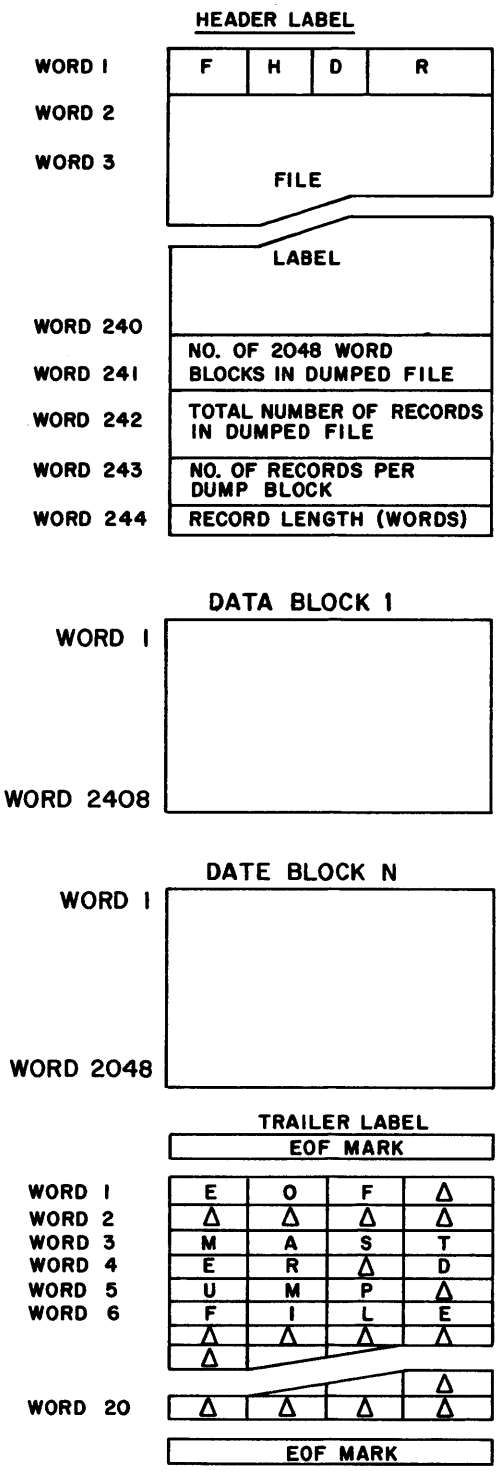
fo File number of the file to be dumped.

The file label from the LABELFILE (appendix G) is dumped on unit lu as part of the tape header label. The format of the header and trailer labels are shown in Figure 5-2. This format is compatible with the MASTER operating system. A file dumped by MSUTIL dump function can be loaded with a MASTER *FMU LOAD card.

Example:

```
DUMP, 20, 21
```

† Pass code is for compatibility with MASTER systems. It is not referenced by MSOS.



Each data block is 2048 words long. The number of records per block is dependent upon the device type as follows:

Device	Records/Block
853	2
854	2
863	2
813	1
814	1
841	1/2

When the DUMP file is written, the trailer label is written and the tape is backspaced past the first EOF. Thus, if another DUMP request is made, that file is written over the trailer. In this way, a string of dumped files are written with one trailer.

Figure 4-1. MSUTIL File Dump Format

LOAD FUNCTION

The load statement loads files from tapes that were written in the MSUTIL dump format. The dump may have been taken by either the MSOS or the MASTER† operating system.

LOAD, lu, fo

lu File that the dump tape is mounted on.

fo File number of an open mass storage file into which the dump is loaded.

The load function searches the tape for a file with a header label that matches the file label for file fo. The load function then loads the file from file lu into the mass storage file area allocated for file fo.

To allocate a new file, the old file must be released and a FET, ALLOCATE, and OPEN statement used before reloading the dumped file. The following descriptions for the new file (file fo) must match those on the tape file header label.

Name

Owner

Edition

Device type

Access privacy code

Modification privacy code

The block size must be the same as in the original mass storage file, and the number of blocks allocated must be equal to or greater than the number of blocks written on the old file or an error message occurs.

The sequence for reorganizing files on a device is as follows.

1. Open all of the files on the device.
2. Dump all files.
3. Release all dumped files.
4. Re-RAT, re-FET, re-ALLOCATE, and re-OPEN each of the files.
5. Load each of the files.

Example:

LOAD, 20, 21

LIST MSD FUNCTION

The LIST MSD statement lists all or part of the entries in the mass storage device label file.

LIST, lu, MSD, dt/dn
or
LIST, lu, MSD

lu Logical unit for the list output.

dt/dn Device type and device number to have its label listed. If dt/dn is omitted, the labels for all devices in the system are listed.

†With an *FMU Dump Card.

A storage map is also provided with each label. The storage map is a list of each assigned and unassigned (available) track on the device.

LIST FLD FUNCTION

The LIST FLD statement lists all or part of the entries in the mass storage LABELFILE in the order that they appear on the file.

```
LIST,lu,FLD,dt/dn
     or
LIST,lu,FLD,owner,name,edition
     or
LIST,lu,FLD,owner
     or
LIST,lu,FLD
```

lu	File number for the list output.
dt/dn	Device type and number. If this parameter is used, the labels of all files and file segments on the device are listed.
owner	The file owner name. †
name	The file name. †
edition	The edition number of the file.

If a file name, owner, and edition number is specified, only the label for that file is listed. If the file name and edition are omitted, labels for all files with the specified owner name are listed. If owner, name, edition, and dt/dn parameters are omitted, the entire label file is listed. The access and privacy codes are omitted from the file label listings.

MAP FUNCTION

The MAP statement provides a map of the track usage of a specified mass storage device.

```
MAP,lu,dt/dn
```

lu	Logical unit (or file) the map is listed on.
dt	Mass storage device type.
dn	Mass storage device number.

The map consists of a listing of the file label directory (IDFILE) entries for each file on the device. The file label entries are sorted by track numbers.

Example:

```
MAP,61,854/803
```

---

† Refer to FET card for description of owner and name.

OWNER	FILE NAME	ED	C-DATE	E-DATE	L-DATE	USE-CT	F-SIZE	B-SIZE	BLK-CT	SEG-CT	SEG	DT	DN	LTL	SL
RTS-MSIC	LABELFILE	00	^ 0-	86-07-38	^ 0-	0	60	392	24	1	1	854	8541	2	60
RTS-MSIC	IDFILE	00	^ 0-	86-07-38	^ 0-	0	6	480	2	1	1	854	8541	62	6
RTS-MSIC	MSGFILE	00	^ 0-	86-07-38	01-11-74	0	7	4	5	1	1	854	8541	68	7
MSOS	L-MSIO	00	08-01-73	99-99-99	06-15-74	1574	15	1280	0	1	1	854	8541	75	15
RTS-MSIC	RESFILE	SS	06-21-74	99-99-99	07-15-74	3	9	4	138	1	1	854	8541	90	9
RTS-MSIC	RESFILE	SS	06-21-74	99-99-99	07-15-74	3	27	4	429	1	1	854	8541	99	27
RTS-MSIC	LIBDIRFILE	SS	06-21-74	99-99-99	07-15-74	3	1	500	3	1	1	854	8541	126	1
RTS-MSIC	LIBFILE	SS	06-21-74	99-99-99	07-15-74	3	86	960	344	1	1	854	8541	127	86
MSOS	FILE54	00	04-17-74	99-99-99	06-16-74	5	200	512	0	1	1	854	8541	213	200
MSOS	FILE55	00	04-25-74	99-99-99	06-16-74	3	450	512	0	1	1	854	8541	413	450
MSOS	FILE56	00	04-17-74	04-17-74	06-16-74	4	100	960	0	1	1	854	8541	863	100
RTS-MSIC	ABSFILE	SX	07-15-74	99-99-99	07-15-74	1	154	4	2463	1	1	854	8541	963	154
RTS-MSIC	RESFILE	SX	07-15-74	99-99-99	07-15-74	1	9	4	137	1	1	854	8541	1117	9
RTS-MSIC	LIBFILE	SX	07-15-74	99-99-99	07-15-74	1	274	960	1095	1	1	854	8541	1126	274
RTS-MSIC	LIBDIRFILE	SX	07-15-74	99-99-99	07-15-74	1	4	500	30	1	1	854	8541	1400	4
MSOS	AUXLIB	00	07-15-74	99-99-99	07-15-74	2	58	960	230	1	1	854	8541	1404	58
RTS-MSIC	ABSFILE	EX	07-15-74	99-99-99	07-15-74	2	28	4	443	1	1	854	8541	1462	28
RTS-MSIC	RESFILE	EX	07-15-74	99-99-99	07-15-74	2	10	4	149	1	1	854	8541	1490	10
RTS-MSIC	LIBFILE	EX	07-15-74	99-99-99	07-15-74	2	87	960	345	1	1	854	8541	1500	87
RTS-MSIC	LIBDIRFILE	EX	07-15-74	99-99-99	07-15-74	2	1	500	3	1	1	854	8541	1587	1
* FREE *													8541	1588	442

OWNER	Name of the file owner	BLK-CT	The block count. The highest block number written.
FILE NAME	Name of the file.		
FREE	Unused tracks.	SEG-CT	The number of segments the file is divided into.
ED	The file edition.	SEG	Which segment this entry is for. Each file segment has a separate entry.
C-DATE	The file creation date.		
E-DATE	The file expiration date.	DT	The type of device the files are on.
L-DATE	The last date the file was used.	DN	The number of the device containing these files.
USE-CT	The number of times the file was used.	LTL	The track number that the file, or the segment of the file, starts on.
F-SIZE	The file size in tracks, including all segments		
B-SIZE	The block size used in the file.	SL	The number of tracks or segments in the file.

Figure 4-2. Sample Device Map

SAMPLE JOB DECKS 5

BATCH JOBS

The following are examples of MSOS control statements used to compile, load, and execute batch programs.

1. Assemble a COMPASS program, punch a binary program deck (object program), and print a COMPASS output listing at the system list output unit.

$JOB,ACT1,ASSEMBLE,10	Job name is assemble; ACT1 is account number, 10 is time limit
$COMPASS,P,L (COMPASS subprogram deck 1) . . . (COMPASS subprogram deck n) FINIS	Library program name card calls the COMPASS assembler from the library to assemble the program and punch a binary output deck. †
77 88	End-of-COMPASS program decks. End-of-file card.
$EOJ	End-of-job, print job accounting information.

2. Execute the object binary program deck produced by the COMPASS assembly in example 1. Open a mass storage input file and an output file for the job.

$JOB,ACT1,SCHEDULES,10	Job name is schedules.
$DUMP	Dump program if job aborts.
$FET,SUE,UPDATES,640,1,PRV1	Job input file.
$OPEN,10,I	Open the input file.
$FET,BEVERLY,SCHEDULES,640, 2,PRV2,MPV2	Job output file.
$EXPAND,100	Expand and open the output file.
$OPEN,11 (Binary deck with IDC header card)	COMPASS program deck from the assemble job.
$RUN	Execute program.
$FET,BEVERLY,SCHEDULES,640 2,PRV2,MPV2	
$CLOSE,11	Close schedule file.
$RELEASE,UNUSED	Release unused tracks on output file.
77 88	
$EOJ	

†Refer to the COMPASS Reference Manual for a description of the COMPASS card.

3. Compile an ANSI COBOL program, write object code on a mass storage file, and print a COBOL output listing at the system output unit.

```
$JOB,ACT1,TIME-CARD-PROCESSOR,1
$RAT,841/20,841/21                    Select devices for output file.
$FET,LINDA,TIME-CARDS,960.1,          Defines an LGO output file. LGO file
   PRV3,MPV4                          block size must always be 960 char-
                                      acters.
$ALLOCATE,10,741231,,,841             Allocate output file.
$OPEN,12                              Open output file.
$UCBL,L,X=12                          Library program name card writes
                                      object code on file 12. †
(COBOL program decks)
         FINIS                        End of COBOL program.
77
88
$EOJ
```

4. Execute the COBOL program compiled in example 3, using the output file prepared by the COMPASS program in example 2 as the input file. Write output on a tape file.

```
$JOB,ACT1,NEW-SCHEDULES
$FET,BEVERLY,SCHEDULES,640,2,         Define output file from schedules job
   PRV2,MPV2                          as the input file.
$OPEN,13,I
$FET,LINDA,TIME-CARDS,960,1,          Use the output file from the time card
   PRV3,MPV4                          processor job as the COBOL object
                                      program file.
$OPEN,14
$EQUIP,20=MT                          Define unit 20 as a magnetic tape.
$LOAD,14                              Load COBOL program.
$RUN                                  Execute COBOL program.
77
88
$EOJ
```

†Refer to the ANSI COBOL Reference Manual for a description of the UCBL statement.

5. Compile and execute an ANSI FORTRAN program. Prepare a punched output deck of the object program. Use an auxiliary library for object-time routines and use tape units for input data and the output file.

```
$JOB,ACT1,F21-PROJECT
$CTO,MOUNT F21-INPUT TAPE ON
   DRIVE 3
$EQUIP,20=MTC1E1U03,21=MT          20=input file; 21=output file.
$UFORTRAN,X,L,P                    Write object code on load-and-go file
(FORTRAN program deck)             (file 56), and punch a binary object
      FINIS                        program deck.†
$FET,MSOS,AUXLIB,960,1,UF          Define auxiliary library file.
$OPEN,10                           Open auxiliary library file.
$AUX,10
$LOAD,56                           Load object code in core.
$RUN                               Execute object code.
$UNLOAD,21                         Unload output tape.
$PAUS,MARK TAPE F21 AND STORE      Pause so operator can remove the F21
                                   tape.
$UNLOAD,20                         Unload input tape.
77
88
$EOJ
$JOB...                            Next batch job.
```

6. Compile an ANSI FORTRAN program. Absolutize the object code and place the absolutized code on a mass storage file for quick loading by ABSTSK. Do not execute the job at this time.

```
$JOB,ACT1,ACRONYM-GENERATOR,10
$RAT,841/21                        Select a device for the absolutized
                                   FORTRAN object code file.
$FET,NANCY,ACRONYMS,640,AX,        Define an output file for absolutized
   PRV6,MPV6                       FORTRAN object code.
$ALLOCATE,B40,741231,,NOSEG,841
$OPEN,22                           Open file for absolutized binary object
                                   code.
$XFER,56                           File 56 is load and go scratch. XFER
                                   transfers only binary cards.
M22                                MAIN overlay header card. ††
```

†Refer to the ANSI FORTRAN Reference Manual for a description of the UFORTRAN statement.

††The MAIN overlay header card specifies on which file the loader will assemble the absolutized program before loading the program in core. If the MAIN card is omitted, the loader uses scratch file 55. Refer to section 7 for a description of the MAIN overlay card.

```
$UFORTRAN,X,L                    X parameter writes binary output on
                                 file 56.
(FORTRAN program deck)           Acronym generator program
$LOAD,56,M                       Dummy load to assembly absolutized
                                 code on file 22 in accordance with the
                                 MAIN card.
77
88
$EOJ
```

7. Prepare an absolutized binary program from a relocatable binary program deck and place the absolutized program on a mass storage file for quick loading by ABSTSK.

```
$JOB,ACT 1,QUICK-UPDATE,10
$FET,PUBS,UPDATE,640,01,PRV7,
   PRV7
$ALLOCATE,B10,741231,,NOSEG,841  Allocate and open a file for the
                                 absolutized program.
$OPEN,25
$LOAD                            Dummy load to assemble absolutized
                                 code on a file.
M25                              MAIN overlay header card specifies
                                 file 25 for absolutized object code.
(Binary deck with IDC header card)  Program deck
77
88
$EOJ
```

8. Execute an absolutized program file.

```
$JOB,ACT1,LOCATE-MANUAL,20
$FET,TONI,PUB-NUMBERS,640,
   APV9,MPV9
$OPEN,26                         Open file containing absolutized program.
$ABSTSK,26,P1,...,PN             Load absolutized code from file 26.
                                 P1 through PN are parameters passed
                                 to the program.
(data cards)                     Optional
77
88
$EOJ
```

INITIALIZING PRIORITY PROGRAMS

The following are examples of jobs which load and initialize priority programs in core. Refer to section 15 for additional description of priority program loading.

1. Assemble a COMPASS program and load it as a priority 2 program.

```
$JOB,ACT1,BUZZWORD-DECODER,10
$FET,JOY,DECODER,960,AY,PRV8,      960-character block required for LGO
   MPV8                            files.
$ALLOCATE,B10,760604,,NOSEG
$OPEN,23
$COMPASS,X=23,L,R                  X parameter puts LGO output on file 23.
(COMPASS deck)
$CLOSE,23
$RELEASE,UNUSED
77
88
$EOJ
$PRIORITY,P2
$FET,JOY,DECODER,960,AY,PRV8
$OPEN,26
$LOAD,26                           Load priority program.
$RUN                               Initialize priority program.
77
88
$EOJ
```

2. Load a priority program from a relocatable binary deck.

```
$PRIORITY,P4                       Relocatable binary deck with IDC header
(binary deck)                      Program deck
$RUN
77
88
$EOJ
```

UTILITY FUNCTIONS

The following are examples of using the MSOS storage utility functions to copy, print, dump, and load files.

1. Print a tape file.

```
$JOB,ACT3,PRINT
$CTO,MOUNT LIST-A TAPE ON DRIVE 3
$EQUIP,6=MTC1E6U3
$UTILITY
COPY,6,,61,1F                    File 61 is system output file.
END
77
88
$EOJ
```

2. Copy a card deck onto a mass storage file and print a second card deck which follows the first deck.

```
$JOB,ACT3,COPY-REPRINTS
$RAT,841/20                      Specify a mass storage device for the
                                 file.
$FET,PAT,REPRINTS,960,R1,APV2,
   MPV2
$ALLOCATE,100,760704,NOSEG,841
$OPEN,44                         Open reprints file.
$XFER,44
(Reprints deck)
$CLOSE,44
$RELEASE,UNUSED                  Release all space not written on in the
                                 reprints file.
77                               Task separator, optional.
88
$UTILITY                         Second group of tasks in the job.
COPY,60,,61,1F
(List deck)
77
88                               EOF indicates end of the file being copied
END
77
88
$EOJ
```

3. Punch a tape file.

```
$JOB,ACT3,PUNCH
$CTO,MOUNT PERS-23 TAPE ON DRIVE 3
$EQUIP,21=MTC2E7U3
$UTILITY
COPY,21,62,,1F                    62 System punch unit.
END
77
88
$EOJ
```

4. Copy a tape with 25 files and verify the results.

```
$JOB,ACT3,COPYFILES
$CTO,MOUNT FILES-A TAPE ON TAPE
   DRIVE 1.
$PAUS
$EQUIP,01=MTC0E1U1,02=MT          01 Tape with files; 02 new tape.
$UTILITY
COPY,1,2,,25F                     Copy from tape 1 to tape 2.
VERIFY 1,2,61,25F,B               Compare tapes 1 and 2 in back-
                                  wards mode. Write compare
                                  errors on file 61.

UNLOAD,1,2
END
77
88
$EOJ
```

5. Dump and reload three mass storage files on the same device in order to reclaim unused (released) tracks between files and make all files contiguous.

```
$JOB,ACT3,REALLOCATE
$FET,HARRY,BLONDS,640,6,A1,A1        Define first file on the device.
$OPEN,1
$FET,HARRY,BRUNETTES,640,3,B,B       Define second file on the device.
$OPEN,2
$FET,HARRY,RED-HEADS,640,1,C,C       Define third file on the device.
$OPEN,3
$EQUIP,10=MTC1E1U3                   Dump tape.
$MSUTIL
DUMP,10,1
DUMP,10,2
DUMP,10,3                            Dump redheads.
END
$CLOSE,1
$CLOSE,2
$CLOSE,3
$FET,HARRY,BLONDS,640,6,A1,A1
$RELEASE,ALL
$FET,HARRY,BRUNETTES,640,3,B,B
$RELEASE,ALL
$FET,HARRY,RED-HEADS,640,1,C,C
$RELEASE,ALL
$REWIND,10                           Rewind dump tape.
$RAT,841/20                          Select same device for new files.
$FET,HARRY,BLONDS,640,6,A1,A1
$ALLOCATE,200,760704,,NOSEG,841      Reallocate blonds.
$OPEN,4
$FET,HARRY,BRUNETTES,640,3,B,B
$ALLOCATE,75,760704,,NOSEG,841
$OPEN,5
$FET,HARRY,RED-HEADS,640,1,C,C
$ALLOCATE,25,760704,,NOSEG,841
$OPEN,6
$MSUTIL
LOAD,10,4
LOAD,10,5
LOAD,10,6
END
77
88
$EOJ
```

RELOCATABLE BINARY PROGRAM DECKS 6

BINARY DECKS

The compilers and assemblers used with MSOS produce assembled output (object code) in relocatable binary format. The relocatable binary output may be directed to the card punch to produce punched card output, or the output may be written on a mass storage file or a tape file.

The relocatable binary output for each program consists of a separate deck for each subprogram in the program. Each deck is headed by an IDC (identification) card and ended with the TRA (transfer address) card. Each deck may contain symbolic reference to addresses in other subprograms and to library routines. The library routines are loaded and linked when the program is loaded for execution.

To execute a program, the relocatable binary output deck must be input to the relocatable loader (refer to section 9). The relocatable loader converts the relocatable addresses to absolute binary addresses, and loads the program and all referenced library routines in core for execution. The LOAD card calls the loader which loads the program decks following the LOAD card, or loads programs from a file specified on the LOAD card.

RELOCATABLE BINARY CARDS

There are three types of relocatable binary card images.

1. Cards produced by the assembler or compiler
2. Optional cards that are punched and inserted in the deck by the operator
3. Overlay cards used as headers when overlay and overlay segments are used in a program

The following is a summary of the binary cards produced by the assemblers and compilers.

IDC card	Program identification card produced by the compiler or assembler. It must be the first card in each subprogram deck. The only exception is the overlay cards used with program overlays.
RIF card	Relocatable information card. Contains 32 relocatable binary instruction or data words per card.
EPT card	Entry point card. Contains symbolic entry point names that may be used as entry points in the subprogram by the other subprograms. Each name is followed by its relocatable address. Library subprograms and routines are called with the entry point name used on the first EPT card in the deck.

XNL — External name and linkage card. Lists external symbolic addresses referenced within the subprogram. These addresses may be entry points in other subprograms or in library routines.

BDT card — Blocked data table card. Contains name and length of each labeled data block field used in the program. Produced only for ANSI COBOL or ANSI FORTRAN programs. Labeled data blocks are illegal in all but these programs.

LRL card — Local reference list card. When a program references a symbolic address before the address is defined, the compiler or assembler produces an LRL card for the symbolic address. The LRL card contains the relocatable address that defines the symbolic address.

TRA card — Transfer address card. Indicates the end of a subprogram deck. Must be the last card in each deck. If the subprogram contains the program's main or secondary entry point, the symbolic entry point name appears on the TRA card.

The following cards are punched for output deck identification purposes. The identification information is punched in large legible block letters across the center of the card.

Job sequence card — Job sequence number. Last card punched in a job output deck. Contains the sequence number of the job that punched the deck so that the job's list and punch output can be matched. This card is offset and can be used as an end-of-file card.

Flip card — First two cards punched in an MS FORTRAN binary output deck. First flip card contains the subprogram name and the second file card contains the compilation date and the library edition number. This card is ignored by the relocatable loader.

The following is a summary of the binary cards that may be punched and added to a subprogram deck by the operator.

LED card — Loader equipment declaration card. Assigns a logical I/O unit to a specific hardware unit (by channel, unit, equipment number) or to a specific hardware type (by hardware type number). The special I/O unit assignment applies only for the program being loaded.

EXS card — External symbol card. Declares additional symbolic addresses or library routines as external to the subprogram, or equates several external symbols to the same entry point in another subprogram.

ELD card — End load card. Indicates the end of the deck being loaded from the card reader. Does not indicate end-of-job or end-of-file. Follows a TRA card. Can be used only when a program calls the loader to load a binary deck from the card reader.

SNAP card — Snap dump card. Produces dumps of user specified memory locations while the program is in execution.

OCC card — Octal correction card. May be used to change the contents of any of the memory addresses in the program while the program is being loaded. The change applies only to the current load. Provides a quick program fix without recompiling the source deck.

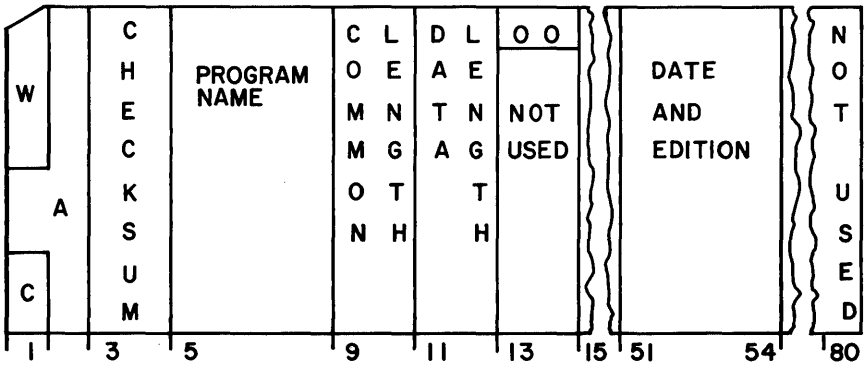
The following binary cards are used in overlay programs. These cards are the headers for the main, overlay, and segmented sections of the overlay subprograms. They precede the IDC card in each subprogram. The overlay cards are described in detail in section 7.

Main card	Indicates that the subprogram following this card is the main program.
Overlay card	Indicates that the subprogram following this card is an overlay section of the main program.
Segment card	Indicates that the subprogram following this card is a segment of an overlay of the main program.

IDC CARD

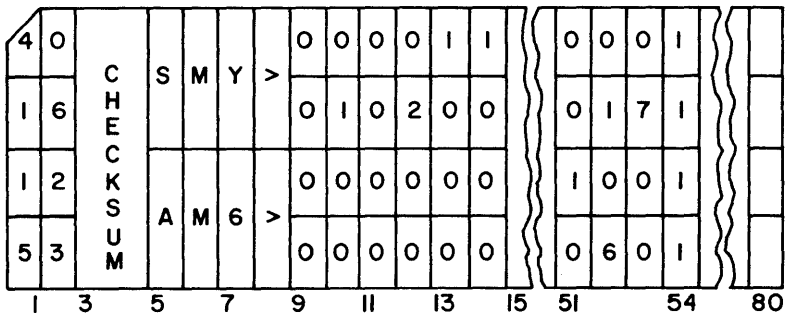
The IDC card is the subprogram identification card. It must be the first card in the subprogram deck, with the exception of overlay cards (refer to section 7). In addition to the subprogram name, the IDC card specifies the subprogram length, the data area length, and the common area length. In a deck containing more than one subprogram, the first data area size specified on an IDC card is the maximum size that can be used in any of the remaining subprograms.

In addition to supplying the subprogram name and core requirements, the IDC card causes MSOS to call the relocatable loader which absolutizes the relocatable addresses and loads the subprogram following it for execution.



Field	Significance
W	Six-bit field that contains 41_8 (octal). Identifies the card as an IDC card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates loader should not make a checksum test using columns 3 or 4. A 5 indicates the loader would make a checksum test using columns 3 and 4.
A	15-bit field that contains the subprogram length in words. The most significant digit (msd) is in column 1.
columns 3 and 4	24-bit sum of columns 1, 2, and 5 through 80. An end around carry is used if overflow occurs.
columns 5 through 8	Subprogram name. 1 to 8 internal BCD characters left justified. If the name has fewer than 8 characters, octal 60's complete the field.
columns 9 and 10	Number of storage locations reserved for common memory; must be unpunched in priority program.
columns 11 and 12	Number of storage locations reserved for data.
columns 13 and 14	A one or a zero (punch or no punch) in the top row of both columns indicates starting address of common memory. 0 Starting address of common memory not adjusted. 1 Starting address of common memory adjusted to follow MSIO overlays.
columns 15 through 50	Not used.
columns 51 through 54	Date and edition (optional) mmddyyed (refer to ACCOUNTS Table, section 16.
columns 55 through 80	Not used.

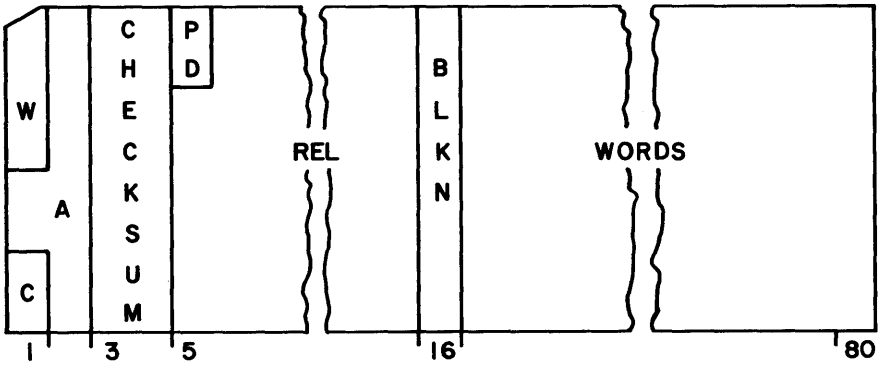
Example:



The subprogram SAMMYS is 10623_8 words long; common memory is 100_8 words long and data is 200_8 words long. The common memory starting address is adjusted. The date is 8/16/70 and edition number is 99.

RIF CARD

The RIF card contains up to 32 24-bit relocatable binary instruction or data words, and a 4-bit relocation character for each word. The relocation character specifies how its corresponding word is to be relocated. These cards may appear in any order in the subprogram deck.



Field	Significance
W	Six-bit field that identifies the card as an RIF card and specifies the number of instruction or data words on the card. May range from 1 to 40.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using columns 3 and 4. A 5 indicates the loader should make a checksum test using columns 3 and 4.
A	15-bit field containing the relocatable address of the first instruction or data word on the card. The address of the second word is a +1, the third word is a +2, etc. The msd is a column 1. When the subprogram is relocated, the relocation address is supplied by the loader and added to this address.
columns 3 and 4	24-bit sum of columns 1, 2, and 5 through 80. An end-around carry is used if overflow occurs.
PD	Four-bit field which specifies whether field A contains a program address or a data address. A 2 indicates a program address, a 3 indicates a labeled data address, and a 4 indicates a data address.

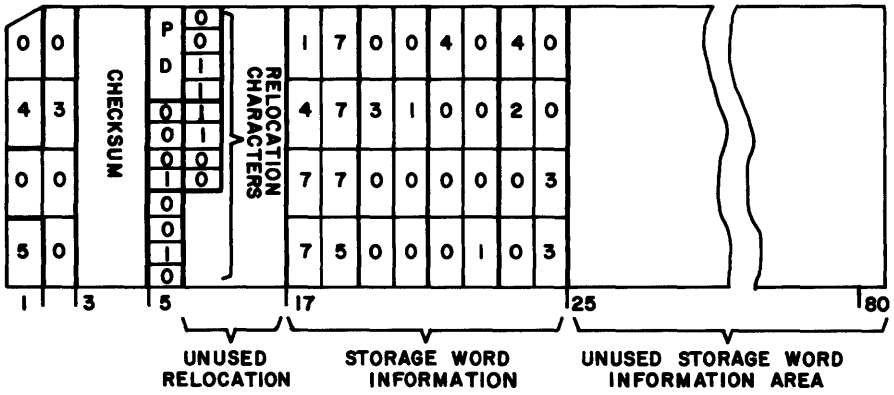
Field	Significance
REL	Contains one four-bit relocation character for each instruction or data word on the card. Modification of each word is in accordance with its relocation character as follows:

Relocation Character	Significance
1000	Indicates that data block increment (or data block decrement) is to be changed by the fwa (or complement fwa) of a new block.
0000	Unused; constitutes an error.
x001	No modification (absolute address).
x010	Instruction; increment relocation address with bits 14 through 00 of the word.
x011	Common block; increment relocation address with bits 14 through 00 of the word.
x100	Data block; increment relocation address with bits 14 through 00 of the word.
x101	Instruction; decrement relocation address with bits 14 through 00 of the word.
x110	Common block; decrement relocation address with bits 14 through 00 of the word.
x111	Data block; decrement relocated address with bits 14 through 00 of the word.

In each word that x=0, the modified 15-bit relocated word address replaces the old address in bits 14 through 00. In each word that x=1, the modified 17-bit character address replaces the old address in bits 16 through 00.

BLKN	Contains block number for relocation of labeled data.
columns 17 through 80	Up to 32 storage words. The address portion of the first word (bits 14 through 00) is modified according to the first relocation character in the relocation field; the address field of the second word is modified according to the second relocation characters, etc.

Example:



Relocatable Address	Relocation Character	Relocatable Word
00300	0001	14777775
00301	0010	03000100
00302	0011	40000001
00303	1100	42000033

If the loader relocates the subprogram so that:

Program starts at 60000

DATA starts at 57400

COMMON starts at 4000

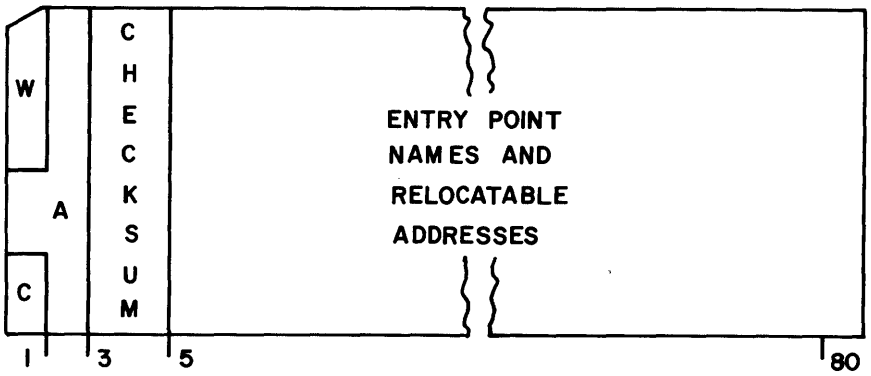
The results after relocation are:

Address	Word
60300	14777775
60301	03060100
60302	40014002
60303	42276033

EPT CARD

The EPT cards list the symbolic names of the entry points in the subprogram. Each entry point is followed by its relocatable address. Other subprograms must use one of these addresses to enter the subprogram. These cards may appear anywhere in the subprogram deck, but must be in correct sequence.

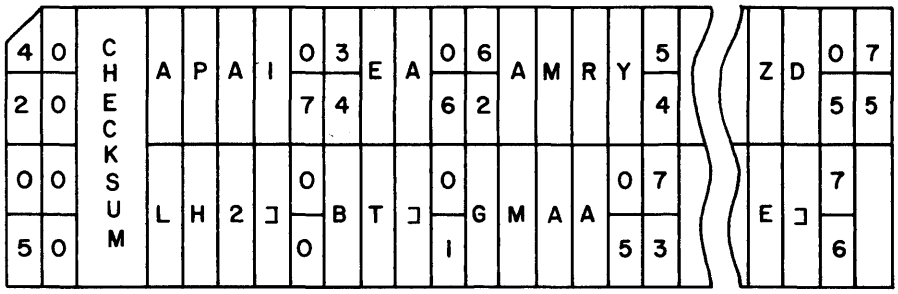
When a subprogram (or routine) resides on the system library or an AUX library, one EPT card must be located between the IDC card and the first RIF card. This EPT card contains the primary entry points that are the names used to call the subprogram from the library. A subprogram cannot be called from the library unless it has a primary entry point. All other entry points are secondary entry points and can be referenced only after the subprogram is loaded.



Field	Significance
W	Six-bit field that contains an octal 42. Identifies the card as an EPT card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using column 3 or 4. A 5 indicates the loader should make a checksum test using columns 3 and 4.
A	15-bit field that contains the card sequence number. The field contains a zero for first card, a one for second card, etc. Column 1 is the most significant digit of the number. Cards must be in correct sequence, but do not have to be next to each other. The cards may be anywhere between the IDC and TRA card.
columns 3 and 4	24-bit sum of columns 1, 2, and 5 through 80. An end-around carry is used if overflow occurs.
columns 5 through 80	Variable length fields, each containing an entry point name in BCD and its address. An entry point name may be 1 to 8 alphanumeric characters, when it is fewer than 8 characters. 72_8 terminates the field. The address field for each entry point name begins after the last character of the name or after 72_8 and is 18 bits long.

The address, with any leading zeros, occupies the least significant 15 bits of the field. Bits 18 through 16 are always zero, because entry points cannot be character address. An entry point name and address field cannot be continued from one card to another.

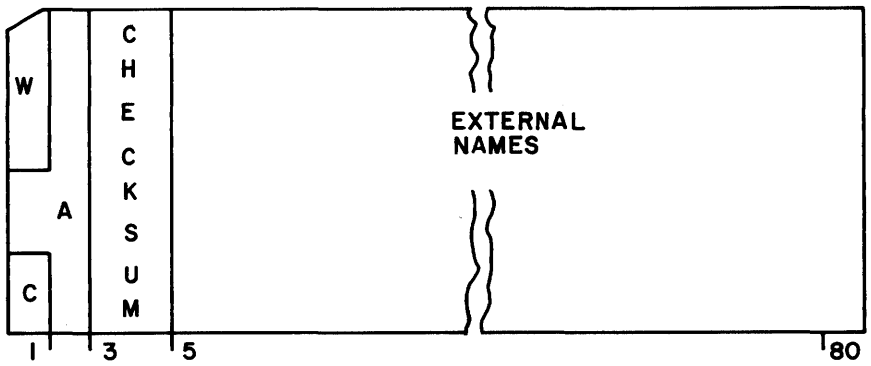
Example:



Entry Point Name	Address
ALPHA21	70034
BETA	60162
GAMMARAY	55473
.	.
.	.
.	.
ZED	57675

XNL CARD

The XNL card lists external entry points in other subprograms or library routines that are referenced in the subprogram.



Field	Significance
W	Six-bit field that contains an octal 43. Identifies the card as an XNL card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using columns 3 and 4. A 5 indicates the loader should make a checksum test using columns 3 and 4.
A	15-bit field that contains the card sequence number. The field contains a zero for the first card, a one for the second card, etc. Column 1 contains the most significant digit of the number. The cards must be in sequential order in the subprogram deck but may be spread anywhere between the IDC and the TRA card.
columns 3 and 4	24-bit sum of columns 1, 2, and 5 through 80. An end-around carry is used if overflow occurs.
columns 5 through 80	Variable length fields, each containing an external name and the last address in the subprogram at which the external name is referenced. An external name may be one to eight alphanumeric characters; when it has fewer than eight characters, it is terminated by a 72_8. The address field for each external name begins after the last character of the name or after 72_8 and is 18 bits long. The address, with any leading zeros, occupies the low order 15 bits of the field. The most significant three bits are zero for the word address and nonzero for a character address.

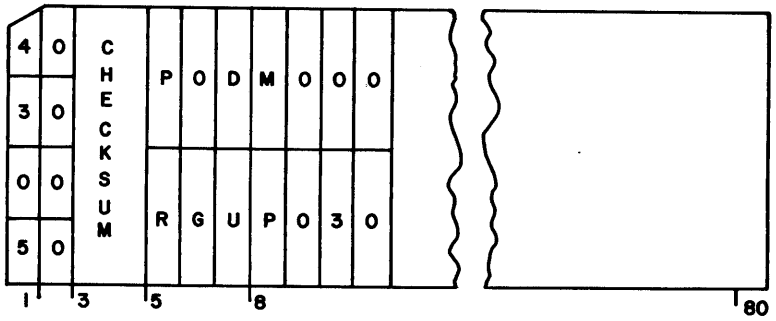
If the symbol is declared external, but is not referenced in the subprogram, the address on the XNL card is 77777_8.

The address given on the XNL card may specify another location in the same subprogram where reference is made to the external name. A series of reference addresses is called an external string. The low order 15 bits of the last entry in the string contain 77777_8. External strings may run in any order through the subprogram.

All external references are to subprogram relocatable word addresses. However, the external reference may be made by either a word- or character-type instruction. Entries in a string may be from both word-type instructions and character-type instructions.

XNL cards may occupy any position between the IDC card and the TRA card in a program deck. An exception is, when the binary deck is loaded from the system or auxiliary library file, the XNL card must follow the first EPT card. An external string may refer to previously encountered external symbols only after the relocatable information has been loaded for them.

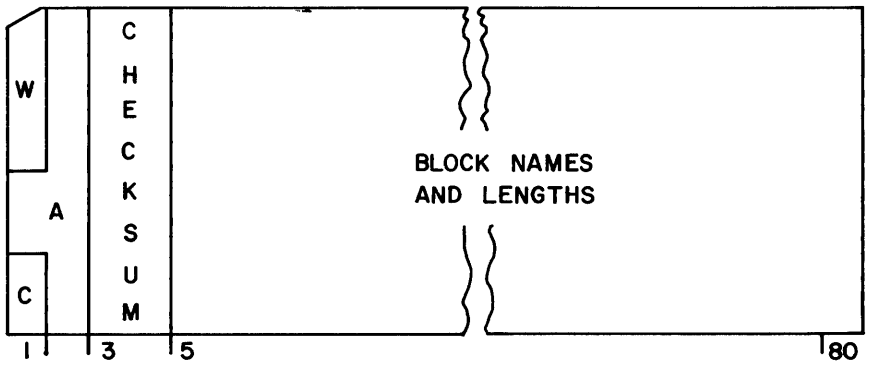
Example:



The external name PROGDUMP was last referenced in the subprogram at relocatable address 000300.

BDT CARD

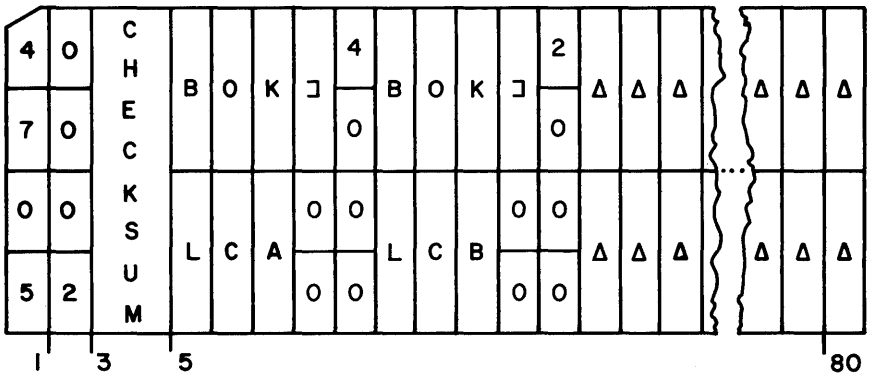
The blocked data table (BDT) card contains the name and length of any labeled data block fields used by the subprograms. The labeled data areas are legal only for subprograms written in ANSI FORTRAN or ANSI COBOL. A maximum of 63 labeled data areas may be used per subprogram and ANSI FORTRAN allows a maximum of 14 BDT cards per subprogram. The BDT cards must be in sequential order. However, they may be spread out between the IDC and TRA card.



Field	Significance
W	Six-bit field that contains an octal 47. Identifies the card as a BDT card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using columns 3 and 4. A 5 indicates the loader should make a checksum test using columns 3 and 4.
A	15-bit field that contains the card sequence number. The field contains a one for the first card, a two for the second card, etc. Column 1 contains the most significant digit in the number.
columns 3 and 4	24-bit checksum of columns 1, 2, and 5 through 80. An end-around carry is used if overflow occurs.
columns 5 through 80	Variable length fields, each field containing a block name in BCD, and the corresponding block length in characters. Block names and block lengths (octal) for any single block can not be continued from one card to another. Block name 1- to 8-alphanumeric characters. If the name has fewer than 8 characters, a 72_8 terminates the field. Block length 18-bit positions, right justified, and begins after the last character of the name or a 72_8.

Each individual block name is assigned a number sequentially starting with 1.

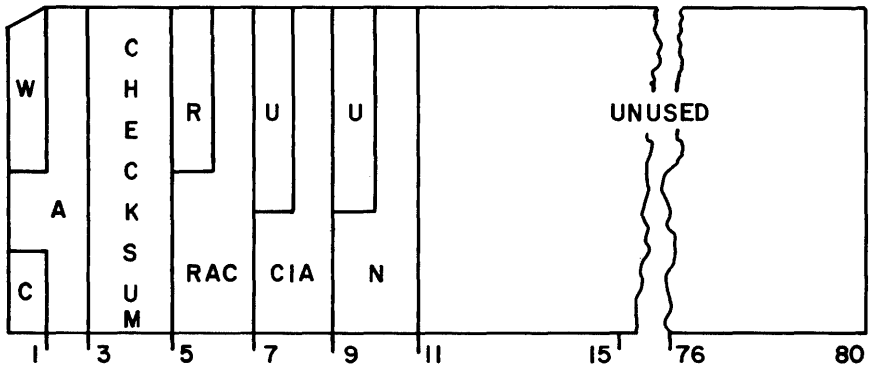
Example:



Sequence number	2	
First data block	BLOCKA	4000_8 characters
Second data block	BLOCKB	2000_8 characters

LRL CARD

The LRL card is produced only by single pass assemblers and compilers. The card is produced when the program references a symbolic address before the address occurs. The LRL card contains the relocatable address that defines the symbol, the last relocatable address that used (referenced) the symbol, and the number of times the symbolic address was referenced.



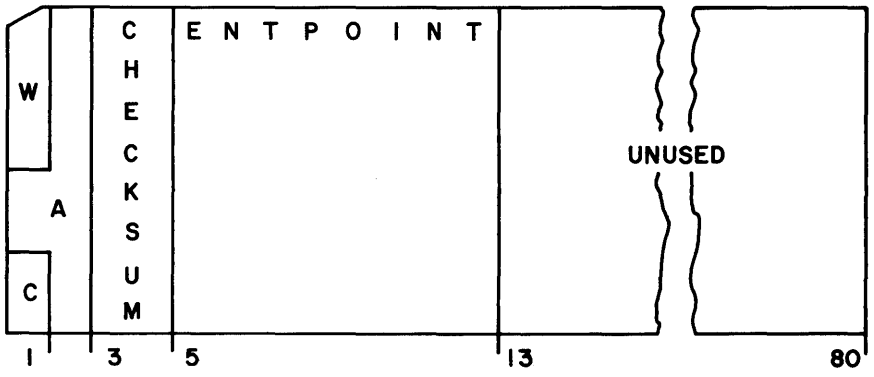
Field	Significance
W	Six-bit field that contains an octal 45. Identifies the card as an LRL card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using columns 3 and 4. A 5 indicates that the loader make a checksum test using columns 3 and 4.
A	15-bit field that indicates whether the address specified in the RAC field is a word or character address. A = zero for word address A ≠ zero for character address
columns 3 and 4	24-bit sum of columns 1, 2, and 5 through 80. An end-around carry is used if overflow occurs.
R	Seven-bit field that is unused (zero filled).
RAC	17-bit field that contains the relocatable address that defines the symbol. Column 5 contains bits 12 through 16 and column 6 contains bits 00 through 11. If the address is a word address, bits 16 and 15 in rows 8 and 9 are zero-filled.
U	Nine-bit fields that are unused (zero-filled).
CIA	15-bit field that contains the relocatable address of the last reference to the symbol. Column 7 contains bits 12 through 14 and column 8 contains bits 00 through 11.
N	15-bit field that contains an octal digit indicating the number of times the symbol was referenced. Column 10 contains bits 00 through 11 and column 9 contains bits 12 through 15.

TRA CARD

The transfer address (TRA) card indicates the end of a subprogram or library routine. It must be the last card in each subprogram deck. If a subprogram contains the program's entry point, the entry (transfer) point symbol appears on the subprogram TRA card.

MSOS uses the transfer point symbol to initially enter all programs. If a second transfer point is used, it appears in the Q register when the program is entered. The user may then elect to reenter the program at the secondary entry point via a jump instruction or to start execution at the main entry point.

Only two transfer point addresses can be used in a program. If one transfer point address is used, the loader assigns it as the main entry point to the program. If two transfer point addresses are used, the LOADER assigns the first transfer point encountered as the secondary transfer point and the last transfer point encountered as the main transfer point. † The loader terminates loading a library routine when it reads the routine's TRA card.

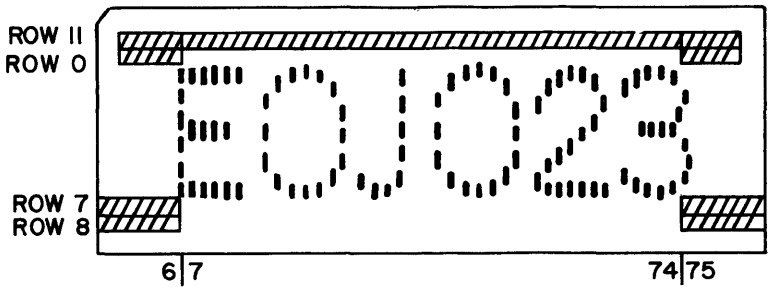


Field	Significance
W	Six-bit field that contains an octal 44. Identifies the card as a TRA card.
C	Three-bit field that contains an octal 5 or 7. A 7 indicates the loader should not make a checksum test using columns 3 and 4. A 5 indicates the loader should make a checksum test using columns 3 and 4.
A	15-bit field that is not used (unpunched).
columns 3 and 4	24-bit checksum of columns of all cards in the subprogram. There is no checksum for the TRA card alone. An end-around carry is used if overflow occurs.
columns 5 through 12	Symbolic program entry point name (punched in Hollerith). Does not appear on all cards.
columns 13 through 80	Unused and unpunched.

† The transfer point addresses may be on the same or different TRA cards.

JOB SEQUENCE CARD

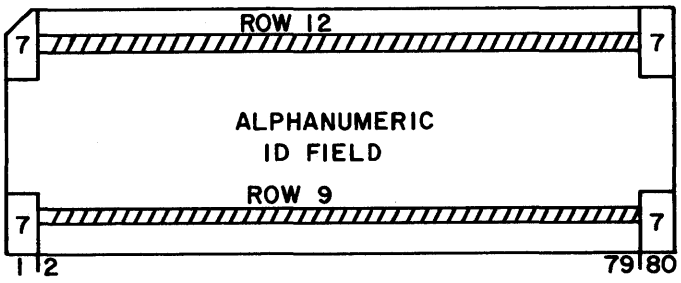
The job sequence card is the last card punched in a job's punched card output. It is an EOF card and is offset to indicate the end of the job output. An EOJ and the job sequence number are punched in block letters across the middle of the card (refer to job sequence numbers in section 16). The card may be used as an end-of-file card when loading the deck, and may be turned end-around to separate the input decks.



row 11	1's punched in columns 2 through 79
row 0	1's punched in columns 2 through 6 and 75 through 79
rows 7 and 8	1's punched in columns 1 through 6 and 75 through 80

FLIP CARD

The flip cards are the first two cards in subprogram decks produced by the MS FORTRAN compiler. The first flip card contains the subprogram name punched in block letters, and the second flip card contains the date of compilation and the library edition number punched in block letters. The block letters are inverted so that they are readable when the card is flipped upside down. Flip cards are produced only for the punch output unit; the cards do not appear at the list output unit. The loader ignores flip cards when reading a binary input deck.



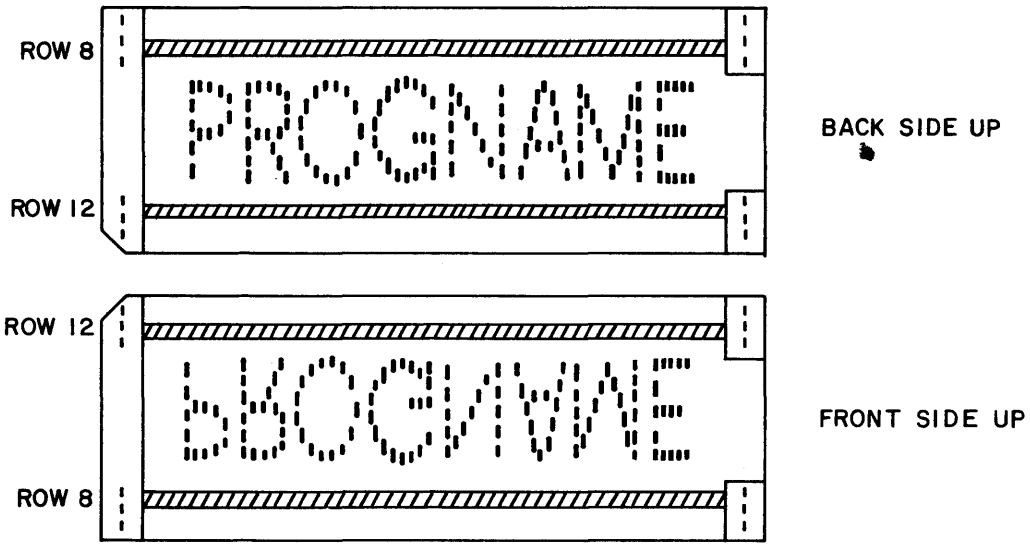
column 1	Contains an octal seven in the upper and lower three rows.
rows 8 and 12	Contain binary 1's punched in each column of the card.
ID field	Contains either an 8-character subprogram name punched in inverted block letters or an 8-digit date/edition number punched in inverted block letters. The data/edition number is in the following form.

mmddyyed

m	Month
d	Day
y	Year
ed	Library edition number

column 80	Upper three and lower three rows contain octal sevens. Rows 1 through 6 are part of the ID field.

Example:



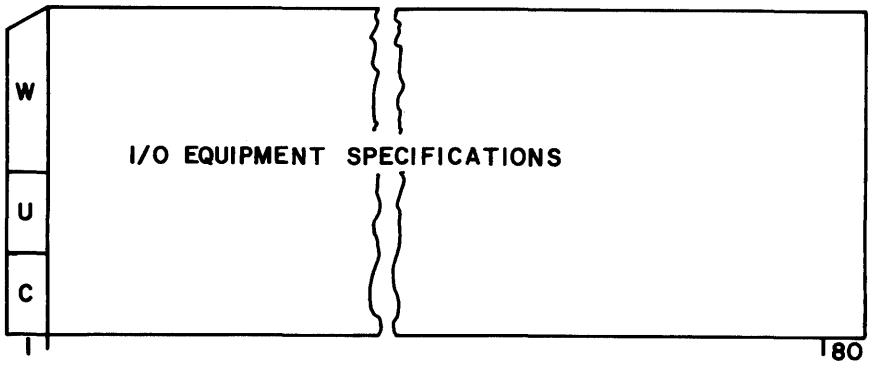
LED CARD

The loader equipment declaration (LED) card may be punched and inserted in a subprogram deck by the user. The card is produced only by the ANSI COBOL compiler.

The LED card performs the same function as an EQUIP statement. It assigns logical unit numbers to I/O equipment. If a logical unit has been assigned previously by an EQUIP statement, the EQUIP statement takes precedence; the LED declaration is ignored. A LED card may contain more than one hardware declaration.

The use of LED cards is an assembly option which may be selected or omitted when the system is installed. If the option is omitted, LED cards are ignored by the loader. Units with a dialable equipment number that are assigned by LED cards are logged on CTO and the operator is asked: READY?.

Priority programs may not contain LED cards.

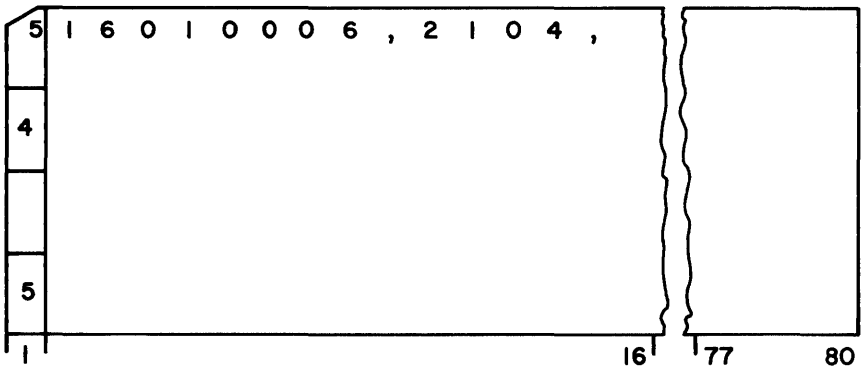


Field	Significance
W	Six-bit field that contains an octal 54. Identifies the card as a LED card
U	Unused, unpunched.
C	Three-bit field that contains an octal 5.
columns 2 through 80	uuhh or uuhheeuu — Parameters appear in Hollerith code, occupying 4 or 8 columns.
	uu — Logical unit, 01 through 56.
	hh — Hardware type, 01 through 31_8 (appendix C).
	c — Channel number, 0 through 7.
	e — Equipment, 0 through 7
	uu — Unit designator, 00 through 77_8
	Parameters may be blank. If so MSOS assigns an available unit of the hardware type specified. If one parameter is present, all must be present.

Declarations are separated by commas. Embedded blanks are ignored. Any number of LED cards may appear in a program deck. If the logical unit indicated in a declaration has been previously assigned to a physical unit, the declaration is ignored.

Example:

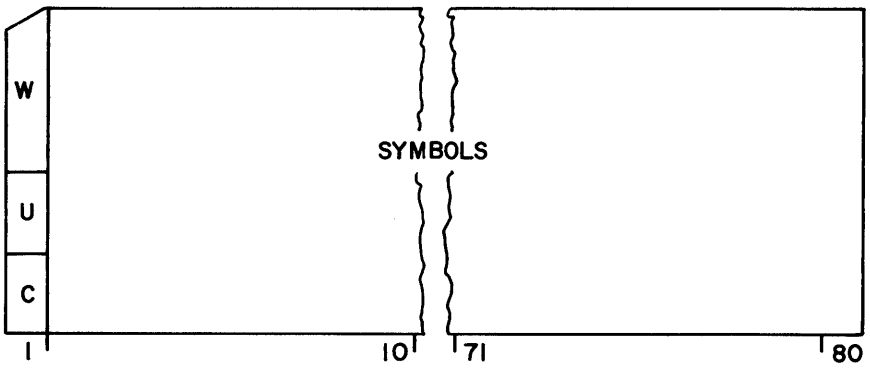
This card assigns logical unit 16 to the magnetic tape designed by the hardware code C0E0U06, and logical unit 21 to any available card punch.



EXS CARD

The external symbol card (EXS) may be punched and inserted in a subprogram deck by the user. This card is not produced by an assembler or compiler.

The EXS card declares external symbols not listed on an XNL card or equates one or more external symbols to a single entry point. A common use is to load additional library routines not referenced in the subprograms being loaded.

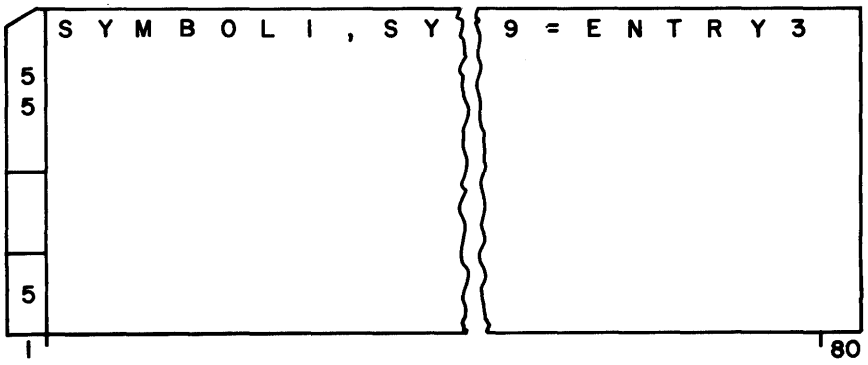


Field	Significance
W	Six-bit field that contains an octal 55. Identifier card as an EXS card.
U	Three-bit field that is unused and unpunched.
C	Three-bit field that contains an octal 5.
columns 2 through 80	Contain Hollerith characters. To declare symbols external, the form is: external symbol, external symbol,...,external symbol To equate symbols to an entry point, the contents are: external symbol, external symbol,...,external symbol=entry point

When EXS declares SNAPSHOT external, a SNAP control card must provide linkage. When EXS declares any other subroutines external, the source program must provide linkage (jump instruction) to enter the subroutine.

If EXS is used to equate symbols to an entry point, only the program containing the entry point is loaded. EXS declarations override later EPT declarations. If a subsequent entry point name is identical to an external symbol in an EXS declaration, a duplicate symbol error results.

Example:

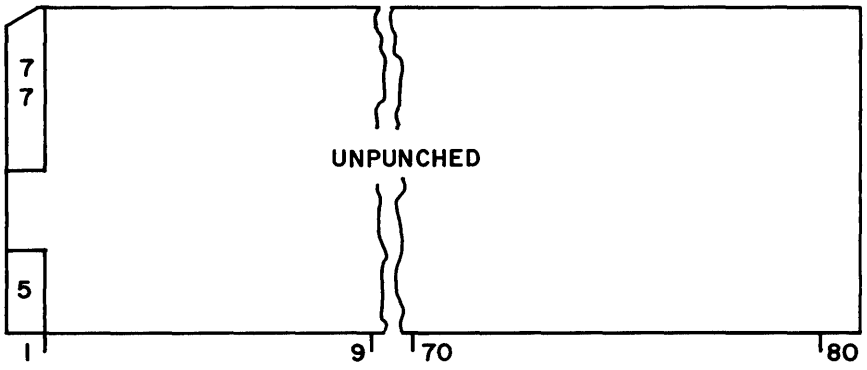


On this card, symbols SYMBOL1 through SYMBOL9 are equated to ENTRY 3.

ELD CARD

The end-of-load (ELD) card may be punched and added to the end of a binary program deck by the user. This card will not be produced by an assembler or compiler.

The ELD card terminates loading a binary card deck from the card reader and causes control to revert back to the program. For example, this option permits a batch program to call the loader to load a binary data deck from the card reader. Upon reading the ELD card, the loader terminates reading and returns control to the program.



SNAP CARD

A SNAP card may be used to obtain a periodic dump of a specified area of core while a program is running. SNAP cards can be used only in conjunction with relocatable binary program decks in batch programs. SNAP cards cannot be used with priority programs. †

When loading a binary deck for execution, SNAP cards may be inserted in any of the following positions in the job deck.

1. Behind the last TRA card in the program deck.
2. In front of an OVERLAY card.
3. Behind an OCC card.

The SNAP card can be used behind a TRA card only when the binary deck is being loaded from the card reader. A separate SNAP card must be used for each area of core to be dumped.

When a SNAP card is read, MSOS replaces one of the binary instructions of the process with a return jump to SNAP. The replaced instruction is saved for execution by the SNAP routine.

When the jump to SNAP occurs, SNAP dumps the area of core specified by the SNAP card. Then SNAP executes the saved instruction, and returns to the next instruction in the program.

```
$SNAP,(subprog)n,fwa,lwa,mode,id
```

(subprog)n — Location of the instruction to be replaced by a return jump to SNAP. (subprog) is a one- to eight-BCD subprogram name and n is a five-digit relocatable octal address. MSOS will modify this address with the absolute address of the subprogram in order to obtain the absolute address of the instruction to be replaced. The parentheses must be used around the subprogram name.

Example:

```
,(PROGNAME)1537,
```

fwa,lwa — First and last word address of the area to be dumped by SNAP. Only the locations between fwa and lwa will be dumped. fwa and lwa will not be dumped.

The fwa and lwa parameters must be punched in one of the following formats. Both must be in the same format.

1. D/id/address

D	Indicates the location to be dumped is in a data block.
id	One- to eight-character data block name used in the program.
address	A five-digit relocatable octal address in the data area. MSOS will modify the address with the absolute address of the first word in the data area in order to obtain the absolute address of the data area to be dumped.

Example:

```
,D/DATA1/0053,
```

† The SNAP function is an assembly option in the loader. In some systems, the SNAP function may have been omitted at system installation time to save core.

2. C address

C	Indicates the area to be dumped is in a common area.
address	A five-digit relocatable octal address in the data area. MSOS will modify the address with the absolute address of the first word in the common area in order to obtain the absolute address of the common area to be dumped.

Example:

```
,C00053,
```

3. (subprog) address

(subprog)	The name of the subprogram containing the core area to be dumped. May be one- to eight-BCD characters. The parentheses must be around the subprogram name.
address	A five-digit relocatable octal address within the subprogram. MSOS will modify the address with the absolute address of the first word in the subprogram in order to obtain the absolute address of the program area to be dumped.

Example:

```
,(PROG6)00053,
```

mode — Format the dump will be printed in. One of the following values must be used.

- O Print the dump in octal numbers and omit the register file dump.
- C Print the dump in BCD characters and omit the register file dump.
- F Print the dump in floating point numbers and omit the register file dump.
- R Include the register file dump. The R parameter may precede or follow the O, C, or F parameter.

Example:

RO	Print octal with register file dump.
OR	Print octal with register file dump.
O	Print octal without register file dump.

If any other values are used for mode, the SNAP dump will be bypassed and an error message will be typed on the output unit.

id — One to four characters used to identify the dump. The id character precedes each line of the dump as it is printed.

The following rules should be observed when using SNAP cards.

1. The location of the inserted return jump to snap must:
   a. not be a location that is modified during program execution.
   b. not be a location that is indirectly addressed.
   c. not be part of a two or more word instruction (that is, searches, skips, and BDP instructions).
   d. not be modified by an OCC correction card.
2. To avoid excessive printouts, avoid using SNAPSHOT in a loop.

The format of a SNAP dump is shown in Figure 6-1.

Examples:

```
$SNAP,(PROG1)00123,(PROG1)10236,(PROG1)10237,C,STAT
$SNAP,(PROG1)00124,C00000,C00777,C,RSLT
```

OCC CARD

The octal correction card (OCC) may be used to change the contents of any address in a program while loading the program for execution. The OCC card is used in conjunction with relocatable binary batch program decks. The OCC card cannot be used with priority programs or source language decks. †

When loading a relocatable binary program for execution, OCC cards may be inserted in any of the following positions of the job deck.

1. Behind the last TRA card in the program deck.
2. In front of an OVERLAY card.
3. Behind a SNAP card.

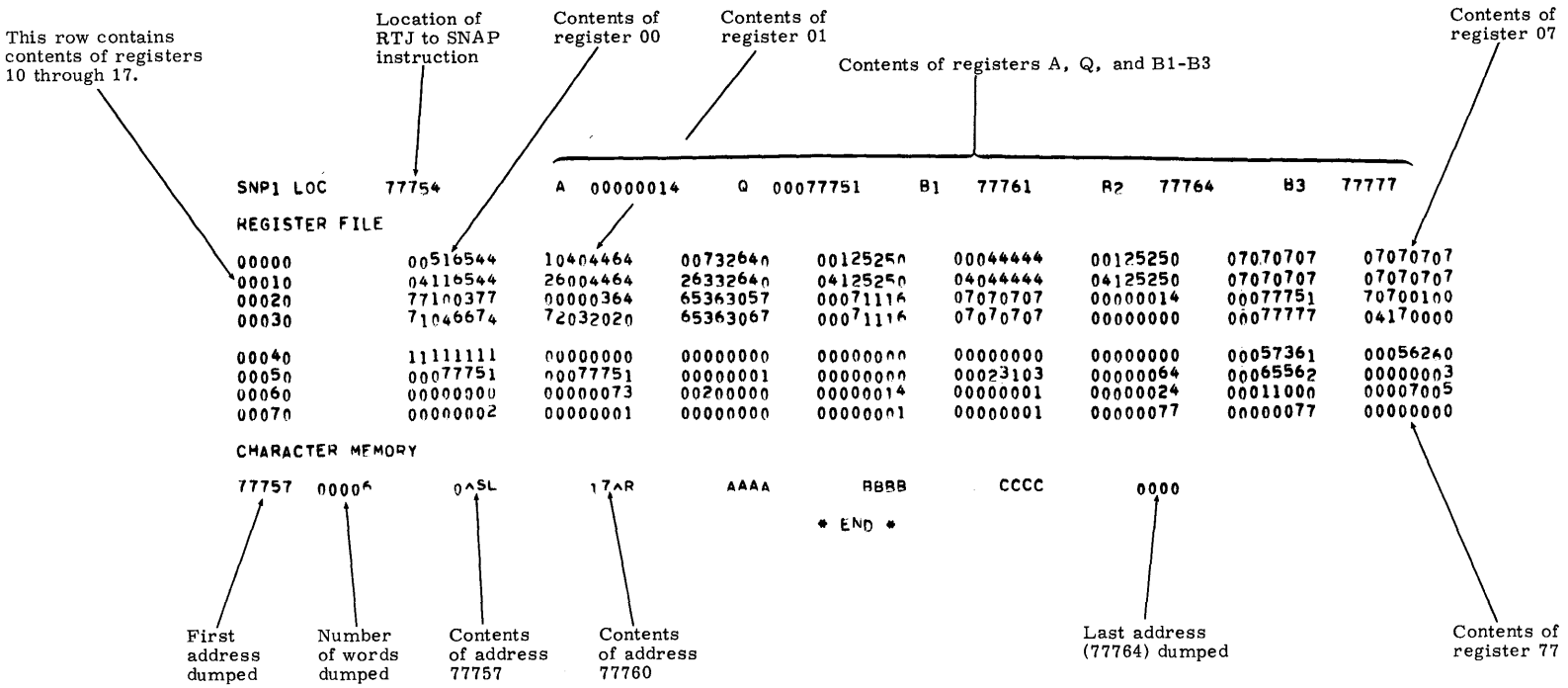
The OCC card can be used behind a TRA card only when the binary deck is being loaded from the card reader.

The OCC card changes only the loaded program. The binary deck or file used to load the program will not be changed. The OCC card may be used to make the following changes to a program during loading.

1. Change the contents of any address in the program.
2. Define an extension to the program area in which additional instruction can be added.
3. Change the contents of any address in a data area or block.††
4. Change the contents of library routine (in core only) which was declared external in the program.

†The OCC function is an assembly option in the loader. In some systems, the OCC function may have been omitted at system installation time in order to save core.

††The OCC function cannot be used to make corections in a labeled data area defined in an ANSI FORTRAN or ANSI COBOL program, or the instructions which reference the labeled data area.



In batch programs run with the extended core variant of MSOS, the loader converts jump instructions that reference executive subroutines to HLT instructions that reference the subroutines (refer to executive interrupts in section 18).

Figure 6-1. Sample SNAP Dump

CHANGING THE CONTENTS OF A PROGRAM ADDRESS

The format of the OCC card used to change the contents of memory locations in a program is as follows:

```
$OCC,(subprog)addr,oc,...,oc    $OCC,+,oc,...,oc    $OCC,+n,oc,...,oc
```

(subprog)addr — Address of the first word to be changed. MSOS advances the address by one for each additional correction word on the card.

(subprog) is the name of the subprogram to be changed. The parentheses must be used.

addr is the first address to be changed. If word addressing is to be used, use a five-digit number. If character addressing is to be used, use a six-digit number. MSOS adds addr to the first word address of (subprog) to obtain the absolute address of the word to be changed.

\+ — Continuation card with additional octal corrections. Corrections will be continued at next sequential address.

+n — Continuation card with additional octal corrections. The address of the last correction on the previous card will increase by n_8 and be used as the address for the first correction word on this card (that is, skip n_8 addresses).

oc — Octal correction word with relocation factor. Consists of a new word which is an eight-digit octal number with a relocation suffix. The new word is right-justified. The leading zeros may be omitted.

The relocation suffix modifies only the lower 15 or 17 bits of the correction word. The values are as follows:

omitted	No modification of the word.
(subp)	Relocate the lower 15-bit address in the word relative to the first word address of the subprogram (subp). The parentheses must be used around the subprogram name.
(subp)C	Same as (subp) except that character addressing is used. The lower 17 bits are modified.
D	Relocate the lower 15-bit address in the word relative to the first word address of the data area assigned to (subprog).
DC	Same as D, except that character addressing is used. The lower 17 bits are modified.
C	Relocate the lower 15-bit address in the word relative to the first word address of the common area assigned to (subprog).
CC	Same as C, except that character addressing is used. The lower 17 bits are modified.
*	Relocate the lower 15-bit address in the word relative to the first word address of the last (subp) name used on this card or on a preceding OCC or SNAP card.

*C	Same as *, except that character addressing is used. The lower 17 bits are modified.
X	Relocate the lower 15-bit address in the word relative to the first word address of a program extension area. An OCC card defining the program extension area must precede this relocation suffix.
XC	Same as X, except that character addressing is used. The lower 17 bits are modified.

Any number of correction words may be added to a program by using OCC,+ cards. To skip an address in a series of corrections, omit the octal correction word and retain the trailing comma. The address counter will be advanced by the comma, but will not change the contents of a location for which there is no correction word.

Examples:

```
$OCC,(PROG1)70,15600100
```

This example writes 15600100 (INA by 100) in address PROG1+70. If subprogram PROG1 starts at address 73355, 20000100 would be written in address 73445.

```
$OCC,(PROG1),20000100*
```

This example modifies the lower 15 bits of the octal correction word with the first word address of PROG1 and then stores the word in the first address of PROG1. If subprogram PROG1 starts at address 73355, 20073455 (LDA from 73455) will be written in address 73355.

```
$OCC,(SUB1)20,00000036,,00036,,036,36
```

This example stores the octal value 00000036 into locations 20, 22, 24, and 25 of subprogram SUB1. Because values are right-justified, all the octal corrections, 00000036, 036, etc., are stored as 00000036. If SUB1 was loaded beginning at address 63652, the results of this OCC are:

Address	Contents
63672	00000036
63673	Unchanged
63674	00000036
63675	Unchanged
63636	00000036
63637	00000036

```
$OCC,(PROG1)70,20000100(PROG2),30000100D,,40000100C
```

Enter octal correction 200xxxxx at address 00070 relative to subprogram PROG1. Relocate word address portion of this octal correction relative to subprogram PROG2. Enter octal correction 300xxxxx at address 00071 relative to subprogram PROG1; relocate word address portion of this correction relative to the data area. Enter 400xxxxx at address 00073 relative to subprogram PROG1; relocate word address portion relative to the common area. Absolute initial locations for PROG1. PROG2, data, and common are:

PROG1	45333
data	45534
PROG2	45635
common	46036

The results of this OCC are:

45423 = 20045735
45424 = 30045634
45425 = (unchanged)
45426 = 40046136

DEFINING A PROGRAM EXTENSION AREA

The format of an OCC card used to define an extension of the program area for additional instructions is as follows.

```
$OCC,Xn
```

n Length of program extension area. One to five octal digits.

If length n exceeds available memory, the area is adjusted to the size of available memory. A message is printed on OUT and the program executes if this is the only error in the run. A memory overflow error occurs when the number of corrections for loading exceeds the defined area. The job will abort.

Only one program area extension statement may be used. Any octal corrections appearing on this statement will be ignored.

All subsequent OCC,X statements will load octal instructions in the extended program area. The format is as follows:

```
$OCC,Xn,oc,...,oc
```

n Relative load address for first word to be stored in the extended program area. First correction word on the card will be stored at first location in extended program area plus the value of n.

oc Octal correction word and relocation suffix.

Example:

```
      4. $OCC,X10,20000620(SUB4),40000621*,20000622(SUB5),40000623*
   3. $OCC,+,20000400(SUB2)40000401*,20000402(SUB3),40000403*
2. $OCC,X,20000100(SUB1)40000101*,20000102*,40000103*
1. $OCC,X20
```

1. Assign a program extension area of 20 locations.
2. Load corrections into locations 0 through 3 of the program extension area. Relocate word address portions of these corrections relative to subprogram SUB1.
3. Load corrections into locations 4 through 7 of the program extension area. Relocate word address portions of the first two corrections, relative to SUB2. Relocate word address portions of the second two corrections, relative to SUB3.
4. Load corrections into locations 10 through 13 of the program extension area. Relocate word address portions of the first two corrections, relative to SUB4. Relocate word address portions of the second two corrections, relative to SUB5.

CHANGING THE CONTENTS OF THE DATA AREA

The format of the OCC card used to change the contents of the data area is as follows:

```
$OCC,Dk,oc,...,oc
```

k	Address (relative to first address in data area) of first correction. If k is omitted, first correction word is written in first address of data area. Otherwise, first correction word is written in first word address plus k addresses.
oc	Octal correction word and relocation suffix.

Example:

```
   $OCC,+,73535353,1060
$OCC,D,4,10,14,20,,,57632,114567
```

Enter four octal values in successive data area locations starting with the first word. Skip two addresses and enter four more values. If the data area begins at location 70000, the results of these cards are as follows:

(70000) = 00000004
(70001) = 00000010
(70002) = 00000014
(70003) = 00000020
(70004) = (unchanged)
(70005) = (unchanged)
(70006) = 00057632
(70007) = 00114567
(70010) = 73535353
(70011) = 00001060

OVERLAYS 7

DESCRIPTION

A batch or priority program that is larger than available core may be divided into a main program and overlay elements. The main program resides in core and the overlay elements reside on mass storage files. During execution of the program, the main program calls and gives control to the overlay elements as they are needed.

OVERLAY ELEMENTS

In MSOS, overlay programs are divided into the following elements: MAIN, OVERLAY, and SEGMENT. Each of these elements is written and compiled or assembled as individual subprograms, the same as any other large program. Then, the relocatable binary decks are loaded as a single program. The only difference between entering an overlay program and a regular program is that each element (one or more subprogram decks) in an overlay program must be preceded by a binary overlay header card. The overlay header card indicates an overlay program is being loaded and identifies each element by type and with an identification number.

MAIN

Each overlay program must have only one MAIN element. MAIN resides in core and calls in overlays as they are needed. When the relocatable loader loads the binary deck of MAIN, it assigns an overlay area in core immediately below MAIN. Each overlay will be loaded in this area when called from MAIN.

OVERLAYS

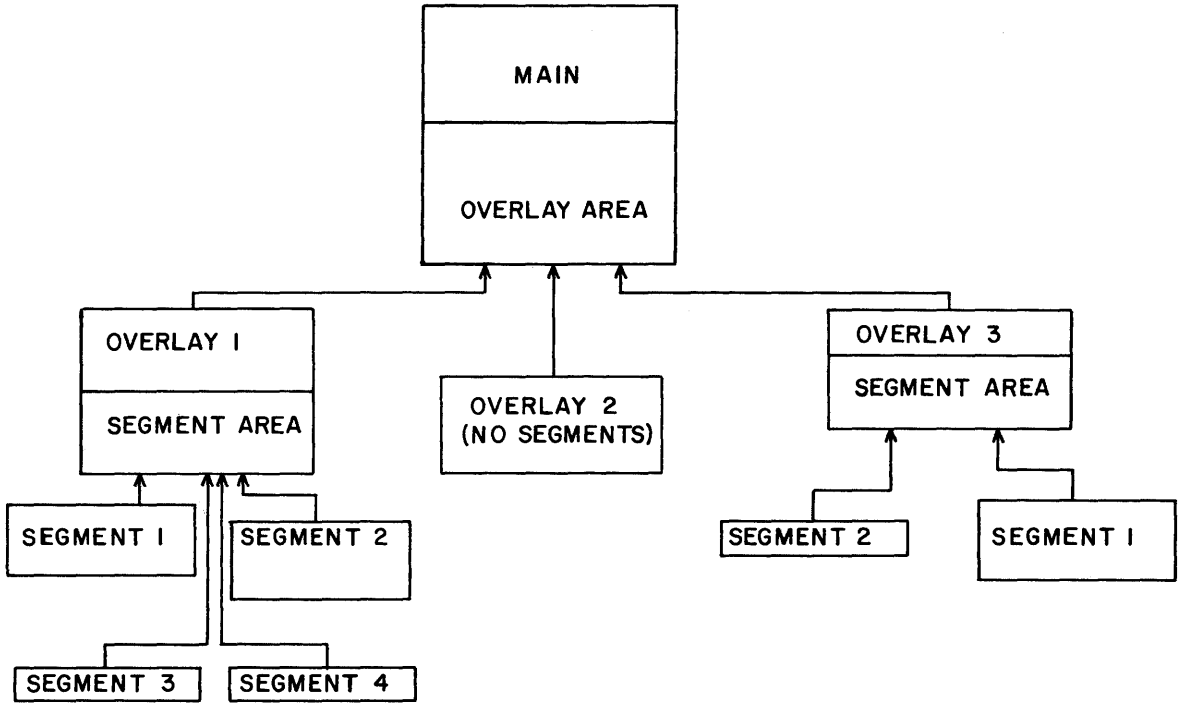
The overlay elements reside in absolute binary format or a mass storage file until called by MAIN. Only one overlay can be in core at a time. Each new overlay is loaded over the previous overlay in the MAIN overlay area. A program can have up to 99 different overlays.

Overlays can call and use segments, the same as MAIN calls and uses overlays. The loader assigns an area for segments in the overlay area.

SEGMENTS

Segments reside in absolute binary format on a mass storage file, until called into core by an overlay. Only one segment can be in core at a time. Each new segment is loaded over the previous segment in the overlay segment area. Each overlay can have up to 99 different segments.

Each segment is subordinate to and assigned to only one overlay. The same segment cannot be used by more than one overlay. Segments are assigned to an overlay element by loading the binary segment decks immediately following the binary deck of the overlay.



SAMPLE CORE CONFIGURATIONS

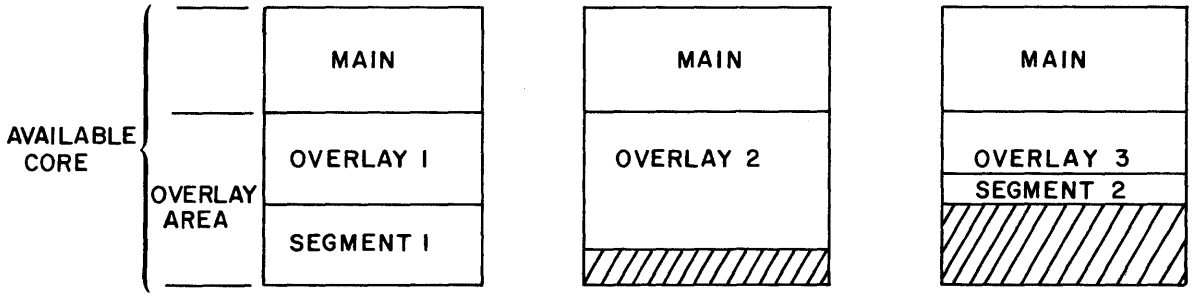


Figure 7-1. Overlay and Segment Organization Block Diagram

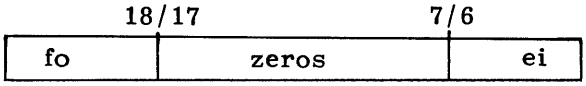
OVERLAY PROGRAMS

Overlay programs can be written in ANSI FORTRAN, MS FORTRAN, ANSI COBOL, and COMPASS. For programs written in FORTRAN or COBOL, refer to the FORTRAN or COBOL reference manual for instructions on preparing the overlay elements and for calling the overlay elements into execution.

When writing overlay programs in COMPASS, each element must be written as one or more separate subprograms. The overlay elements, both overlays and segments, are then called into execution by doing a return jump to EXECOVR.

The procedure for using EXECOVR is as follows:

1. Declare EXECOVR as an external.
2. Enter a file number and an element identification number in register A.



Register A

fo — Mass storage file number that the element is stored on. May be any decimal number from 1 through 55. The number must be the same as that used on the binary overlay or segment header card that will be used to load the element (refer to OVERLAY card or SEGMENT card).

ei — Element identification number. Must match the overlay or segment identification number used on the binary overlay or segment header card used to load this element. May be any octal number from 1 through 143_8.

3. Do a return jump to EXECOVR

   EXECOVR will load and give control to the overlay or segment specified in register A.

The following is an example of using EXECOVR to load and give control to overlays and segments.

Example:

```
P       EXT     EXECOVR         Declares EXECOVR as external.
P+1     LDA     M               Load A with file number and element identifi-
                                cation number.
P+2     RTJ     EXECOVR         Pass control to overlay 4 or segment 4.
P+3     next instruction        Return from overlay or segment.
```

EXECOVR may be used only in the MAIN program and overlays. EXECOVR cannot be used in segments.

SYMBOLIC ADDRESS REFERENCES

The MAIN element of an overlay program cannot reference symbolic addresses in any of its overlays or overlay segments. MAIN can read symbolic addresses in resident EXEC and tables.

Overlay elements cannot reference symbolic addresses in their segments or in any other overlay. An overlay element can read and write in symbolic addresses in MAIN and can read symbolic addresses in resident EXEC and tables.

Segments cannot reference symbolic addresses in any other segment or in overlay elements other than their own. A segment can read and write in symbolic addresses in its own overlay and in MAIN. A segment can also read symbolic addresses in resident EXEC and tables.

DATA BLOCKS

Each element in an overlay program may define one data area (block). When an element defines a data block, only the element and its subordinate elements may use the block.

A data block defined in a subordinate element will be linked to (assigned in) the data block defined in the higher level element. For example, if a data block is defined in MAIN, all data areas defined in the overlays and overlay segments will be assigned in the same data block defined in MAIN. If a data block was not defined in MAIN, each overlay element may have its own individual data block. An overlay's data block can be referenced only by the overlay itself and its segments.

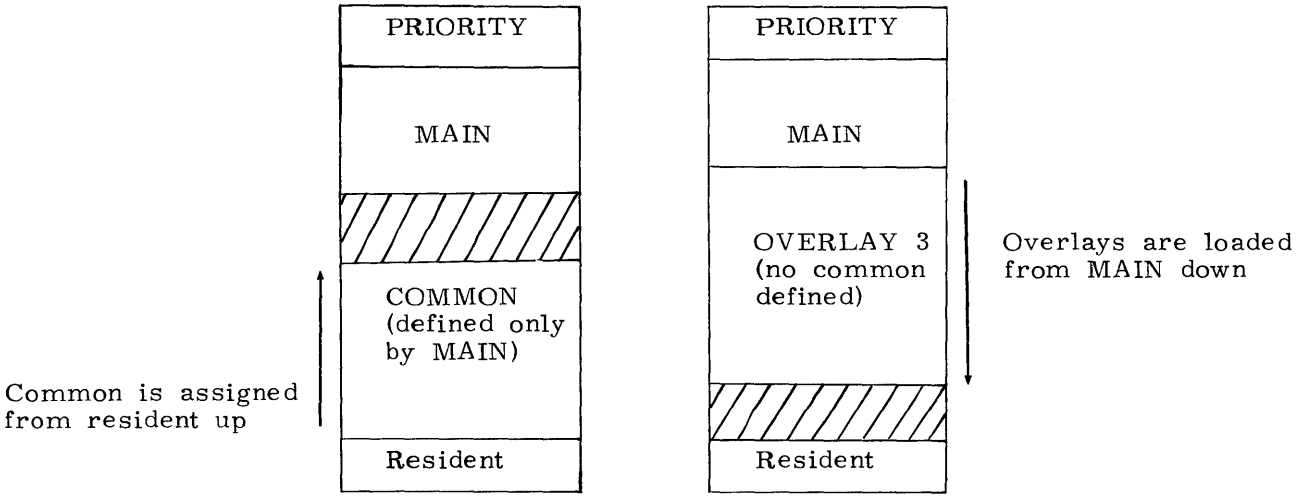
If a data block is not defined in MAIN or in an overlay, each of the overlay's segments may have its own individual data block. A segment data block can be referenced only by the segment that defined it.

Data can be pre-set in a data block only by the highest level element using the block. The size of a data area defined in a subordinate element must not be larger than the data block defined by the highest level element.

COMMON BLOCKS

Each element of a batch program may define a common block. The length of common memory is the greatest length defined by any program element in core unless some element of the program is loaded into the common memory area. Priority programs using overlays cannot define or use common memory.

An element of the program that does not require common memory need not define it. It is possible that an overlay element that does not define common memory may be loaded in an area of common memory used by some other element, as shown in the following diagram.



BINARY OVERLAY HEADER CARDS

The relocatable binary subprogram decks of an overlay program are loaded with the relocatable loader. The overlay program is loaded the same as any other program except that each element (MAIN, OVERLAYS, and SEGMENTS) must be preceded by a special binary header card. The overlay header cards are as follows:

MAIN card	The subprograms between this card and the next overlay header card comprise the main element of the program.
OVERLAY card	The subprograms between this card and the next overlay or segment header card comprise an overlay for the main element.
SEGMENT card	The subprograms between this card and the next overlay or segment header card comprise a segment for the preceding overlay deck.

When loading overlay programs, all segments of an overlay must follow their associated overlay deck. An overlay can use only those segments inserted in the deck prior to the next OVERLAY card.

MAIN CARD

The MAIN card must be the first card after the LOAD card in the binary program deck when overlays are used.

⌜wfo

w	An octal 50 in the upper six rows of column 1 and a 7/9 punch (octal 5) in the lower three rows of column 1. Identifies the card as a MAIN header card.
fo	A two-digit decimal mass storage file number for main. The fo may range from 01 through 55 in batch programs and 01 through 49 in priority programs. No space is allowed between w and the fo. If the program is to be loaded and executed and there is no need to save main on a file, fo may be omitted. When fo is omitted, main is written in core for execution, but is not saved on a file.

A MAIN card may also be placed ahead of a relocatable binary subprogram deck to save absolutized binary code on a permanent file for quicker loading with the ABSTSK statement in future jobs. The loader assembles the absolute code on the file specified by the MAIN card, rather than file 55, before loading the subprogram in core (refer to Figure 8-2). The block size should be in accordance with the sector size of the device.

OVERLAY CARD

The OVERLAY card indicates that the subprograms following it are an overlay. The overlay consists of all subprograms between this OVERLAY header card and the next binary header card or MSOS control card.

```
wfo,id
```

w — An octal 51 in the upper six rows of column 1 and a 7/9 punch (octal 5) in the lower three rows of column 1. Identifies the card as an OVERLAY header card.

fo — A two-digit decimal mass storage file number for the overlay. The fo may range from 01 through 55 in a batch program and from 01 through 49 in a priority program. The fo must be followed by a comma. No space is allowed between the w and fo.

id — A one-or two-digit decimal identification number for the overlay. May range from 1 through 99 and must be unique for all overlays in the program. No spaces are allowed between the comma and the id number.

SEGMENT CARD

The SEGMENT card indicates that the subprograms following it are a segment of the preceding overlay. The segment consists of all subprograms between this SEGMENT header card and the next binary header card or MSOS control card.

```
wfo,id
```

w — An octal 52 in the upper six rows of column 1 and a 7/9 punch (octal 5) in the lower three rows of column 1. Identifies the card as a SEGMENT header card.

fo — A two-digit decimal mass storage file number for the segment. The fo may range from 1 through 55 in batch programs and from 1 through 49 in priority programs. The fo must be followed by a comma. The fo parameter is punched in columns 2 and 3.

id — A one-or two-digit decimal identification number for the segment. The id may range from 1 through 99 and must be unique for all segments of the preceding overlay. No spaces are allowed between the comma and the id number.

SAMPLE OVERLAY PROGRAM

The following is an example of a job that will load and execute the binary subprogram deck of a batch overlay program. All overlays and segments will be stored in file 2. Each binary deck includes all cards for a complete subprogram (IDC through TRA). Block size must be 512 characters if files 54 or 55 are used. Otherwise, block size should be in accordance with the sector size of the device.

```
$JOB...
$EQUIP...
$RAT...
$FET...
$ALLOCATE...
$OPEN,2
$LOAD,M                 The M parameter will produce a memory map for each element
                        of the overlay program.
M55†
(first binary deck)
  .
  .
  .
(last binary deck)
O02,1††                 First overlay element header card
(first binary deck)
  .
  .
  .
(last binary deck)
S02,1†††                First segment for overlay 1
(first binary deck)
  .
  .
  .
(last binary deck)
S02,2                   Second segment for overlay 1
  .
  .
  .
S02,n                   Last segment for overlay 1
(binary decks)
  .
  .
  .
O02,n                   Last overlay segment for MAIN
(binary decks)
S02,1                   First segment for overlay n
(binary decks)
S02,n                   Last segment for overlay n
(binary decks)
$RUN
77
88
```

†M indicates a MAIN header card (octal 50 and a 7/9 punch in column 1).
††O indicates an OVERLAY header card (octal 51 and 7/9 punch in column 1).
†††S indicates SEGMENT header card (octal 52 and a 7/9 punch in column 1).

The following is an example of using overlays with a compile and run FORTRAN program. A similar job could be used to assemble COMPASS programs with overlays. ALGOL and COBOL have special control statements for segmenting compiled decks.

```
$JOB,...
$FET,owner,name,960,...          Block size of 960 characters required for all
                                 load and go (LGO) files.
$ALLOCATE...
$OPEN,10
$XFER,10                         XFER binary main card to LGO file.
(M)                              Binary main header card.
$FORTRAN,X=10                    Compile main and write on LGO file.
(FORTRAN source statements)
        FINIS
$XFER,10                         XFER binary overlay 1 card to LGO file.
(O54,1)                          Binary overlay 1 header card with fo=54.
$FORTRAN,X=10                    Compile overlay 1 deck and write on LGO file.
(FORTRAN source statement)
        FINIS
$XFER,10
(O54,2)                          Binary overlay 2 header card with fo=54.
$FORTRAN,X=10
(FORTRAN source statements)
        FINIS
$LOAD,10                         Load main from LGO file.
$RUN
(data cards)
77
88
$EOJ
```

In the preceding example, the overlay program will be saved on file 10 for future use. Each time, new data cards can be inserted for the program after the RUN statement.

LIBRARY FILE OVERLAYS

Overlayed programs may be written on the system library or a user auxiliary library. The following rules must be observed when placing an overlayed program on a library file.

1. Only one subprogram deck (IDC through TRA) may follow a MAIN, OVERLAY, or SEGMENT card. The TRA card must contain a transfer point address.
2. If an overlay element (MAIN, OVERLAY, or SEGMENT) has more than one subprogram, the additional subprograms must follow the TRA card in the last overlay element (overlay or segment) on the file. They may be in any order and their TRA cards may not contain a transfer point address.

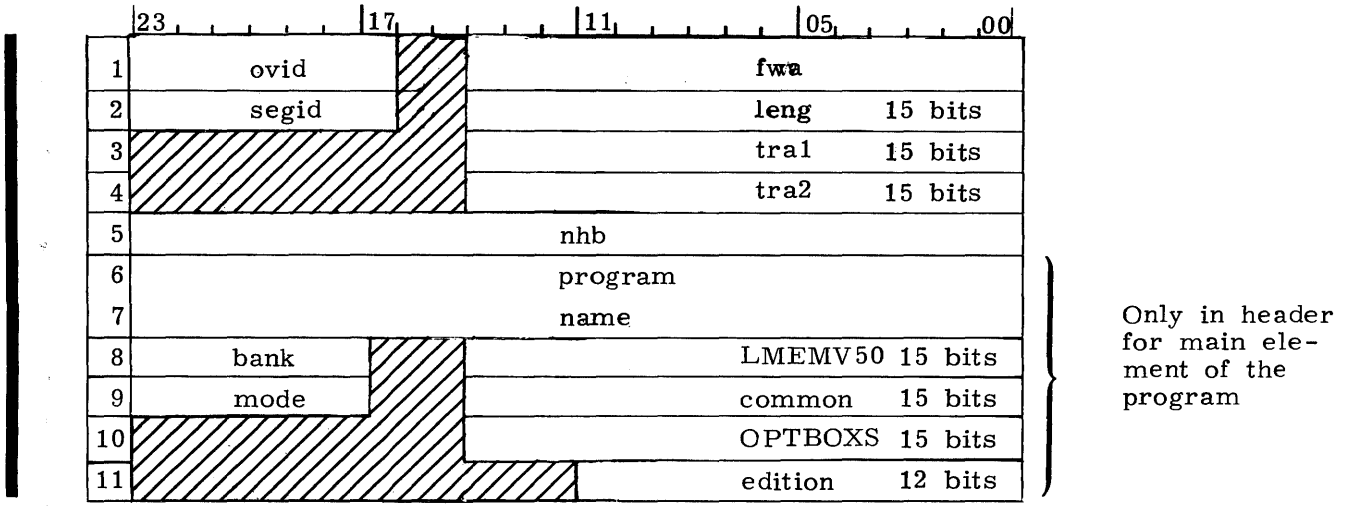
The following is an example of an overlay program on an AUX library. Refer to section 22 for a description of AUX Libraries.

```
$JOB,...
$FET,JOB,AUX,960
$ALLOCATE,50,991231
$OPEN,20
$PRELIB,,AUX,20,2
MAIN card
IDC                                          )
EPT                                          )
RIF                                          )   Main Program
  .                                          )
  .                                          )
  .                                          )
RIF                                          )
XNL                                          )
TRA (with transfer point address)            )
OVERLAY card
IDC                                          )
  .                                          )   Overlay
  .                                          )
  .                                          )
TRA (with transfer point address)            )
SEGMENT card
IDC                                          )
  .                                          )   First Overlay Segment
  .                                          )
  .                                          )
TRA (with transfer point address)            )
SEGMENT (last element in overlay program)
IDC                                          )
  .                                          )   Second Overlay Segment
  .                                          )
  .                                          )
TRA (with transfer point address)            )
IDC                                          )
  .                                          )
  .                                          )
  .                                          )
TRA                                          )   <-- Additional subprograms that are part of the overlay elements.
IDC                                          )       The subprograms may be in any order.
  .                                          )
  .                                          )
  .                                          )
TRA                                          )
7
9 FILE
77
88
$EOJ
```

SEGMENT AND OVERLAY FILE HEADERS

Each overlay or segment file is preceded by a special five-word file header that identifies the overlay or segment. This header is used by EXECOVR to locate and load the overlay or segment. The file header is not loaded into memory.

The format of the special file header for overlays and segments is as follows:



ovid (bits 23-17) — Overlay identification number (001 through 143_8). Octal equivalent to decimal number used on header card for the overlay binary input deck and same as element number used in the EXECOVR call to load the overlay. OVID=000 if the file contains main.

segid (bits 23-17) — Segment identification number (001 through 143_8). Octal equivalent to decimal number used on header card for the segment binary input deck and same as element number used in EXECOVR call to load the segment. segid=000 if the file contains an overlay or main.

fwa (bits 14-0) — Absolute address for loading the first word of the overlay or segment in core.

leng (bits 14-0) — Length of the overlay or segment in words (octal). Does not include the header. Does include the last word that is an end-of-file symbol. The end-of-file symbol is not loaded in core (that is, leng-1 words are loaded).

tra1 (bits 14-0) — Primary transfer address for return to MAIN or the overlay (that is, last encountered transfer symbol on a TRA card).

tra2 (bits 14-0) — Secondary transfer address for return to MAIN or the overlay (that is, next to last transfer symbol on a TRA card).

nhb (bits 23-0) — Next block number in the file that contains an overlay or segment file header (octal).

program name (two words) — Eight-character program name from IDC card.

bank (bits 23-18)	Memory bank that this program is loaded in; 00 for a 16 or 32K system, 01 or 00 for a 48K or 64K system.
LMEMV50 (bits 14-0)	The lowest absolute address that the program can use (from memory limit table).
mode (bits 23-18)	The mode of program execution (octal). 01 Batch 02 Priority 4 03 Priority 3 05 Priority 2 06 Priority 1
common (bits 14-0)	Highest address assigned to common.
OPTBOXS (bits 14-0)	The address of OPTBOXS if the program is using software simulators for BCD or floating point hardware.
edition (bits 11-0)	Edition of the library that was used to link this program (two BCD characters).

OVERLAY MAPPING

When an overlay program is loaded with an M parameter on the LOAD card, a loading map for each overlay and segment is printed on the standard output unit. No map will be produced if an M does not appear on the LOAD card.

The overlay map contains the following information for each overlay and segment.

1. First word address of each subprogram in the overlay or segment.
2. All entry points defined within the overlay or segment.
3. Overlay or segment extension, if any, to contain corrections.

Triple spacing separates the load map for each overlay or segment. The following heading appears on the first line of each overlay map.

e fo,n OVLAY ovn SEG segn FILE fn

e	Type of overlay element M main O overlay S segment
fo	Number of the file that the element is stored on.
n	Overlay or segment identification number. Omitted for the main element.
ovn	Overlay number
segn	Segment number
fn	File number

Example:

O 55,2

```
  M 40              OVLAY  0       SEG  00        FILE  40
    SUBP
63635   RAAR        63715   EXECOVR       64104    MAIN
    ENTR
63715   EXECOVR     12464   FDPBOXS       64104    MAIN      63635   RAAR
        MAINTEST    77514   RAAR
    LDTA
        NONE
    COMM
        NONE
    DATA
        NONE
    PEXT
        NONE


  O 40,10           OVLAY  10       SEG  00        FILE  40
    SUBP
53742   OVERLAY
    ENTER
53742   OVERLAY
    LDTA
        NONE
    COMM
        NONE
    DATA
        NONE
    PEXT
        NONE


  S 40,30           OVLAY 10        SEG  30        FILE  40
    SUBP
47750   SEGMENT
    ENTR
47750   SEGMENT
    LDTA
        NONE
    COMM
        NONE
    DATA
        NONE
    PEXT
        NONE
```

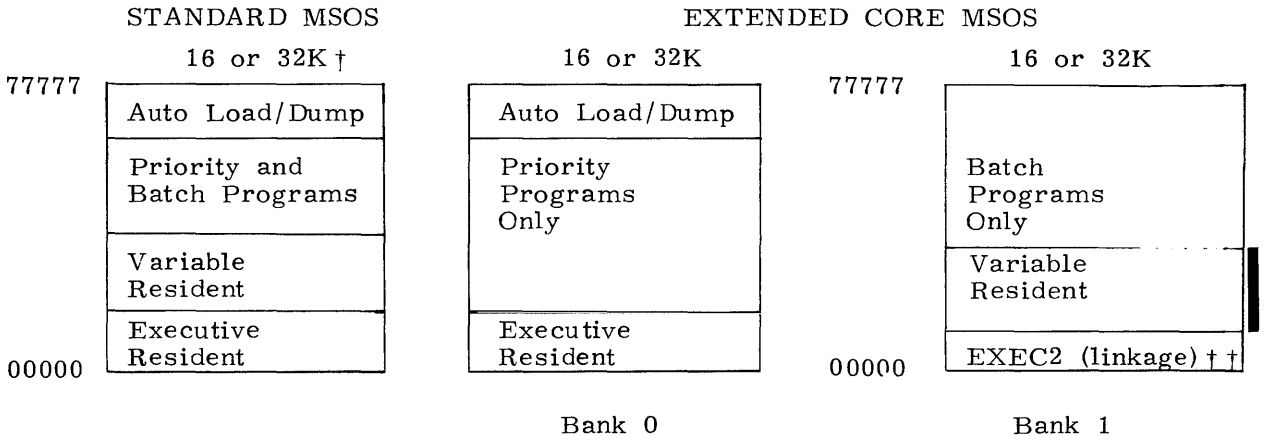
SUBP Subprogram names and first word addresses.
ENTR Entry points in the element.
LDTA Name and first word address of a labeled data area.
COMM First word address of common area.
DATA First word address of data area.
PEXT First word address of a program extension area.

Figure 7-2. Sample Overlay Map

MEMORY ORGANIZATION

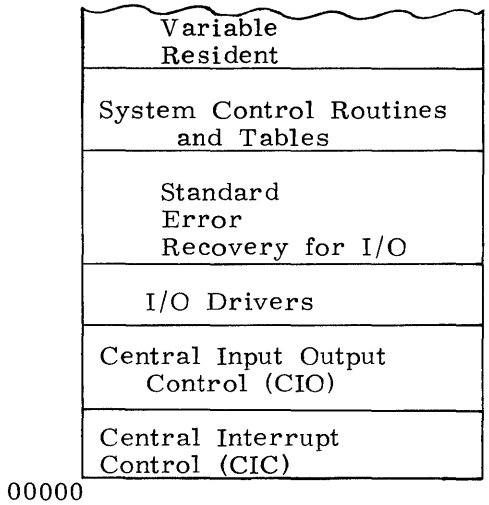
MSOS divides memory into five areas.

1. Executive resident
2. Variable resident
3. Batch program
4. Priority program
5. Autoload/autodump



EXECUTIVE RESIDENT

The executive resident section of memory contains all the MSOS operating system routines that must remain permanently in core.



Executive Resident

System	Approximate EXEC Size
16K Standard	$12,100_8$ locations
32K Standard	$15,200_8$ locations
32K Memory Protection	$17,000_8$ locations
Extended Core	$17,000_8$ locations

†With 16K, use is restricted to either batch or priority programs.
† †Less than 64 words.

VARIABLE RESIDENT AND COMMON

The variable resident area is used for nonresident executive overlay routines that are read into core to perform special job control functions.

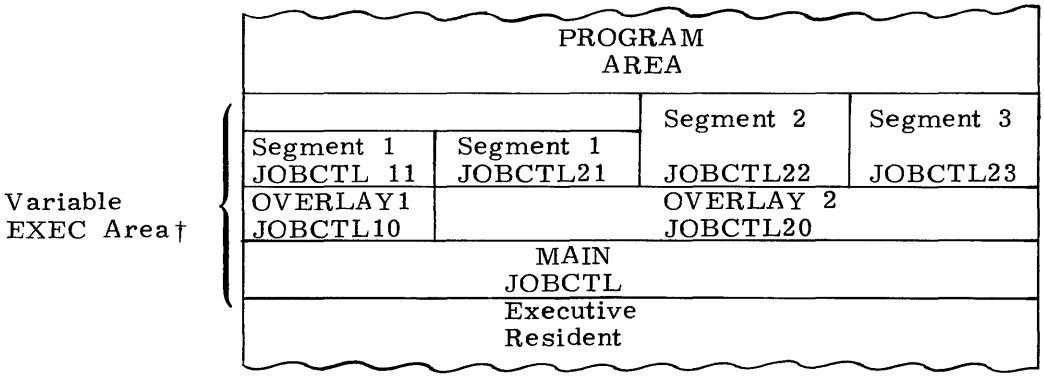


TABLE 8-1. VARIABLE EXEC JOB CONTROL FUNCTIONS

Section	Control Statements Processed
JOBCTL	JOB, SEQUENCE, PRIORITY, ENDREEL, EOJ, ENDSCOPE, LOAD, RUN, end-of-file, Library Pros. Name, DUMP, PAUS, CTO, CTL
JOBCTL10	EQUIP, AUX, FMT, DUMP, UNLOAD, XFER, REWIND, ABSTSK
JOBCTL11	TRAIN, operator control, AUT, MST
JOBCTL20	RAT, FET, RRAT
JOBCTL21	OPEN, CLOSE
JOBCTL22	ALLOCATE, EXPAND
JOBCTL23	RELEASE, MODIFY
JOBCTL24	RONL

The variable resident area is occupied by INITIAL at autoload time, and by the loader when relocatable binary program decks are being loaded for execution. The loader is loaded upward starting at the beginning of the executive resident area.

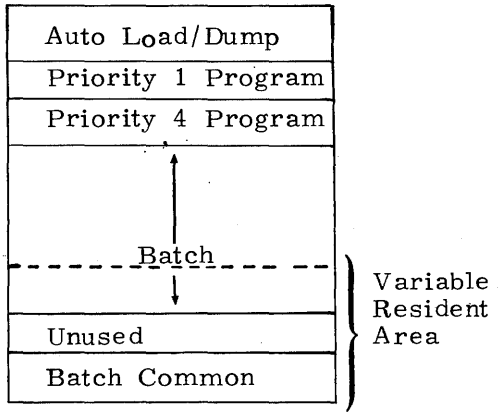
When a batch program with a common area is loaded, the loader assigns the common in the variable resident area. The loader assigns common upward from the executive resident area and loads the batch program downward from the priority program area. The batch program may be loaded downward into the variable resident area. † †

†Approximately $11,600_8$ memory locations.

† †In the extended core variant, common is assigned upward from EXEC2 in bank 1.

PRIORITY PROGRAM AREA

The use of priority programs is described in section 19. Priority programs are loaded downward from the autoload/autodump area toward the variable resident area. They cannot extend into the variable resident area or use common. The remaining area between the priority programs and variable resident is available for batch programs. In the extended core version of MSOS, priority programs can reside only in bank 0.

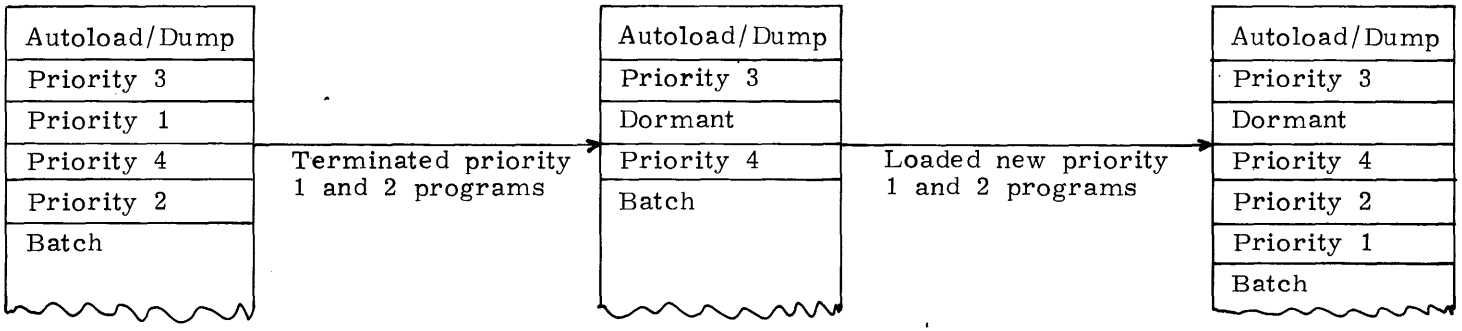


Only one program of each priority level may be in core at the same time. In standard MSOS without memory protection and in MSOS with programmable memory protection, the priority programs can be loaded in any order. In MSOS with standard memory protection (memory protect switches), priority 1 and 2 programs must be loaded first in any order. Then priority 3 and 4 programs can be loaded in any order below the priority 1 and 2 programs. A priority 1 or 2 program cannot be loaded after a priority 3 or 4 program has been loaded.

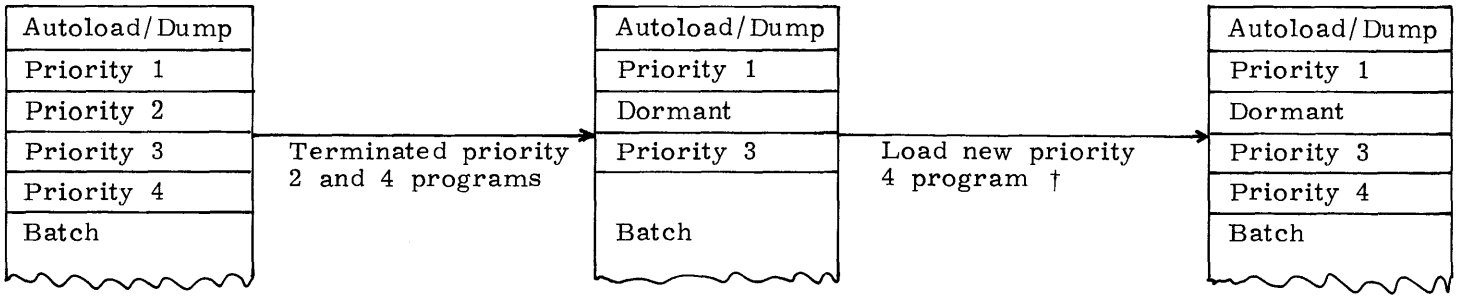
When a priority program terminates (refer to termination of priority programs in section 16), its block of memory will be returned to the system for batch use only if it was located in the lowest memory block assigned to priority programs. Otherwise, the memory block will become dormant or unusable.

Dormant memory blocks are recovered by terminating all priority and batch programs below the dormant memory block, and then reloading them.

Example 1: Standard MSOS or MSOS with programmable memory protection



Example 2: MSOS with standard memory protection.



†The priority 2 program cannot be reloaded unless priority 3 and 4 programs are terminated and reloaded.

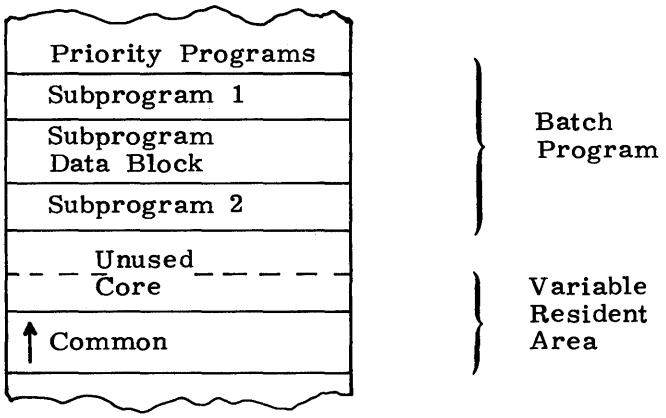
BATCH PROGRAM AREA

All memory locations between the priority programs and executive resident areas are available for batch programs. Common will be shared by all subprograms using it. Common will be assigned upward from the executive resident area. The size of the common will be the greatest length defined in any of the subprograms.

Only one subprogram may define a preset data area. All other subprograms in the program may reference and use the data area. † All batch programs, subprograms, and data blocks are loaded downward from the highest addresses in available core.

In the extended core variant of MSOS, bank 1 is reserved for batch programs only. The loader assigns batch common upward from EXEC2 and loads the batch program downward from the highest address in bank 1. Calls to executive subroutines (Table 18-2) can be made only with an RTJ instruction which contains the symbolic address of the routine. All other methods will connect to an address in bank 1 rather than the subroutine in bank 0 (refer to executive interrupts, section 18).

During execution of batch programs, the variable resident area in bank 0 is available for I/O buffers. Special CIO I/O calls may be made to use this area for I/O buffers (refer to read and write functions in section 11).



AUTOLOAD/AUTODUMP AREA

The autoload/autodump area contains short routines which load and initiate the autoload and autodump programs from mass storage. The autoload/autodump programs are permanently stored in the upper 40_8 address of memory. † †

MEMORY LIMIT TABLE

MSOS has two eight-word tables which are used to control core memory assignment and usage. The format is given in Table 8-2. The user references this table from a program written in COMPASS by declaring UMEMV50 and LMEMV50 as externals and reading the contents of the symbolic address into register A or Q. These values are used to determine the memory limits assigned to the program by the loader.

† Labeled data areas may be used only in programs written in ANSI COBOL or ANSI FORTRAN.

† † Starts at address 37737 (16K memory) or 77737 (32K memory). For 3100 and 3150 systems without enhanced block control, autoload/autodump area starts at address 37637 (16K memory) or 77637 (32K memory).

TABLE 8-2. MEMORY LIMIT TABLES

Upper Memory Limit		Lower Memory Limit	
Location	Contents (bits 14 through 00)		Contents (bits 14 through 00)
UMEMV50	Upper address protection switch settings (switches 0 through 5). †	LMEMV50	Lower address protection switch settings (switches 9 through 14). †
UMEMV50+1	Highest address in core used by the batch program.	LMEMV50+1	Lowest address in core used by the batch program. ††
UMEMV50+2	Highest address of the priority 4 program in core. Undefined if no priority 4 program is in core.	LMEMV50+2	Lowest address of the priority 4 program in core. Undefined if no priority 4 program is in core.
UMEMV50+3	Highest address of the priority 3 program in core. Undefined if no priority 3 program is in core.	LMEMV50+3	Lowest address of the priority 3 program in core. Undefined if no priority 3 program is in core.
UMEMV50+4	Highest address assigned to batch common.	LMEMV50+4	Lowest address assigned to batch common.
UMEMV50+5	Highest address of the priority 2 program in core. Undefined if no priority 2 program is in core	LMEMV50+5	Lowest address of the priority 2 program in core. Undefined if no priority 2 program is in core.
UMEMV50+6	Highest address of the priority 1 program in core. Undefined if no priority 1 program is in core.	LMEMV50+6	Lowest address of the priority 1 program in core. Undefined if no priority 1 program is in core.

†Undefined for dynamic memory protection or standard MSOS without memory protection.
††Does not include batch program common area.

MEMORY PROTECTION

There are three variants of MSOS with respect to memory protection.

1. Standard MSOS (no memory protection)
2. MSOS with standard memory protection (address protection switches)
3. MSOS with dynamic memory protection (memory bounds registers)

In all three variants, the autoload/dump area is permanently protected from all levels of programs (EXEC, priority 1 and 2, etc.) by a special wired protection circuit. Any program or task that attempts to write in the autoload/dump area will be terminated.

STANDARD MSOS

Standard MSOS operates in the nonexecutive mode. In the nonexecutive mode, all memory protection is disabled and any program can write anywhere in memory. Only the autoload/dump area is protected.

All user programs should be thoroughly tested before being used during normal system operation. A program error could write over and affect the performance of the system executive and other programs in core.

In the standard version of MSOS, priority 1 through 4 programs may be loaded in any order and they can reside in any order in the priority program area.

STANDARD MEMORY PROTECTION

The standard memory protection variant of MSOS uses address protection switches to protect the areas of memory occupied by the resident executive, priority 1 and priority 2 programs. If a batch, priority 3 or priority 4 program attempts to write in one of the protected areas, the write is inhibited. An illegal write interrupt is generated, and the violating program is normally aborted (refer to illegal write interrupts in section 18). In the extended core variant of MSOS, only memory bank 0 is protected with the address switches.

The address protection switches, located on the power control panel, must be set by the operator each time MSOS is autoloaded or whenever a priority 1 or priority 2 program is loaded or terminated. MSOS supplies the switch settings at the console typewriter whenever the switches need to be set or changed.

The memory protection switches set an upper unprotected bounds limit and a lower unprotected bounds limit. Batch and priority 3 and 4 programs cannot be loaded in nor can they write in any memory area above the upper bounds limit or below the lower bounds limit.

Since priority 1 and 2 programs run in the monitor state of executive mode with all memory protection disabled, these programs should be thoroughly tested before being run during normal system operation.

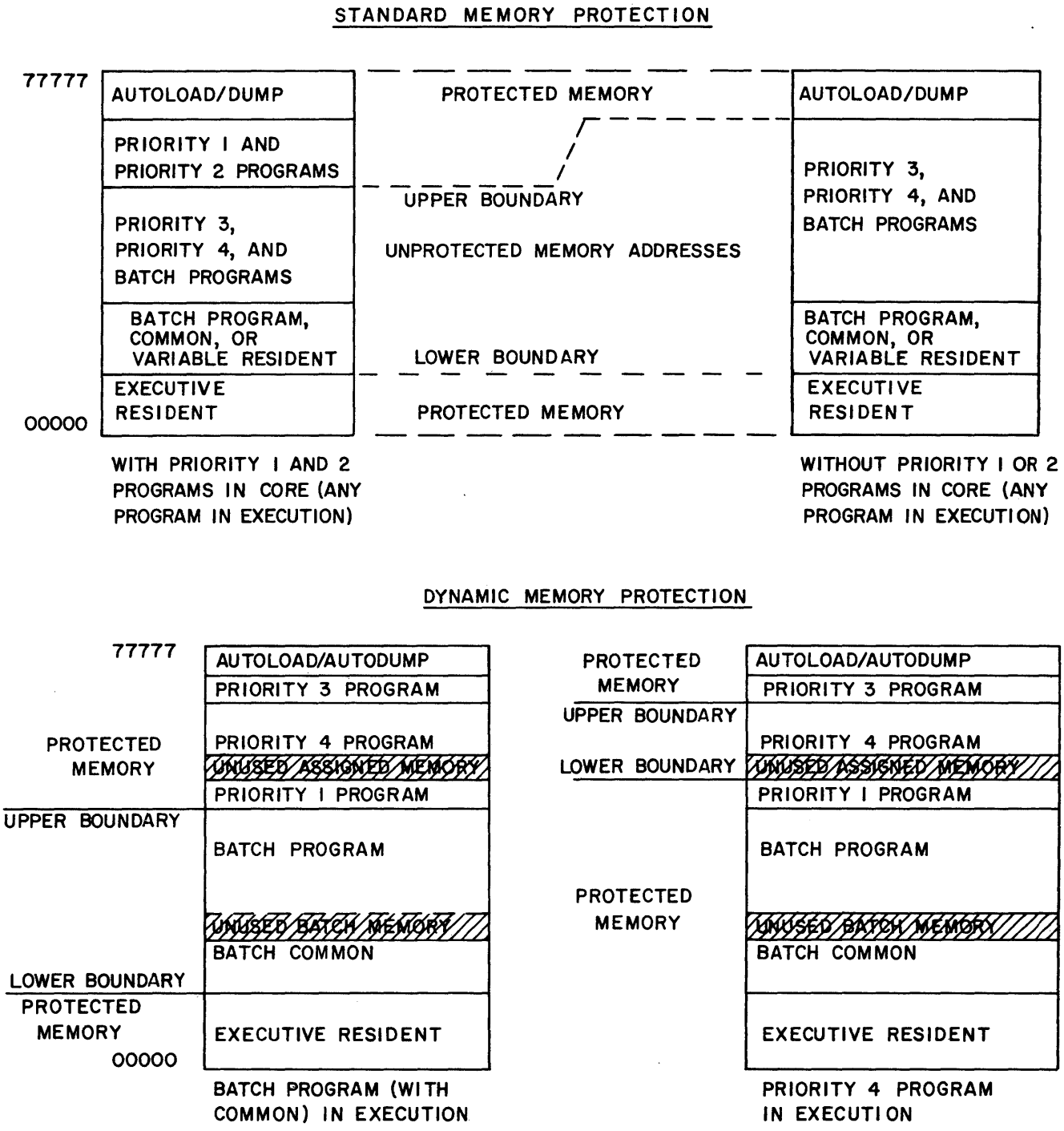


Figure 8-1. Standard Memory Protection

DYNAMIC MEMORY PROTECTION †

The dynamic memory protection variant of MSOS uses an address protection register to indicate the upper and lower unprotected memory bounds. MSOS automatically resets the bounds limits each time a new batch or priority 3 or priority 4 program is initiated. The new bounds are set in accordance with the programs assigned memory area in the memory limits table. In this system, all programs in core (EXEC, priority 1 through 4, and batch) are protected when a batch or a priority 3 or 4 program is in execution.

Priority 1 and 2 programs run in the monitor state the same as in standard memory protection. Memory protection is disabled when these programs are in execution. Since they can write any place in memory, these programs should be thoroughly tested before being run during normal system operation.

Priority 1 through 4 programs may be loaded in any order and can reside in any order in the priority program area.

MEMORY PROTECTION INCREMENTS

Standard and dynamic memory protection provide memory protection in increments of 1000_8 (512 decimal) words. This requirement causes the loader to adjust each program memory limits (in the memory limits table) so that they occur at an address evenly divisible by 1000_8. Therefore, to prevent wasting core, programs are written to fit in an area of memory that is an even multiple of 1000_8. For example, a program using 1005_8 instructions will be assigned 2000_8 words of memory. This wastes 773_8 words of memory, unless the extra memory could be used as a scratch area or as a double buffer area to increase program efficiency. The 512 word memory increments do not apply to standard MSOS without memory protection.

CORE MEMORY UTILIZATION

There are five main phases in processing programs with MSOS.

1. Autoloading MSOS
2. Job control (processing MSOS control statement)
3. Loading batch or priority programs
4. Executing a batch program
5. Terminating a batch program

Each of these phases uses a different core memory configuration.

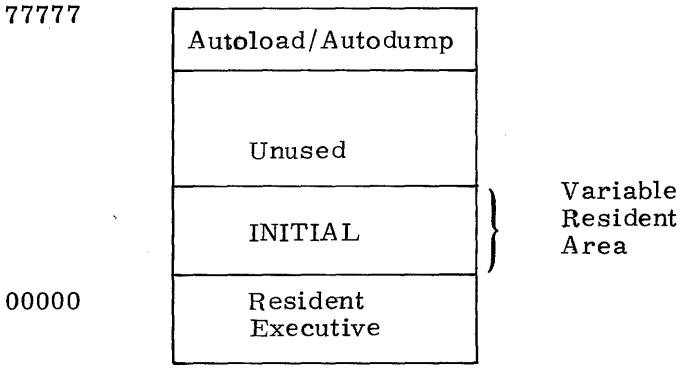
AUTOLOADING

When the operator presses the AUTOLOAD button, a forced jump is made to the autoload routine in upper core. The autoload routine loads the autoload program from mass storage, which in turn loads the executive resident and INITIAL from the RESFILE.

---

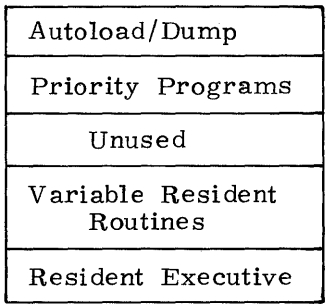
†Available only on 3100, 3150, and 3200 computer systems.

INITIAL is loaded in the variable resident area of core. INITIAL assigns standard systems units, opens system files, initializes system tables, etc., and returns control to resident exec for processing jobs. INITIAL will also initialize the 512 line printer, the MMTC image memories, and set up the linkage necessary to load software simulation routines if BCD and floating point hardware are not present.



JOB PROCESSING

Between jobs and during the initial phase of each job, the system control routines in executive resident load the variable resident routines into the variable resident area as they are needed. These routines process MSOS control statements input from the system input unit or from the console typewriter.



PROGRAM LOADING

When a control statement requests MSOS to load a user program, MSOS loads the relocatable loader into the variable resident area and uses all unused batch space for a symbol table area and a loading and linking area. One by one, the relocatable binary subprogram and referenced library routines are read into the loading and linking area. Here, all addresses are absolutized. If the loading and linking area fills, the absolutized subprograms are written on mass storage to make room for additional subprograms and library routines.

When all subprograms are absolutized, the programs on mass storage are reloaded. Then, if a $RUN card is included in the deck, the program will be executed. Loading is downward from the lowest priority or batch program in core.

When a RUN statement or card is processed, MSOS enters the program at the point specified by the primary transfer symbol on the program TRA card.

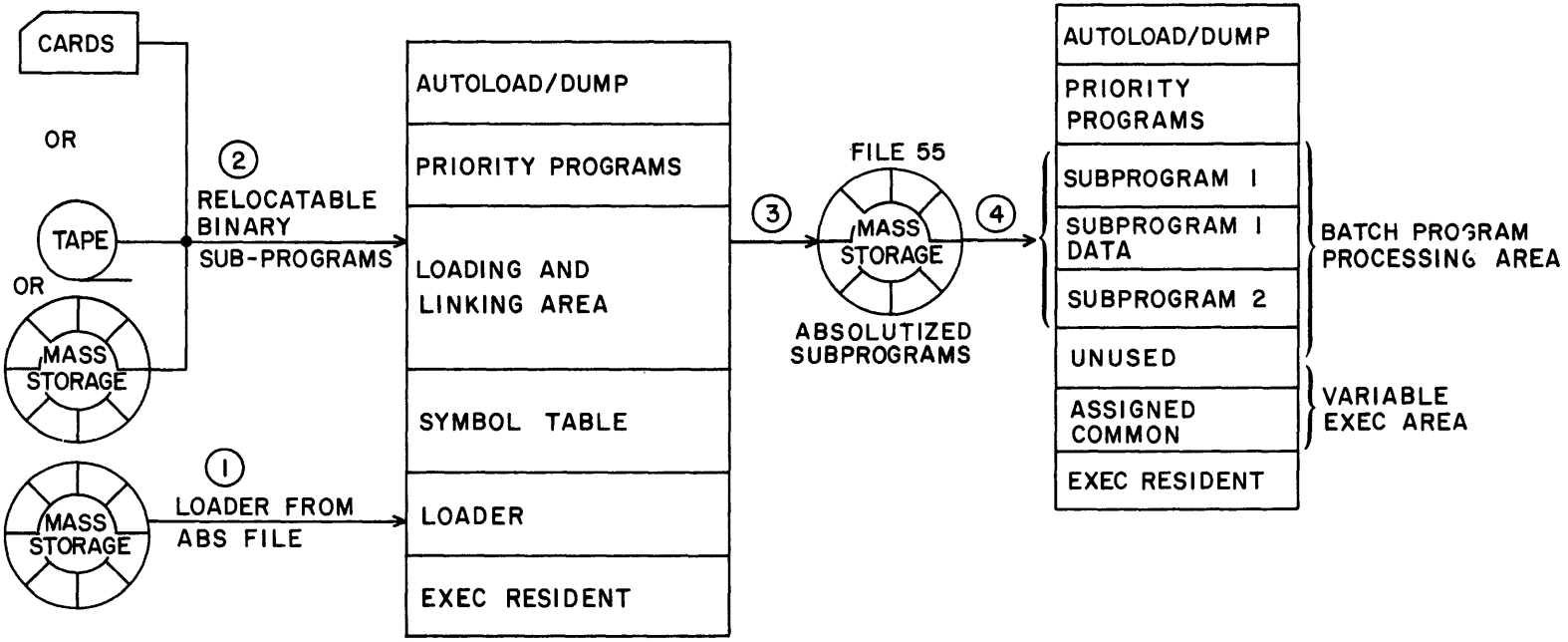
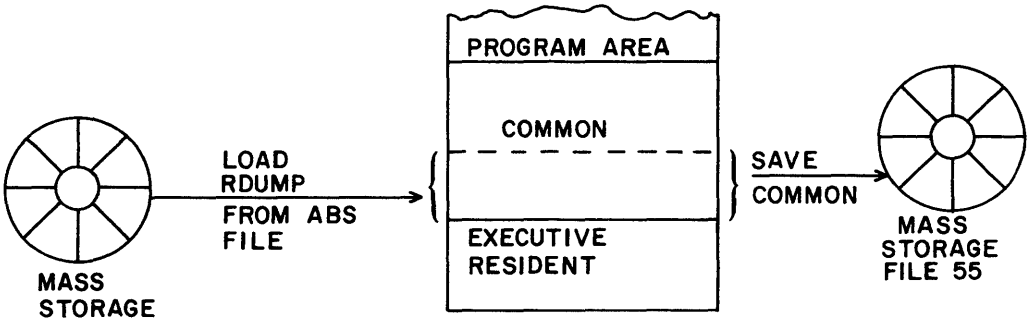


Figure 8-2. Loading Relocatable Binary Programs

PROGRAM TERMINATION PHASE

If an abnormal termination occurs and a DUMP card was included in the job deck, EXEC control routines load the recovery dump routine (RDUMP) into the lower portion of the variable resident area. All data in the area that RDUMP is loaded into is saved on mass storage. Normal program termination or abnormal termination without a DUMP card causes the system to revert to the job processing phase in order to process the next job and operator statements. In this case, no dump is taken for an abnormal termination. Only diagnostic error messages are printed. Refer to section 17 for abnormal program termination methods.



EXECUTIVE TABLES

Batch and priority programs may read from, but not write in, executive tables such as the memory limits and accounts tables. Executive tables are summarized in appendix F. In 16/32K MSOS systems, the tables may be read with load A or Q instructions using the symbolic table address and indexing.

In the 48/64K extended core variant of MSOS, batch programs reside in memory bank 1 and the tables are in memory bank 0. Therefore, a batch program must execute an ROS instruction before reading a table entry from bank 0, and the batch program must execute an RIS instruction before storing the table entry in bank 1. The ROS instructions cause all operand references to be made in bank 0 and the RIS instruction causes all operand references to be made in bank 1.

Example:

```
EXT   UMEMV50
  .
  .
  .
ENI   4,1              Enters index 1 with 4
ROS                    Selects bank 0 (operand state)
LDA   UMEMV50,1        Reads highest common address from UMEMV50+4
                       (memory limits table) in bank 0
RIS                    Selects bank 1 (program state)
STA   HICOMMON         Writes highest common address at HICOMMON in bank 1
  .
  .
  .
```

Batch programs that do not use RIS and ROS instructions when referencing executive tables are not run in an extended core system. However, batch programs that do not use RIS and ROS instructions when referencing executive tables are run in both the regular 16/32K system and in the 48/64K extended core system.

Indirect addressing cannot be used to reference executive tables in bank 0. Indirect addressing always references the same memory bank the program is in for new addresses, regardless of the use of ROS and RIS instructions. For example, after executing an ROS instruction in a batch program in bank 1, an LDA,I 53214 instruction reads the new address from location 53214 in bank 1, and then loads register A from bank 0 using the new address.

REGISTER FILE USAGE

Register file registers 00 through 37_8 are reserved for operating system use. Batch and priority programs can read these registers, but they must not write in them. These registers are protected in the memory protect and extended core variants of MSOS. Writing in them causes an illegal write interrupt.

Registers 40 through 77_8 are reserved for batch program use. Priority programs can read these registers, but they should not write in them.

The loader uses registers 50, 51, 52, and 54 and does not restore them before exiting to the caller. Batch programs that use the register file must save the contents of the registers being used before calling the loader, and restore the registers upon return from the loader.

RELOCATABLE LOADER 9

DESCRIPTION

The MSOS loader loads relocatable binary decks produced by compilers and assemblers into core memory in preparation for execution.† The loader performs the following functions.

1. Accepts relocatable binary input decks from the card reader, mass storage files, or tape files.
2. Links independently compiled subprograms that reference each other through external symbolic entry point names (that is, with XNL and EPT cards).
3. Loads library routines that are called for in a subprogram or library routine and links them to the subprogram or routine that called them.
4. Prepares absolutized overlays and writes them on a file from which they can be called by the main section of the program.
5. Loads and links BCD hardware simulation and floating point hardware simulation (FDP) routines when the hardware is not present and BCD or floating point instructions are used.
6. Prints a loader map of the memory area allocated to the program and any overlays that are used.
7. Returns the main and second program entry addresses to the program that called the loader.

NOTE

The loader does not load I/O drivers from the library. All I/O drivers to be used must have been part of executive resident when the system was installed.

When linking subprograms, the loader assumes that all external reference symbols from XNL and EXS cards that were not matched with an entry point on an EPT card are entry point symbols for library subprograms or routines. The loader searches the library directory for the symbols and loads and links the subprograms from the library. If a symbol is not located in the library directory, an undefined symbol error occurs.

LOADING THE LOADER

The relocatable loader may be loaded and called into execution by any of the following methods.

1. By use of a LOAD or library program name statement on the system input unit.

† The relocatable binary cards are described in section 6.

2. By use of the IDC card in a relocatable binary deck.
3. By the operator typing a library program name or LOAD statement at the console typewriter.
4. By calling RDCKF1 to load the loader in a COMPASS batch program.

Methods 1 through 4 load the loader from the ABS file, and then initiate the loading function. If the loader is loaded within a batch program (method 5), a jump must be made to the loader entry point (LOADER) to initiate the loading function.

RDCKF1 is a routine in executive resident that loads absolutized programs from the ABS file on mass storage.† The contents of the A, Q, and B3 registers must be set as follows before using RDCKF1 to load the loader. Then a return jump may be taken to RDCKF1.

(A) register = LOAD
(Q) register = ER$\Delta\Delta$
(B3) register = 77777_8

The contents of A and Q specify the name of the subprogram to be loaded, and the contents of B3 specify how many words are to be loaded from the ABS file. Setting B3 equal to 77777 specifies all of the subprogram.

RDCKF1 loads the loader into assigned batch common in the variable executive area. However, RDCKF1 does not load the loader if batch subprograms extend into the area of variable resident that is needed by the loader. RDCKF1 returns with a zero in register A if it cannot load the loader.

LOADING PROGRAMS

Once loaded, the loader loads subprograms downward in core, starting at the end of the batch program. The loader loads subprograms into the variable resident area up to the resident executive area (that is, the loader overlays itself). Before entering the loader, the loaders function must be specified by an entry in the A and Q registers. Then the loader may be entered by a return jump to LOADER.

†Refer to section 10 for a description of RDCKF1.

Contents of A and Q	Function	Results
A = zero Q = zero	Loads from the system input unit.	1. Loads and links relocatable binary subprogram decks from the system input unit until an ELD card or an end-of-file is detected. 2. Loads and links library routines that are referenced in the subprograms.
A = zero Q = 0,f1,f2,f3 f1 through f3 are file numbers. The file numbers must be converted to octal and entered in bits 17-00 of register Q. f1 bits 17-12 f2 bits 11-06 f3 bits 00-05	Loads from files f1 through f3.	1. Loads and links relocatable subprograms and routines from files f1, f2, and f3. File f1 is read first, then f2, and then f3. A file number of 00 is skipped. 2. Loads and links library routines that are referenced in the subprograms.
AQ = cccccccc cccccccc is a 1 to 8 character BCD library entry point name. The entry point name must be left-justified in the A and Q registers with blank fill. The left most character must be an alphabetic character.	Loads the named library routine from the library or auxiliary library.	1. Loader loads the named library program and all other library programs referenced by it.

Upon return from the loader, bits 14 through 00 of the A register contain the transfer point address of the program that was loaded. Bits 14 through 00 of the Q register contain either a second transfer point address for the program or zero, and bits 23 through 18 of the Q register contain the number of errors that occurred, if any. The contents of the index registers are undefined.

Example:

```
        LDAQ    =2HLOADER
        ENI     -0,3            Set Be to 77777
        RTJ     RDCKF1          Load the loader
        AZJ,EQ  ERR             Check for error
        LDAQ    =2HUPDATE
        RTJ     LOADER          Load UPDATE
        AZJ,EQ  ERR             Check for no transfer address
        SWA     TAG             Save transfer address
        SHAQ    6               Shift error bits from Q to A
        SHQ     18              Reposition secondary transfer address
        ANA     77B             Check for error bits
        AZJ,NE  ERR             Jump to ERR if loader error
TAG     RTJ     **              Enter UPDATE
ERR     Error processing routine
```

When a batch program is loaded, the first unused address below the batch program is stored in memory location LDMEMV50. If the loader is called in a batch program, it starts loading subprograms or routines downward starting at the address contained in location LDMEMV50. The loader also updates LDMEMV50 so that at the completion of loading, LDMEMV50 contains the first unused address below the last subprogram or routine that was loaded.

The contents of LDMEMV50 can be read and modified in a batch program before calling the loader. This allows loading new subprograms or routines over old subprograms and routines currently in core.

LDMEMV50 can be read with LDA and LDQ instructions. The following set of instructions can be used to change the address in LDMEMV50.

```
ENA     LDMEMV50
ENQ     new address
RTJ     RSTOREQ
```

The following points must be noted when using the loader in COMPASS programs.

1. The loader is not reenterant. It cannot be called twice in the same program unless each subsequent call is preceded by a call to RDCKF1 to reload the loader.
2. The loader is loaded over the batch common area.
3. Binary input programs on mass storage files must be in 240-word blocks (six binary cards per block). On tape files they must be in 40-word binary blocks.
4. On mass storage, a card image with 17000000 in the first word position (7_8 punch in column 1) is interpreted as an end of file mark and terminates loading from the file.
5. The loader cannot be called by COMPASS in a priority program.
6. The loader uses file registers 50, 51, 52, and 54, and does not restore their initial contents before returning to the users program.

LOADER MAP

The loader produces a map of the memory area assigned to each program it loads. A printout of the map will occur if the M parameter is included on the LOAD card or in a call to an assembler or compiler. The map headings are as follows:

Heading	Description
SUBP	Subprogram names and first word addresses
ENTR	Entry points in the subprogram
LDTA	Name and first word address of a labeled data area
COMM	First word address of common area
DATA	First word address of data area
PEXT	First word address of a program extension area

Example:

```
    SUBP
77514   RAAR        77574   EXECOVR   77772   MAINTEST
    ENTR
14040   BCDBOXS     77574   EXECOVR   14040   FDPBOXS
77772   MAINTEST    77514   RAAR
    LDTA
        NONE
    COMM
        NONE
    DATA
77763   NONE
    PEXT
        NONE
```

In the SUBP and ENTR maps, the absolute address precedes the subprogram name. The word NONE is printed when no information accompanies a heading. Each map starts on a new page.

ABSOLUTE LOADER 10

The absolute loader (RDCKF1) loads absolutized subprograms and routines from the ABS file into core.† The procedure for using RDCKF1 in COMPASS programs is as follows:

1. Declare RDCKF1 as an external.
2. Load registers A and Q with the name of the routine (in BCD) to be loaded. The name must be left-justified and blank-filled to the right.
3. Enter the number of words to be loaded in register B3. Use 77777 if the whole routine is to be loaded.
4. Do a return jump to RDCKF1.
5. On return, check contents of register A. If A is zero, RDCKF1 could not load the requested routine due to unavailable core or inability to locate the routine in the ABS file. If A is not zero, A contains the first word address of the routine that was loaded.
6. If parameters are used by the subprogram or routine, enter parameters or values in the A and Q registers.
7. Do a return jump to the primary entry point of the loaded routine. For the LOADER and other routines that have SEPOINTS in the library, the symbolic entry point may be declared external and used for the entry. For routines without SEPOINT entries in the library, the user must use an absolute entry point address (that is, usually the first word address plus a known constant).

NOTE

In the extended core variant of MSOS, batch programs cannot use an absolute address to enter ABS file routines that load in memory bank 0. An address in bank 1 will be referenced rather than the bank 0 address.

If RDCKF1 cannot load the specified subprogram or routine from the ABS file, RDCKF1 calls ABNORMAL and terminates the job. The only exception is if the requested routine could not be found in the ABS file (invalid name in the A and Q registers). In this case, RDCKF1 returns to the requesting program with zero in register A.

†The ABS file contains absolutized library routines. Special user subprograms and routines may also be added to the ABS file at system installation time. The ABS file holds up to 71 absolutized subprograms or routines. The ABS file is described in the MSOS V5 Installation Handbook. To load absolutized user subprograms from other mass storage files, use the ABSTSK card (refer to section 4).

LDABSV50 - LOAD ABSOLUTE TASKS

LDABSV50 is called by COMPASS to load and execute absolute tasks from a file. This routine may be used only for batch mode. The I/O must be idle since there is no return to the user from this call; the loaded task terminates the job. The calling sequence for LDABSV50 is:

```
EXT LDABSV50
RTJ LDABSV50
```

The user must load the task name in AQ (AQ contents are left-justified and 0-filled) and the file number in B1. The absolute file that LDABSV50 loads must be the MAIN task of an overlay program. The new task must be set up as an overlay program. The new task can share common with the old task.

INPUT OUTPUT CONTROL (CIO) 11

DESCRIPTION

Central input/output (CIO) provides MSOS COMPASS users with a standard method for reading, writing, and controlling the system I/O equipment. With the memory protection or extended core variants of MSOS, CIO must be used for I/O functions in batch and priority 3 and 4 programs. Any attempt to use COMPASS I/O instructions in these programs generates an executive interrupt that aborts the job.

In standard MSOS (no memory protection) and in priority 1 and 2 programs, the user has the option of using CIO or COMPASS I/O instructions. When doing I/O on units defined to the system and entered in the AUT table, the user should not use COMPASS instructions. If COMPASS I/O instructions are used on units defined in the system, they must be used with the utmost caution. COMPASS I/O instructions could conflict with those issued by CIO for the same unit and cause an irrecoverable error in both programs.

A return jump to CIO is the standard method of using CIO. The return jump calling sequence will generally take the following form.

```
P       RTJ     CIO
P+1     f       u,i
P+2     jump    raddr
P+3     Additional parameters depending upon the
        function that was selected.
 .       .
 .       .
 .       .
P+n     Next instruction in the program (normal return address).
```

f	Function code (read, write, locate, rewind, backspace, etc.).
u	File number
i	Selects interrupt upon completion of I/O (optional).
raddr	Reject address. Address of first word of user supplied routine for processing rejected CIO requests.

When entered, CIO reads the parameters (P+1 through P+n) and processes the call. The I/O request is rejected and control is returned to address P+2 in the program if an illegal function code was found, a wrong file number was used, the unit was not ready, the restrictions on using the system files were violated, or the unit is busy. If the function is legal and the I/O unit is ready, CIO initiates the I/O operation and returns control to the program at address P+n.

If an interrupt on I/O completion was not selected in the CIO call, the user must check the equipment status to determine when the I/O function has been completed. If an interrupt is selected, the unit status is in the A and Q registers when control is switched to the interrupt processing routine. Refer to input/output interrupts in section 18 for a discussion on the use of interrupts with CIO.

CIO saves the contents of the B registers, but not the A and Q registers. The user must save the contents of the A and Q registers before calling CIO.

READ/WRITE FUNCTION

The return jump sequence for a read or write function is as follows:

P	RTJ	CIO†
P+1	f	u,i
P+2	jump	raddr
P+3	m,c	fwca
P+4	c	n
P+5	00	iaddr

P+6 Next instruction in program (normal return address). For mass storage reads and writes, register A contains the number of the next sequential block in the file.

f One of the following read or write functions.

- 01 Read
- 02 Write
- 03 Read backward (tapes only)
- 11 Write end-of-file mark (only for tapes and punches)
- 17 Write check (mass storage only)
- 41 Read with error recovery
- 42 Write with error recovery

u File number (two decimal digits)

i Interrupt selection

- 0 No interrupt selection
- 1 Interrupt on abnormal I/O termination
- 2,3 Interrupt on normal completion or abnormal termination

raddr First word address of the reject processing routine. Register A contains a reject code if control is returned to this address (refer to reject processing). If register A equals zero, the unit is busy. Retry the I/O at a later time.

m Six-bit mode designator that specifies BCD or binary I/O. Also indicates density for magnetic tapes. For all I/O except the incremental plotter and magnetic tape unit, the following codes should be used.

- 40 BCD mode
- 41 Binary mode

For magnetic tape units, one of the following codes should be used.

	7-Track Drives		9-Track Drives	
Code	Mode	Density	Mode	Density
40	BCD	No change	Convert	No change
41	Binary	No change	Pack	No change
50	BCD	200 bpi	Convert	800 cpi
51	Binary	200 bpi	Pack	800 cpi
60	BCD	556 bpi	Convert	800 cpi
61	Binary	556 bpi	Pack	800 cpi
70	BCD	800 bpi	Convert	1600 cpi
71	Binary	800 bpi	Pack	1600 cpi

†For paper tape, use PTIOV50.

For incremental plotter, 40 specifies character mode and 41 specifies word or disassembly mode.

Mode	With Character I/O	With Word I/O
40	All characters plotted	Only bits 00 through 05 (lower character) of each word is plotted.
41	All characters plotted	All characters in each word are plotted.

,c — Indicates whether fwca is a character address or a word address.

- ,C — Character address. Character address is illegal for mass storage, card punches, card readers, and reverse tape reads.
- omitted — Word address

fwca — First address of I/O buffer area. If mode designator (m) is followed by a C, fwca is character address (bits 16 through 00). Otherwise, fwca is word address (bits 14 through 00). fwca must be a symbolic address (that is, start with an alpha character).

Example:

m,C	fwca	Character address
m	fwca	Word address

c — Indicates if data transfer is to be by character or words.

- 00 — Word I/O
- 40 — Character I/O (illegal for mass storage, card readers, card punches, and reverse magnetic tape reads).

n — Number of words or characters to be input or output. One to six octal characters. Normally equal to the block size of the file.

iaddr — First word address of the interrupt routine. If an interrupt was not selected, iaddr is not used. Address P+5 is the normal return address instead of P+6.

The read or write function starts the read or write operation and then returns to the program. The program must use either an interrupt selection or a status check to determine when the I/O operation is completed. If interrupt on normal completion or abnormal termination is selected, the unit status word must be checked in the interrupt processing routine to determine whether normal completion or abnormal termination occurred. The unit status words are in registers A and Q when the system enters the I/O interrupt processing routine.

The maximum buffer size that should be used for line printers is 136 characters. Each write prints one line (135 characters plus one control character); the remaining characters in the buffer are truncated. Refer to appendix B for print control characters.

The maximum buffer size for the console typewriter (CTO) is 80 BCD characters. Refer to appendix A for typewriter character codes and functions.

The maximum buffer size that should be used for punching cards is 20 words for BCD mode, and 40 words for binary mode. Each write punches only one card; the remaining characters in the buffer are truncated.

The write-check function is used after a mass storage write function to ensure the data is written correctly. The write-check function reads the last block written and compares it with the contents of addresses fwca through fwca+n. If only one output buffer is being used, it cannot be updated for the next write until CIO completes the write-check function.

After a write-check function is initiated, CIO returns control to the program. When CIO completes the write-check function, the user should check the unit status for a write-check error before issuing the next I/O request on the unit. The user may periodically check status or use an interrupt to determine when the write check has completed.

The write with or read with error recovery function (f=41 or 42) does not return control to the program until the I/O operation is completed. When a write function with error recovery is selected for mass storage, a write check is also performed before returning to the program. If an I/O error is encountered, the standard system error recovery routines are used to recover from the error and complete the I/O operation.

The interrupt on I/O completion is illegal when the read with error recovery or write with error recovery function is selected. Therefore, normal return (I/O completed) will be at address P+5. If an irrecoverable error or a reject condition occurs, the return is at address P+2 (reject return). Register A will contain an error code of 12 for an irrecoverable error (refer to Table 11-4). Refer to the paragraph on I/O error recovery for programming alternatives when an irrecoverable I/O error occurs.

When a write function is used with a line printer, the first character in each line of print is a control character that is not printed. Refer to appendix B for a list of these characters and their functions.

In the extended core variant of MSOS, the bank 0 common area, which is memory locations between the lowest priority program and executive resident, may be used for I/O buffers by batch programs residing in bank 1. To use bank 0 common, the following CIO calling sequence must substitute for the previously described CIO calling sequence.

```
P       RTJ     CIO
P+1     VFD     A6/f,A3/i,O6/0,A2/2,A7/u
P+2     jump    raddr
P+3     m,c     fwca
P+4     C       n
P+5     00      iaddr
```

The parameter values are the same as those used in the previous CIO calling sequence. However, the fcwa is an address in bank 0 rather than bank 1.

When using I/O buffers in bank 0, RIS and ROS instructions must be used to bracket the load and store instructions, the same as when referencing executive tables in bank 0 (refer to referencing executive tables in section 8). The available buffer area is determined by referencing the memory limits table.

For function codes 01, 02, 17, 41, and 42 on mass storage on a normal return (P+5 if I=0; P+6 if I≠0), the A register contains the next block number (NBN). For example, if a request transmits data to or from blocks 1 and 2, on normal return A=3. On reject return, an error code appears in the A register.

MASS STORAGE LOCATE FUNCTION

On mass storage files, a locate function must be used to select the next block before a new record is read or written. A locate sets the block number pointer to the next block or to any specified block number. Once a block is located, all I/O is restricted to that block until another locate function is requested. The return jump sequence for a locate function is as follows:

P	RTJ	CIO
P+1	15	u,i
P+2	jump	raddr
P+3	xy	zeros
P+4		s
P+5	00	iaddr
P+6	Next instruction in the program (normal return address). Register A contains the number of the block that was just located.	

u — File number (two decimal digits)

i — Interrupt selection

0	No interrupt selected
1	Interrupt on abnormal termination
2,3	Interrupt on normal or abnormal I/O termination

jump — UJP or RTJ instruction

raddr — First word address of a reject processing routine. Register A contains a reject code when a return is made to P+2 (refer to reject processing).

xy — Two octal digits that specify optional read/write head movement and select the next block for I/O.

x = 0	Position the read/write heads
x = 1	Do not position the read/write heads, † and negate any interrupt selected with the i parameter.
x ≠ 1,0	Illegal
y = 0	Replace the current block number with the s parameter (value in address P+4).
y = 1	Replace the current block number with the highest block number written plus one.
y = 2	Replace the current block number with the current block number plus the s parameter (value in address P+4)
y = 3	Replace the next available block number with the s parameter (value in address P+4). Does not change current block number.
other	Illegal

s — A 24-bit octal number used in conjunction with the y parameter (described above).

iaddr — First word address of the interrupt processing routine. The mass storage status word is in registers A and Q when control is passed to this address.

If an interrupt was not selected (with i in address P+2), iaddr is not used. Address P+5 becomes the normal return address instead of P+6.

---

†If the heads are not positioned by the locate function, the next read or write function positions the heads over the track containing the current block.

The following is an example of using the locate and write-check functions to write blocks of data on a mass storage file.

1. Open the file.
2. Locate to the next block to be written on.
3. Load the buffer area.
4. Issue a write request to CIO. On normal return, continue processing until ready to use the buffer area again.
5. Check status to assure the write function completed. If the status check indicates an I/O error, skip to step 8.
6. Issue a write-check function request to CIO. On normal return, check status until the function completes. If the status check indicates an I/O error, skip to step 8.
7. Repeat steps 2 through 6 to write the next block of data.
8. Make a call to the system I/O error recovery routine (SCARV50).
9. If a normal return occurs (register A = zero), repeat steps 2 through 7 until all data has been written.
10. If an abnormal return occurred (register A ≠ zero), abort the job, try another unit, or accept the error. Register A contains an error code indicating the type of error.

Some users may elect to select interrupts on completion of I/O or write check functions and do all or some of the above processing in the interrupt routine. Depending upon the program application, interrupts normally provide faster I/O and more processing time between I/O functions.

UNIT RECORD DEVICE CONTROL FUNCTIONS

The return jump sequence for unit record device control functions are as follows:

P	RTJ	CIO
P+1	f	u,i
P+2	jump	raddr
P+3	00	iaddr
P+4	Next instruction in program (normal return address)	

f One of the following unit record control functions:

04	Rewind
05	Rewind and unload†
06	Backspace
07	Skip one file mark forward
10	Skip one file mark backward
11	Eject page
12	Erase

u File number (two decimal digits)

i Interrupt selection

0	No interrupt selected
1	Interrupt on abnormal termination of the function
2,3	Interrupt on normal completion or abnormal termination of the function

†For 601 units, rewind and drop ready status.

TABLE 11-1. CIO FUNCTION CODES

FUNCTION	601/603/ 606 Tape Drives	604/607/ 608 Tape Drives	657/659 Tape Drives	415 Card Punches	405 Card Readers	813/814/ 841/853/ 854/863 Mass Storage	501/505/ 512/3254/ 580 Line Printers	Console Typewriter	3293 Plotter	3691 Paper Tape Station		211 Display/ Entry Station
										Paper Tape Reader	Paper Tape Punch	
Read one record	01	01	01	--	01 Read 1 card	01	--	01	--	01	--	01
Read one record backward	--	03	03	--	--	--	--	--	--	--	--	--
Write one record	02††	02††	02††	02 punch card	--	02	02††† Print one line	02	02	--	02 punch 1 record	02
Write end of file	11††	11††	11††	11 punch EOF card	--	--	11 eject page	--	--	--	11	--
Rewind to load point†††††	04	04	04	--	--	--	--	--	--	--	--	--
Rewind and unload†††††	05†	05	05	--	--	--	--	--	--	--	--	--
Backspace one record††††	06	06	06	--	--	--	--	--	--	--	--	--
Skip one file mark forward	07	07	07	--	--	--	--	--	--	--	--	--
Skip one file mark backward	10	10	10	--	--	--	--	--	--	--	--	--
Erase six inches of tape (forward)	12††	12††	12††	--	--	--	--	--	--	--	--	--
Read with error recovery	41	41	41	--	41 read 1 card	41	--	--	--	--	--	--
Write with error recovery	42††	42††	42††	42 punch 1 card	--	42	42 print 1 line	--	--	--	--	--
Locate	--	--	--	--	--	15	--	--	--	--	--	--
Write check	--	--	--	--	--	17	--	--	--	--	--	--

†On 601 tape drives, rewind and drop READY status (no unload).
††Write ring must be on the tape reel to perform these functions.
†††First character of line is carriage control character. Second character is first character printed.
††††Space forward one record if last function was 03 or FORMAT code 06 was selected.
†††††End of operation interrupt is illegal with this function. Controller will be available for servicing other tape drives as soon as the unload function has been initiated.

raddr First word address of the reject processing routine. Register A contains a reject code (refer to reject processing).

iaddr First word address of the interrupt routine. If an interrupt is not selected with i in P+2, iaddr is not used and P+3 is the normal return address rather than P+4.

UNIT RECORD DEVICE FORMAT FUNCTIONS

The format function selects specific operating modes for a specific unit record device, without doing any I/O. The return jump sequence for a format function is as follows:

P	RTJ	CIO †
P+1	14	u,v
P+2	jump	raddr
P+3	Next instruction in program (normal return address)	

u,v The file number and the function code. Use u,v for all but the 512 printer.

u File number
v Format function that is described in Table 11-2

For the 512 printer, use the following format.

06/14, A5/v, A13/u

u File number
v Format function that is described in Table 11-2

raddr First word address of the reject processing routine.

The format function is not applicable to mass storage or the card reader.

† For paper tape, use PTIOV50.

TABLE 11-2. FORMAT CODES FOR UNIT RECORD DEVICES

Format Code	601/603/606 Tape Drives	604/607/608/657 Tape Drives	659 Tape Drive	512/580 Printer	501/505/3254 Printers	415 Card Punch	3691 Paper Tape Read/Punch
00	--	--	--	No operation	Page eject (skip to level 8)	No operation	--
01	Select BCD mode	Select BCD mode	Select conversion (BCD) mode	Eject page (skip to level 1)	Skip to level 1	Punch in binary mode	Select conversion mode
02	Select binary mode	Select binary mode	Select pack (binary) mode	Skip to level 2 punch	Skip to level 2	Punch in Hollerith/ ASCII mode	Select binary mode
03	Select 200 bpi density	Select 200 bpi density	Select 800 cpi density	Skip to level 3 punch	Skip to level 3	Select offset stacker†	Select BCD mode
04	Select 556 bpi density	Select 556 bpi density	Select 800 cpi density	Skip to level 4 punch	Skip to level 4	Check read last card punched††	Select ASCII mode
05	--	Select 800 bpi density	Select 1600 cpi density	Skip to level 5 punch	Skip to level 5	Clear punch	--
06	--	Select reverse read	Select reverse read	Skip to level 6 punch	Skip to level 6	No operation	--
07	--	Select forward read	Select forward read	Skip to level 7 punch	Skip to last line (level 7)	No operation	--
10	--	--	--	Skip to level 8 punch	--	--	--
11	--	--	--	Skip to level 9 punch	--	--	--
12	--	--	--	Skip to level 10 punch	--	--	--
13	--	--	--	Skip to level 11 punch	--	--	--
14	--	--	--	Skip to last line (level 12)	--	--	--
15	--	--	--	Select extended character array	--	--	--
16	--	--	--	Clear extended character array	--	--	--
17	--	--	--	Enable load-image-memory	--	--	--
20	--	--	--	Release load-image-memory enable	--	--	--

†Applicable only to next card to be punched, not card currently being punched.
††Only when using a 3644 card punch controller. No operation for all other card punch controllers.

TABLE 11-3. UNIT STATUS CODES RETURNED BY CIO

Hardware Type	Status Word Bits							
	15	14	13	12	11	10	09	08
601/603/606 Tape drives	Channel parity error: 1=yes 0=no	--	--	--	Mode selection 1=binary 0=BCD (conversion mode on 659 drives).	Tape parity error: 1=error 0=no error	End of operation: (1) 1=yes 0=no	Lost data (2) 1=yes 0=no
604/607/608/657/659 Tape drives		Reverse motion selected: 1=yes 0=no	--	Type of drive: 1=9 track 0=7-track				
512/580 Line printer	Channel parity error: 1=yes 0=no	Lines/inch mode: 1=8 lines/inch 0=6 lines/inch	Auto eject mode: 1=set 0=not set	6 to 8 lines/inch switching position (4) 1=yes 0=no	All output buffer characters are on print train: 1=no 0=yes	Print error: (5) 1=yes 0=no	End of operation: 1=yes 0=no	Post print page eject selected 1=yes 0=no
501 line printer		--		--	--	--		
Optical character reader	Channel parity error: 1=yes 0=no	--	Scan mode: 0=scan3 1=scan2	Mirror at specified coordinates: 1=positioned 0=not positioned	--	Line locate error: 1=error 0=no error	End of operation: 1=yes 0=no	Lost data: 1=yes 0=no
3691 Paper tape station		Unconvertable ASCII character was read: 1=yes 0=no	Last function was: 1=a read 0=not a read	Code type is 1=8 level ASCII 0=6 level flexowriter	Mode is: 1=binary 0=code conversion (Refer to bit 12)	Parity error (used only with ASCII code): 1=error 0=no error		--
405 Card reader	Parity channel error: 1=yes 0=no	--	Mode selection: 1=binary 0=BCD to Hollerith/ASCII	Compare or pre-read error: (6) 1=yes 0=no	Last card read was a binary card: (7) 1=yes 0=no	--	End of operation: 1=yes 0=no	Reader malfunction: (3) 1=yes 0=no
415 Card punch		Current card is: 1=offset 0=not offset	Next card selected for offset: 1=yes 0=no	--	Binary punch mode selected: 1=yes 0=no	Read compare error: 1=yes 0=no		Card feed failure: 1=yes 0=no
3293 Plotter	Channel parity error: 1=yes 0=no	--	--	--	Assembly mode selected: 1=yes 0=no	--	End of operation: 1=yes 0=no	--
Console typewriter	--	--	--	--	--	Repeat key pressed: 1=yes 0=no		--
813/853/854/841 Mass storage (disk)	Channel parity error: 1=yes 0=no	--	--	--	--	Read sum check error: 0=no 1=yes	End of operation: 1=yes 0=no	Lost data or 841 operation error: 0=no 1=yes
863 Mass storage (drum)		--	--	--	--	Read parity error: 0=no 1=yes		--
211 Display/entry station	Channel parity error: 1=yes 0=no	--	--	--	--	Poller error: 0=no 1=yes	Alert interrupt: 0=not active 1=active	End of operation interrupt: 0=not active 1=active

Special Notes

1. End of operation status bit is cleared when a new operation is started.
2. Hardware failure.
3. Feed failure, output stacker full, or card jam.
4. Form is positioned so switching from 6 to 8 (or 8 to 6) lines per inch will not cause overprinting.
5. Non-printable character was output to printer.
6. Second read did not match first read, or read before card was in read station did not sense all one's.
7. 7, 9 punch in column one.

General Note

-- Indicates unused and undefined.

TABLE 11-3. UNIT STATUS CODES RETURNED BY CIO (Cont'd)

Hardware Type	07	06	05	04	03	02	01	00
601/603/606 Tape drives	--	Density: 1=556 cpi 0=200 cpi	End of tape mark sensed: 1=yes 0=no	Tape at load point: 1=yes 0=no	EOF mark read or written: 1=yes 0=no	Write enable set: (8) 1=set 0=cleared	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
604/607/608/ 657/659 Tape drives	Density: 0=200 bpi (not for 659s) 1=556 bpi (not for 659s) 2=800 bpi (cpi for 659s) 3=1600 cpi (for 659s only)							
512/580 Line printer	Post print skip last line selected 1=yes 0=no	Buffer memory busy: 1=yes 0=no	Paper fault (out of paper or paper jam) 1=fault 0=no fault	Form positioned at level 9: 1=yes 0=no	Form positioned on last line of page: 1=yes 0=no	Image buffer input enabled: 1=yes 0=no	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
501 Line printer		--		--		--		
Optical character reader	Mode is: 0=alphanumeric 2=numeric 1=alphabetic 3=mark sense		Mirror is far forward position: 1=yes 0=no	Mirror in far reverse position: 1=yes 0=no	Buffer memory full: 1=full 0=not full	Mirror position error: 1=error 0=no error	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
3691 Paper tape station	--	--	Tape supply 1=low 0=not low	--	End of file mark read: 1=yes 0=no	--		
405 Card reader	--	End of file switch set: (9) 1=yes 0=no	Input hopper empty: 1=yes 0=no	--	End of file card read: (10) 1=yes 0=no	--	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
415 Card punch	--	--	--	--	--	--		
3293 Plotter	--	--	Manual stop switch pressed: 1=yes 0=no	--	--	--	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
Console typewriter	--	--	--	--	--	--		
813/853/854/841 Mass storage (disk)	Write lockout, defective track or address error 0=no 1=yes	Write check no compare error 0=no error 1=compare error	--	--	--	--	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready
863 Mass storage (drum)	End of device 0=no 1=yes		--	--	--	--		
211 Display/ entry station	Ready and not busy interrupt: 0=not active 1=active	Station interrupt: 0=not active 1=active	Alert request: 0=no 1=yes	Poll message error: 0=no 1=yes	Print request: 0=no 1=yes	Send request: 0=no 1=yes	Unit busy: 1=busy 0=not busy	Unit ready: 1=ready 0=not ready

Special Notes

8. Cleared if write ring is not mounted or tape is unloaded.
9. Input hopper empty and END OF FILE switch depressed.
10. 7,8 punch in column one.

General Note

-- Indicates unused and undefined.

UNIT STATUS REQUEST

A status request provides a means of checking the current condition of any I/O unit connected to the system. A request for status of a unit is made before, during, or after an I/O or control operation. When the request is made, CIO returns a status code to the program in registers A and Q.

CIO obtains the status from each I/O unit at the completion (normal and abnormal) of each I/O and control function. CIO edits the status and saves the edited status words in the unit's FDT table entry. (Refer to appendix F.) The status words in the FDT table are returned to the program when a static status request is made. When a dynamic status request is made, CIO connects to the unit and updates the status words in the FDT before returning them to the program.

The return jump sequence to request a unit status check is as follows:

```
P      RTJ     CIO†
P+1    13      u,d
P+2    Next instruction in the program.
```

u File number (two decimal digits)

d Type of status requested:

0 Static status requested
1 Dynamic status requested

Type of Status	Unit Busy	I/O or Control Function Completed
Static Status	The status words in the FDT table are returned in registers A and Q.	The status words in the FDT table are returned in registers A and Q.
Dynamic Status †	CIO updates the u parameter in the second status word in the FDT table. The updated status words are returned in registers A and Q.	CIO connects to the unit, obtains new status information, and updates the status words in the FDT. Table 11-3.1 lists the status bits that are updated. CIO returns the updated status words in registers A and Q.

† For paper type, use PTIOV50.
†† Unit busy applies to channel, equipment, or unit for dynamic status.

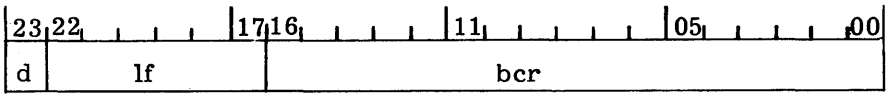
TABLE 11-3.1. BITS UPDATED BY DYNAMIC STATUS CHECKS

I/O Unit	Updated Bits
Tape Drive	U parameter in register Q and status bits 00, 01, 02, 04 06, and 07
Mass Storage Unit	U parameter in register Q and status bits 00 and 01
Line Printer	U parameter in register Q and status bits 00, 01, and 05
Card Reader	U parameter in register Q and status bits 00 and 01
Card Punch	U parameter in register Q and status bits 00, 01, 08, 09, and 10
Paper Tape Reader/Punch	U parameter in register Q and all status bits
211 Display Unit†	U parameter in register Q and all status bits
Console Typewriter	U parameter in register Q and all status bits
Plotter	U parameter in register Q and all status bits
Optual Character Reader	U parameter in register Q and all status bits

†A dynamic status request performed on a 211 display that is not busy may result in the loss of poll message error or poll error status bits in the second status word of the FDT table. Another unit may have connected to the controller or a channel clear may have cleared these bits. The error condition cannot be resensed until it reoccurs.

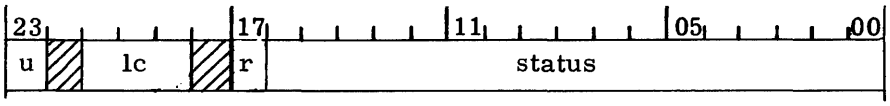
The format of the status words stored in the FDT and returned in registers A and Q is as follows:

Status Word 1 (Register A)



Register A

Status Word 2 (Register Q)



Register Q

d Condition of the unit either dynamic or static.

1 Unit is static. The last I/O or control function has been completed; unit is not busy.

0 The unit is dynamic. An I/O or control function has been initiated and has not been completed; unit is busy.

lf Last function code issued to the unit other than status or format. If lf is zero, no operation has been started at the unit.

bcr — Contents of the block control register for last read or last write function. The buffer control register contains one plus the address of the last character that was referenced by an I/O operation. For reverse reads, bcr is the last character address minus two.

u — Availability of the unit.

- 0 Unit is available and can be connected. This bit is cleared by the end of operation interrupt which is independent of the ready and busy status bits.
- 1 Unit is busy with an I/O or control function and cannot be connected at this time.

lc — Number of the last channel that was connected to unit.

r — Controller reserved flag; applicable only to dual access mass storage controller.

- 1 Controller is reserved by the other channel.
- 0 Controller is not reserved by the other channel.

status — A 16-bit status word that is dependent on the hardware type. (Refer to Table 11-3.)

To simplify checking status in either the interrupt routine or within the program, a routine called SCARV50 has been provided. SCARV50 checks the I/O status and retains control until the I/O completed or an irrecoverable error occurred. † SCARV50 is described in the error recovery paragraph in this section.

I/O REJECT PROCESSING

When a CIO request is rejected, one of the following conditions occurred.

1. Either unit or channel is busy.
2. Unit is not ready.
3. No write ring for write requests on magnetic tape units.
4. Illegal function request.
5. Hardware failure that prevents the unit from connecting or from accepting the function code.
6. File is not defined in the system.
7. Illegal use of system files.

When CIO returns control to the program, register A contains a decimal error code (Table 11-4) and register Q contains the second unit status word from CIO (refer to unit status request).

†SCARV50 uses standard system error recovery methods to recover from all I/O errors and complete the I/O function.

TABLE 11-4. I/O REJECT AND ERROR CODES IN REGISTER A

Error Code Octal	Type of Error
0	A not ready or busy condition exists. Check status word in register Q. On returns from RAARV50 or SCARV50, zero indicates successful recovery.
1	Illegal value in a locate request. The y parameter was not a 0, 1, 2, or 3.
2	Block number selected in a locate request was less than one.
3	File limit was exceeded in a locate request.
4	Illegal x parameter in a locate request. The x parameter can only be 0 or 1.
5	Attempted a read beyond the highest block written.
6	Exceeded mass storage file limits on a read, write, or write-check function.
7	Write request made on a read-only mass storage file.
8	Illegal function code or a request for a zero-length data transfer.
9	Illegal call to system files.
10	Irrecoverable hardware malfunction such as continuous connect reject or select reject. Check status in register Q.
11	Undefined file number used.
12	Irrecoverable read or write error. Standard system error recovery could not recover from the error.
13	Irrecoverable read error on magnetic tape due to request to read request in the wrong mode (that is, binary or BCD).
14	A checksum error occurred while reading a file label in the system mass storage label file.
15	Lost position during write recovery on magnetic tape. A recovery retry will probably cause loss of one or more blocks of data.
17	Illegal I/O request on the library file (LIB).
18	Illegal I/O request on the system punch output unit.
19	Illegal I/O request on the system list output unit.
20	Illegal I/O request on the system input unit.
21	Illegal I/O request on the system console typewriter (CTO) unit.
24	Bad data on the block header (preamble) or record header for a system input file block, when system input unit is a mass storage file.
25	Out of data or out of space, when the system list output, punch output, or input unit is on mass storage.

Rejects with a nonzero value in register A usually indicate a programming error or a hardware error. Mass storage error codes 3, 5, or 6 can provide useful file status information.

If a reject return occurs and register A contains zero, the second status word of the I/O unit in register Q may be tested to determine the cause of the reject. If bit 23 (u parameter) in register Q is zero, the unit and channel are available and the unit can be connected. However, the unit cannot accept control functions that normally indicate the unit is not ready. If bit 00 in register Q is a zero, a message should be sent to the operator requesting that the unit be readied. If bits 00 and 01 in register Q are ones, the unit may still be busy with some previous operations even though it has been released and can be connected. Examples would be postprint page motion or magnetic tape units that are still rewinding to loadpoint.

If a reject occurred and bit 23 in register Q is one, the unit is busy with an I/O function. The busy condition clears on completion of the function. The I/O request should be reissued at a later time. Periodic static status checks may be used to determine when the unit is ready.

If the user does not wish to write his own reject processing routines, he may use RAARV50 or RAAR to recover from rejects. RAARV50 is reentrant; RAAR is not. RAAR can be used only in batch programs.

RAARV50

RAARV50 is an I/O reject processing routine that can be used by both batch and priority programs. If used, RAARV50 must be called from a reject processing routine that contains a return jump (RTJ) to RAARV50. An unconditional jump (UJP) to an open-ended reject routine must be coded in the reject address (P+2) of the CIO call.

With RAARV50, the user can select a return to the program when a not ready condition exists or let RAARV50 retain control until the unit is ready. RAARV50 informs the operator of the not ready condition and requests operator action before returning to the program. For all other conditions, RAARV50 returns with a status code in register A.

With the call to RAARV50, the AQ register must contain the values that were returned from CIO.

The return jump sequence for using RAARV50 is as follows:

```
R       RTJ     RAARV50
R+1     nr      P
R+2     Return from RAARV50
```

nr — No return flag for a not ready or no write ring condition. nr must be 01 or 00.

- 01 — If a not ready or no write ring condition exists, RAARV50 requests the operator to ready the I/O unit and then returns to the program at address R+2. Register A contains a reject status code of -1 upon return.
- 00 — If a not ready or no write ring condition exists, RAARV50 retains control until the condition is cleared or until an irrecoverable condition is detected. Then RAARV50 returns to address R+2 with a reject status code in register A.

P — First word address of the CIO return jump sequence.

All returns from RAARV50 are to address R+2. RAARV50 does not reissue I/O requests or abort the job. This must be done in the program. In all cases, upon return to R+2, register A contains a status code. The code is as follows:

Value in Register A	Significance
Zero	Reject condition was cleared. I/O request should be reissued.
Negative one	RAARV50 sent a ready request to the operator and immediately returned. Not ready or no write ring condition occurred and nr flag was set to 1.
All other values	Reject error codes given in Table 11-4.

The following is an example of RAARV50 processing for a not ready condition.

```
P       RTJ       CIO              CIO call
P+1     01        20               Select read from file 20
P+2     UJP       R                Reject return, return from CIO
P+3     40        Buffer address
P+4     00        512
P+5     Next program instruction   Normal CIO return

R       RTJ       RAARV50          RAARV50 call
R+1     01        P
R+2     AZJ,EQ    P                A=0, reissue I/O request
R+3     AZJ,LT    NR               A=-1, delay reissue of I/O request
R+4     Process reject condition
  .       .
  .       .
  .       .
R+n     UJP       P+5              Return to program

NR      Set I/O repeat flag
NR+1    Select clock interrupt
NR+2    UJP       P+5              Return to program
```

RAAR

RAAR is a reject processing call used in earlier versions of MSOS. The RAAR call is retained so that programs written for earlier versions of MSOS will run on the current version. RAAR processes rejected CIO requests the same as RAARV50, except that no return is made to the program until the busy or not ready condition is cleared.† All other rejects abort the job with an explanatory message on the console typewriter and system output unit.

†Not ready includes no write ring on a magnetic tape drive.

RAAR is used by coding a return jump to RAAR in the reject address (P+2) of a CIO call. RAAR returns to the program at address P. The following is an example of an I/O read request using RAAR to process reject returns.

```
P       RTJ     CIO
P+1     01      20          Read from file 20.
P+2     RTJ     RAAR        Reject address.
P+3     00      BUFFER      Store first word at address BUFFER.
P+4     00      512         Read 512 words.
P+5     Next instruction    Normal return address.
```

I/O ERROR RECOVERY

After CIO initiates an I/O operation, it returns control to the program to allow additional processing while the I/O function is being performed. Therefore, the user must always use an interrupt or check status with a static request to determine if normal I/O completion without error occurred. †

If abnormal termination occurred, the user may attempt error recovery with his own routine or with the standard system error recovery routines. If error recovery is not attempted, or is not successful, the user has the following options.

1. Repeating the I/O on a different unit.
2. Continuing the program without doing the I/O.
3. Aborting the job.
4. Accepting the I/O with error.

If the user does not wish to write his own error recovery routines for unit record devices, he may use one of the standard system recovery routines, SCARV50, SCAR, or read/write with error recovery. These routines process the abnormal I/O termination.

If an abnormal I/O completion occurs, the recovery routine repeats the I/O function several times in accordance with the standard system error recovery procedures (refer to section 13). If the recovery routine is able to successfully complete the I/O function, it returns control to the program the same as for a normal I/O completion. If the recovery routine was unable to do the I/O function, it returns an error code to the program indicating that an irrecoverable I/O error occurred. The user must then decide whether or not to abort the job.

The standard recovery routines provide error recovery for the following devices and functions.

1. All forward and backward reads from mass storage, magnetic tapes, and card readers.
2. All writes on mass storage, magnetic tapes, printers, and card punches.
3. Write a file mark on tape.
4. Backspace a magnetic tape.
5. Check last card in card reader.

---

†An exception is a read or write with error recovery selected.

The recovery routines provide error recovery only for the I/O units and functions listed above. For other I/O units and functions, the recovery routines check for normal/ abnormal completion and return control to the program with error code.

SCARV50

SCARV50 calls the standard system error recovery routines to provide recovery for mass storage and unit record device I/O errors (refer to section 17).

The call to SCARV50 is made at the point of normal return from a CIO call, or in an I/O completion interrupt routine. If the call is made on normal return from the CIO call, SCARV50 will retain control until the I/O is completed. Calling SCARV50 in an interrupt routine allows extra processing time while the I/O is in process. SCARV50 is reentrant and may be used in both batch and priority programs.

Only one call to SCARV50 can be made for each I/O function performed. System error recovery uses and updates the unit status word in the status table. For the second call, the updated status word may not reflect the status of the unit when the error occurred. The I/O function must be repeated before making a second call to SCARV50.

If SCARV50 is used for error recovery on magnetic tape files, it is important that SCARV50 be called after every tape function except status, even if a normal completion occurs. SCARV50 checks each read for a possible noise record (interrecord gap noise read as a data block). In addition, SCARV50 saves a check sum value for the last good block written. The check sum value is used to ensure correct repositioning if an error occurs in the next I/O function. CIO does not perform these functions.

The return jump sequence for calling SCARV50 is as follows:

```
P       RTJ     SCARV50
P+1     nr      cio
P+2     Return to program.
```

nr Card reader compare error flag.

- 0 If a card reader compare error occurred, request operator to reload and reread the card. Repeat procedure until a successful read or an irrecoverable error occurs.
- 1 If a card reader compare error occurred, request operator to reload the card and ready the reader. Then return to the program.

cio First word address of the return jump sequence to CIO.

When SCARV50 returns control to the program, register A contains the following codes.

- 0 Normal I/O completion or successful I/O error recovery. The second CIO status word is in register Q. Continue with the program.
- -1 Card reader compare error occurred. SCARV50 returned control to the program before rereading the card (nr parameter was set to 1). The second CIO status word is in register Q.
- other Error code defined in Table 11-4. Second CIO status word is in register Q.

If a -1 is returned in register A, the user must reissue the I/O request later in the program.

If the CIO return jump sequence was destroyed prior to entering SCARV50, a dummy sequence can be reconstructed at a new address for SCARV50. SCARV50 only needs the parameter values used in the original return jump sequence to CIO.

The following is an example of the use of SCARV50 to check for abnormal I/O termination.

```
P         RTJ      CIO          CIO call.
P+1       01       20,1         Select interrupt on abnormal I/O completion.
P+2       UJP      reject       Reject return from CIO.
P+3       40       buff         Store first word at address buff.
P+4       00       40           Read 40 words.
P+5       00       ERROR        Address of interrupt processing routine for
P+6       Next program instruction  abnormal I/O completion.
  .
  .
  .
ERROR     UJP      **           I/O completion interrupt routine.
ERROR+1   RTJ      SCARV50
ERROR+2   01       P            Select return on card reader compare error;
                                P equals address of CIO call.
ERROR+3   AZJ,LT   RDCOMP       A is negative; compare error.
ERROR+4   AZJ,NE   IOERROR      A is not zero; irrecoverable I/O error.
ERROR+5   UJP      ERROR        Normal return.
```

SCAR

SCAR is the error recovery call used in earlier versions of MSOS. SCAR is retained so that programs written for earlier versions of MSOS will run on the current version. SCAR is not reentrant. However, SCAR may be used in both batch and priority programs. Each program level receives its own copy of SCAR.

SCAR works the same as SCARV50 except that SCAR does not return control to the program during card reader not ready conditions and SCAR has two return addresses, normal return and error return.

To use SCAR, enter the first word address of the CIO calling sequence (P) in register A and then use the following return jump sequence to enter SCAR.

```
P         UJP      77777
P+1       ENA      addr        Enter address of CIO call in A.
P+2       RTJ      SCAR        Call to SCAR.
P+3       00       00  }
P+4       00       00  }       Addresses P+3, P+4, and P+5 are not checked
P+5       00       00  }       by MSOS.
P+6       jump     addr        Irrecoverable error return. Registers A
                               and Q contains error codes.
P+7       UJP      P           Normal I/O completion or successful error
                               recovery return.
```

jump UJP or RTJ instruction.

addr Address of irrecoverable error processing routine.

When SCAR returns control to the program at address P+6, error codes are returned in registers A and Q. The error code in register A is defined in Table 11-4. The error code in register Q is the CIO status word (refer to Table 11-3).

If the CIO return jump sequence was destroyed prior to entering SCAR, a dummy sequence can be reconstructed at a new address for SCAR. SCAR only needs the parameter values used in the original return jump sequence to CIO.

READ/WRITE WITH ERROR RECOVERY

The read or write with error recovery function (function codes 41 and 42) can be used in two ways.

1. Always used for reads and writes.
2. Used after an abnormal I/O termination on mass storage devices.

The simplest method for using standard system error recovery is to always use function code 41 or 42 when reading or writing. However, this method is wasteful of processing time because there is no return to the program from CIO until the I/O is completed.

If function code 01 or 02 is used, CIO returns control to the program and additional processing time is available while the read or write is in process. However, the user must use an interrupt or check the unit status later in the program to ensure that normal I/O completion occurred.

Mass storage I/O may be repeated using read or write with error recovery if an interrupt or status check indicates an abnormal I/O termination. The read or write with error recovery can be initiated within the users program or within an interrupt routine. A normal return from the read or write with error recovery can be followed with the next I/O request. If a reject return occurs, the user must decide whether or not to abort the job. Registers A and Q will contain an error code (refer to Table 11-4).

Use of a read or write with error recovery after an error on I/O units other than mass storage should be avoided. All other units must be repositioned before the I/O can be repeated. Correct repositioning cannot be ensured unless SCARV50 is used for error recovery.

ASCERTAINING EQUIPMENT TYPE

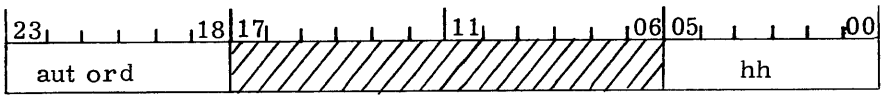
The user may determine the hardware type being used for any file number by using the WHATKIND routine. WHATKIND may be called from a batch or priority program. The procedure is as follows:

1. Enter the file or logical unit number right-justified in register A.
2. Do a return jump to WHATKIND.

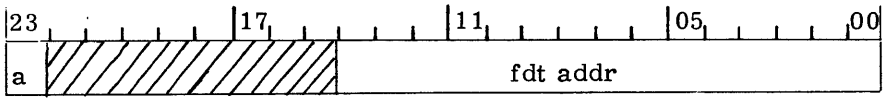
```
P       ENA   lun              Lun - file number.
P+1     RTJ   WHATKIND
P+2     Next instruction.      Return from WHATKIND.
```

WHATKIND passes the following information to the program in registers A and Q.

Register A



Register Q



aut ord	Ordinal of the entry for this unit in the AUT table; zero for spooling and system input files on mass storage.
hh	Hardware type code for unit record devices or the device type code for a mass storage file. For system I/O spooling and system input files on mass storage, the associated low speed hardware type is given.
a	1 if the unit is ASCII. 0 if the unit is not ASCII.
fdt addr	Address of the entry for this unit in the FDT table.

If register A contains all zeros, the file number is not assigned.

CIO MACRO CALLS †

All CIO functions except reject and I/O error recovery can be called via COMPASS macros instead of the return jump sequence to CIO. Normal return is to the instruction immediately following the macro. In addition to the following macros, the CLOSE macro (section 13) may be used to release unit record devices so they can be EQUIPED and used by another program.

Read

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD                COMMENTS
1       8   10                    20                           41
            READS                 (l,r,fwa,n,i,ia,m,c,ch)
```

Write

```
LOCATION    OPERATION, MODIFIERS  ADDRESS FIELD                COMMENTS
1       8   10                    20                           41
            WRITES                (l,r,fwa,n,i,ia,m,c,ch)
```

† CIO macros cannot be used for paper tape units.

Read Backward

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	READB	(l, r, fwa, n, i, ia, m, c, ch)	

Rewind to Load Point

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	REWIND	(l, r, i, ia)	

Rewind and Unload

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	UNLOAD	(l, r, i, ia)	

Skip File Mark Forward

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	SEFF	(l, r, i, ia)	

Skip File Mark Backward

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	SEFB	(l, r, i, ia)	

Write End-of-File Mark

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	WEOF	(l, r, i, ia)	

ERASE

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	ERASE (l, r, i, ia)	

STATUS

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	STATUS (l, d)	

FORMAT

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	FORMAT (l, r, c, dt)	

BACKSPACE

LOCATION	OPERATION, MODIFIERS ADDRESS FIELD	COMMENTS
1 8	10 20	41
	BKSP (l, r, i, ia)	

LOCATE

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	LOCATES (l, r, x, y, n, i, ia)	

WHATKIND

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	WHATKIND (fo)	

Parameter	Description
c	For read and write functions. C — Character I/O; illegal for mass storage, card reader, card punch, and reverse magnetic tape reads. omitted — Word I/O. For format functions, c is the format code (refer to Table 11-2).
ch	Specifies which memory bank an I/O buffer is in; used only for batch programs in an extended core system. 0, 1 — Bank 1 2 — Bank 0 Omitted for priority programs and standard 16/32K systems.
d	Specifies dynamic or static status. 0 — Static status 1 — Dynamic status
dt	Indicates whether or not the device is a 512 line printer. 512 — 512 line printer zero or omitted — Not a 512 line printer
fo	File number.
fwa	Address of first word or character in a buffer for reads or writes; may be a character address.
i	Interrupt request. 0 — No interrupt on completion. 1 — Interrupt on abnormal termination. 2 — Interrupt on normal completion or abnormal termination.
ia	Interrupt address; address of interrupt processing routine for the interrupt selected by the i parameter. This may be omitted if i equals 0.
l	File number.
m	Six-bit mode designator which specifies BCD or binary I/O; also indicates density for magnetic tapes. For all I/O except the incremental plotter and magnetic tape unit, the following values should be used. 40 — BCD mode 41 — Binary mode

For magnetic tape units, one of the following codes should be used.

Code	7-Track Mode	9-Track Mode	Density 7-Track Drive	Density 9-Track Drive
40	BCD	Convert	No change	No change
41	Binary	Pack	No change	No change
50	BCD	Convert	200 bpi	800 cpi
51	Binary	Pack	200 bpi	800 cpi
60	BCD	Convert	556 bpi	800 cpi
61	Binary	Pack	556 bpi	800 cpi
70	BCD	Convert	800 bpi	1600 cpi
71	Binary	Pack	800 bpi	1600 cpi

For incremental plotter, 40 specifies character mode and 41 specifies word (disassembly) mode.

Mode	With Character I/O	With Word I/O
40	All characters plotted.	Only bits 00 through 5 (lower character) of each word is plotted.
41	All characters plotted.	All characters in each word are plotted.

n For reads and writes, n is the number of words or characters read or written.

For mass storage locate functions, n is an eight-digit octal number used in conjunction with the y parameter to select the next block number.

r Reject address. If * or RAAR is used for reject address, rejects are automatically processed by RAAR. An RTJ to RAAR is assembled in the macro.

If RAARV50 or a user's reject routine is to be used, a reject address must be supplied, and a UJP to the reject address is assembled in the macro.

x One octal digit which specifies read/write head movement.

- 0 Position the read/write heads.
- 1 Do not position the read/write heads, and negate any interrupt selected with the i parameter.
- other Illegal

y One octal digit which selects the next block number.

- 0 Replace the current block number with the n parameter.
- 1 Replace the current block number with the current block number plus one.
- 2 Replace the current block number with the current block number plus the n parameter.
- 3 Replace the next available block number with the n parameter. No change is made to the current block number.
- other Illegal

CIO MACRO EXPANSIONS

The following job can be run to obtain a copy of the expansions for any of the preceding macros.

```
$JOB,...
$COMPASS,L
    IDENT       EXPAND
    LIBM        READS,WRITES,READB,REWIND,UNLOAD
    LIBM        BKSP,SEFF,SEFB,WEOF,ERASE,STATUS
    LIBM        FORMAT,LOCATES,WHATKIND
    END
    FINIS
77
88
$EOJ
```

SPECIAL FORMS CONTROL 12

DESCRIPTION

In batch programs, mounting and aligning special forms can be controlled by calling the special forms control routine, FORMSV50. The subroutine requests the operator to mount a specific form on a printer, or to load special cards in the card punch. Forms control can be used with APC but cannot be used in priority programs.

If the request was to mount a form in the printer, control is returned to the program to print an alignment pattern. After each alignment pattern is printed, the operator has the option of requesting an additional alignment pattern until the form is correctly aligned.

Calls to the forms control routine can be made in COMPASS, COBOL,† and FORTRAN† programs. The COBOL and FORTRAN calls are described in the COBOL and FORTRAN reference manuals.

SPECIAL CARD FORMS

The COMPASS calling sequence for the forms control routine to load special card forms in the card punch is as follows:

```
P       RTJ     FORMSV50
P+1     u       loc1
P+2             loc2
P+3     next program instruction
  .
  .
  .
loc1    0
loc2    name
```

loc1,2 Any symbolic word address.

u Logical unit number of a card punch. u must be 62 or be a number that was previously equated to 62 with an EQUIP card.

name Four-character BCD name of the card forms to be loaded in the punch. The name cannot be four blanks.

The contents of loc1 is zero in the call to FORMSV50 for card forms. On return to address P+3, loc1 contains one of the reply codes listed in Table 12-1. The user should add a routine to test and process the reply codes.

† Both ANSI and MS.

TABLE 12-1. FORMS CONTROL REPLY CODES†

Code	Description
0	Operator response was negative. a. For initial forms mounting call, 0 indicates operator cannot or did not mount the form. A possible wrong form name was used in the call. b. For printer alignment calls, 0 indicates the form is not aligned. The operator is requesting that another alignment pattern be printed.
1	Operator response was positive. a. For initial forms mounting call, 1 indicates the form has been mounted or the cards loaded, and the operator is waiting for the test pattern to be printed. b. For printer alignment calls, 1 indicates the form is aligned. No more alignment patterns need to be printed.
2	Wrong value used in location loc1. The value must be either 0 to indicate an initial call, or 1 to indicate an alignment call.
3	The u parameter is not defined in the system.
4	The u parameter is not a printer or a card punch.
5	The u parameter is not 61 or 62, or a unit number that has been equated to 61 or 62.
6	a. An alignment call to forms control was not preceded by the initial call to mount the forms. b. The preceding initial call to forms control returned an error and the call was not repeated correctly before making the alignment call.
7	a. The u parameter in the initial call was not the same as in this alignment call. b. A second initial call occurred between the first initial call and its alignment calls.

† The forms control reply codes are returned as an octal number in bits 02 through 00 in location loc1.

SPECIAL PRINTER FORMS

Two or more calls to FORMSV50 are required to mount and align special forms on the printer. The first (initial) call is to request the operator to mount the form, and the second and all subsequent calls are to supply alignment test patterns.

If a positive reply is returned after the initial call, a test pattern should be written and then a second FORMSV50 call should be made. The second and all subsequent calls request the operator to indicate if the form is or is not correctly aligned. If the form is not correctly aligned, another test pattern is printed and another call to FORMSV50 made. The calling sequence to FORMSV50 is as follows:

Location	Instruction	
P	RTJ FORMSV50	Initial call to FORMSV50.
P+1	u loc1	
P+2	loc2	
P+3	Return from initial call to FORMSV50.	Refer to Table 12-1 for reply codes.
TEST ⋮ TEST+n	Call CIO to write a test pattern on unit u.	
ALIGN	RTJ FORMSV50	Alignment call to FORMSV50.
ALIGN+1	u loc1	
ALIGN+2	NOP	
ALIGN+3 ⋮	Return from alignment call to FORMSV50.	Refer to Table 12-1 for reply codes.
loc1	0 or 1	loc1 must be zero for initial calls and one for alignment calls.
loc2	name	

loc1, 2	Any symbolic word address.
u	Logical unit number of the printer. u must be 61 or be a number that was previously equated to 61 with an EQUIP card.
name	Four-character BCD name of the form to be mounted on the printer.

On return from the call to FORMSV50, loc1 contains one of the codes in Table 12-1. The contents of loc1 must be checked and reset to 1 before each alignment call.

NOTE

The operator uses the manual interrupt to reply to each request from FORMSV50. If a manual interrupt was selected prior to the FORMSV50 call, the interrupt must be reselected. FORMSV50 does not restore a manual interrupt which was previously set.

MASS STORAGE FILE CONTROL MACROS (OCAREM) 13

FILE CONTROL MACROS

Each mass storage file control function described in section 3 has a corresponding COMPASS macro. These macros may be used within a COMPASS program instead of the corresponding MSOS control statement which must either precede or follow a user program.

FILEID MACRO

The FILEID macro performs the same function as the FET control statement. It provides file identification for a user file by establishing a file environment table (FET) in core.

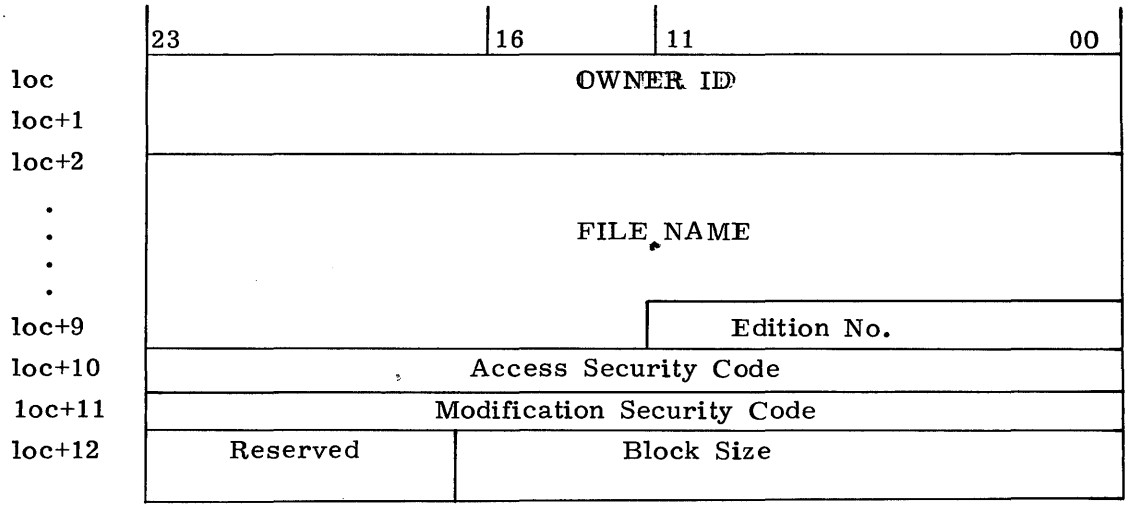


Figure 13-1. FET Table

The FET may be referenced in a program by using the loc address from the FILEID macro and LDA instructions.

LOCATION		OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1	8	10	20	41
loc		FILEID	(owner, name, blksize,	edition, accsec, modsec)

loc — Symbolic location of the first word of the FET table. This location is specified in the calling sequence for all pre-I/O/post-I/O macro functions except CLOSE.

owner — Eight-character identification; if less than eight characters, the field is blank-filled on the right.

name — 30-character mass storage file name; if less than 30 characters, the field is blank-filled on the right. Imbedded blanks in the name are not permitted.

blksize — Number of characters in each data block, not exceeding 131071_{10}.

edition — Edition number, two alphanumeric characters or one decimal digit; set to zero if omitted.

accsec — Four-character access security code; if omitted, the field is blank-filled. If less than four characters, the field is blank-filled on the right.

modsec — Four-character modification security code; blank-filled if omitted. If less than four characters, the field is blank-filled on the right.

The macro fields are separated by commas and must not contain imbedded blanks. If a field is omitted, its trailing comma must be supplied unless all fields to the right are also omitted. The owner, name, and blksize parameters are mandatory; edition number and security codes are optional.

ALLOCATE MACRO

The ALLOCATE macro reserves space for the file. It performs the same function as the ALLOCATE control statement. The ALLOCATE macro adds new entries in the LABELFILE and IDFILE, and updates the MSD file.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD	COMMENTS
1 8	10 20	41
	ALLOCATE (loc, n, exp, , seg, dt, b)	

loc — Core address for a 13-word FET that was created by the FILEID macro. It contains the file identifiers, security codes, and number of characters in each data block.

n — Number of mass storage tracks or blocks to allocate for the file; it is a positive integer not exceeding 262143.

exp — Expiration date to be inserted in the file label. It is a six-digit decimal integer in the form yymmdd. If omitted, the current date is used.

seg — NOSEG specifies contiguous allocation. Omitted or any other value indicates the file may be segmented.

dt — Specifies the type of mass storage device. Default is library device type. Values may be 853, 854, 813, 841, or 863. An 814 is defined as two 813s.

b — Specifies block or track allocation.

1	Allocate n blocks
omitted or 0	Allocate n tracks

If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 00 through 11. Address m+1 should contain a jump instruction to an error processing routine.

m	MACRO
m+1	Reject return
m+2	Normal return (next instruction)

Calling sequence to ALLOCATE:

P	RTJ	ALLOCATE
P+1	VFD	O6/dt, A2/0, A1/seg, A15/loc
P+2	VFD	A1/b, A5/1, A18/n
P+3	DEC	exp
P+4	Reject return	
P+5	Normal return	

b, loc, n and exp — Same as in the macro description.

seg — Segment flag

0 File may be segmented
1 File must be contiguous

dt — 6-bit code for device type

Error Codes

31	RFLD full (class-R files only)
32	RFLD misalignment (class-R files only)
51	OCAREM is busy; previous function has not completed.
52	Calling sequence contains an illegal device type and/or recording mode.
54	File size exceeds the maximum permitted by the installation.
55	File identifier, words 1 through 10 in the FET, already exists.
56	File label directory is full.
57	An illegal device number was used on the previous RAT card (that is, device number not entered in system).
58	Not enough tracks are available.
59	Segment count exceeds the installation maximum.
60	Contiguous space requested but no contiguous block of n tracks is available.
79	Illegal block size.
81	Illegal value for n parameter.

OPEN MACRO

The OPEN macro performs the same functions as the OPEN control statement. It uses the FET to locate the file label in LABELFILE, checks to ensure the required mass storage devices are on-line, makes a file entry in the file ordinal table to assign the file number, and generates a file description table (FDT).

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	OPEN	(loc,fo,use,alpha)	

loc — Core address of a 13-word FET containing file identifiers and access security code. (Refer to FILEID macro.)†

fo — Unique file number (0 through 62) which is used for subsequent I/O requests to this file.

use — I specifies a read only file while it is open. Any other symbol indicates file is available for reading and writing. If I value is used, a file can be opened concurrently by other programs and shared for input. If I is not used, the file cannot be concurrently opened and shared by other programs.

†Only the first 11 words of the FET are referenced when opening a file.

alpha Core address of a three-word parameter table (optional).

alpha	L
alpha+1	P
alpha+2	N

L First character position in label ($92 \leq L \leq 195$) or ($92 \leq L \leq 115$ for class R) to be read.
P Starting character address in core.
N Number of characters to move ($1 \leq N \leq 104$) or ($1 \leq N \leq 24$ for class R).

If alpha is included, the file label is read. N characters are moved from the file label, starting with character L, to core starting at character address P. If alpha is omitted, the file label is not read.

The alpha parameter is optional and may be omitted. If used, it specifies the location of a three-word table that defines a record definition area in core. The characters specified in the three-word table will be read from the file label (refer to appendix G) and written in the record definition area for inspection and use within the program. The data in the three-word table must be set before using the macro.

On a normal return to m+2, register A contains the octal number of tracks in the file. If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 11 through 00. Address m+1 should contain a jump instruction to an error processing routine.

m Macro
m+1 Reject return
m+2 Normal return (next instruction)

Calling sequence to OPEN:

```
        ENA     alpha
P       RTJ     OPENB
P+1     VFD     A6/fo,A1/0,A1/m,A1/use,A15/loc
P+2     Reject return
P+3     Normal return
```

alpha, fo, loc: Same as in macro description.

use Usage indicator
- 0 Allows output to the file while it is open.
- 1 Does not allow output to the file while it is open.

m Store indicator
- 1 A register contains alpha.
- 0 Alpha not present; no characters to move.

Normal return is to address P+3 with register A containing the octal number of tracks in the file. Control is returned at P+2 with an error code in the A register if an error occurred.

Error Codes

30	File inoperative (class-R files only)
51	OCAREM is busy; previous function has not completed.
53	L or N, illegal value. $92 \leq L \leq 195$, or $1 \leq N \leq 104$ is legal.
61	File identifier specified in the FET could not be located in the label directory.
63	Access security code in the FET does not match the access security code in the file label.
64	Specified file ordinal has already been assigned, or is an illegal value.
65	Resident tables are full. Too many files are open.
66	The file is already open for output.
67	Use indicates output but the read only flag is set in the file label.
68	Use indicates output but the file is already open.
69	The file cannot be placed on-line. Not enough drives are available.

CLOSE MACRO

The CLOSE macro performs the same function as the CLOSE control statement. For mass storage files, it releases file entries in the FDT and file ordinal tables, and updates the file label in the LABELFILE. The file cannot be used until it is reopened with an OPEN macro or statement.

For unit record devices, the CLOSE statement releases a file entry from the FDT and file ordinal tables. The unit must be reequipped with an EQUIP statement before it can be used again. The unit will be available for use by another program. The CLOSE macro does not update or change tape labels. The alpha and t parameters are ignored.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	CLOSE	(fo, alpha, t)	

fo	File number	
t	Label update flag	
	1	Update the label but leave the file open for continued use in the job.
	omitted or 0	Update the label and close the file.

alpha Core address of a three-word parameter table (optional).

alpha	L
alpha+1	P
alpha+2	N

L First character position in the file label. L may range from 92 to 195 or 92 to 115 for class R.

P Starting character address in core.

N Number of characters to move. N may range from 1 to 104 or 1 to 24 for class R.

If the alpha parameter is included, N characters are read from core, starting at character address P, and written on the file label, starting at character position L. The system always updates the last access date, number of blocks written, etc., when a mass storage file is closed.

If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 11 through 00. Address m+1 should contain a jump instruction to an error processing routine.

m	Macro
m+1	Reject return
m+2	Normal return (next instruction)

Calling sequence to CLOSE:

	ENA	alpha
P	RTJ	CLOSEB
P+1	VFD	A6/t,A1/0,A1/m,A10/0,A6/fo
P+2	Reject return	
P+3	Normal return	

Normal return is to P+3 with the last block number in octal written in register A and the octal number of unused tracks in Q.

alpha,fo,t Same as in macro description.

m Store indicator

1 A register contains alpha.
0 Alpha is not present; no characters to move.

Control is returned at P+2 with an error code in the A register if an error occurs.

Error Codes

51 OCAREM busy; previous function has not completed.

53 Illegal value for L or N.

71 The specified file number has not been assigned or an illegal value was used.

RELEASE MACRO

The RELEASE macro performs the same function as the RELEASE control statement. It releases all or part of the space assigned to mass storage files with the ALLOCATE statement. A mass storage file must be closed before it can be released.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	RELEASE	(loc, n, b)	

loc — Core address of a 13-word FET containing file identifiers and security codes (refer to FILEID macro). †

n — Decimal integer indicating the number of tracks or blocks of mass storage to release. Values may be 1 to 262143, UNUSED or ALL.

b — Specifies block or track release. Optional.

1	Blocks
omitted or 0	Tracks

If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 11 through 00. Address m+1 should contain a jump instruction to an error processing routine.

m	MACRO
m+1	Reject return
m+2	Normal return (next instruction)

Calling sequence to RELEASE:

P	RTJ	RELEASE
P+1	VFD	A9/0,A15/loc
P+2	VFD	A1/b,A5/c,A18/n
P+3	Reject return	
P+4	Normal return	

b and loc — Same as in the macro description.

c — Control value

0	Release n tracks or blocks.
1	Release the entire file. Ignore values for n and b.
2	Release the unused space. Ignore values for n and b.

n — Number of tracks or blocks to release.

Control is returned at P+3 with an error code in the A register if an error occurs.

†Only the first 12 words of the FET are referenced to release a file.

Error Codes

30	File inoperative (class-R files only)
33	All segments not on-line (class-R files only)
51	OCAREM busy; previous function has not completed.
61	File identifier specified in the FET cannot be located in the label directory.
62	The file is currently open.
63	One or both security codes in the FET do not match security code in the file label.
70	Calling sequence contains an illegal control value.
81	Illegal value for the n parameter.

EXPAND MACRO

The EXPAND macro performs the same function as the EXPAND control statement. It increases the number of tracks reserved for a file.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	EXPAND	(loc, n, seg, b)	

loc	Core address of a 13-word FET containing file identifiers and security codes (refer to FILEID macro). †
n	Number of new tracks or blocks to add to the existing file.
seg	NOSEG specifies only one new segment, contiguous with the current file when possible. Omitted or any other term indicates the new area may consist of more than one segment.
b	Specifies blocks or tracks to be added. 1 Expand by n blocks. omitted or 0 Expand by n tracks.

If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 11 through 00. Address m+1 should contain a jump instruction to an error processing routine.

m	MACRO
m+1	Reject return
m+2	Normal return (next instruction)

†Only the first 12 words of the FET are referenced when expanding a file.

Calling sequence to EXPAND:

```
P       RTJ       EXPAND
P+1     VFD       A9/seg,A15/loc
P+2     VFD       A1/b,A5/0,A18/n
P+3     Reject return
P+4     Normal return
```

loc, b, and n — Same as in the macro description.

seg — Segment flag

- 0 — Expansion may be segmented.
- 1 — Expansion must be contiguous.

Control is returned at P+3 with an error code in the A register if an error occurred. File cannot be expanded across different device classes (that is, R and nonclass R).

Error Codes

- 30 — File inoperative (class-R files only)
- 34 — Class of file does not match class in RAT (class-R files only)
- 51 — OCAREM busy; previous function has not completed.
- 54 — File size (current size + n) would exceed the maximum permitted by the installation.
- 57 — An illegal device number was used on a previous RAT card (that is, device not entered in the system).
- 58 — n additional tracks are not available.
- 59 — Segment count exceeds installation maximum.
- 60 — Contiguous space requested but no contiguous block of n tracks is available.
- 61 — File identifier specified in the FET could not be located in the label directory.
- 62 — The file is currently open.
- 63 — One or both security codes in FET do not match security code in file label.
- 81 — Illegal value for n.

MODIFY MACRO

The MODIFY macro performs the same function as the MODIFY control statement. It allows the user to change the file label.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	MODIFY	(loc, prot, exp, newfet)	

loc — Core address of a 13-word FET containing file identifiers and security codes (refer to FILEID macro). †

prot — Inserts a file protection indicator in a file label.

- I — Restrict file usage to input (read only) usage.
- O — Allows both input and output (read or write) usage.
- Omitted — No change in file's protection.

†Only the first 12 words of the FET are referenced when expanding a file.

exp — Inserts new expiration date in file label. Exp is a six-digit decimal integer in the form yymmdd. If omitted or zero, the expiration date in the label is not changed.

newfet — Core address of a new FET (refer to FILEID macro). If zero or omitted, newfet is ignored. Otherwise, contents of the new FET are inserted in the file label.

If an error occurs, the return from the macro is to address m+1 and register A contains an error code in bits 11 through 00. Address m+1 should contain a jump instruction to an error processing routine.

m	MACRO
m+1	Reject return
m+2	Normal return (next instruction)

Calling sequence to MODIFY:

P	RTJ	MODIFY
P+1	VFD	A9/0,A15/loc
P+2	VFD	O9/prot,A15/newfet
P+3	DEC	exp
P+4	Reject return	
P+5	Normal return	

prot — New protection flags for file label.

001	File may be read or written.
011	File may not be written.
000	Does not change protection.

Control is returned at P+4 with an error code in the A register if an error occurs.

Error Codes

30	File inoperative (class-R files only)
51	Another nonresident MSIO function has not yet run to completion.
55	File identifier specified by the new FET (newfet) already exists in the label directory.
61	File identifier specified in the FET cannot be located in the label directory.
62	File specified in the FET is currently open for processing.
63	Wrong access and/or modification code in the FET.
79	Block size specified in new FET is illegal.

MACRO EXPANSIONS

The following job may be run to obtain a list of OCAREM macro expansions.

```
$JOB...
$COMPASS,L
    IDENT       EXPAND
    LIBM        FILEID,ALLOCATE,OPEN,CLOSE
    LIBM        RELEASE,EXPAND,MODIFY
    END
    FINIS
77
88
$EOJ
```

The macro expansion is the code that replaces the macro call statement when the program is assembled.

LOGICAL MSIO 14

DESCRIPTION

Logical MSIO (L-MSIO) is a set of I/O routines for reading and writing logical records. L-MSIO provides the COMPASS user with a set of routines that perform the following functions.

1. Automatic blocking and deblocking of logical records
2. Automatic file label processing
3. Automatic buffering
4. Random file access
5. Restart function
6. Multireel file and multifile reel handling functions

The L-MSIO file and record formats are compatible among COMPASS, ANSI COBOL, MS COBOL, and LISA. The L-MSIO routines can be used with tape, mass storage, and punched card files, and they can drive any device controllable by MSOS.

FILE REQUIREMENTS AND INITIALIZATION

All mass storage files to be referenced by L-MSIO must be initially allocated with the ALLOCATE card or the ALLOCATE macro to build a file label. Similarly, all unit record devices must be equipped by assigning a logical unit number, with an EQUIP statement, before they can be referenced by L-MSIO.

Before a file can be referenced by L-MSIO, the file and record formats must be defined with L-MSIO file description routines. These routines build a special 32-word file environment table (FET) for the file (refer to appendix G for FET format). The L-MSIO FET is different from the OCAREM FET built by the FET control statement or the FILEID macro.

The ANSI COBOL compiler generates and uses a modified version of the L-MSIO FET which cannot be used by COMPASS, LISA, or MS COBOL subprograms or routines. For example, if a COMPASS subprogram were entered from an ANSI COBOL program, the COMPASS subprogram would have to build a new L-MSIO FET to use an ANSI COBOL file. The FETs generated by COMPASS, LISA, and MS COBOL are compatible.

After a file's FET has been constructed, the file must be opened with the OPENF routine. Then logical records can be read from and written on the file with GET and PUT macros (refer to section 15).

File record sizes may vary from 1 to 4095 characters for files used by COMPASS, LISA, and MS COBOL. Record sizes may vary from 1 to 32767 characters for files used with ANSI COBOL. COMPASS, LISA, and MS COBOL cannot process files generated by ANSI COBOL if the record size exceeds 4095 characters.

L-MSIO uses an overlay file which was allocated at system installation time. L-MSIO opens this file as file 53. Therefore, 53 cannot be assigned as a file number in any user program.

LOGICAL RECORDS

Logical records may be fixed or variable in length according to the five formats listed below. All records within a file must have the same format. The format of these records is described in detail in appendix F.

Fixed length	All records are the same length. A parameter in the file description macro specifies the length.
Variable length	A key field, set in BCD format by the user, indicates the record length in characters. If the key field is within the logical record, it must occupy the same position and be the same size within each record.
Variable length with trailer	Logical records consist of a fixed length base and a variable number of fixed length trailer items. A key field in the fixed length base specifies the number of trailer items in the record. The user sets the key field in BCD format.
Record delimiter	Records may be fixed or variable in length. A special delimiting character terminates each record. The user may specify the character in the file environment table (FET). If none is specified and this format is selected, the system assigns the special character 72_8.
Universal	A record blocking routine (PUT) prefixes each record with a 24-bit binary control field. This field contains the mode of the record as defined by the file description macro and the number of 6-bit bytes in the record. The record deblocking routine (GET) removes the control field before delivering the record to the calling program. In an address specified by the VARIABLE macro, the system stores in binary format the length of a record and its mode. † The record length must be set in the specified address prior to a PUT.

RECORD BLOCKS

Logical records are grouped together into blocks in order to provide more efficient I/O. A whole block is written or read at once when L-MSIO does input or output functions. All records on tape and mass storage are stored in blocks.

MASS STORAGE BLOCKING

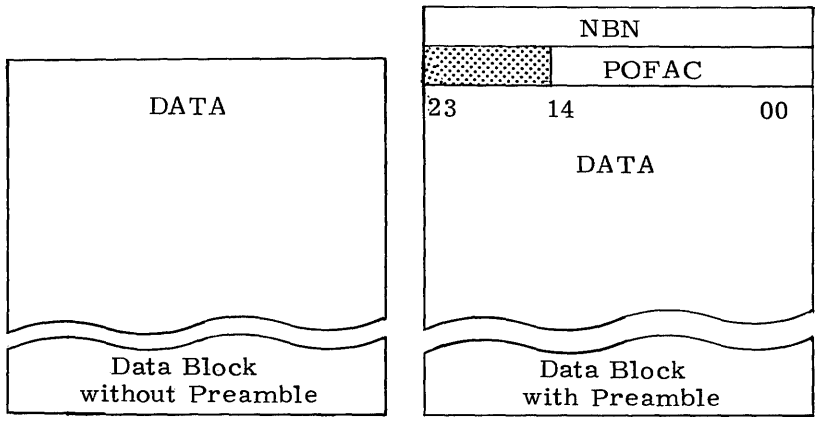
Record blocks on mass storage must be fixed length. The user determines the block size and specifies the size when allocating and establishing a file.

† Refer to the MODEBIT option in the MSOS Installation Handbook.

The user may specify through the block format parameter of the FILEDESC macro that each block contains a two-word preamble. If present, the preamble contains the following:

Block number of the next block to be read in logical sequence. This permits creating and reading files that are not in sequential order physically.

Position of the first available data character within the block relative to the block origin. This allows a variable number of records per block.



NBN — Next block number to be read when file is accessed sequentially

POFAC — Position of first available data character

When no preamble has been specified, L-MSIO assumes a single record per block.

NONMASS STORAGE BLOCKING

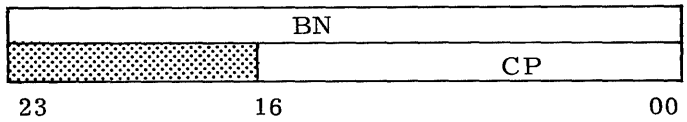
The block size for line printers is 136 characters. Each write prints one line of 135 characters plus one control character. Any remaining characters in a larger output buffer are truncated.

The block size for card punches is 20 words for BCD mode and 40 words for binary mode. Each write punches only one card. Any remaining characters in a larger output buffer are truncated.

For all other units, the block size may range from five to 4090 words, depending upon the application. † The block size is selected in the FILEDESC macro.

FILE ACCESS

L-MSIO provides random access to file records under control of a seek address key (SAK) in the file's FET. The SAK consists of a relative block address. When there is more than one record per block, SAK has a pointer to the relative position of the first data character of a record within the block.



BN — Relative block number within file.

CP — Relative character position within block BN. For the first record in block BN, CP equals 0.

†Maximum of 32,767 characters for tape files in the modified version of L-MSIO generated by the ANSI COBOL compiler.

SEQUENTIAL ACCESS

Records within a file are read or written sequentially under control of the SAK. For input or output files, L-MSIO automatically increments the SAK to the next record for each GET or PUT request. For mass storage files opened with OPENF as I/O, L-MSIO advances the SAK only for GET. When a PUT follows a GET, both reference the same record. Since the I/O option implies the existence of data in a file, this option cannot be used when a mass storage file is initially created or expanded.

For sequential output files, new data is written starting only at the highest block written plus one. Thus, before rewriting over existing data, the file is released and reallocated. The ANSI variant always begins writing at block one. Therefore, the whole file is rewritten in order to rewrite the last block.

RANDOM ACCESS

To access a file randomly, the user declares random access mode in the FILEDESC macro. L-MSIO does not modify the SAK when GET or PUT requests are executed. Prior to each GET or PUT, the user calls LOCATE to set the SAK for the record to be referenced. Random access is allowed on mass storage only.

FILE SECURITY

Each mass storage file label has a provision for privacy codes. The user supplies both an access privacy code and a modification privacy code when he allocates the file. The access privacy code protects a file from unauthorized use. If the access privacy code in the FET specified for an OPENF call does not match one in a file label, L-MSIO rejects the call and terminates the job. RELEASE, EXPAND, and MODIFY require a modification privacy code. If the codes are mismatched, the system rejects the request and terminates the job.

The user may specify mass storage files as read only through a call to MODIFY. L-MSIO rejects requests to open files for output which have been defined as read only.

BUFFERING

Buffering permits CPU processing to occur concurrently with I/O data transmissions. The extent of the buffering depends upon the size of the record and buffer areas. A record area is a memory block assigned for a logical record to reside in. A buffer area is a memory area assigned to hold a block of records, including the two-word preamble if specified for mass storage files. The user may specify one to five configurations of record and buffer areas as follows:

1. Record area only

   No buffering or blocking/deblocking occurs. Each input or output request results in transmission of one record to or from an I/O unit.

2. Buffer area only

   With a buffer area, L-MSIO performs automatic I/O of record blocks. The user may use the GET routine to obtain the first character address of the next record in the input buffer and the record length in characters. L-MSIO returns the first character address in register A and the record length in register Q. The user may also use the PUT request to get the first character address of the next record slot in an output buffer. However, the reads and writes used to block or deblock a record, that is, write or read a new record in the buffer area, must be done in the program. When an output buffer is full, or an input buffer is empty, L-MSIO automatically initiates I/O to write the block on an output unit or read a new block from an input unit.

3. Buffer area and a record area

   With both a buffer and a record area, L-MSIO provides automatic I/O and blocking/deblocking. The user can use the PUT request to read a record from the record area and block (write) the record in the output buffer. The user can also use the GET request to deblock (read) a record from the input buffer and write the record in the record area for inspection or processing. When an input buffer is empty or an output buffer is full, L-MSIO automatically initiates I/O to empty or refill the buffer area.

4. Double buffer areas and no record area

   With two buffer areas, when one buffer area is either full or empty, L-MSIO automatically switches the blocking/deblocking functions to the second buffer and initiates I/O on the first buffer to empty or refill it. Blocking/deblocking functions must be performed within the user program.

5. Double buffer area and a record area

   This is the most efficient configuration for automatic blocking/deblocking and I/O buffering. PUTs or GETs may be used in one buffer for blocking/deblocking while I/O is being performed with the other buffer.

Two or more files may share buffer areas, a record area, or both. Files which share buffer areas may not be open simultaneously. Files which share only the record area may be open at the same time; the user must determine the file to which the current record belongs.

Only one file on a multifile reel may be open at one time; thus the buffer areas for only one of the files on that reel can be in use at one time. To conserve space, all files on a multifile reel should share the same buffer areas. A file on a multifile reel may not share the same buffer areas with files not on the reel.

LABELS

For mass storage files, L-MSIO references the standard device and file labels. These labels are required. Device labels are prepared by OCAREM when the devices are entered in the system. Mass storage file labels are prepared by OCAREM when the files are allocated.

Files on unit record devices have standard, nonstandard, or omitted labels. The labeling macro is used to define all standard and nonstandard labels, including mass storage, when the file is initially defined with the FILEDESC macro. The formats of the mass storage file and unit record file labels are given in appendix G.

Files on tape drive devices may have header and trailer labels. A header label is a single BCD record used as an identifier at the beginning of a file and at the beginning of intermediate reels of a multireel file.

A trailer label is a single physical BCD record used to mark the end of a file (EOF label) or the end of a reel of a multireel file (EOT label). A trailer label is always in standard format and is preceded and followed by an EOF mark.

A CLOSEF file request for an output file writes an EOF label. A close reel function (CLOSEF macro) writes an EOT label on the file, and a PUT function writes an EOT label when it finds an end-of-tape mark while writing an output file.

STANDARD TAPE LABELING

Standard labeling implies both header and trailer labels in standard format on every reel of the file. The labels are single physical BCD records 80 characters in length. If the user wants to utilize the optional information within either the header or trailer labels, the information must appear in the specific character positions (33 through 80 for header labels, 9 through 80 for trailer labels) in the record area when the label is written. The system moves this information to the record area when the label is read. For multireel files, this optional information may be processed through user label processing routines.

NONSTANDARD TAPE LABELING

Nonstandard labeling implies nonstandard header labels and standard trailer labels for output files, and nonstandard header labels with no trailer label checking for input files. A nonstandard header label can be 16 to 4095 characters in length and may be of two types, data name or NON-STAN. For data name, the address and length of the label buffer area must be stored in the FET by the LABELING macro. Data name label type may be used with all types of logical records. For the NON-STAN type of header label, L-MSIO uses the record area as the label buffer and writes the record area as the label in accordance with the logical record type. Universal type records cannot be used with NON-STAN labels.

If ANSI COBOL is being used, L-MSIO is automatically linked to a COBOL compiler object time routine that checks the nonstandard header label information.

OMITTED TAPE LABEL

Neither header labels nor trailer labels are written or checked for files defined with omitted labels. The optional user label routines are not applicable when labels are omitted.

MULTIREEL FILES

A user may create a multireel file with any one of the three labeling modes. Since L-MSIO recognizes input multireel files from the existence of an EOT trailer label, and the system checks trailer labels for input files only if labels have been declared as standard, output files to be used subsequently as input must have standard labeling. L-MSIO assumes an input file with nonstandard or omitted labels to end at the first end of file mark on the first reel. A multireel file with nonstandard or omitted labels can be processed by treating each reel as a separate file.

MULTIFILE REELS

A multifile reel consists of a set of files contained on a single reel of magnetic tape. A file cannot be on a multifile reel and also be multireel. Each file must be defined by the STOPOPEN macro (refer to section 4). These files may have any type of labeling, but all files on a reel must have the same type.

Files on a multifile reel are handled as all other files except that OPENF and CLOSEF perform additional functions to locate each of the files on the reel. When opening an input file on a multifile reel, L-MSIO compares the position of the file being opened to the current position of the tape. If the new file is farther along the tape than the current file, the tape is spaced forward the appropriate number of files. Otherwise, the tape is rewound and then spaced forward. Any file on an output multifile reel can be accessed and opened for use as input without affecting the status of the remaining files on that reel.

Files on an output multifile reel are written in the order dictated by the program rather than the order in which they are defined in the FET by the STOPOPEN macro. Therefore, if they are not in the same order, the system issues an error message at object time giving the actual file position and the file position specified in the FET. Program execution continues normally. If an output multifile reel is to be used later as input, the system assumes the files to be in the order specified in the FET. The user must ensure that a tape is positioned at the end of the last file written when adding new files to the tape.

Because no two files on a multifile reel may be open simultaneously, they should share buffer and record areas.

USER LABEL PROCESSING ROUTINES

Label processing occurs at the time a reel of a tape file is closed or opened. The following conditions close the current reel and open a new reel.

A CLOSEF request with the close reel option

The discovery of an EOT trailer label on an input file

The detection by PUT of an end of tape mark on an output file prior to a CLOSEF request

L-MSIO enters user specified routines each time a label with which they are associated is processed. These are closed subroutines entered through a return jump. These routines cannot call or use L-MSIO routines (that is, L-MSIO cannot be reentered).

HEADER LABEL PROCESSING

If the user specifies a preheader label routine, the system enters the routine for an input file after reading the label and moving it to the record area but before checking it. For an output file, L-MSIO enters the user label routine before the label has been formed in the buffer.

If a post header label user routine is specified, L-MSIO enters the routine for input files after reading, moving to the user record area, and checking the label. For output files, the system enters the user routine after forming the label in the buffer but before writing it.

TRAILER LABEL PROCESSING

These routines apply only to nonmass storage files. Mass storage files have no trailer labels. L-MSIO enters pretrailer label user routines after reading the trailer label on an input file and moving it to the record area but before checking it or before forming the trailer label for an output file.

A post trailer label user routine is entered for input files after the label has been read, moved to the user record area, and checked. For output files, the user routine is entered after the label has been formed but before it has been written.

The user can modify the label from within the user routine.

DESCRIPTION

The L-MSIO routines are divided into four classes.

- File description routines which build a file environment table (FET) for a file
- Open and close routines which open and close files
- Logical I/O routines which read and write logical records
- Restart routines which restart a program if an irrecoverable I/O error occurs

FILE DESCRIPTION MACROS

L-MSIO uses macros to describe the characteristics of files to be processed. These macros generate an FET for each file that MSIO references during I/O operations. Refer to appendix F for the FET format.

There are five file description macros:

1. FILEDESC
2. LABELING
3. VARIABLE
4. STOPOPEN
5. RERUN

Only the FILEDESC macro is required, and it must be the first macro used. It builds the file FET. The other macros insert values in special purpose fields in the FET which are left blank by the FILEDESC macro. The other macros may appear in any order, but must follow the FILEDESC macro they apply to and must precede the OPENF routine which opens the file.

FILEDESC MACRO

The FILEDESC macro builds the FET and provides the basic information necessary to reference the file.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
filename	FILEDESC	(recl, reca, bufl, bufa, bufb, lu, alu, den, mod, bf, opt, acc)	

filename — Symbolic location of the first word of the FET. It is used to reference the FET in subsequent I/O requests on the file.

recl — Logical record length. Size, in characters, for fixed length logical records or for the fixed length base of variable length records which have a recurring item. If the end of a record is indicated by a record mark or if the total number of characters is variable, this field is specified as zero or is omitted. The maximum size is 4095 characters.

reca — Record area address. Expression representing the word address of the first character of the record area. Files which share the same record area should have the same or equated symbols. If there is no record area for the file, this parameter is omitted.

bufl — Buffer length. Number of whole words, 5 to 4095 are needed to contain the maximum record block in the file. If bufa is not specified, this parameter should be omitted.

bufa — Buffer address. Expression representing the first word address of the buffer for this file.

bufb — Alternate buffer address. Expression representing the first word address of the alternate buffer for this file.

lu — First file number. This field is required for nonmass storage files and omitted for mass storage files.

alu — Alternate file number. Alternate file number used if the file is multireel. If this field is omitted and the file is multireel, the first file number is used.

For mass storage files, this parameter is interpreted as a write check parameter.

Zero or blank	No write check
Other	Write check operations are issued after every write on an output unit

den — Recording density on magnetic tape only.

		7-Track	9-Track
1	low	200 bpi	800 cpi
2	med	556 bpi	800 cpi
3	high	800 bpi	1600 cpi

If this field is omitted, medium density is assumed. The density is recorded in an output label but is not checked on an input label.

mod — Recording mode. Indicates parity of the file.

0 even parity (BCD)†

1 odd parity (binary)††

If this field is omitted, even parity (BCD) is assumed.

---

† Conversion mode on 659 tape drives
†† Pack mode on 659 tape drives

bf — Block format. Indicates whether records are to be written one per block or blocked to the capacity of the buffer area. If the file is on mass storage, this parameter governs whether a preamble is attached to each block.

- 0 Records are blocked to capacity of buffer area. Preamble present for mass storage files.
- 1 One logical record per record block.

opt — Optional file indicator. Indicates optional file. It may or may not be present at object time. For output files, output requests only, this field should be omitted.

- 0 Not optional
- 1 Optional

acc — Access mode for mass storage files.

- 0 Sequential
- 1 Random

LABELING MACRO

This macro is required for all mass storage files and for nonmass storage files with standard or nonstandard header labels.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	LABELING	(owner,fileid,ed,reel,reten)	

owner — Owner of mass storage files. Address of an eight-character owner identification. This field is omitted for nonmass storage files.

fileid — File identification. Address of the file identification which is compared with file name in the label record. For nonmass storage files, this is a 14-character field. For mass storage files, it is a 30-character field. For nonmass storage files, the special symbol NON-STAN specifies that the file has nonstandard labels, and parameters ed and reel have special meanings. If NON-STAN is used and ed and reel are omitted, the label is read into or written from the record area according to the record type.

ed — Edition number. Address in memory of a two-digit (BCD) edition number. If fileid is NON-STAN, this parameter is the address of the nonstandard label area.

reel — Reel number on magnetic tape. Address in memory of a two-digit BCD reel number. It is used when reading and writing labels. If not specified, reel numbers are not checked on input. If fileid is NON-STAN, this is the length in characters of the nonstandard label area.

Access security in mass storage. Address in memory of the four-character access security.

reten — Retention cycle for nonmass storage only. Address in memory of a three-digit BCD retention cycle field. The value can range from 000 to 999.

VARIABLE MACRO

This macro describes the characteristics of files containing variable length records. It is not required if the logical records within a file are fixed length.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	VARIABLE	(type, keyl, key, kfm, max, tis, id, idl, idm)	

type — Logical record type (required)

- 1 — Key field contains the total number of characters within the record.
- 2 — Key field contains the number of times a fixed length trailer item occurs.
- 3 — Universal format. The parameter key specifies the address which contains the size of the record when writing and reading records.
- 4 — Record mark of 72_8 terminates each record.
- n — Record mark of n terminates each record. n can be any six-bit octal number except 01 through 04.

keyl — Size of key field. Number of characters, from 1 to 31, in the key field. For record types 1 and 2 above, the value in the key field may not exceed 4095.

key — Location of key field. Character position for the left boundary of the key field. If the field is within each record of the file, this number indicates its position relative to the beginning of the record. If the field is not in the record, this parameter represents the character address for the first character of the BCD field. If type indicates that the record format is universal, this is the address which contains in binary the length of the record to be written or, when reading, the address where the length and mode (bit 23) of the record read can be stored. †

Bit 23 = 0 — BCD records

Bit 23 = 1 — Binary records

† Refer to the MODEBIT assembly option in the MSOS Installation Handbook.

kfm — Key field indicator

Omitted or 0	Key field is within the record or not used
1	Key field is elsewhere.

If this field is omitted, the key field is assumed to be located within each record of the file. This is not applicable for universal format.

max — Maximum record size. Maximum size for the variable portion of the file records. If the records consist of a fixed base and a trailer item, this field contains a number equal to the size of the trailer item times the maximum number of possible appearances. If the records vary in any other way such as record mark or total number of characters, this parameter is the maximum possible size for any record of the file. If a record mark terminates the record, the maximum length should include this character.

tis — Trailer item size. Character size of the trailer item. If records vary in any other way, this field should be omitted.

id — Position of the left character of the identification field within the record. †

idl — Character size of the identification field. †

idm — Mode of the identification field. †

0 Numeric

1 Alphanumeric

STOPOPEN MACRO

This macro provides information on a file within a set of files. No more than one file within the set is open at the same time. Files which share buffer areas may be described by a STOPOPEN macro. STOPOPEN is required for files on the same reel.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	STOPOPEN	(mnam, mi, mfi)	

mnam — Master file name; name of the file to which all other files in the set are linked. One file from each set is arbitrarily chosen as the master file. The same file name must be specified in all STOPOPEN instructions for files of this set, including the master file.

mi — Master indicator

0 or omitted	Indicates this is not the master file.
1	Indicates master file.

† Optional usage. Currently not used for record description but value is written in FET and mass storage file label by L-MSIO.

mfi	Multifile indicator	
	0 or omitted	Not a multifile reel, but buffer areas are shared.
	1-63	Specifies the position of this file on a multifile reel.

Output files for a multifile reel are written in the order in which they are presented to the I/O system. The user must insure that this order is the same as the position numbers specified for the files.

Only input files to be processed need be declared. Position numbers for every input file need not be present.

The buffer area addresses declared for each file mentioned in a STOPOPEN instruction should be the same.

RERUN MACRO

This macro is required for every file when rerun dumps are taken during execution. If rerun dumps are requested, all FETs for that job must be in consecutive memory locations.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	RERUN	(rlu,fwafets,numfets,freq)	

rlu — Rerun indicator. Indicates that rerun dump is to be taken as defined by freq.

- 0 No rerun dump associated with this FET.
- nn Unit on which rerun dumps are to be taken. For nonmass storage files this is the logical unit. For mass storage files this is the file number assigned the rerun file.

fwafets — First word address of the first FET within this job.

numfets — Number of FETs in this job. The maximum number allowed is 63.

freq — Rerun dump control.

- 0 Rerun dump at end of reel.
- nnnnn Rerun dump taken after each nnnnn physical records of this file (nnnnn equals 1 to 32767).

Parameters fwafets and numfets must always be present for every FET when using rerun dumps. Parameters rlu and freq need to be present only for FETs which control the rerun dumps.

OPEN AND CLOSE ROUTINES

L-MSIO contains an open file routine (OPENF) and a close file routine (CLOSEF). These routines are used in place of the OCAREM open and close functions for files to be processed with L-MSIO routines.

OPENF ROUTINE

OPENF initializes a file for processing. It processes file header labels, positions mass storage output files and multireel tape files, inputs a block of records†, and provides automatic entry to user label processing routines. An OPENF request is made prior to any processing of the file.

OPENING FILES ON UNIT RECORD DEVICES

The following conditions must exist before an OPENF request is made for a nonmass storage file.

1. If labels are used in the file, they are described with the LABELING macro.
2. For output files, any optional information for the file header label is stored in characters 33 through 80 of the record area unless a user preheader routine has been defined for this purpose.
3. If nonstandard data name labels have been specified, the FET must contain the address and length of the label buffer area.
4. Any user routines executed before or after label processing must be specified in the OPENF calling sequence.
5. Files which share buffer areas, including multifile reels, are described by the STOPOPEN macro.
6. The FET is defined by the FILEDESC macro.
7. If a mass storage file was specified as a dump unit with the RERUN macro, the file is opened with an OPEN function.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	OPENF	(filename,use,rewind	key,error,L1,L2,L3,L4)

filename — Symbolic address of a FET which contains the file identifier and other information necessary to open the file (refer to FILEDESC macro).

use — I/O key for the file.

- I Read only
- O Output
- IO Input/output (mass storage only)

†Only for input or I/O files

rewind key Applies only to magnetic tape files and is ignored for mass storage.

zero or blank	Rewind
1	No rewind

error Address of a routine entered after executing the standard I/O error recovery procedure.

L1 Address of a user routine executed before header label processing. Preheader label routine.

L2 Address of a user routine to be executed after header label processing. Postheader label routine.

L3 Address of a pretrailer label routine.

L4 Address of a post trailer label routine.

Calling sequence to OPENF:

```
p      RTJ     MOPENF
p+1    use     fet, rewind
p+2    rev     error
p+3            L1
p+4            L2
p+5            L3
p+6            L4
p+7    Normal return
```

use	01	Input
	02	Output
	03	Input/output (mass storage files only)
rewind	0	Rewind
	1	No rewind

fet Symbolic address of the FET

rev	00	Forward
	01	Reverse (ANSI FET only)

error, L1-L4 Addresses of optional user routines. If absent, addresses in calling sequence are zero-filled.

If an input file was declared as optional in the FILEDESC macro and was not assigned in an EQUIP statement before opening the file, an end-of-file exit is taken on the first GET request for that file. If the file was equipped before opening, normal processing occurs.

OPENF processes header labels according to the specified labeling mode (standard, nonstandard, or omitted). After label processing, L-MSIO positions the file and, for input files, inputs a record or a block of records in accordance with the assignments in the FET.

STANDARD LABELING

For an output file, the system transfers optional user information from characters 33 through 80 of the record area to the label area and writes the label. For input files, L-MSIO reads the label and checks it against the LABELING parameters, identification, edition, and reel number, in the FET.

If the label is valid, L-MSIO transfers it to the user record area. If it is not valid, the system types a message to the operator identifying the file, the label expected, and the label received. The operator may, by typing the proper key word, select one of the following options.

Keyword	Option
X	Ignore the error condition and proceed.
A	Terminate the job.
R	Reread the label, either on the same tape or on a new one mounted by the operator.

NONSTANDARD LABELING

At the time an output file is opened and for each reel of a multireel file, L-MSIO writes the label from the nonstandard label area, if specified in the FET, or from the record area if the label type is NON-STAN. The first record of an input file is read into the nonstandard label area if one is specified, or into the record area. No label checking occurs.

If ANSI COBOL is being used, nonstandard labels are placed into the nonstandard label area where they are checked by the ANSI nonstandard label checking routine.

OMITTED LABELING

No checking or writing of labels occurs.

OPENING FILES ON MASS STORAGE

The following conditions must exist at the time an OPENF request is made for a mass storage file.

1. The area for the file must have been previously allocated.
2. The label must have been specified by the LABELING macro and must be standard.
3. The OPENF calling sequence must specify any user routines to be executed before or after label processing.
4. The FET must have been described by the FILEDESC macro.
5. Files which share buffer or record areas must have been described by the STOPOPEN macro.
6. If a mass storage file was specified as a dump unit with the RERUN macro, the file must have been opened with an OPEN function before the OPENF routine can be called.

For a mass storage file, an OPENF request directs the system to search the label directory for the specified file, get the proper devices on line, update the file usage counter, set the file to open status, and check labels. Information concerning file structure may be in the file label. If so, it is read into the FET, replacing any of the parameters which may have been supplied by the user. Then, the OPENF routine reads a block of records into the buffer area, if the file was opened as an input or I/O file.

During file opening, L-MSIO checks the file label for the following error conditions.

1. The file is being opened but is currently open for output.
2. The file is being opened for output or I/O but is currently open for input.
3. The file is being opened for output or I/O but the read only protection flag is set in the label.
4. Wrong access security code.
5. The file identification cannot be found in the label directory and the file was not declared optional.
6. The file cannot be placed on line and the file was not declared optional.
7. Too many files are open and there is no more room in the resident tables.

If one of these errors exists, L-MSIO sends a message to the operator and to the standard output unit, giving owner identification, file name, edition number of the requested file, and type of error.

If the input file is declared optional and the file label does not appear in the file label directory or the operator elects not to place the file on-line, the first GET request for the file causes an end-of-file or invalid key exit. The job is terminated if an optional output file does not exist or cannot be mounted.

A mass storage input file may be opened any number of times without intervening close requests in the same program as long as a different FET is supplied for each OPENF request. For example, a mass storage input file may be in an open condition for both random and sequential accessing. An output file can be opened only once without an intervening CLOSEF.

CLOSEF ROUTINE

CLOSEF terminates processing on a file for a specific FET, prevents any subsequent accessing of that file until it is opened again, updates the file label, and releases mass storage resident tables. CLOSEF applies to both magnetic tape and mass storage files. A close reel option is available for use with files on magnetic tape. It closes a reel of a tape file and opens the next reel. CLOSEF is called by the following macro.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	CLOSEF	(filename, p2, p3)	

filename — Symbolic address of a FET containing the file identifier and other information necessary to close a file (refer to FILEDESC macro).

p2 — A rewind option for magnetic tape files.

zero or blank	Rewind to load point
1	Unload
2	No rewind

This parameter is omitted for mass storage.

p3 — Close reel option intended for use with multireel magnetic tape files only and used to close a reel of a multireel file prior to the normal end.

R	Close current reel and open next reel.
other	Close file.

For ANSI COBOL, the close reel option rewinds and unloads the reel.

Calling sequence to CLOSEF:

```
p     RTJ     MCLOSEF
p+2   rw      fet,r
p+3   Normal return
```

fet — Symbolic address of the FET.

r — Reel/file option

1	Close current reel and open next reel.
0	Close file.

rw — Rewind option

tape rw	00	Rewind
	01	Rewind and unload
	02	No rewind
mass storage rw	00	

When closing an output or I/O file, L-MSIO writes any records remaining in the buffer areas on the I/O unit.

CLOSING FILES ON UNIT RECORD DEVICES

After emptying the buffers for output files, L-MSIO writes an end of file mark, an ending label, and another end-of-file mark. L-MSIO writes an EOT label for a close reel request and an EOF label for a close file request.

The ending label always has the following standard format.

Character Position	Content	Explanation
1-3	EOT	End of tape for an intermediate reel
	EOF	End of file for a final tape
4-8	nnnnn	Physical record (block) count for the reel
9-80	any	Optional information

The optional information is obtained from characters nine through 80 of the user record area. If the user wishes to have this information in his labels, he must define a pretrailer label routine or put the information there prior to a CLOSEF request for output files. The optional information from the EOF label is in the user record area when the end of file routine is called.

If an EOT label is encountered on an output file, L-MSIO produces a rerun dump if requested and opens the next reel of the file. An EOF label on an input file causes an exit to the user end-of-file routine.

When closing an input file with the reel option, L-MSIO does not check the end reel label but assumes it to be an EOT label and opens the next reel in the sequence. A CLOSEF request must be given for the last reel of a multireel file.

CLOSEF stores the optional information in the user record area when encountering a trailer label on an input file. A post trailer label user routine can be used to examine this information in the case of a multireel file.

CLOSING MASS STORAGE FILES

Mass storage files do not have ending labels. The user ending label routines specified in the OPENF request have no meaning for mass storage files. A CLOSEF request for a mass storage file updates the FET and file label and closes the file to further processing through that FET. The file ordinal and the label and error user routine parameters are cleared from the FET, and control returns to the calling program.

LOGICAL READ/WRITE ROUTINES

The L-MSIO read/write routines allow the user to read and write logical records in L-MSIO formatted files.

GET ROUTINE

The GET routine deblocks input records by reading them from an input buffer and moving them to the record area. When an input buffer has been deblocked, GET automatically reads a new block of records from the input unit. A file is read, one record at a time, by looping on a GET request until an end of file is sensed. GET is used on any L-MSIO formatted file that is open for input or I/O.

After a file is opened, the first GET request transfers the first record in the input buffer to the record area for inspection or processing. If no record area was specified in the FILEDESC macro, GET returns the first character address of the record in register A and returns the record size, in characters, in register Q. This allows the user to do his own deblocking. The process is repeated for the next record in the buffer area each time a GET request is made.

When all records in a buffer have been processed, the next GET reads the next block of records into the buffer area, and then moves the first record of the new block into the record area. If there is no record area, the record address and size is returned in the A and Q registers. If two or more buffer areas were specified, the GET function reads the next records from the second buffer area while the first buffer area is being refilled.

Optionally, the user can request that all or part of the record be transferred to an additional area by specifying an INTO-area and an INTO-size. If the INTO-size is less than or equal to the logical record size, L-MSIO transfers only the number of characters specified. If the INTO-size is greater than the logical record size, the unused portion of the receiving field is blank filled.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	GET	(filename,ur,INTO-area,INTO-size)	

filename — Symbolic address of the FET (refer to FILEDESC macro).

ur — Address of a user routine to be entered if an end-of-file or invalid key condition exists when a program calls GET. This is an open subroutine which L-MSIO calls by a UJP.

INTO-area — Character address of an area in addition to the record area into which the record is read.

INTO-size — Size in characters of the INTO-area. This must be specified if INTO-area is present.

Calling sequence for GET:

```
p      RTJ    MGET
p+1           fet
p+2           ur
p+3           INTO area
p+4           INTO size
```

fet — address of the FET

ur

INTO area } same as in the macro description; if absent, addresses in calling sequence are zero-filled

INTO size

The system enters the routine specified by ur when GET encounters either an end of file condition on a file declared as sequential access or an invalid key condition on a file defined as random access. An invalid key condition for a mass storage file in random access mode occurs when a program attempts a GET request with a SAK which is outside the physical limits of the file. Any attempt to execute a GET statement after an end-of-file or invalid key user routine has been called is illegal unless subsequent CLOSEF and OPENF requests have been executed for the same file. The CLOSEF and EXPAND macros can be used to expand the size of the file when an invalid key condition occurs. To clear an invalid key condition on random files, execute a LOCATE request. If a GET request detects an irrecoverable error, L-MSIO moves the current record to the record area prior to calling the user error routine.

PUT ROUTINE

The PUT routine writes logical records in an output buffer and, when the buffer is full, automatically writes the block of records on an output unit. The PUT routine can be used on all files opened as output or I/O files.

A PUT request writes the contents of the record area in an output buffer and advances a record counter by one for each record written. If no record area was specified in the FILEDESC macro, the PUT request returns the first character address of the next available space in the output buffer to register A, and advances the record counter by one. With no record area, the user must do his own blocking.

When a buffer has been filled with records, the next PUT writes the block on an output unit. If two buffers are not available, PUT waits for the buffer to be written on the output unit and then writes the next record in the emptied buffer.

The PUT routine does not split records across a block. If a record does not fit in the remaining space of a buffer, PUT outputs the partially filled block and then writes the record as the first record of the next block.

The user can specify an area, other than the record area, from which a record is to be blocked with the FROM option. When the FROM option is selected, records are transferred from the FROM area directly to the output buffers. If there are no specified output buffers, the record areas serve as the output buffer and receive the records transferred from the FROM area. The FROM option used with the ANSI COBOL variant moves records from the FROM area to both the record area and output buffer areas if both areas are specified.

If the records of the file are fixed length, the FROM size indicates the number of characters to be transferred. It may be less than, equal to, or greater than the length of a logical record. If the length is greater than that of a logical record, the system truncates the record to fixed length size. If equal to or less than the size of the logical record, L-MSIO transfers the number of characters specified.

If the records of the file are variable length, the number of characters transferred depends on the value of the key field or the presence of a record mark, as well as the FROM size. If the key field is not in the record, the field must be in the same location as for all other records of the file. If the key field is normally in the record, it must be in the same relative position in the FROM area. If the FROM size is greater than the length indicated by the key field, L-MSIO truncates the record to the indicated size and moves it to the buffer. If the FROM size is less than the length specified by the key field, L-MSIO fills the unused portion of the record with blanks. For universal record format, the key field is outside the record and contains the length in binary of the record to be transferred.

If a record mark determines the length of each record in the file, L-MSIO searches the FROM area until it encounters a record mark. L-MSIO then transfers the record (starting with the FROM address and including the record mark) to the buffer. If a record mark is not found before the maximum logical record length is exceeded, the system issues a diagnostic and terminates the run.

PUT is referenced by the following macro:

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	PUT	(filename,ur,FROM-area,FROM-size)	

filename — Symbolic address of the FET (refer to FILEDESC macro).

ur — Address of a user routine to be entered if an invalid key condition exists. This routine is an open subroutine which L-MSIO calls by a UJP. It is not required for nonmass storage files.

FROM-area — Character address of the area from which the record is to be transferred. This overrides the record area specified in the FET.

FROM-size — Size in characters of the FROM-area. This must be specified if FROM-area is present.

Calling sequence for PUT:

```
p      RTJ     MPUT
p+1            fet
p+2            ur
p+3            FROM-area
p+4            FROM-size
```

fet — address of the FET

ur, FROM-area, FROM-size — same as in the macro description; if absent, addresses in calling sequence are zero-filled

The invalid key condition occurs for mass storage files when an attempt is made to write outside the physical limits of the file. In sequentially accessed files, L-MSIO enters the user invalid key routine when a PUT request encounters the end of the last block of a mass storage file. The invalid key routine is entered immediately after the last record is written in the buffer, but before the buffer is written on the file. To write the last buffer, the file must be closed in the invalid key routine with CLOSEF. Closing causes all records in the file's output buffer to be written. Then the file can be expanded and reopened in the invalid key routine if more data must be written.

In randomly accessed files, L-MSIO enters the invalid key routine when a PUT request follows a locate which located to a block outside of the physical file limits.

Both GET and PUT requests can be made on an I/O file. † For sequential files the seek address key (SAK) is updated or advanced only by the GET request. All PUT requests that follow a GET request refer to the last record referenced by the GET request.

LOCATE ROUTINE

LOCATE references a specific record on a mass storage file. Randomly accessed files require a LOCATE request before each GET or PUT to position the file to the record to be referenced. LOCATE may be used while processing a sequentially accessed input file in order to position the file to a noncontiguous record from which sequential processing resumes. LOCATE is referenced by the following macro.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	LOCATE	(filename, rlp)	

filename — Symbolic address of the FET (refer to FILEDESC macro).

rlp — Record location parameter.

- 0 or blank — File is positioned to the highest block written plus one for writing a record.
- other — Word address of a user supplied two-word SAK which L-MSIO uses to position the file (refer to file access in section 13).

Calling sequence to LOCATE:

```
p      RTJ     MLOCATE
p+1            fet
p+2            rlp
```

fet — Core address of the FET.

rlp — Same as in the macro description.

When locating to other than the first record in a block, the two word preamble should be included in the calculation of the relative character position of the record.

† Mass storage only.

LOCATE is illegal on a sequential output file. For a sequentially accessed input or I/O file, L-MSIO initiates buffering following the position operation in accordance with buffer assignments in the FET. Since L-MSIO cannot anticipate the next record address for random access files, only the current buffer is filled.

If an attempt is made to LOCATE outside the allocated area of a file, an invalid key condition is flogged so that the next GET or PUT causes an invalid key exit.

RELSE ROUTINE

The RELSE macro releases the records in the current buffer area and prepares the buffer for the next block of records. For output buffers, RELSE writes the current partially filled buffer on the output unit. All records in the buffer, up to and including the last record referenced with a PUT or GET, are written. The remainder of the record block is written as blank fill on the output unit. The next PUT macro writes the next record in the first record position of the cleared buffer.

For input files, RELSE fills the buffer area with the next block of records from the input unit. The next GET macro reads the first record in the refilled buffer area.

If two RELSE commands are given consecutively, the second is treated as a no operation. RELSE is referenced by the following macro.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	RELSE	(filename)	

filename Symbolic address of the FET (refer to FILEDESC macro).

Calling sequence for RELSE:

```
p      RTJ     MRELSE
p+1            fet
```

fet Address of the FET.

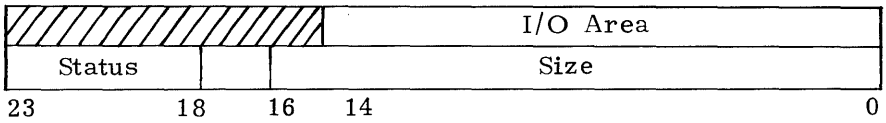
READF ROUTINE

READF reads a record or block of records into a record area or an input buffer. All read functions are sequential unless preceded by a file positioning routine (LOCATE). Either of the two possible buffers specified in the FET or a record area specified in the READF calling sequence may be used as the input area.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	READF	(filename,fwa x)	

filename Symbolic address of a FET (refer to FILEDESC macro).

fwa x First word address of an optional two word table in the following form:



I/O Area First word address of the I/O record area.

Size Size in characters of record to read.

Status Buffer status code returned by L-MSIO as follows:

Status	Description
00 Internal	Buffer available for use
01 External	Buffer currently in I/O processing
02 Error	Irrecoverable error on this buffer
20 EOF	EOF mark on this buffer
30 EOT/Invalid Key	EOT mark on this buffer if nonmass storage file; invalid key condition if mass storage file
32 EOT/Error	EOT and irrecoverable error on this buffer

There is no end of file status for reverse reads. A job aborts if a reverse read is attempted past the load point.

Calling sequence to READF:

```
p      RTJ     MREADF
p+1            fet
p+2            fwa x (optional)
```

fet Address of the FET.

fwa x First word address of an optional two word table; if omitted, location p+2 is zero-filled.

The fwa x parameter is optional and, if omitted, L-MSIO initiates a read into the currently active buffer within the FET. If the user wishes to control buffering and does not require GET, he may specify the fwa x parameter. In this case, buffer areas must not be assigned within the FET.

If no user error routine was specified in the OPENF request for the file, an irrecoverable error gives control to the operator. The operator must type in one of the following:

X Ignore the error condition and proceed.

B Skip the erroneous record and proceed.

A Terminate the job at this point.

R Repeat the standard error recovery for this record.

If the OPENF request specified an error routine and no buffer areas are specified in the FET, L-MSIO sets the A register to the first word address of the buffer containing the error record and the Q register to the buffer size. Then the system enters the user error routine through the following calling sequence.

```
p      RTJ    user error routine
p+1    return
```

Before returning to L-MSIO, the user error routine must enter a code in register A to specify one of the following three alternatives.

0 Accept

1 Bypass

2 Retry

In addition to the above options, a call to ABNORMAL can be made to terminate the program.

If the OPENF request specified an error routine and the FET assigned buffers, STATUS is set to indicate an irrecoverable error.

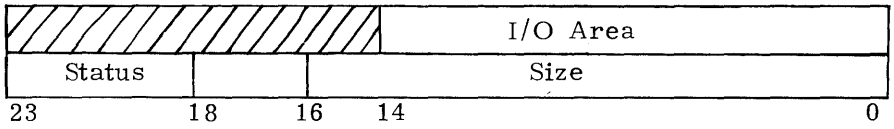
WRITEF ROUTINE

WRITEF writes a record or a block of records on an output unit. All write functions are sequential on the output unit unless the WRITEF request is preceded by a file positioning function (LOCATE request). Either of the two possible buffers specified in the FET or a record area specified in the calling sequence may be used as the output buffer.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	WRITEF	(filename,fwa x)	

filename Symbolic address of a FET (refer to FILEDESC macro).

fwa x First word address of an optional two word table in the following form:



I/O Area First word address of the I/O area.

Size Size in characters of the record to be written.

Status Refer to buffer status codes for READF macro.

Calling sequence to WRITEF:

```
p      RTJ     MWRITEF
p+1            fet
p+2            fwa x (optional)
```

fet — Address of the FET.

fwa x — First word address of an optional two word table; if omitted, location p+2 is zero-filled.

The fwa x parameter is optional. If it is omitted, L-MSIO initiates a write from the currently active buffer within the FET. If the user wishes to control buffering and does not required PUT, the fwa x parameter may be specified. In this case, buffers should not be assigned within the FET.

Error recovery action is similar to that described under READF.

PAUSEF ROUTINE

PAUSEF creates a pause until the previous READF or WRITEF request, including error recovery, is completed. If the READF or WRITEF used the optional parameter, fwa x, the status field contains the status of the last operation.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	PAUSEF	(filename)	

filename — Symbolic address of the FET (refer to FILEDESC macro).

Calling sequence to PAUSEF:

```
p      RTJ     MPAUSEF
p+1            fet
```

fet — Address of the FET.

RESTART FUNCTION

The L-MSIO restart routines CHECKPT and RESTART make periodic restart dumps while a program using L-MSIO formatted files is running. If an irrecoverable I/O or other error occurs which would normally require restarting the program from the beginning, the program can be restarted at the point of the latest restart dump rather than at the beginning.

Prior to using CHECKPT and RESTART, the RERUN macro must be used to enter the dump unit number and dump frequency in the FETs of all files used in the program. At the points in the program specified by the RERUN macro, a dump of all memory and registers being used by the program is taken, and the position of all open L-MSIO formatted files is recorded in the dump file. The dumps are written sequentially on the dump unit specified in the RERUN macro. Any of the dumps on the dump unit can be selected for a restart.

CHECKPT ROUTINE

CHECKPT is used to produce rerun dumps any place in the program that a CHECKPT macro or call is inserted. The dumps taken by CHECKPT are in addition to those caused by the RERUN macro.

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
	CHECKPT	(filename)	

filename — Symbolic address of the FET related to the rerun dump (refer to FILEDESC macro).

Calling sequence to CHECKPT:

```
p      RTJ     MCHECKPT
p+1            fet
```

fet — Address of the FET.

RESTARTING A PROGRAM

To restart a program, the user must ensure that the program is operating in the same physical environment as it was when the rerun dump was taken.† The RESTART statement relocates each open file to the position where the last rerun dump was taken, and then restarts the program.

† The system must have the same units equipped as well as the same tape reels, library editions, core addresses, and file numbers as before.

Use caution when restarting a program which uses files that are not on magnetic tape. When an input file consists of cards, the entire file must be reloaded. Mass storage I/O files may be modified between rerun dumps. Therefore, the nature of the program determines whether restarts are acceptable.

```
$RESTART,uu,n
```

uu	Logical unit number or file number on which the rerun dumps were written.
n	Number of the rerun dump to be used in the restart operation. If blank, the last dump written is used.

If rerun dumps were taken on mass storage, the file containing the dumps must first be opened by using the OPEN statement. If uu is a file number, it must not be the same as a logical unit number used within the program.

L-MSIO MACRO EXPANSIONS

The following job may be run to obtain a list of the L-MSIO macro expansions.

```
$JOB...
$COMPASS,L
    IDENT       EXPAND
    LIBM        FILEDESC,LABELING,VARIABLE
    LIBM        STOPOPEN,RERUN,OPENF,CLOSEF
    LIBM        GET,PUT,LOCATE,RELSE
    LIBM        READF,WRITEF,PAUSEF,CHECKPT
    END
    FINIS
77
88
$EOJ
```

STANDARD JOB OUTPUT AND SYSTEM JOB ACCOUNTING 16

STANDARD LIST OUTPUT

All batch jobs have list output at the system list output unit. The list output consists of job identification and accounting information, a copy of each MSOS control card used in the job, and the data written on the list unit by the program which was executed. The last line on the list output gives the amount of CPU time, in minutes and seconds, that was used by the job, plus the lines printed, and number of cards punched.

If a job deck was preceded by a SEQUENCE statement, the SEQUENCE statement is written on the second line. If no SEQUENCE statement was used, the JOB statement is the first statement written.

In addition to the normal information supplied on the JOB statement, MSOS writes the following additional information on the same line the JOB statement is written on.

1. Job sequence number
2. MSOS version number
3. Library edition number
4. Date
5. Job initiation time

After the SEQUENCE and JOB statements, all MSOS control statements are listed in the order that they were read and processed. Program list output immediately follows its associated library program call statement, or the RUN statement. For example, COBOL compiler output follows the COBOL statement and the output from a binary program deck follows the RUN statement.

JOB SEQUENCE NUMBERS

MSOS generates a sequence number for each batch job and writes the sequence number on the same line as the JOB statement on the list output.

For jobs without a SEQUENCE statement, MSOS assigns sequence numbers from 1 through 999 in accordance with the order that the jobs are processed (that is, first job processed after autoloading the system is job 1, second job is job 2, etc.). The job sequence number reverts to 1 after the 999th job.

If a SEQUENCE statement is used, the number on the SEQUENCE statement replaces the sequence number generated by MSOS. The sequence counter is advanced for the next job and the number generated by MSOS for the job is skipped.

If a job has punched output, its sequence number is punched in block letters on a job sequence card which is the last card in the deck (refer to job sequence card in section 6). This card identifies the job punch output. The sequence cards are offset to separate the punched output decks.

ELAPSED TIME ACCOUNTING

An accounting routine in MSOS accumulates elapsed time for each batch job and prints this information at the CTO and at the end of the system list output.

```
I  SYS  400  hh/mm/ss  id  L=n  C=c
```

hh	Hours
mm	Minutes
ss	Seconds
id	Account number from the JOB card
n	Number of lines printed
c	Number of cards punched

Accounting starts when MSOS reads a new job statement. Elapsed time accounting stops when MSOS reads an EOJ statement or senses an EOJ condition. The accounting routine subtracts the time used by any priority programs which interrupted the batch program so that the batch program is not charged for priority program time. MSOS does not perform accounting functions for priority programs.

SPECIAL USER ACCOUNTING ROUTINES

If additional accounting information is desired, a custom accounting routine may be added at system installation time. This routine must be supplied by the user. File number 57 is reserved as a system accounting file for the user supplied accounting routine.

SYSTEM ACCOUNTS TABLE

MSOS has a system accounting table which batch and priority programs written in COMPASS source language may reference by using the symbolic address ACCOUNTS. The accounts table holds only information pertaining to the current batch program. Accounting information is not kept for priority programs.

Accounts Table Format:

	123 — 18	17 — 12	11 — 6	5 — 00
ACCOUNTS	m		d	
ACCOUNTS +1	y		ed	
ACCOUNTS +2	*	h	min	s
+3	*			
+4	*	j		
+5	ts			
+6	act			
+7				
+8	i			
+9	te			
+10	*			
+11	nl			
+12	lp			
ACCOUNTS +13	nc			
ACCOUNTS +14	cp			

m	Month. Two BCD digits from autoload.
d	Day. Two BCD digits from autoload.
y	Year. Two BCD digits from autoload.
ed	Current edition of library. Two BCD digits from autoload.
h	Hours. Two octal digits updated at the beginning of each job.
min	Minutes. Two octal digits updated at the beginning of each job.
s	Seconds. Two octal digits updated at beginning of each job.
j	Sequence number from the last SEQUENCE card processed (BCD) or sequence number from system counters.
ts	Machine clock time (octal), from register file register 22, at the beginning of the job.
act	Account number from JOB card. Eight BCD characters.
i	Job name from job card. First four BCD characters.
te	Time estimate in seconds from JOB card. Eight octal characters.
nl	Maximum number of lines that can be printed on the standard output unit for the job. Value is from JOB card.
lp	Actual number of lines printed. This field is set at the end of the job.
nc	Maximum number of cards that can be punched for the job. Value is from JOB card.
cp	Actual number of cards punched. This field is set at the end of the job.
*	Reserved for future use.

DATE AND TIME UTILITIES

The DATE and TIME utilities place the current date and time in the address specified by the buffer. These statements may be used by COMPASS or ANSI FORTRAN. The calling sequence for the data or the time is:

```
EXT     DATE or TIME
RTJ     DATE or TIME
n       BUF
```

Where n specifies the buffer size (ANSI FORTRAN requires that n equal 75). All values returned are in BCD format. If the buffer size is two words; the current wall clock time is returned with format of hh $\leq$ mm $\leq$ ss, or the current date is returned with format of mm/dd/yy.

If the buffer size is one word, the date is returned with format of mm dd or the job-start time (not current time) is returned with format of hh/m.

ABNORMAL PROGRAM TERMINATION AND PROGRAM DUMPS

17

ABNORMAL TERMINATION

When a user wishes to terminate his program because an error or program condition has made further processing impractical, he may make a return jump to ABNORMAL. ABNORMAL terminates the program. If a DUMP statement was inserted in the program (refer to DUMP statement in section 4), ABNORMAL causes the core dump to be taken. A core dump of a priority program is delayed until the batch program that was interrupted by the priority program is completed. Figure 17-1 is a sample core dump.

Example:

```
RTJ     ABNORMAL
```

An alternative method of abnormally terminating a program is to increase the return address by one. The address is in the lower fifteen bits of the program entry point. Then, set a nonzero value in register A and take a normal exit. The program is abnormally terminated but no core dump is taken, regardless of the presence or absence of a DUMP statement. This method of termination may be used to suppress a dump when a DUMP statement is included in the program.

When a program in a job is abnormally terminated, all remaining control statements and/or programs on the job's input file are bypassed, until an end of file statement is sensed. All control statements and/or programs in the job that follow the end of file statement are executed.

PROGRAM DUMPS

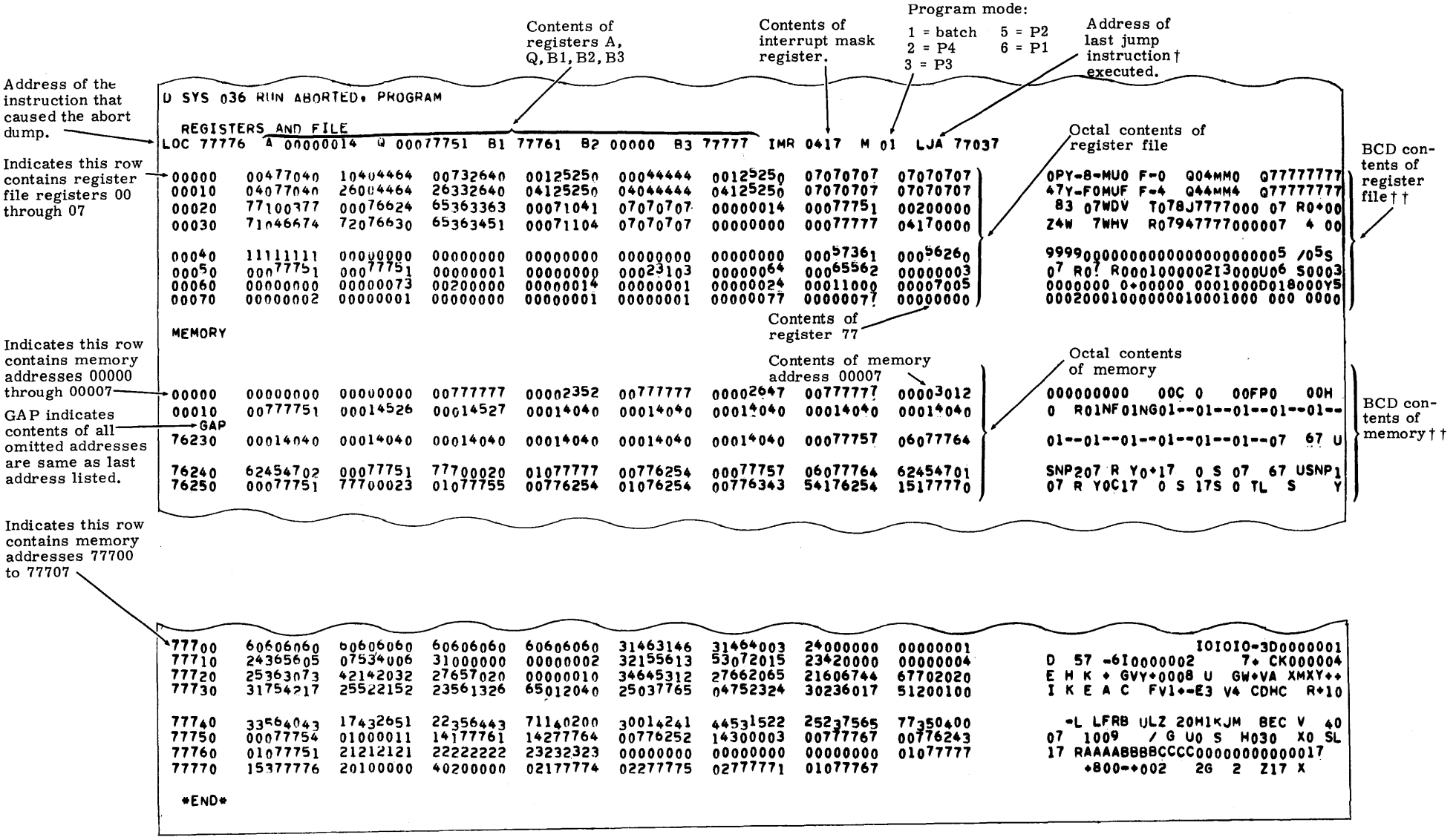
COMPASS and MS FORTRAN users can call a dump routine to print any portion of the program in octal, BCD, or floating point characters. The dump also prints the contents of the registers and optionally, the register file. The format of the dump is the same as the SNAPSHOT dump shown in Figure 4-1.

In COMPASS programs, the dump is obtained by a call to PROGDUMP. In MS FORTRAN programs, the dump is obtained by a call to FORTDUMP.

PROGDUMP

Calling sequence to obtain a dump with COMPASS:

LOCATION	OPERATION, MODIFIERS, ADDRESS FIELD		COMMENTS
1 8	10	20	41
p	RTJ	PROGDUMP	
p+1		b	
p+2	m	e	
p+3	BCD	1, id	
p+4	*		



†This value is listed only by 3100 or 3200 systems with dynamic memory protection, or by 3300 or 3500 systems with standard memory protection. In the extended core variant, the loader converts the jump instructions which reference the executive subroutines to HLT instruction. (Refer to executive interrupts in section 18).

††On 512 line printers, special characters on the print train (that is, characters that appear only once) are not printed to decrease the time required to print the dump.

Figure 17-1. Sample Program Abort Dump

b Beginning address of the dumped region.

m Mode in which the dump of selected core memory occurs.

Octal Code	Mode
1	Octal
2	BCD character
3	Floating point
4	Register file
5	Octal; register file
6	BCD character; register file
7	Floating point; register file

e Ending address of dumped region.

id Dump identifier of 0 to 4 BCD characters.

* Control returns at this instruction.

PROGDUMP enables an interrupt and should not be called during an interrupt subroutine. PROGDUMP must be declared as an external symbol in the subprogram that contains the calling sequence.

FORTDUMP

FORTRAN statement to call the system dump routine:

```
1    5 7
      CALL FORTDUMP(b,e,m,d)
```

b Simple or subscripted variable of first word to be dumped.

e Simple or subscripted variable of last word to be dumped.

m Mode; octal constant with same meaning as COMPASS PROGDUMP, or a variable identifier for the location of the constant.

d Hollerith or internal BCD constant or variable identifier giving the location of four BCD characters that identify the dump.

FORTDUMP need not be declared as an external symbol in the source program.

PROGDUMP AND FORTDUMP EXAMPLES

Examples:

To dump the area from BUFY to BUFY+24 and identify the dump as PRG1:

COMPASS Calling Sequence

```
RTJ     PROGDUMP
        BUFY
 1      BUFY+24
BCD     1,PRG1
 :        :
 :        :
```

FORTRAN Call

```
CALL FORTDUMP(BUFY(1),BUFY(25),1,4HPRG1)
```

INTERRUPT CONTROL 18

INTERRUPT PROCESSING

MSOS has a Central Interrupt Control routine (CIC) which is part of executive resident. Return jumps to CIC subroutines may be coded in COMPASS programs to select and clear internal program interrupts. External (I/O) interrupts must be selected with CIO.

MSOS interrupts can be divided into three types: user interrupts, executive type interrupts, and system interrupts. User interrupts are selected by coded statements within the user program. These interrupts enable the user to sense and process internal program fault conditions, I/O completions, time intervals, and operator commands. When a program is interrupted, CIC gives control to a user supplied interrupt routine to process the interrupt.

Executive type interrupts apply only when the system is in the program state (that is, only to batch and priority 3 and 4 programs) and only for the memory protect and extended core variants of MSOS. These interrupts occur if the user attempts to execute an illegal instruction or do an illegal write in a protected memory area. CIC processes the interrupt and usually terminates the program. Illegal instructions are listed in appendix D. The interrupt routines used to process this class of interrupts are part of the system executive.

System type interrupts are activated when a system condition, such as an I/O channel ready or trapped instruction, occurs which requires immediate action by the system executive.

The user has no control over the occurrence of these interrupts. The system returns control to the interrupted program after processing the interrupts.

When a program is interrupted, control is given to CIC which:

1. Saves the contents of all registers being used by the program.
2. Clears the interrupt condition so that the interrupt will not be sensed a second time without being reselected.
3. Gives control to the associated interrupt routine to process the interrupt.

If additional interrupts occur while a previous one is being processed, the interrupt processing routine is interrupted and control is returned to CIC again. If the new interrupt is a higher priority, it is processed before control is returned to the first interrupt processing routine. If the new interrupt is of lower or equal priority, CIC stacks it in an interrupt table for later processing according to its priority level, and then returns control to the interrupted routine. CIC itself cannot be interrupted, except by a memory parity error or power failure interrupt.

Table 18-1 lists the interrupts and programs in accordance with their priority level. Priority 1 through 4 programs are interrupt processing routines which are core resident. They are entered (executed) each time their associated interrupt occurs. When an interrupt gives control to a priority program, that program is allowed to complete execution before any interrupt routine below the program's priority level is processed. The system processes all stacked interrupts before reentering the batch program.

TABLE 18-1. INTERRUPT AND PROGRAM PRIORITY LEVELS

Priority Level	Interrupt	Interrupt Selected in	Interrupt Type	Processing State	Remarks
1	Real-time interrupt	Priority 1 program	User	Monitor	CIC clears the interrupt condition (with INCL) but does not enable the interrupt system before entering the interrupt processing routine. The interrupt processing routine will be interrupted and reentered if the interrupt system is enabled (with EINT) and a subsequent priority 1 real-time interrupt occurs. Priority 1 real-time interrupts are the only ones that interrupt CIO processing.
2	I/O channel equipment interrupt	System executive	System	Monitor	----------
3	System real-time clock	System executive	System	Monitor	CIC writes the clock interrupt's next value in its table.
3	Manual interrupt	System executive	System	Monitor	CIC stacks the logical interrupts to the user and moves message to user buffer.
4	Real-time interrupts	Priority 2 program	User	Monitor	CIC clears the interrupt condition but does not enable the interrupt system before entering the interrupt processing routine. P2 real-time interrupts are stacked if priorities 1 through 3 are executing.
5	Trapped instruction interrupt (only for occurrence during priority 1 processing)	System executive	System	Monitor	----------
6	Internal fault interrupt	Priority 1 program	User	Monitor	CIC clears the interrupt condition but leaves the interrupt selection before entering the interrupt routine.

TABLE 18-1. INTERRUPT AND PROGRAM PRIORITY LEVELS (Cont'd)

Priority Level	Interrupt	Interrupt Selected in	Interrupt Type	Processing State	Remarks
6	Manual interrupt	System executive	System	Monitor	CIC stacks the logical interrupts to the user and moves message to user buffer.
7	I/O interrupt (equipment only)	Priority 1 program	User	Monitor	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine.
7	Clock interrupt	Priority 1 program	User	Monitor	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine. The user must reselect the clock interrupt in his interrupt routine or program if he needs further clock interrupts.

TABLE 18-1. INTERRUPT AND PROGRAM PRIORITY LEVELS (Cont'd)

Priority Level	Interrupt	Interrupt Selected in	Interrupt Type	Processing State	
8	Trapped instruction interrupt (only for occurrence during priority 2 processing)	System executive	User	Monitor	----------
9	Internal fault interrupt	Priority 2 program	User	Monitor	CIC clears the interrupt condition but leaves the interrupt selection before entering the interrupt routine.
9	Manual interrupt	System executive	System	Monitor	CIC stacks the logical interrupts to the user and moves message to user buffer.
10	I/O interrupt (equipment only)	Priority 2 program	User	Monitor	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine.
10	Clock interrupt	Priority 2 program	User	Monitor	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine. The user must reselect the clock interrupt in his interrupt routine or program if he needs further clock interrupts.
11	Executive interrupt during priority 3 processing	System executive	Executive	Monitor	----------
11	Illegal write interrupt during priority 3 processing	System executive	Executive	Monitor	----------
11	Trapped instruction interrupt during priority 3 processing	System executive	System	Program	----------

TABLE 18-1. INTERRUPT AND PROGRAM PRIORITY LEVELS (Cont'd)

Priority Level	Interrupt	Interrupt Selected in	Interrupt Type	Processing State	Remarks
12	Internal fault interrupt	Priority 3 program	User	Program	CIC clears the interrupt condition but leaves the interrupt selection before entering the interrupt routine.
12	Manual interrupt	System executive	System	Monitor	CIC stacks the logical interrupts to the user and moves message to user buffer.
13	I/O Interrupt (equipment only)	Priority 3 program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine.
14	Clock interrupt	Priority 3 program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine. The user must reselect the clock interrupt in his interrupt routine or program if he needs further clock interrupts.
15	Executive interrupt during priority 4 processing	System executive	Executive	Monitor	----------
15	Illegal write interrupt during priority 4 processing	System executive	Executive	Monitor	----------
15	Trapped instruction interrupt during priority 4 processing	System executive	System	Program	----------
16	Internal fault interrupt	Priority 4 program	User	Program	CIC clears the interrupt condition but leaves the interrupt selection before entering the interrupt routine.
16	Manual interrupt	System executive	System	Monitor	CIC stacks the logical interrupts to the user and moves message to user buffer.

TABLE 18-1. INTERRUPT AND PROGRAM PRIORITY LEVELS (Cont'd)

Priority Level	Interrupt	Interrupt Selected in	Interrupt Type	Processing State	Remarks
17	I/O interrupt	Priority 4 program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine.
17	Clock interrupt	Priority 4 program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine. The user must reselect the clock interrupt in his interrupt routine or program if he needs further clock interrupts.
18	Executive interrupt in batch program	System executive	Executive	Monitor	----------
18	Illegal write interrupt in batch program	System executive	Executive	Monitor	----------
18	Trapped instruction interrupt in batch program	System executive	System	Program	----------
19	Internal fault interrupt	Batch program	User	Program	CIC clears the interrupt condition but leaves the interrupt selection before entering the interrupt routine.
20	I/O interrupt	Batch program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine.
20	Clock interrupt	Batch program	User	Program	CIC clears the interrupt condition and the interrupt selection before entering the interrupt routine. The user must reselect the clock interrupt in his interrupt routine or program if he needs further clock interrupts.
21	Batch program	-----	-----	Program	----------

REAL-TIME INTERRUPTS

Real-time interrupts have the highest priority of all user interrupts. These interrupts can be selected only in a priority 1 or 2 program. Real-time interrupts are activated by equipment interrupts from a special I/O channel.

The channels used to activate real-time interrupts must be dedicated channels. The channels cannot be defined in the system (entered in the AUT table), and cannot be used or referenced by the system executive, priority 3, priority 4, or batch programs. If an existing channel is to be used, the channel definition must be removed from the AUT by downing all I/O units on the channel before the channel is used for real-time applications.

Since the dedicated channel is not available to the system, CIO cannot be used to do I/O, interrupt processing, error checking, or error recovery functions on the channel. This can be done only by a priority 1 or 2 program using standard COMPASS I/O instructions.

Real-time interrupts must be preselected by a call to SETCHV50 in order to inform CIC of the channel and equipment number that the interrupt occurs on. The procedure is as follows:

1. Set the entry point address of the interrupt routine in bits 14 through 00 of register A.
2. Set the channel and equipment interrupt number (octal) in the Q register. Bits 02 through 00 are the channel number and bits 05 through 03 are the equipment (interrupt line) number.
3. Return jump to SETCHV50.

After the return from SETCHV50, the priority program can set the interrupt with an SEL instruction. The priority program must perform all the functions at the real-time channel (connects, selects, reads, writes, etc.) as the system has no access to the channel for use by CIO and CIC.

Interrupts may be set on all eight or any combination of the eight interrupt lines on the real-time channel. Each interrupt to be set requires a separate call to SETCHV50. All subsequent calls to SETCHV50 for the same channel and equipment number replace the old interrupt processing address with a new address supplied in register A. Setting the contents of A equal to zero and the channel and equipment number in Q clears the interrupt selection.

When a real-time interrupt occurs, CIC clears the interrupt condition, saves the contents of the registers being used by the interrupted program, disables the interrupt system with a DINT, and then gives control to the interrupt processing routine. The register file registers are not changed or saved by CIC, and therefore should not be used in the interrupt processing routine, unless site programming standards reserve some of the registers for real-time programs.

Priority 2 real-time interrupts do not interrupt CIO processing and are stacked for execution. Priority 2 real-time may interrupt itself or lower priority processing.

Priority 1 real-time processing may interrupt all priorities of processing.

Priority 1 real-time and priority 2 real-time processing should not share a common channel, because the termination of either of them would terminate both.

Calling system executive routines from an interrupt routine will enable the interrupt system. This allows priority 1 real-time processing routines to be reentered by subsequent priority 1 real-time interrupts. Some of the system routines enable the interrupt system when they are entered, and some when they exit.

Routine	Enable Interrupts
BJSV50	Exit
CIO	Entry
RAARV50	Entry
RLSMV50	Exit
SCARV50	Entry
SETCHV50	Exit
SETFTV50	Exit
SETCLV50	Exit
SETMIV50	Exit
WHATKIND	Exit

Priority 1 real-time interrupt routines that call system executive routines should be written as reentrant routines. Non-reentrant priority 1 real-time routines may use executive routines that enable interrupts at exit if the real-time routine executes a DINT instruction immediately upon return.

Before allowing itself to be reentered, the priority 1 interrupt routine must resave the contents of the registers which were initially saved when the first real-time interrupt occurred.

Each time the priority 1 interrupt processing routine is entered or reentered, the following information is supplied in the A, Q, and B1 registers.

- A First-word address of the register save area that the registers were saved in (bits 14 through 00)
- Q Number of words in the registers save area (bits 05 through 00, octal)
- B1 Mode of the program that was interrupted
  - 1 Batch Program
  - 2 Priority 4 program
  - 3 Priority 3 program
  - 5 Priority 2 program
  - 6 Priority 1 program

If there is a possibility of a subsequent real-time priority 1 interrupt occurring, the priority 1 interrupt routine must copy the contents of the register save area into another save area before executing the EINT instruction. Then, if the interrupt routine is interrupted and reentered, the initial register save area must be restored upon returning to the initial interrupt processing. If multiple reentries occur, multiple register saves must be made.

Any fault interrupts selected for the priority partition are ignored while in the real-time portion of the priority program.

A DINT instruction should be executed before restoring a register save table to prevent the restore operation from being interrupted. Interrupting a restore operation may result in a partial restore; a resulting loss of data.

When a priority 1 interrupt subroutine is reentered, the subsequent interrupt can be flagged for later processing. Then, an exit to CIC through the routine entry point returns control to the initial interrupt processing. The subsequent interrupts can be processed upon completion of the initial interrupt processing. An alternative would be to complete the subsequent interrupt processing before exiting and returning to the initial interrupt processing.

The priority 1 and 2 real-time interrupt processing routines should use CIO to execute I/O and control functions on all nonreal-time channels. Otherwise, there may be a conflict in usage from which both CIO and the interrupt processing routine cannot recover (that is, one routine clears a channel while another is reading from the same channel).

Priority 1 real-time interrupts are the only ones that can interrupt CIO. Since CIO is not reentrant, priority 1 real-time interrupt processing routines should check for a CIO busy flag at location IOBLKV50 before calling CIO. If IOBLKV50 is zero, CIO is not busy and may be called. If the IOBLKV50 is not zero, CIO was interrupted. In such cases, the CIO call must be made from a nonreal-time routine. For example, a clock interrupt may be selected to make the CIO call and then an exit taken to allow CIO to complete processing. The clock interrupt will be stacked until CIO completes processing.

Priority 2 real-time interrupt processing routines will not run if the dedicated channel contains a non-standard I/O unit which generates interrupts that cannot be cleared by CIC with an INCL instruction. Only priority 1 real-time processing routines can be used. The priority 1 real-time routines must be written reentrantly and an I/O function code must be issued in the real-time routine to clear the interrupt.

Before a priority 1 or 2 program with a real-time interrupt processing routine terminates, all its real-time interrupt selections should be cleared. If a real-time interrupt occurs after the priority program has terminated, CIC issues a channel clear instruction (IOCL) which clears all interrupt selections on the real-time channel, including those which may have been selected by another priority program that is still active.

Table 18-2 illustrates the hierarchy/mode processing of the central interrupt control (CIC) for logical interrupts. Only a hierarchy of three may interrupt itself. In all other cases, a program does not gain control unless its mode is greater than the mode currently in execution. If the mode is equal, the hierarchy must be greater than the current hierarchy.

A program in execution with a hierarchy of 3 is considered real-time. It may not make CIO calls until it checks the IOBLOCK for availability. A hierarchy of 7 indicates that a priority AP processor gained control to process a batch I/O request.

TABLE 18-2. CENTRAL INTERRUPT CONTROL

Hierarchy	Mode	Description
3	7	P1 real-time (true mode = 6)
2	7	System I/O or CIO in XIOR
1	7	System I/O or CIO in driver initialization processing
0	7	System
3	6	P2 real-time (true mode = 5)
2	6	P1 fault-manual interrupt subroutines
1	6	P1 I/O-clock interrupt subroutines
0	6	P1 initialization
2	5	P2 fault-manual interrupt subroutines
1	5	P2 I/O-clock interrupt subroutines
0	5	P2 initialization
0	4	Illegal write processor
2	3	P3 fault-manual interrupt subroutines
1	3	P3 I/O-clock interrupt subroutines
0	3	P3 initialization
2	2	P4 fault-manual interrupt subroutines
1	2	P4 I/O-clock interrupt subroutines
0	2	P4 initialization
2	1	FG fault-manual interrupt subroutines
1	1	FG I/O-clock interrupt subroutines
0	1	Batch running
0	0	System or batch
7	X	AP processing batch request No associated RSA table

CLOCK INTERRUPTS

Each priority program and the batch program may select one clock interrupt through the use of the SETCLV50 routine. If more than one interrupt selection is active at the same time, the selections are stacked in a clock interrupt table.

To select a clock interrupt, place an interrupt processing routine address and a time interval in the A and Q registers and then do a return jump to SETCLV50 as follows:

1. Set bits 14 through 00 of register A equal to the entry point address of the interrupt processing routine.
2. Set bits 22 through 00 of register Q equal to the interrupt time interval in milliseconds (octal). Bit 23 of Q must be zero.
3. Do a return jump to SETCLV50.

Upon return from SETCLV50, register A contains zero if the interrupt was set, and negative one if the request was rejected. The SETCLV50 request is rejected if any of the following conditions exist.

1. The interrupt was previously set and has not been cleared.
2. An interrupt processing routine address was not supplied in register A.
3. A time interval larger than 37777777_8 (2 hours 20 minutes) was used.

The minimum clock interrupt time that can be selected depends upon the program mode. Selecting a smaller time interval results in the minimum time interval.

Program Mode	Minimum Time
Batch	63 milliseconds
Priority 4	48 milliseconds
Priority 3	32 milliseconds
Priority 2	10 milliseconds
Priority 1	3 milliseconds

The clock interrupt may be cleared any time up to the time the interrupt occurs. This can be done by setting register Q to zero and making a return jump to SETCLV50. Register A must contain the same address that was used to set the interrupt. After clearing the interrupt, SETCLV50 returns control to the user program with register A equal to zero. If register A does not equal zero, the interrupt was not cleared. The wrong address was used in register A.

After clearing the clock interrupt, the interrupt can be reset with a different interrupt address and/or a new time interval. This may be done by setting a new time in the Q register, setting an interrupt address in the A register, and executing another return jump to SETCLV50.

When a clock interrupt is selected, SETCLV50 adds the current clock time to the interrupt time interval in register Q and then writes the sum in the clock interrupt mask register (register 32) and in the clock interrupt table. If more than one clock interrupt is active, the interrupt with the shortest time interval is written in the mask register to set the interrupt.

When a clock interrupt occurs, CIC clears the interrupt condition and clears the interrupt selection so the same clock interrupt cannot reoccur during the interrupt processing. If interrupt selections were stacked, CIC searches the clock interrupt table for the next shortest time interval, and places the new time value in the clock mask register (register 32) to set the next clock interrupt. Then CIC enters the clock interrupt processing routine.

Due to higher level interrupt processing, a clock interrupt processing routine may be entered late. If the interrupt time is critical, the user may check for a late interrupt by reading the clock register (register 22_8) just before setting the interrupt, and again when the interrupt routine is entered.

INPUT/OUTPUT EQUIPMENT INTERRUPTS

The user may select input/output equipment interrupts in calls to CIO in order to interrupt his program upon normal or abnormal completion of an I/O function. The user must specify the type of interrupt and the first word address of an interrupt processing routine in the return jump sequence to CIO (refer to section 11).

The interrupt processing routine may be used to make additional CIO calls on the same (or different) unit, to select other interrupts, to process errors, to check status, to communicate with the operator, etc. If another I/O interrupt is selected in the interrupt processing routine, and the interrupt occurs, the interrupt is stacked until the current interrupt processing has completed. However, higher level I/O interrupts interrupt the current interrupt processing routine.

Before giving control to an I/O interrupt routine, CIC places the I/O unit status word in the A and Q registers. This facilitates checking for normal or abnormal termination within the interrupt routine. The return from the interrupt processing routine must always be back to CIC through the routine entry point.

MANUAL INTERRUPTS

Manual interrupts have the second highest priority of all user interrupts. One manual interrupt can be selected for each user program in core (priorities 1 through 4 and batch). Manual interrupts allow the operator to manually interrupt any program in execution in order to type a message to any program in core. The program that the message is directed to is given control to process the message.

To send a message to a program, the operator presses the MANUAL INTERRUPT switch. He selects which program the message is directed to (priorities 1 through 4 or batch program) by use of a prefix attached to the message. The prefixes are as follows:

Prefix	Program
/FG,(message)	Batch
/P4,(message)	Priority 4
/P3,(message)	Priority 3
/P2,(message)	Priority 2
/P1,(message)	Priority 1

The SETMIV50 routine is used to select manual interrupts as follows:

1. Place the entry point address of the message processing routine in register A, and place number of words (bits 23 through 15) in user buffer and starting word address of user buffer (bits 14 through 0) in register Q.
2. Do a return jump to SETMIV50.

The SETMIV50 routine sets the interrupt and places the entry point of the message processing routine in a table in accordance with the level of the program that requested it. There is one entry in the table for each program level (priorities 1 through 4 and batch).

The prefix of the operator message is matched with the table entry to select which message processor routine is given control.

Subsequent calls may be made to SETMIV50 to select a new message processor routine or to clear the interrupt. To select a new message processor, do a return jump to SETMIV50 with the entry point address of the new message processor in registers A and Q. To clear the selection, do a return jump to SETMIV50 with zeros in register A. Upon return from all SETMIV50 calls, registers A and Q contain previous setting of manual interrupt processor, if there was one.

When the manual interrupt is activated by an operator message, CIC transfers operator message to user's buffer before giving user control. Then CIC gives control to the user's message processor routine.

The user's buffer is not initialized before writing each new message. The number of characters (octal) in a message is passed to the message processor routine in the lower six bits of register A when the processor is entered.

CIO cannot be used or called in the message processing routine; therefore, all processing should be as brief as possible. Processing should be limited to routines such as setting flags for processing the data later, selecting other interrupts, activating a priority program, etc.

Instructions that can cause an internal fault condition should be avoided in the manual interrupt routine. An internal fault interrupt selected in any routine of the same mode (batch, P1 through P4) as the manual interrupt can cause the fault interrupt to occur in the manual interrupt routine. A fault interrupt occurring within a manual interrupt subroutine aborts the job.

INTERNAL FAULT INTERRUPTS

Batch and priority programs may select any one or any combination of the following four internal program fault interrupts.

ARITHMETIC OVERFLOW FAULT

The arithmetic overflow fault is set when the capacity of the adder is exceeded. Its capacity, including sign, is 24 or 48 bits for 24-bit precision and 48-bit precision, respectively.

DIVIDE FAULT

The divide fault sets if a quotient, including sign, exceeds 24 or 48 bits for 24-bit precision and 48-bit precision, respectively. Therefore, attempts to divide by too small a number, including positive and negative zero, result in a divide fault.

A divide fault also occurs when a floating point divisor is either equal to zero or not in floating point format. The results in the A, Q, and E registers are insignificant if a fault occurs. A divide fault can be correctly sensed only after the current instruction has been executed.

EXPONENT OVER/UNDERFLOW FAULT

During all floating point arithmetic operations, exponential overflow occurs if the exponent exceeds $+1777_8$ or is less than -1777_8. The fault is also set if the SFPF instruction is executed.

BCD FAULT

In a 3100, 3150, or 3200 computer system, a BCD fault is generated by the BCD module if:

1. The lower four bits of any character (except the sign character) exceed 11_8. The characters are tested only during LDE, ADE, and SBE instructions.
2. The upper two bits of any character (except the sign character) do not equal zero.
3. An attempt is made to load register D with 15_8, 16_8, or 17_8.

In a 3100 through 3500 computer system with a BDP module, a BCD fault is generated if:

1. The lower four bits of any character in field A, except the sign character, exceed 11_8 during the numeric character operation.
2. The lower four bits of the sign character in field A exceed 12_8 during a numeric character operation.
3. The upper two bits of any character in field A, except the sign character, do not equal 00 during a numeric character operation.
4. An arithmetic carry out of the highest order character of field C occurs during an ADM or SBM instruction.
5. Field length $l_s > l_r$ for an ADM or SBM instruction.
6. Field length $l_s \neq l_r$ for a FRMT instruction, including provision for insertion characters.
7. A carry occurs out of the 14th character position during a CVBD instruction.
8. A field of more than 14 BCD characters is specified during a CVDB instruction.
9. Bits 05 and 06 of an ASCII character are both 1's or both 0's during the execution of an ATD instruction.

The BCD fault may also be set by executing the SBCD instruction.

INTERNAL FAULT PROCESSING

If an internal fault interrupt is selected, a routine to process the interrupt must be included in the program. The internal fault interrupt may be selected by a return jump to the SETFTV50 routine as follows:

1. Enter the address of the interrupt processing routine in bits 14 through 00 of register A.
2. Enter an interrupt mask code in bits 03 through 00 of register Q.
3. Execute the following instruction.

```
RTJ     SETFTV50
```

On return to the next instruction, register A is zero if the interrupt was set, and register A is negative one if the request was rejected. Selecting more than one interrupt processing routine in the same program or incorrect selection causes a reject.

The interrupt mask code in register Q selects one or any combination of the internal fault interrupts. The interrupt mask codes are as follows:

Code	Interrupt
01	Arithmetic Fault
02	Divide Fault
04	Exponent Overflow
10	BCD Fault

Only one internal fault interrupt processing routine can be used in the same program. To select more than one fault, use the sum of the codes for the interrupt wanted. For example, to select an interrupt upon divide fault and an interrupt upon exponent overflow, enter 06_8 (2+4) in register Q. The remainder of the Q register must be filled with zeros.

Each subsequent call to SETFTV50 clears the previous call. Setting a new value in register Q and setting the same interrupt address in register A selects the new code that is in register Q. A zero in register A clears the interrupt selection.

If a fault interrupt occurs and more than one fault interrupt is selected, the interrupt processing routine must determine which fault caused the interrupt before the fault can be processed. Therefore, before giving control to the interrupt routine, CIC always places an interrupt code in register Q to indicate which fault activated the interrupt, and the address of the instruction that caused the interrupt in register A. The contents of the A and Q registers may not be valid if the interrupt is not processed immediately.

Interrupt Code in Q	Fault
0111_8	Arithmetic overflow
0112_8	Divide fault
0113_8	Exponent overflow
0114_8	BCD Fault

Care should be taken not to cause an internal fault interrupt within an internal fault interrupt routine. This causes the same interrupt to loop on itself.

ILLEGAL WRITE INTERRUPTS

An illegal write occurs only in systems using the memory protect feature and only when the system is operating in the program state. This interrupt occurs whenever a batch or priority 3 or 4 program attempts to write in or alter the contents of any address in the protected memory area, including return jumps into protected memory.

The illegal write interrupt routine is a part of the system executive. It allows return jumps only to the executive routines listed in Table 18-3. The interrupt routine terminates the program if a return jump is made to any other location in protected memory, or if an attempt is made to write in protected memory.

EXECUTIVE INTERRUPTS

In MSOS Version 5, all batch, priority 4, and priority 3 programs must use system executive routines (CIO and CIC) to perform I/O and interrupt functions. In addition, MSOS does not allow batch or priority 3 and 4 programs to write in registers 00 through 37_8 of the register file. Any attempt to perform these functions causes an executive interrupt and terminates the user program.

TABLE 18-3. EXECUTIVE ROUTINES CALLABLE BY BATCH AND PRIORITY PROGRAMS

Routine Name	Batch or Priority Use	Function
ABNORMAL	Batch, Priority	Abnormal termination of program
BJSV50	Priority	Submit a batch job from mass storage
CIO†	Batch, Priority	I/O functions on logical units and mass storage files
DINT.	Batch	Disable batch interrupts
EINT.	Batch	Enable batch interrupts
LDABSV50	Batch	Load another main program
LOADER	Batch	Load programs, data, etc., from input unit on the library
RAARV50†	Batch, Priority	I/O unit not ready recovery
RDCKF1	Batch	Load the loader
REQSUV50	Batch	Reassign system units to the priority program
RLSMV50	Priority	Release memory to the system
SCARV50†	Batch, Priority	I/O error recovery
SETCHV50	Priority 1, 2 only	Set real-time interrupts
SETFTV50	Batch, Priority	Set internal fault interrupt
SETCLV50	Batch, Priority	Set clock interrupt
SETMIV50	Batch, Priority	Set manual interrupt
WHATKIND	Batch, Priority	Determine device type and location of AUT entry for any logical I/O unit number

†To retain compatibility with MSOS V4.2, calls to MSIO, MSIO.SP, SCAR, and RAAR may be made. These calls generate a call to the associated MSOS V5.0 routine (CIO, SCARV50, or RAARV50) which performs the requested function.

The COMPASS instructions which are illegal, or legal for limited usage only in batch and priority 3 and 4 program, are listed in appendix A. Any attempt to execute one of these instructions generates an executive interrupt. If the instruction is a limited usage instruction, and is being used within its limits, the system executive executes the instruction and returns control to the next instruction in the user's program. In all other cases the system executive will terminate the job.

In an extended core system, when the loader loads a batch program in memory bank 1, the loader converts all RTJ instructions that reference executive subroutines (Table 18-2) to halt instructions that reference the subroutines.

Example:

```
RTJ  CIO        Converts to   000, CIO entry point address
```

When the batch program is executed, each halt instruction causes an executive interrupt. In the interrupt processing, EXEC checks each halt instruction for a reference to a subroutine that is legal for batch programs. If a legal address is sensed, EXEC executes the subroutine for the batch program and then returns control to the batch program. If the code does not reference a legal subroutine, EXEC aborts the job.

Note that an RTJ instruction with a symbolic address is the only method that can be used to call on executive subroutines in an extended core system. All other methods will connect to an address in bank 1.

TRAPPED INSTRUCTION INTERRUPTS

The trapped instruction interrupt occurs if an attempt is made to execute a floating point, BCD, or BDP instruction in systems without the associated hardware. The interrupt causes the system executive to switch control to an interrupt routine which executes floating point and BCD instructions with software rather than with hardware. Trapped BDP instructions will abort the job. The following is a list of the instructions which are trapped and processed by the interrupt routine.

Floating Point Instructions (hardware option for all 3000L systems)

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

BCD Instructions (hardware option only for 3100, 3150, and 3200 computer systems)

ADE	EOJ	SET
EZJ, EQ	LDE	SFE
EZJ, LT	SBE	STE

BDP Instructions (hardware option for 3100 through 3500 computer systems)

ADM	JMP, LOW	SCAN, LR, EQ
ATD	JMP, ZRO	SCAN, LR, EQ, DC
ATD, DC	LBR	SCAN, LR, NE
CMP	MVBF	SCAN, LR, NE, DC
CMP, DC	MVE	SCAN, RL, EQ
CVBD	MVE, DC	SCAN, RL, EQ, DC
CVDB	MVZF	SCAN, RL, NE
DTA	MVZS	SCAN, RL, NE, DC
DTA, DC	MVZS, DC	TST
EDIT	PAK	TSTN
FRMT	SBM	UPAK
JMP,HI	SBR	ZADM

USER INTERRUPT ROUTINES

All user interrupt routines are entered only from CIC. The return to the interrupted program must also be through CIC via the first address of the interrupt routine. CIC either returns control to the interrupted program or processes other stacked interrupts, depending upon the program and interrupt priorities.

In protected memory or extended core variant of MSOS, interrupt routines may not reside outside of the program assigned memory area (refer to memory limit table in section 8). Any attempt to store an interrupt routine outside of the program address limits causes the job to abort.

DINT. AND EINT. ROUTINES

The DINT. routine temporarily inhibits the processing of all interrupts selected in a batch program. All batch program interrupts which occur are stacked until the EINT. routine is executed. This allows the batch programmer to shut off batch interrupts without affecting priority or real-time program interrupts. For example, a DINT. function could be used to ensure an I/O buffer area was processed before the next I/O interrupt was processed.

The EINT. routine enables batch interrupt processing. All stacked batch interrupts are processed in accordance with their priority.

The DINT. and EINT. routines may be entered with a return jump to DINT. or EINT. The A, Q, and B registers either are not used or are restored by the DINT. and EINT. routines.

USE OF DINT AND EINT INSTRUCTIONS

Use of the disable interrupt system (DINT) and enable interrupt system (EINT) instructions should be avoided in programs and routines operating in the monitor state. Use of these instructions could cause unnecessary or destructive delays in processing real-time, clock, manual, or system interrupts for other programs in core. DINT and EINT instructions are illegal in program state.

ILLEGAL INTERRUPTS

The adjacent processor and search move interrupts cannot be selected or used in any program or routine operating under the MSOS operating system.

On some 3100 and 3200 systems, special hardware is used to generate an adjacent processor interrupt if an indirect addressing loop occurs. The hardware imposes a time limit on instructions using indirect addressing. Exceeding the time limit triggers the adjacent processor interrupt which terminates the program.

PRIORITY PROGRAMS 19

DESCRIPTION

Priority programs are special interrupt routines which reside in core and are periodically activated by a preselected interrupt. If a LOAD statement or a binary IDC card is used to load the program, a RUN statement must be used to initiate the program. During this initial run phase, the priority program selects an interrupt to activate itself at a subsequent time, and then exits through its primary entry point.

The first exit through the primary entry point establishes the priority program in core. The second exit through the primary entry point terminates the priority program. Figure 19-1 illustrates the use of a priority program to type the name of the batch job currently in core each time the operator presses the MANUAL INTERRUPT switch and types /P3.

Up to four priority programs may reside in core memory at the same time. The PRIORITY statement assigns a priority level, P1 through P4, to each program. P1 is the highest and P4 is the lowest. Each level can be assigned to only one program at a time.

In the protected memory variants of MSOS, priority P1 and P2 programs operate in the monitor state. These programs have unrestricted use of core memory and all of the computer instructions. † Priority P3, P4, and batch programs operate in the program state. They write only in unprotected memory and are restricted from using COMPASS I/O and interrupt control instructions (refer to appendix D). In addition, the dynamic memory protection option restricts priority P3, P4, and batch programs from writing outside their own assigned memory areas.

In the standard version of MSOS, all programs operate in the nonexecutive mode. As a result, the programs have free access to all memory and may execute all instructions. † †

LOADING PRIORITY PROGRAMS

Priority programs are loaded from the standard system input unit, from a tape or mass storage file, or from the library. All priority programs must be in the relocatable binary format. Source language cannot be entered as priority programs. Only FORTRAN or COMPASS produced relocatable binary decks are used.

When using MSOS with standard memory protection, priority P1 and P2 programs must be loaded prior to priority P3 and P4 programs. For standard MSOS or MSOS with programmable memory protection, the priority programs are loaded in any order.

† An exception is the autoload/dump area in 3100, 3150, and 3200 computers, which is protected by hardware from all programs in core.

† † Sixteen special executive mode instructions are executed as no operation instruction (refer to appendix D).

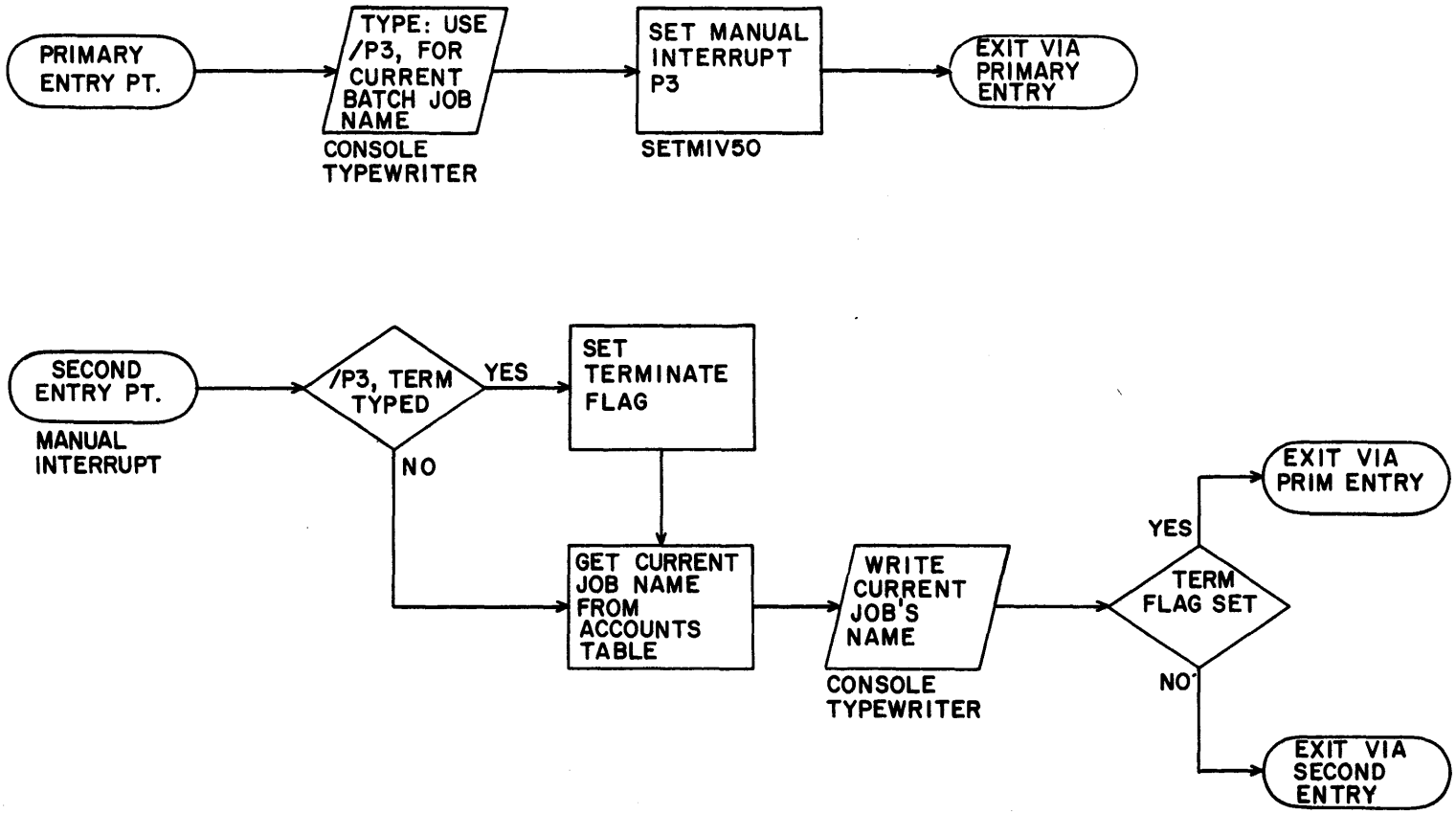


Figure 19-1. Priority Program, Initialization and Execution

All priority programs must begin with a PRIORITY statement and end with an EOJ statement. The following control statements are optional. They may be inserted after the PRIORITY statement and ahead of a LOAD or a library program name statement, or a binary program deck.

EQUIP	DUMP	ALLOCATE	CLOSE	PAUS
FMT	TRAIN	OPEN	RELEASE	
CTO	RAT	EXPAND	ABSTSK	
CTL	FET	MODIFY	LED	
AUX	OCC	REWIND	UNLOAD	

Each priority program has its own table for logical unit and file numbers. Therefore, the same file numbers can be duplicated in each priority and batch program residing in core. However, once a physical hardware unit is assigned to a program with an EQUIP statement, the unit is not available to any other program in core.

The following is an example of two priority programs: one loaded from a mass storage file, the other loaded from the card reader.

```
$PRIORITY,P3
$FET,...
$OPEN,03              Open mass storage file containing priority program deck.
$EQUIP,04=MT          Assign tape drive to priority program.
$LOAD,03              Load priority program from mass storage file.
$RUN
77
88
$EOJ
$PRIORITY,P4
(Binary deck with IDC Load priority program from card reader.
header)
$RUN
77
88
$EOJ
```

TERMINATING PRIORITY PROGRAMS

Once established in core, a priority program remains in core until it is terminated by the operator, terminates itself, or is aborted by the operating system.

OPERATOR TERMINATION

The operator may terminate a priority program at any time by pressing MANUAL INTERRUPT, typing: TERM,Pn and pressing MANUAL INTERRUPT. Pn is the priority program level (that is, P1, P2, P3, or P4).

The operator may also terminate a priority program by loading another priority program with the same priority level. When the new priority program is read from the input unit, MSOS sends a message to the operator requesting permission to terminate the current priority program and replace it with the new one. If the operator types YES and presses FINISH or just presses FINISH, the current priority program is terminated and the new priority program is loaded. If the operator types NO and presses FINISH, the new priority program is bypassed and the current one remains active.

SELF TERMINATION

A priority program may terminate itself either normally or abnormally. Normal termination results when a second exit is taken back to MSOS through ENTRY. The first exit through ENTRY is made after the priority program is initially loaded and has activated itself.

Abnormal termination should be used when an error occurs from which the priority program cannot recover. Abnormal termination is accomplished by making a return jump to ABNORMAL. The ABNORMAL routine causes MSOS to abort the priority program. Refer to section 17 for a description of abnormal termination.

SYSTEM TERMINATION

MSOS aborts priority 3 and 4 programs which attempt to write in a protected memory area or attempt to execute an illegal instruction (refer to appendix D).

LOADING NEW PRIORITY PROGRAMS

If a batch program is in core, it must complete processing and terminate itself or be terminated by the operator before any new priority programs can be loaded.

When using standard MSOS† or MSOS with programmable memory protection, all new priority programs load below the current priority programs. This expands the size of the priority program area and reduces the size of the batch program area. The area occupied by old priority programs becomes dormant core (refer to Figure 8-1).

Dormant core can be recovered only by terminating and then reloading all priority and batch programs which reside below the dormant memory.

When using MSOS with standard memory protection††, a new priority 1 or 2 program cannot be loaded below a priority 3 or 4 program. The priority 3 or 4 program must be terminated and can be reloaded after the new priority 1 or 2 program has been loaded.

RLSMV50 (RELEASE MEMORY) ROUTINE

The RLSMV50 routine provides a means of releasing the core that was used to initialize a priority program so that the core can be used by the batch or other priority programs. The initialization portion of the priority program, or all core to be released, must be at the beginning of the binary program. RLSMV50 can be used only in the initialization phase of a priority program.

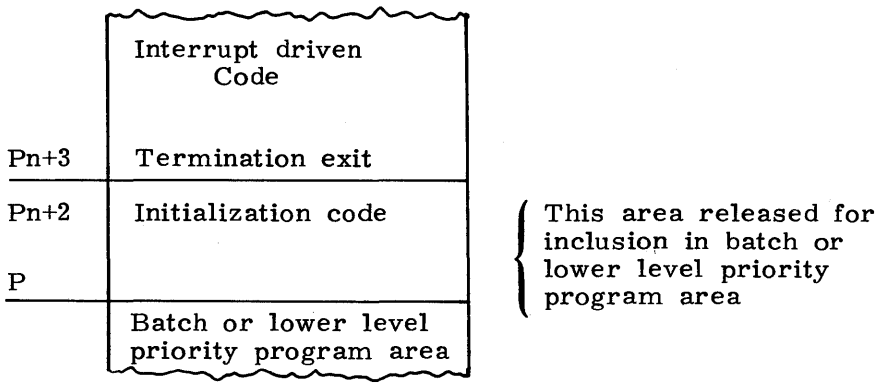
To use RLSMV50, enter the address of the first instruction of the interrupt driven section of the priority program in register A and do a return jump to RLSMV50. All addresses below the address in register A are released to the system.

Example:

Address	Instruction
P	First instruction in initialization phase
.	.
.	.
.	.
Pn	Last instruction of initialization phase

†Without memory protection.
††With memory protection switches.

```
Pn+1    LDA  Pn+2
Pn+2    RTJ  RLSMV50
Pn+3    UJP  77777  (Primary entry point address and exit)
Pn+4    UJP  P
Pn+5    First instruction of interrupt driven phase (secondary entry point
        address)
```



In the above example, after initialization of the priority program, RLSMV50 releases addresses P through Pn+2 to the system. Address Pn+3 (primary entry) is retained as the self termination exit.

In addition to releasing memory, RLSMV50 can be used to acquire dormant core or core from the batch memory area during the initialization phase. For example, I/O buffer requirements may vary, depending on input values supplied during the initialization phase. To acquire memory, the call to RLSMV50 is the same as for releasing memory. The lowest address to be used must be entered in register A.

USING SYSTEM INPUT, LIST OUTPUT, AND PUNCH UNITS

The REQSUV50 routine may be used to assign the system input, output, or punch unit(s) to a priority program for I/O functions. The priority program retains the system unit (or units) until the priority program terminates (that is, operator termination or exit through primary entry point). At termination, the system unit(s) is automatically reassigned to batch programs. A batch program will terminate if it attempts to use a system unit when the unit is assigned to a priority program.

The procedure for calling REQSUV50 is as follows:

1. Set bits 23 through 17 of register A equal to:

   60 to select the system input unit
   61 to select the system output unit
   62 to select the system punch unit

2. Set the file number to be assigned to the system unit in bits 05 through 00 of register A. The file number may range from 1 to 62.

3. Execute a return jump to REQSUV50.

The REQSUV50 call can be made only in the initialization phase of a priority program. If a call is made to REQSUV50 in the interrupt driven phase of a priority program, the program aborts.

REQSUV50 reassigns system units by zeroing out the system units FDT address in the batch FOT table, and writing the units FDT address in the priority programs FOT table. (Refer to appendix F.)

SPECIAL CODING REQUIREMENTS FOR PRIORITY PROGRAMS

Since priority programs can lock out batch and lower level priority program processing, the efficiency of batch/priority multiprogramming depends upon the design of the priority programs. Therefore, the following items should be considered when coding priority programs.

1. After a PRIORITY statement, mass storage files opened for output usage cannot be opened or used by any other batch or priority program until the priority program closes the file or is terminated. Mass storage files opened for input only can be opened for input and used by any other batch or priority programs.
2. Mass storage files can be opened, closed, modified, etc. with OCAREM macros within a priority program. The open rules in item 1 above also apply to mass storage files opened within a priority program.
3. If an OCAREM busy reject occurs when using the OCAREM macros, the priority program interrupted an OCAREM function in another program. The first OCAREM function must be completed before any other OCAREM function can be started. Set a clock interrupt and exit from the priority program in order to allow the first OCAREM function to finish.
4. File numbers 1 through 53 and 60 through 62 can be assigned as user units. In addition, files 58 and 59 may be used to communicate with the operator. Files 54 through 56 (batch scratch) and units 63 through 69 (system files) cannot be referenced in priority programs.
5. Each priority program can have only one preset or blank data area. All subprograms within the priority program can use this data area. The data area must be defined only in one subprogram.
6. Priority programs cannot communicate with each other or with a batch program.
7. No time limit can be imposed on a priority program.
8. Compilers or assemblers cannot be loaded as priority programs, and priority programs cannot be run as batch jobs.
9. Priority programs cannot use common.
10. Except for the console typewriter, priority and batch programs cannot share unit record devices such as magnetic tapes and card punches. Units reserved with an EQUIP statement are not available for batch jobs or other priority programs until they are released with a CLOSE macro within the priority program, or the priority program terminates.
11. The LOADER or calls to load library routines cannot be used within a priority program.
12. For large priority programs, overlays may be used to save core for batch programs. Permanent files must be used for the overlay segments.
13. Priority programs cannot use the register file.
14. Priority 1 and 2 programs must be thoroughly tested before running them with other batch and priority programs. These programs operate in the monitor state which allows them to write any place in core. Priority 3 and 4 programs operate in program state where they can be tested during normal system operation without the possibility of affecting the system executive or priority 1 and 2 programs currently in core.

15. The memory limit table (Table 8-2) may be referenced by a priority program to obtain its memory limits after loading. Since a priority program may be loaded at more than one priority level (that is, loaded as a priority 4 program one time and as a priority 2 program the next time), the priority level the program is loaded at is supplied in bits 02 through 00 of register B1 when the program is initially loaded during the initialization phase. The code in register B1 is as follows:

Code	Program Level
2	P4
3	P3
5	P2
6	P1

The code in register B1 may be used to determine which entries in the memory limit table contain the program memory limits.

16. If a manual interrupt is selected for processing operator messages, the priority program should obtain its priority level from register B1 and inform the operator of the program name and priority level during the initialization phase. Otherwise the operator may not know what level program to direct the messages to (refer to manual interrupts in section 18).

17. In 3100, 3150, and 3200 systems, priority programs cannot use FDP and BCD instructions if BCD hardware is present. If BCD hardware is not present, priority programs can use FDP instructions, with FDP hardware or system FDP hardware simulation using FDPBOXS. Priority programs can also use BCD instructions with system hardware simulation using BCDBOXS.

    In 3170, 3300, and 3500 systems, priority programs can use BDP instructions if the BDP hardware is present. Priority programs can use FDP instructions regardless of the presence or absence of special FDP hardware.

18. In priority 1 and 2 programs, register A must be cleared before executing instructions which use the block control hardware. Block control uses the lower three bits of register A as bank bits. These bits do not get cleared by block control before execution of the instruction in priority programs.

    The block control instructions are:

MOVE	INPW	OUTC
SRCN	INPC	
SRCE	OUTW	

19. Priority 1 and 2 programs can not use CTO or CTI instructions to read from and write on the console typewriter. These instructions use registers 23 and 33, which the system also uses to control I/O to the typewriter. System files 58 and 59 must be used for all console typewriter I/O.

20. Use of register A for I/O functions (INAC, INAW, OTAC, and OTAW) should be avoided in priority 1 and 2 programs. These instructions are not buffered. In addition, they disable the interrupt system while they are in execution which may delay real-time, channel I/O, and clock interrupt processing.

21. SNAP cards cannot be used with priority programs.

22. COBOL (ANSI or MS) source language and LISA and L-MSIO cannot be used because of L-MSIO overlay file.

DESCRIPTION

The automatic peripheral control routine (APC) provides faster I/O by spooling (buffering) the system list output, punch output, and input files on mass storage before processing them. APC transfers data between system files 60, 61, and 62, and the mass storage spooling files. APC calls the alternate processor (AP) which replaces CIO for batch job I/O on the system units (refer to section 21). AP reads and writes on the mass storage spooling files instead of system units 60, 61, and 62. When using APC, the operator may switch the system units to tape drives.

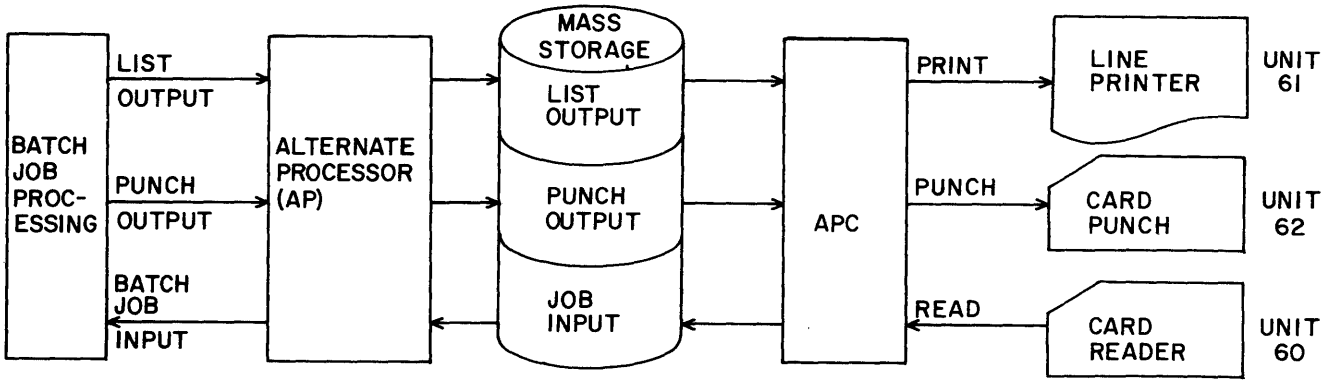


Figure 20-1. System File Spooling

APC OPTIONS

The APC routine has several assembly options and features which must be selected at system installation time. These options are listed in Table 20-1.

TABLE 20-1. APC OPTIONS AND FEATURES

Feature	Description	Options
Restart	If restart was selected and the system is reautoloaded, all jobs that were started but not completed are rerun from the beginning. All jobs that are spooled on mass storage but not started are run. All output files spooled on mass storage but not processed are processed. All output files partially processed are reprocessed from the beginning. If zero, none of the above restart capabilities will be provided.	1. Zero: Restart not selected 2. One: Restart selected
Number of printers	Number of printers that may be assigned to APC. If zero, list output is not spooled. If 2, one or two printers may be selected when APC is initialized.	1. Zero 2. One 3. Two
Number of punches	Number of punches assigned to APC. If zero, punch output is not spooled.	1. Zero 2. One
System unit reassignment	Allows operator to reassign system units such as the card reader, card punch, or printer to a tape drive and to optionally down the unit reassigned.	1. Reassignment of system units 2. No reassignment of system units
ASCII	Allows use of ASCII code and ASCII compatible I/O units.	1. ASCII supported 2. ASCII not supported
Number of spooled jobs	Maximum number of jobs that can be spooled on mass storage at one time.	Any three digit decimal number
Number of spooled list output files	Maximum number of list output files that can be spooled on mass storage at one time.	Any three digit decimal number
Number of spooled punch output files	Maximum number of punch output files that can be spooled on mass storage at one time.	Any three digit decimal number

ALLOCATING SPOOLING FILES

The mass storage spooling files must be allocated before using APC. The files may be permanently allocated, or they can be allocated prior to each use of APC and released after APC is terminated. The files are defined as follows:

File Number	Owner	Filename	Edition
01	APC	SYSTEM-INPUT	00
02	APC	SYSTEM-OUTPUT	00
03	APC	SYSTEM-PUNCH	00
04	APC	RESTART	00

APC opens the spooling files and defines the I/O buffer. Any access and privacy codes may be used to allocate the files, as APC uses the system access and modification privacy code to open and use the files.

The following are suggested file sizes to be used when allocating the spooling files. Except for the restart file, these values may be changed at any time before APC is called. However, the file characteristics must not be changed before a restart.

File† Number	Block Size in Characters		Number of Tracks	Minimum Capacity		
	841 Devices	Other Devices		841's	853/854's	813/814's
01	640	512	100	4200 cards	2400 cards	4800 cards
02	1280	1024	500	28000 lines	14000 lines	28000 lines
03	640	512	100	4200 cards	2400 cards	4800 cards
04	8960	4096	1	---	---	---

Only the INPUT file allocation is required. Allocation of the other files is dependent upon the APC options that were selected (refer to Table 20-1).

Option		File Requirements
Number of printers	1 or 2	System output file required
Number of printers	0	System output file not required
Number of punches	1	System punch file required
Number of punches	0	System punch file not required
Restart	1	Restart file required
Restart	0	Restart file not required

For maximum efficiency, each spooling file should be allocated on a different drive. None of the files should be on the device containing system scratch, OCAREM files, etc. As a minimum, the list output file must not be on a system unit.

† Assigned as the logical file number when APC opens the file. Since APC is a priority program, these numbers may be used in other batch or priority programs. (Refer to loading priority programs in section 15.)

LOADING APC

APC must be loaded as a priority program. Standard units 60, 61, and 62 need not be equipped in the priority program as these units are requested by APC during the priority program initialization.

The only other I/O equipment that APC can drive is a second printer. If a second printer is available, it must be equipped as unit 21. If the second printer is equipped, and the assembly option for a second printer was selected at system installation time, APC uses both printers for faster list output.

The following is a sample job for calling and initiating APC. It is assumed that none of the spooling files have been allocated.

```
$PRIORITY,P2
$FET,APC,SYSTEM-INPUT,...
$ALLOCATE,...
$FET,APC,SYSTEM-OUTPUT,...
$ALLOCATE,...
$FET,APC,SYSTEM-PUNCH,...
$ALLOCATE,...
$FET,APC,RESTART,...
$ALLOCATE,...
$EQUIP,21=PR
$APCV50
77
88
$EOJ
```

Once APC is initiated, all batch jobs have their input and output spooled on mass storage files. APC can be terminated by operator termination of the priority program (that is, TERM,P2 message for the above example).

SPOOLING FILE PROTECTION

Under APC, a file spooled on mass storage normally is retained on mass storage until the whole file has been processed. This allows APC to support the restart feature. The operator also can reposition and reprint or repunch output files in the event of an output unit malfunction, such as paper jam, card jam, etc.

Selecting the no protect option allows APC to release each block of a file segment as soon as the block has been read.† As soon as a block has been released, the next block of a new segment can be written on the newly released block. This allows spooling of larger files and may provide somewhat faster I/O processing, depending upon the job mix.

---

† Segments are the individual job input and output files. The input file for the first job would be the first segment, and the input file for the second job would be the second segment. When the end of the allocated area has been reached, the next block number is incremented end-around to block 1.

Files larger than the allocated spooling file size can be written by using the no protect option. When the end of the area allocated for a spooling file is reached, the next block is incremented end-around to block 1. The file is then written over released blocks until an unread block is reached. Then, the next block of the file is written each time the preceding block has been read and released, until the whole file has been written. This allows files of infinite size to be spooled.

The no protect statement must be the first statement in each job using the no protect option. The no protect statement applies to all job files that are spooled.

```
$APC,NP
```

Using the no protect statement inhibits the restart function, and if a printer or punch malfunction occurs, the amount of the list or punch file that can be reprinted or repunched may be limited. It may require rerun of the job.

SYSTEM UNIT ASSIGNMENTS

The operator may switch system input or output to a tape drive when using APC. However, the printer, punch, or reader cannot be used while a tape is being used as a substitute. If input or output is mostly on tape, tape drive(s) should be assigned as a system unit at autoload time. Then the printer, punch, or reader may be equipped and used in the user programs.

If the operator switches the system list or punch output unit to a tape drive while APC is running, an occasional internal control record may be written on the tape. If the tape is listed or punched by a program or routine running under APC, it removes the extraneous records. If the tape is listed or punched without APC, the extraneous records are printed as extra lines in the listing, or punched as extra cards in the output deck.

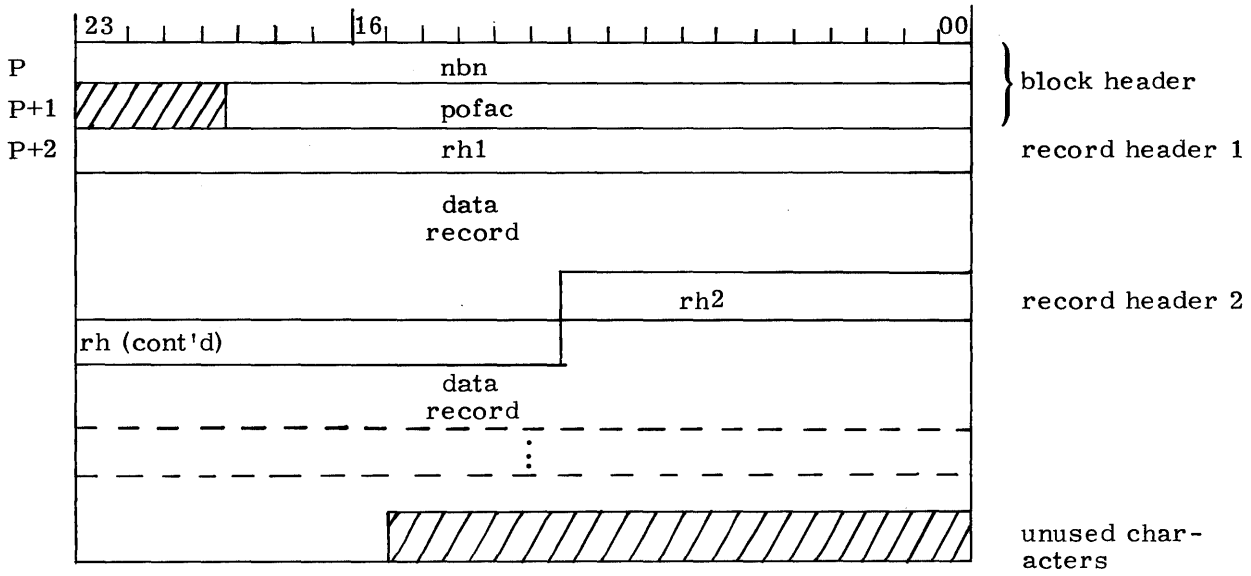
DESCRIPTION

MSOS contains an alternate processor (AP) routine which uses mass storage files for standard system input and output units 60, 61, and 62. When called, the AP is substituted for and simulates the CIO routines that drive the system units. The AP allows the user to submit batch jobs from mass storage and to spool the system list and punch output on mass storage for faster batch job processing.

AP FILE FORMAT

The AP reads and writes mass storage files blocked in AP format. With the exception of the ASCII flag in the record header, the AP file format is identical to the universal record format described in appendix E.

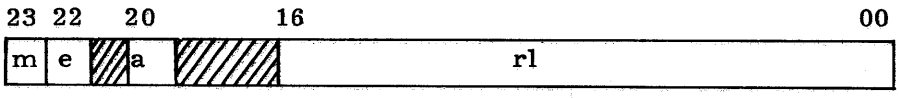
AP Blocking Format



nbn	Next block number. Usually equal to the current block plus one.
pofac	Relative position of first available unused character position in the block (that is, last character written plus 1). †
rh	Record header; rh beings at the end of block header and after the last character of the data record.

†Relative to first character of nbn which is character 0.

Record Header Format



m Mode

1 Binary
0 BCD

e EOF flag

1 EOF record
0 Data record

a ASCII flag

1 ASCII data
0 Hollerith data

rl Record length in characters, not including the record header

The end-of-file record header is written with the following values.

m	0 or 1
l	1
a	0 or 1
rl	4
data	17170000

All spooled input from system file 60 is blocked in AP format. APC blocks mass storage input files in AP format. If APC is not used, the user supplies a routine or uses the AP file processing routine to block and write the files.

AP writes system list and punch files in AP format. APC deblocks AP formatted files. If APC is not used, the user supplies a routine or uses the AP file processing routines to deblock and read the files.

SUBMITTING BATCH JOBS FROM PRIORITY PROGRAMS

For some applications, a user may find it convenient to store batch jobs on mass storage files and submit them for execution by a priority program (that is, initiation via a manual or other type of interrupt).

Each time MSOS completes a batch job, it checks for requests from priority programs for batch job processing. MSOS processes all priority submitted batch jobs before reading the next standard batch job from the system input unit.

Priority programs can submit only one batch job at a time. However, each priority program in core can have one active request for batch job processing at the same time. The priority batch jobs are processed in the order of requests rather than in accordance with the priority level of the submitted program.

Upon completion of a priority batch job, the submitting program can initiate a second batch job which has priority over the next standard batch job from the system input unit.

The procedure for submitting priority batch jobs is as follows:

1. Write job files blocked in AP file format on mass storage.
2. In a priority program, supply a job submission routine. This routine consists of a call to BJSV50 to submit the batch job. The first word address of an initialization/completion routine must be supplied in the call to BJSV50.
3. In the same priority program, supply an initialization/completion routine. The initialization portion consists of a call to IAPV50 to initialize AP, and contains an AP control table (APCT) and an AP mass storage control table (APMCT) to identify and define the batch job files. The completion portion consists of a call to EAPV50 to inform AP of job termination.

When a priority batch job is submitted, MSOS stores the initialization/completion routine address until the current batch job is completed. Then MSOS enters the initialization routine at the secondary entry point to initiate the priority batch processing. After initialization, AP reads the next batch job from mass storage rather than input unit 60.

Upon completion of the priority batch job, MSOS enters the completion routine in the priority program. This routine terminates the AP, and allows the user to submit another priority batch job or do other processing before returning to MSOS for standard batch processing.

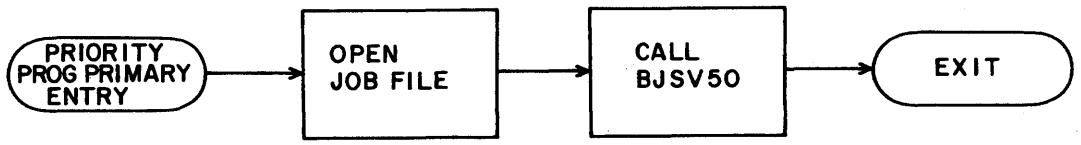
JOB SUBMISSION ROUTINE

To submit a batch job from mass storage, the user must submit a priority program to:

1. Open a mass storage file containing the job.
2. Enter the address of an initialization/completion routine in register A.
3. Return jump to BJSV50.
4. Exit.

Upon return, A equals zero if the request was processed, and A equals negative one if the request was rejected. The request is rejected if a previous call to BJSV50 from the same priority program is still outstanding.

Example:




```
SUBMIT1     UJP         **              Entry
            OPEN        (JOBF,01,I)     Logical file number must be 01
            RTJ         ABNORMAL        for job file.
            ENA         ADDR
            RTJ         BJSV50
            AZJ,LT      ABNORMAL
            UJP         SUBMIT1
ADDR        (Initialization completion routine)
  .           .
  .           .
  .           .
ADDR+n      .
JOBF        FILEID      (OWNER,JOB1,540,1,PRIV1,PRIV2)
```

INITIALIZATION/COMPLETION ROUTINE

The initialization half of the initialization/completion routine consists of providing an APCT and APMCT Table for each batch job to be submitted and calling IAPV50. When the call to AP is made (RTJ to IAPV50), register A must contain the address of the APCT Table.

Upon return from the call to AP, register A equals zero if the call was rejected. A reject occurs if AP was not called from a priority program.

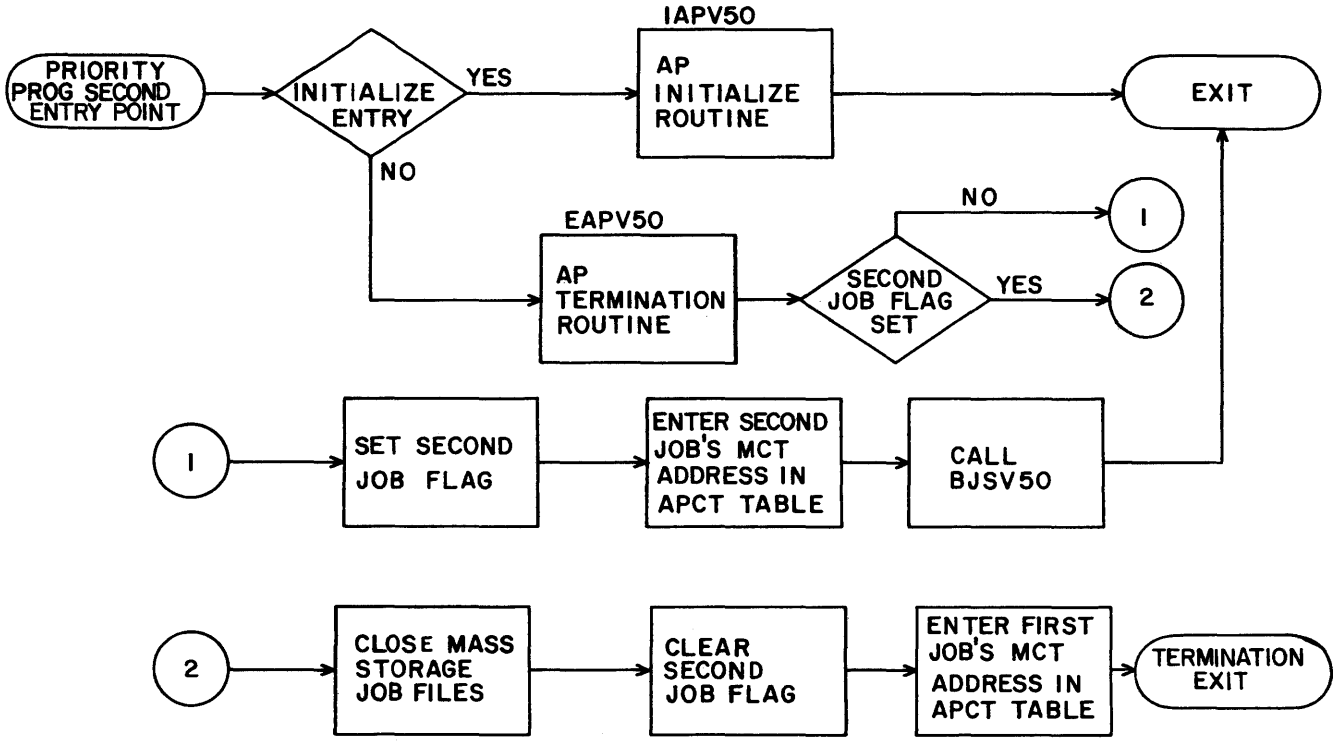
The completion half of the initialization/completion routine consists of a termination call to the AP (RTJ to EAPV50) allowing AP to clear all flags used for processing and to write partially full output buffers on mass storage. In addition, the termination routine is used to submit another batch job to MSOS or do other processing before exiting back to MSOS.

On return from the AP call, A equals zero if AP has terminated normally. Register A equals negative one if AP could not terminate due to a temporary space problem on an output spooling file. This condition does not occur unless the list or punch output is being spooled. If it occurs, a clock interrupt may be used to reenter the completion routine and submit another call to EAPV50.

When MSOS enters the initialization/completion routine to initiate priority batch processing, it sets register A equal to zero. When the AP enters the initialization/completion routine after completing a batch job, it sets register A equal to a negative one.

NOTE

The AP contains an external for loading the APC initialization routine from the library. To prevent loading APC, a dummy external entry point, APC.INIT, must be included in the initialization/completion routine. The dummy external satisfies the APC.INIT external in the AP.



```
INITIAL     UJP       **          Entry
            AZJ,LT    COMPL       Completion entry?
            ENA       APCT
            RTJ       IAPV50      Initialization call to AP.
            AZJ,LT    APC.INIT    Abnormal return?
            UJP       INITIAL     Exit
COMPL       RTJ       EAPV50      Completion call to AP.
            AZJ,LT    APC.INIT    Abnormal return?
            LDA       FLAG
            AZJ,EQ    SUBMIT2     Two jobs submitted?
            CLOSE     (1)         Close first job file.
            CLOSE     (2)         Close second job file.
            ENA       0
            STA       FLAG        Clear second job flag.
            ENA       APMCT1
            STA       APCT        Restore initial APCT value.
            UJP       SUBMIT1     Termination exit through primary entry
                                  point.
SUBMIT2     ENA       7           Start second job.
            STA       FLAG        Set flag for second job start.
            ENA       APMCT2      FWA of new APMTC Table.
            STA       APCT        Change entry in APCT Table.
            ENA       INITIAL
            RTJ       BJSV50      Submit second job.
            AZJ,LT    APC.INIT    Abnormal return?
            UJP       INITIAL     Exit through second entry point.
      FLAG  00        0
            ENTRY     APC.INIT    External to prevent loading APC.
APC.INIT    RTJ       ABNORMAL
APCT        00        APMCT1      FWA of an APMCT Table for the input file.
APCT1       00        0
APCT2       00        0
APCT3       00        0
APCT4       00        0
APCT5       00        0
```

AP CONTROL TABLE (APCT)

An APCT Table must be supplied to AP for each batch program processed. The format of the APCT table is as follows:

Location	23 15	14 01
APCT	reserved	apmct 60 addr
APCT+1		apmct 61 addr
APCT+2		apmct 62 addr
APCT+3		edit 60 addr
APCT+4		edit 61 addr
APCT+5		edit 62 addr

apmct 60 addr	First word address of an APMCT Table for the spooled job file.
apmct 61 addr	First word address of an APMCT Table for the spooled list output file. This entry must be zero unless spooled list output files are being used (refer to user spooling routines).
apmct 62 addr	First word address of an APMCT Table for the spooled punch output file. This entry must be zero unless spooled punch output files are being used (refer to user spooling routines).
edit 60 addr	First word address of a user supplied edit routine for input file records (refer to AP edit routines). Use of an edit routine is optional when submitting a batch job. Value is zero if an edit routine is not being used.
edit 61 addr	First word address of a user supplied edit routine for list output file records. This entry is zero unless spooled list output files are used.
edit 62 addr	First word address of a user supplied edit routine for punch output file records. This entry is zero unless spooled list output files are used.

If the values in the APCT are changed after initiating AP, they are ignored by AP. However, the value may be changed before initiating AP for subsequent jobs if more than one job is to be spooled.

MASS STORAGE CONTROL TABLE (APMCT)

An APMCT Table must be supplied for each mass storage file spooled or unspooled by AP. In addition, if AP file processing routines are used in a user's program, a second APMCT Table must be supplied for each spooling file for use by the AP file processing routines.

The APMCT Table defines a spooling file. Entries llbn (lower block number) and ulbn (upper block number) define the physical limits of the file. Incrementing ulbn by one gives llbn, and decrementing llbn by one gives ulbn. Although the total blocks allocated for the file are greater than the blocks defined by llbn and ulbn, the AP read and write routines restrict usage to the blocks defined by llbn and ulbn.

The beginning and end of a file segment is defined by sbn and ebn. When a file segment is larger than the physical size of the file, the first blocks must be read before the last blocks can be written. Therefore, the ebn (last block in the file) should be set to zero until the last block number of the file is known or is written.

TABLE 21-1. MASS STORAGE CONTROL

Symbolic address used by the AP	23	22	16	14	5	0
MCT.LU	f	reserved			lu	
MCT.LBN	reserved			llbn		
MCT.UBN	reserved			ulbn		
MCT.LEN	reserved			1		
MCT.SBN	reserved			sbn		
MCT.CBN	reserved			cbn		
MCT.DBN	reserved			dbn		
MCT.EBN	reserved			ebn		
MCT.FWA	reserved			fwa		
MCT.POF	reserved			pofac		
MCT.LRP	reserved			lrp		

Symbol	Bits	Description	Initial Value for Simple Submission of a Batch Job†
f	23	First time flag 1 Processing not started 0 Processing started	1
lu	5 through 0	Logical file number on the spooling file.	01 for job input file.
llbn	16 through 0	Lowest logical block number in the file.	First block written in the file (that is, block 1).
ulbn	16 through 0	Highest logical block number of the file.	Last block number written in file.
l	14 through 0	Block size in words.	Value used in FET card converted to words.
sbn	16 through 0	First block of the file segment.	Same as llbn.
cbn	16 through 0	Number of the block currently being processed.	Same as sbn.
dbn	16 through 0	Temporary upper limit for processing.	Zero. Implies no temporary limit.
ebn	16 through 0	Last block of the file segment. (Zero implies unknown.)	Last block number to be written.
fwa	14 through 0	First word address of the I/O buffer in core for this file.	Address of an input buffer for AP to use.
pofac	16 through 0	First character address of the next available space for blocking a record in the output buffer.	10_8† †
lrp	16 through 0	First character position of the last record deblocked in an input buffer.	77777775_8† † †

The dbn is used to avoid block conflicts when reading and writing concurrently on the same segment. The dbn is the block currently being written in an input file segment or the block currently being read from an output file segment. AP can read up to the dbn on an input segment and can write up to the dbn on an output segment. The dbn can be set to zero when there is no possibility of the same segment being written and read concurrently. The segment is protected if the dbn is not advanced beyond the sbn of the segment until after the segment has been fully processed. AP does not set or update the dbn in its APMCT Table. This must be done by APC or a user spooling routine.

The cbn is the block currently being read from an input segment or written on an output segment. The AP advances its cbns by one prior to each new read or write function. If a cbn becomes equal to a dbn, AP must wait for the dbn to be advanced before reading or writing the next block.

Setting the first time flag inhibits advancing the cbn. AP clears the first time flag when a block is read or written without advancing the cbn.

---

†When spooling I/O with user supplied routine, these values depend upon the user spooling routine.

† †First eight characters of each block are a two-word preamble.

† † †Indicates a new block is read before deblocking the next record. The LRP must always be set to a negative two when the APMCT is initialized.

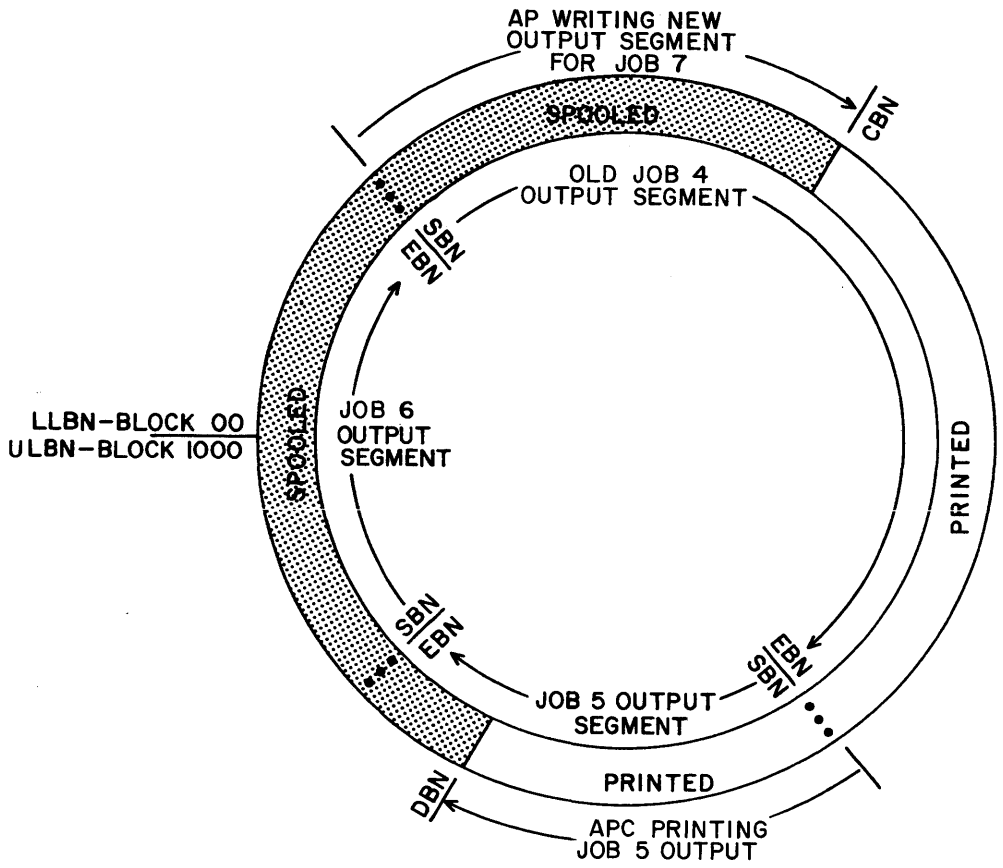


Figure 21-1. List Output Spooling†

AP uses the pofac when blocking output records for spooling. Each time a new record is blocked, the AP blocking routine advances the pofac to the first character address of the next available space in the buffer. †† The AP blocking routine compares pofac with the block size to determine when the buffer is full. When AP writes the buffer, the AP write routine writes the pofac in the block header and sets the APMCT pofac to eight.

AP uses the lrp and the pofac when deblocking input records after unspooling a block. The AP read routine sets lrp to zero each time a new block is unspooled. Then, each time a record is deblocked, AP advances lrp to the first character address of that record.

Each time a block is unspooled, the AP read routine uses the infomation in the block header to set the pofac in the APMCT to the first character address of the last record in the block. Then, during deblocking, when the lrp and the pofac become equal, the buffer is empty and the next block can be unspooled.

---

†The job 5 segment is not protected. To protect the segment, set dbn equal to the job 5 sbn.

††The character address is relative to the first character in the buffer (character zero) rather than the actual memory address.

USER SPOOLING ROUTINES

For special applications, the user may choose to supply his own mass storage spooling routines. A user spooling routine must be part of a priority program. The priority program performs the following functions.

1. Allocate one to three mass storage spooling files. A job input file is required, but the list and punch output files are optional (refer to allocation of spooling files in section 20).
2. Request use of the AP for each batch job with calls to BJSV50.
3. Supply an AP initialization/completion routine (calls to IAPV50 and EAPV50) and an APCT Table for AP.
4. Provide AP with an APMCT Table for each spooling file.
5. Provide a second APMCT Table for each spooling file if the AP file processing routines are used to write the input file and read the output files (that is, in user supplied routines).
6. Read batch jobs from the system input unit and write them on the mass storage input file in AP file format.
7. Read and deblock data from the output files and print or punch the data.
8. Update the dbns in the APMCT Tables for AP. Set the dbn in the APMCT Table for input files after each job is written.

For user spooling programs, the APMCT Table settings for AP are initialized as follows:

f	1
sbn	First block number written or read on the file.
cbn	Same as sbn.
dbn	Same as cbn for the input file. cbn-1 for list and punch output files.
ebn	Zero
lrp	77777775_8

AP FILE PROCESSING ROUTINES

The AP has six subroutines that are also available for us by the programmer for building and processing mass storage spooling files. These routines are used by the AP and may be used in user programs to spool and unspool files.

If the AP file processing routines are used in a priority 1 real-time program, the AP busy flag, APBSYV50, should be checked before entering an AP routine. A priority 1 real-time program can interrupt itself and may attempt to enter an AP routine that was previously entered and is not finished. The AP busy flag at symbolic location APBSYV50 is zero if AP is not busy. If APBSYV50 is not zero, an exit must be taken to allow the AP function to finish before reentering another AP routine.

Figure 21-2 is an example of a program that uses the AP file processing routines to spool input from the system input unit and unspool the system list output.

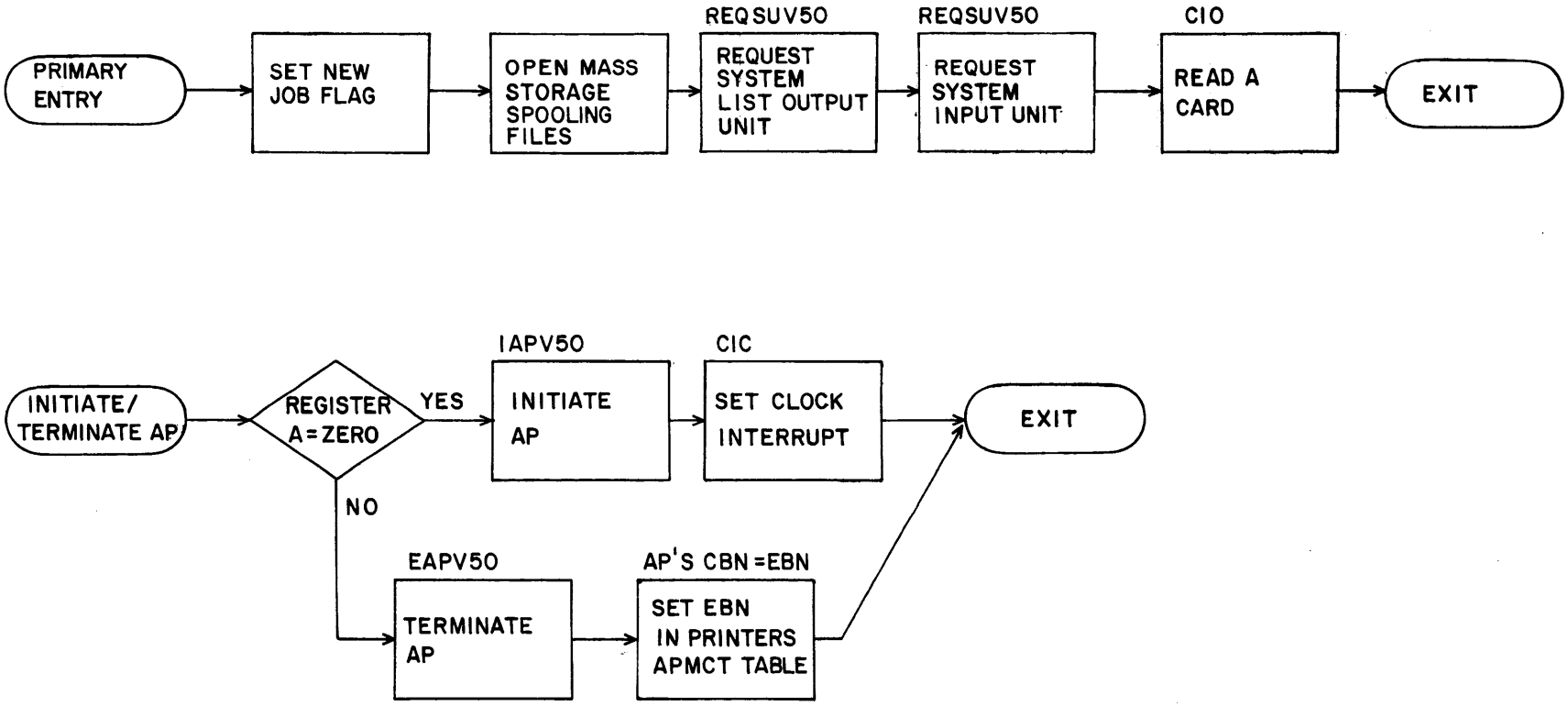


Figure 21-2. Sample User Spooling Program Using the Alternate Processor

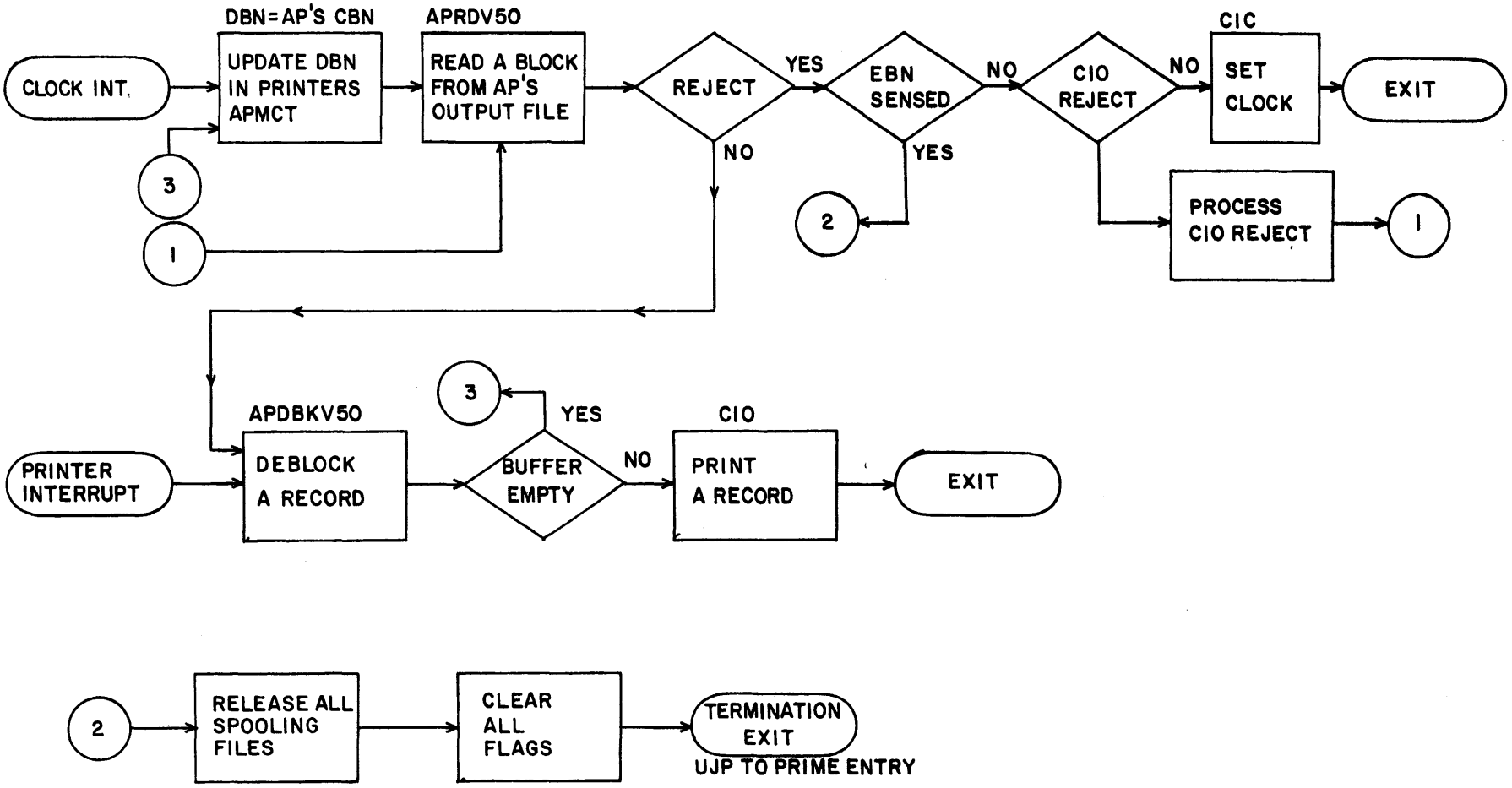


Figure 21-2. Sample User Spooling Program Using the Alternate Processor (Cont'd)

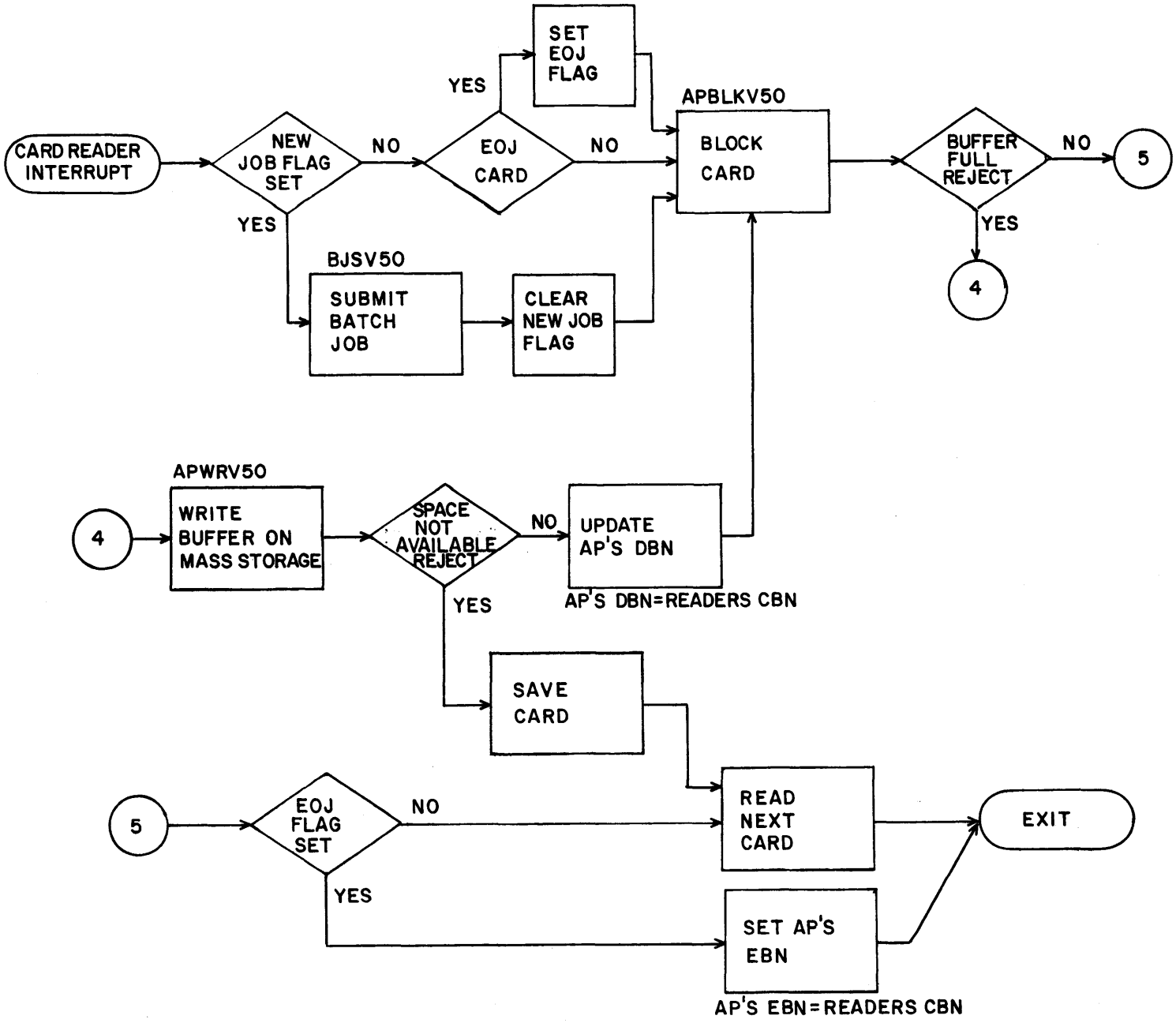


Figure 21-2. Sample User Spooling Program Using the Alternate Processor (Cont'd).

APBLKV50

APBLKV50 blocks records in CDC universal record format. The calling sequence is as follows:

(B3) First word address of an APMCT Table.

(Q) Record header

RTJ APBLKV50

On return:

(A) First character address in memory where the record, excluding the header, must be moved to block it. APBLKV50 writes the record header in the buffer before the return.

Zero if the record does not fit and the data block must be written.

APDBKV50

APDBKV50 deblocks records in CDC universal format. The calling sequence is as follows:

(B3) First word address of an APMCT Table.

RTJ APDBKV50

On return:

(A) First character address in memory where the record is stored, excluding the header.

The contents of register A is:

Zero if buffer is exhausted and the next block must be read.

Negative one if a garbled record header is detected.

Negative two if the lrp was set to that value. Used to indicate an abnormal termination of deblocking for this block. The next block should be read.

(Q) The record header.

APRDV50

The read routine advances the cbn and reads one block from the mass storage file. When ulbn is reached, APRDV50 sets the block number equal to llbn. APRDV50 rejects if ebn or dbn is sensed meaning data is not available, or if an I/O error occurred. The calling is as follows:

(B3) First word address of an APMCT Table.

RTJ APRDV50

On return:

(A) Positive zero if data was read.

Negative one if data is temporarily not available. The dbn is reached.

Negative two if permanently out of space. The ebn has then been processed.

A positive number other than zero is a CIO reject code.

NOTE

This routine requires that the first time flag be set if the cbn is equal to the sbn in order to read the sbn block.

APWRV50

APWRV50 is a write routine that increments the cbn and spools a block on mass storage. When ulbn is reached, APWRV50 sets the next block number equal to llbn. It rejects if space is not available or if CIO rejects the call. If an irrecoverable I/O error occurs, the block in error is flagged and the data written in the next sequential block. The read routine recognizes these flagged blocks and skips to the next sequential block. The calling parameters are as follows:

(B3) First word address of an APMCT Table.

RTJ APWRV50

On return:

(A) Positive zero if data was written successfully.

Negative one if temporary reject due to lack of space. The dbn was reached.

Negative two if a permanent reject due to lack of space. The ebn has been processed.

A positive number other than zero is a CIO reject code.

NOTE

This routine requires that the first time flag be set if the cbn is equal to the sbn in order to write the sbn block.

APINCV50

APINCV50 calculates the next sequential block on a spooled file. The last block plus one is set to the first block and the first block minus one is set to the last block.

(A) Block number.

(Q) One to increment current block number by one.

Negative one to decrement current block number by one.

(B3) First word address of an APMCT Table.

RTJ APINCV50

On return:

(A) Next block number.

APSPCV50

The APSPCV50 checks if a particular block can be accessed by the read or write routines. A temporary reject is returned if the block number equals the dbn and a permanent reject if it equals the ebn+1.

(B3) First word address of an APMCT Table.

(A) Block number.

RTJ APSPCV50

(A) Zero if block may be accessed.

Negative one for a temporary reject. The dbn has been reached.

Negative two for a permanent reject. The ebn has been reached.

AP EDIT ROUTINES

The user may add record edit routines to his spooling program. The address of the edit routine(s) must be inserted in the APCT Table before initializing the AP.

When an edit routine address is specified in the APCT Table, AP does a return jump to the edit routine for each record blocked or deblocked.

If an input edit routine is specified, AP enters the edit routine after each input record is deblocked, but before the record is processed by the batch job. AP places the following information in registers A and Q before entering the edit routine:

(A) First character address of the record.

(Q) The record header.

The edit routine must not increase the length of a record unless the record is moved to a user record processing area.

On return to AP, the edit routine must supply AP with the same type of information in registers A and Q. If the record was not moved from the AP buffer area, the contents of register A would be the same as when AP entered the edit routine. The record header in register Q may or may not be changed, depending upon edit routine processing of the header.

If an output edit routine is specified, AP enters the edit routine before each output record is blocked. AP places the following information in registers A, Q, and OSR before entering the output edit routine.

(A) First character address of the record.

(Q) The record header.

(OSR) The memory bank containing the record. Zero or one for extended core variants only.

The edit routine may not change the length of the record or the data in the record without first moving the record to a record processing location. On return to the AP, the edit routine must supply the same type of information in registers A, Q, and OSR that AP supplied. This information may or may not be changed depending upon the functions the edit routine performed.

AUXILIARY LIBRARY GENERATION 22

DESCRIPTION

MSOS has an auxiliary library feature allowing application users to generate and use their own libraries. COMPASS and FORTRAN users can use auxiliary libraries to store and call programs and subprograms. COBOL users can use auxiliary library to store and call only complete programs.

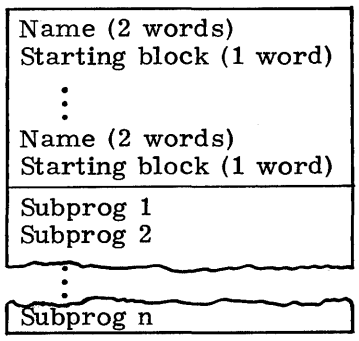
Auxiliary libraries are active only when an AUX statement is included in the job (refer to AUX statement in section 4). The AUX statement applies only to the job it is used with. Other jobs and the operating system are not affected by the AUX statement or by the auxiliary library.

When an AUX statement is used in a job, the loader searches the AUX library directory first for all programs or subprograms called for in a program or in a job. If the subprogram or program is not found in the AUX library directory, the loader searches the system library directory. Therefore, if a subprogram on the AUX library and the system library have the same name, the program on the AUX library is always loaded when the AUX statement is used. The program on the system library is always loaded if the AUX statement is not used.

The AUX library is a mass storage file which is allocated and opened the same as any other mass storage file used in a job. The file contains a directory in the first few blocks, and the remaining blocks contain relocatable binary subprograms in object code. MSOS closes the AUX library at the end of the job, the same as for the other mass storage files. The AUX library device need not be on-line when not in use and the library can be released when it is no longer needed.

Some of the advantages of AUX libraries are:

1. An AUX library is generated, used, and released without disturbing the system library.
2. An AUX library can be used to resolve conflicts in duplicate entry point names.
3. An AUX library provides easy access to special routines and subprograms. Autoloading a special library edition is not required.
4. An AUX library is easy to prepare, easy to use, and is private.



AUX Library Format

DIRECTORY — Three words for each primary entry point name. There are a maximum of 41 entries per block.

LIBRARY — Relocatable binary subprogram decks IDC through TRA are produced from COMPASS, FORTRAN, and ALGOL programs. Each deck starts on a new block.

AUX LIBRARY GENERATION

The procedure for preparing an AUX library is as follows:

1. Allocate a mass storage file for the AUX library. The block size is 960 characters per block, and the number of blocks is the sum of the blocks required for the directory and the library.
   a. For the directory, allow one block for each 41 entry point names.
   b. For the library, allow one block for each six binary cards.

   Each subprogram starts at the beginning of a new block. Therefore, extra space must be allowed between the subprograms for waste.

2. Open the file.

3. Use a PRELIB statement to load the binary relocatable subprograms onto the library. Refer to library file overlays in section 7 for loading overlay programs on an AUX library.

The most commonly used statements for auxiliary library generation are: 7/9PRELIB, 7/9FILE, and 7/9UNIT. Other specialized statements that may be used are described in the MSOS Installation Handbook.

PRELIB STATEMENT

The PRELIB statement generates the AUX library from relocatable binary subprogram card decks or files.

$PRELIB, s, AUX, u, n, I=in

- s — Specifies whether or not a library listing is generated. s equals suppress listing. If omitted, a library listing is printed.
- u — File number of a mass storage file that the library is to be stored in.
- n — The number of allocated blocks that are used for the directory.
- in — Logical unit or file number containing the input subprograms. The parameter may be omitted if the input unit is 60. All subprograms on tape files must be unblocked or blocked in universal format. All subprograms on mass storage must be blocked in universal format. †

Files 1 through 14 are reserved for use by PRELIB. Therefore, 1 through 14 may not be assigned for the u or in parameters.

FILE STATEMENT

The FILE statement terminates PRELIB input. The FILE statement must be the last input statement on the input unit (that is, unit specified with the 'in' parameter on the PRELIB statement).

7/9FILE

UNIT STATEMENT

The UNIT statement switches the PRELIB input from the initial unit specified on the PRELIB statement to a new unit. PRELIB reads input from the new unit until an EOF

† Refer to appendix E for universal format.

is sensed. PRELIB reads the next control statement or subprogram from the initial input unit. The file on the initial unit must be terminated with a 7/9FILE card.

7/9UNIT, u

u = the file number.

Example 1:

```
$JOB, 61, RTDRV
$RAT, 853/131
$FET, DEPT-4F, SPEC-DRIVES, 960, 03, DRIV, DRIV
$ALLOCATE, B101, 741231, NOSEG, 853
$OPEN, 21
7/9PRELIB, , AUX, 21, 1
(binary deck 1)
(binary deck 2)
(binary deck 3)
7/9FILE
$CLOSE, 21
$FET, DEP, T-4F, SPEC-DRIVES, 960, 03, DRIV, DRIV
$RELEASE, UNUSED
77
88
$EOJ
```

Example 2:

```
$JOB, QT, GAMES
$RAT, 853/190
$FET, SPECIAL, ODDS, 960, AR, 7/11, EVEN
$ALLOCATE, B101, 741231, NOSEG, 853
$OPEN, 21
$EQUIP, 20=MT
$FORTRAN, P=20                          Use P parameter instead of X;
(FORTRAN SOURCE DECK 1)                 X blocks the output in 960 character
    FINIS                               blocks. P gives unblocked card
77                                      image output.
88
$FORTRAN, P=20
(FORTRAN SOURCE DECK 2)
    FINIS
77
88
$FORTRAN, P=20
(FORTRAN SOURCE DECK 3)
    FINIS
77
88
$UTILITY
WEOF, 20                                Write EOF on input tape.
REWIND, 20
END
7/9PRELIB, S, AUX, 21, 1, I=60
7/9UNIT, 20                             Switch to unit 20 for input.
7/9FILE                                 Terminate PRELIB input and
                                        generate library on file 21.
```

```
$CLOSE,21
$FET,SPECIAL,ODDS,960,AR,7/11,EVEN
$RELEASE,UNUSED
$UNLOAD,20
77
88
$EOJ
```

SUBPROGRAM CALLING SYMBOLS

The rules for calling and linking subprograms from an AUX library are the same as for the system library. Each binary subprogram deck must have one or more primary entry point names. The entry point names are on an EPT card which is located between the IDC card and the first RIF cards. The symbols on these cards are the names PRELIB enters in the library directory. These symbols are the subprogram calling names. The assembler or compiler uses the information from one of the following source language statements to generate and assemble these EPT cards into the binary output decks.

COMPASS	First ENTRY statement after the IDENT card.
FORTRAN	PROGRAM name statement SUBROUTINE name statement FUNCTION name statement

EPT cards following the first RIF card indicate secondary entry points. Secondary entry points are not used until the subprogram has been loaded.

The loader loads and links all subprograms called by the subprograms it has loaded. When a subprogram references an external name, the loader first searches for the external name in the subprograms it has loaded. If a matching name is found, the loader links the locations. If some names are not found, the loader searches the library directories for matching entry point names. If the matching entry point name is in both an auxiliary library and the system library directory, the subprogram is loaded from:

1. The AUX library if an AUX statement was used in the job.
2. The system library if an AUX statement was not used.
3. The system library if an AUX statement was cleared before the program load function was initiated.
4. The system library if the subprogram containing the external reference was called from the system library. Subprograms called from the system library can not call subprograms from an AUX library.

When selecting entry point names for AUX library subprograms, care should be taken to avoid duplicating a secondary entry point name used in a subprogram on the system library. A duplicate symbol error occurs if both subprograms were loaded in the same program. For example, if subprogram DATE was loaded from an AUX library, subprogram TIME was loaded from the system library, and subprogram TIME had a secondary entry point called DATE, a conflict between the two DATEs would occur.

The system analyst can supply a listing of all primary and secondary entry points used in the system library subprograms.

PROGRAMS ON AUXILIARY LIBRARIES

Programs may be placed on an auxiliary library and called from the auxiliary library with a library program name statement. (Refer to Library Program Name Statement in section 4.)

A program consists of one or more subprograms linked to one another with external references such as XNL and EPT cards. One of the subprograms must have a transfer point address (program entry point) on a TRA card. No more than two transfer addresses are allowed in a program. The assembler or compiler uses information from one of the following source statements to generate the transfer address and place it on the applicable TRA card.

COMPASS	The END statement.
FORTRAN	PROGRAM name statement.
COBOL	Compiler automatically adds the first word address of the procedure division to the TRA card in each deck compiled. The primary entry point name used to call the program is obtained from the PROGRAM-ID statement.

The symbolic name used in the above statements is the name used to call the program off the auxiliary library.

SYSTEM I/O ERROR RECOVERY ALGORITHM 23

DESCRIPTION

The system I/O error recovery routines provide automatic I/O error processing. These routines attempt to determine the cause of the error, and they repeat the I/O function several times in different modes (such as recording modes and forward/reverse reads) in an attempt to perform the I/O function correctly.

The error recovery routines are used with most MSOS products such as COBOL and FORTRAN. COMPASS users have the option of selecting the system I/O error recovery routines or writing their own I/O error recovery routines. †

When standard error recovery is used unsuccessfully, the operator is normally notified by the system and requested to either ready the equipment or request I/O recovery be abandoned.

If the operator requests the system to abandon the I/O attempt, an irrecoverable I/O error return occurs. COMPASS users must check the status word and then perform one of the following steps if an irrecoverable error return occurs.

1. Abort the job (refer to section 17).
2. Select another I/O unit and repeat the I/O request.
3. Continue without the I/O.
4. Accept the error and continue.

System error recovery routines are provided for the following I/O units. For all other units, the user must supply his own error recovery code.

Tape drives	601 through 604, 606 through 608, 657, 659
Mass storage drives	813, 841, 853, 854, 863
Printers	501, 505, 512, 3254
Card reader	405
Card punch	415

MAGNETIC TAPE ERROR RECOVERY

Special noise records and a block checksum are used to facilitate I/O error recovery on magnetic tapes. The error recovery algorithm used varies in accordance with the type of error that occurred. The following paragraphs summarize the procedures error recovery uses to recover from magnetic tape read and write errors.

† System error recovery is selected by calling SCAR, SCARV50, or by using function codes 41 or 42 for I/O functions (refer to section 11).

NOISE RECORDS

A noise record is any data block which is 18 characters or less in length.† A system noise record (SNR) is four six-bit characters written by CIO to bracket bad spots on the tape detected during write error processing.

Noise records, other than SRNs, are considered illegal. The system rereads these records several times to ensure a read error did not occur. If a legal size record of more than 18 characters cannot be read, the system assumes the record is noise. The system then discards the record and reads the next record.

BLOCK CHECKSUM

For each block written on tape, CIO calculates a checksum word which is stored until after the next block is correctly written. This checksum is used during write recovery to ensure the last block has been correctly located before repeating the write function.

READ ERROR RECOVERY

Read errors occur as noise records in the interblock gap or as parity errors within the data block.

Noise Record Recovery

If CIO reads a block of 18 characters or less, it uses the following error recovery procedure.

1. Checks for end-of-file (EOF) mark.
2. If EOF was not sensed, error recovery checks for a SNR, indicating a bad spot on the tape. If an SNR was not read, the routine skips to step 4.
3. If an SNR was read, the error recovery routine reads the next block and returns to step 1.
4. If the noise record was not caused by an EOF or an SNR, error recovery does a reverse read and a forward read three times.†† If the block was read correctly (that is, more than 18 characters and no parity error), CIO makes a normal return to the program.

   If the recovery was obtained with a reverse read, the data is adjusted in the buffer area to simulate a forward read.††† Then a dummy forward read is made to position the tape at the next block of data.

   If a parity error occurs and a nonnoise record is read, a forward or reverse read recovery is attempted.
5. If the block was not read correctly in step 4, meaning less than 18 characters, record is noise. The next block of data is read and if a read error does not occur, CIO makes a normal return to the program.

---

†Standard noise threshold length is 18 characters. This value may be changed with the FMT statement or with an assembly option at system installation time.

††If the hardware is not capable of doing reverse reads, backspaces and forward reads are used.

†††Reverse reads are valid only if the block size is equal to or less than the number of characters requested by the read function. If the block size is greater, the first record or records may be truncated. Refer to FMT statement in section 4 for reverse read options.

If an FMT statement specifies minimum record length of less than 18 characters, the value specified is used for noise recovery.

Noise recovery is bypassed if a noise record of zero characters is specified on a FMT statement. All noise records are read as valid data blocks, if no parity error occurs.

Forward Read Recovery

When CIO reads a block of data with a forward read parity error, it initiates the following error recovery procedures.

1. Error recovery reads the block backward and forward seven times. † If programmable clipping hardware is present on the tape drive, error recovery does each read with a different read head clipping bias. If a successful read is made, CIO does a normal return to the program.
2. For seven-track tapes, error recovery switches the mode (BCD/binary) and repeats step 1 twice. If a successful read occurs, CIO does an irrecoverable error return to the program. This prevents continuous error recovery attempts due to a user error.
3. Error recovery resets the mode to the initial mode and does four reverse reads†† followed by four forward reads to move the bad block back and forth under the tape cleaner.
4. Error recovery repeats steps 1 through 3 four times.
5. Error recovery reads the bad block one more time in reverse direction. If the error still occurs, the error is irrecoverable.
6. If error is irrecoverable, CIO positions tape to the beginning of the next block and sends a message to the operator at the console typewriter.
7. If recovery was obtained with a reverse read, in any of the above steps, the data is adjusted in core. The tape is then repositioned with a dummy forward read, and control is returned to the program.

Reverse Read Recovery

The erroneous block is read in the forward direction and then reread in the reverse direction up to seven times. If the error persists, the block is reread two more times in the opposite mode (BCD/binary).

If the error persists, the original mode is restored, and the bad block is moved back and forth under the tape cleaner. The block is then reread in the reverse direction. If the error persists, the entire procedure is repeated from step one four more times.

After the fourth repetition, the block is read once again in the forward direction. If the error still persists, the error is irrecoverable. The tape is positioned to the beginning of the block with a reverse read, and the operator is notified at the console typewriter.

If recovery was obtained with an opposite direction (forward) read, the data is adjusted in core, a dummy reverse read is done to reposition the tape to the beginning of the block, and control is returned to the program.

---

†Use backspace and forward reads if the tape drive does not have reverse read capabilities. Only valid reverse reads (number of characters in the block is less than the characters in a read request) cause a successful return. For nine-track tapes, a backspace is always used to allow single track CRC correction.

††Uses backspace if reverse read hardware is not on the tape device.

WRITE ERROR RECOVERY

A parity error during a write function causes write recovery processing to occur. In write recovery, error recovery always assumes a bad or marginal area on the tape. No attempt is made to rewrite over the same bad area.

Write recovery is divided into three phases. The first phase positions the tape at the beginning of the bad block of data. The second phase erases six inches of tape and brackets the erased area with SNRs. The third phase rewrites the block and verifies a correct write.

Repositioning Phase

Error recovery uses the following procedures to reposition the tape to the end of the last good block (LGB) written.

1. The tape is repositioned to the end of LGB with control backspaces. When controlled backspace hardware is not available, error recovery uses reverse reads or backspaces.
2. Two reverse reads and one forward read to position tape at the beginning of LGB.
3. LGB read forward. If a parity check error occurs, error recovery does a reverse read and a forward read, four times. † If the error persists, it repeats the procedure from step 2, four times. If the parity error continues to persist, the routine notifies the operator of an irrecoverable position lost error.
4. A sum check is made after reading the LGB correctly. This is to ensure the correct LGB has been located. If a sum check error occurs, the routine notifies the operator of an irrecoverable checksum error.

Erase Phase

There are two types of erase phases, standard and nonstandard. The standard erase phase writes an SNR, erases 6 inches of tape and writes a second SNR. The last SNR is checked for parity. If a parity error occurs, another erase is done and another SNR is written. If the parity error persists after 50 tries, the write error is declared irrecoverable and the operator is notified.

The nonstandard erase phase erases 6 inches of tape without bracketing the erased area with SNRs. This method is used on tapes using phase recording methods and on tapes that are interchangeable with other systems. A nonstandard erase is pre-selected with a FMT statement or an assembly option at system installation time.

Rewrite and Verification Phase

After the last SNR has been written, the original data block is written. If a write error occurs, error recovery returns to the first step in the repositioning phase. An additional 6 inches of tape is erased in the erase phase.

Verification of the erased bad spot is also necessary to ensure unerased noise does not remain. This is especially important if a nonstandard erase was made. The gap verification procedure is as follows:

1. The block just written backward is read and then the tape is positioned at the beginning of the previous (LGB) block. If a record larger than eight characters was read with a parity error, recovery returns to the position phase, step 2.

---

† The routine backspaces if reverse read hardware is not present.

2. The LGB is read. If a length or checksum error occurred, a noise record greater than eight characters remains in the bad spot. That means the tape was positioned to a noise record rather than LGB. Recovery reverts to the repositioning phase, step 2.
3. A forward read is used to reread the new data block. Records of less than eight characters are discarded and another read is made. If a parity error occurs in a read of eight or more characters, the error recovery returns to step 2 of the reposition phase. If no parity error occurs, a new checksum is created and control is returned to the program.

MASS STORAGE ERROR RECOVERY

The following types of I/O errors can occur on mass storage devices.

1. Address error
2. Write check errors
3. Check word error on a read function
4. Lost data caused when memory did not input or output data fast enough
5. Channel parity error

Each time one of these errors occurs, error recovery repeats the read or write function 10 times. Then it declares the error irrecoverable and notifies the operator.

CARD READER ERROR RECOVERY

Two types of input errors can occur at the card reader, a channel parity error and a read compare error. † For both types of errors, typewriter messages request the operator to remove the card from the output tray and place it at the front of the input tray. The operator is requested to repeat this procedure each time the card is read incorrectly. If an error persists for the same card, the operator may declare the error irrecoverable.

CARD PUNCH ERROR RECOVERY

Two types of errors can occur at the card punch, a channel parity error or a punch compare error. †† The punch compare function automatically reads each card after it is punched and compares the total punches with the amount supposed to have been punched.

For both types of errors, the bad card in the read station and the card currently in the punch station are repunched. The bad cards are offset in the output tray, and the operator is notified of the offset cards at the console typewriter so he can remove them. If an error persists, error recovery declares it irrecoverable after five retries.

---

†The second read of a card did not compare with the first read.

††SCARV50 does the punch compare test only with 3446 and 3644 card punch controllers. The test cannot be done with a 3245 card punch controller.

PRINT ERROR RECOVERY

Three types of error conditions can be detected during a print function. They are:

1. Channel parity error
2. Print error (character not printed) †
3. Compare error (illegal character codes) † †

Recovery is the same for all three types of errors. If preprint spacing was used when the line was printed, error recovery repeats the write function up to four times with a no space (+) control character. After the fourth try, error recovery declares the error irrecoverable and notifies the operator.

If postprint line spacing was used, no recovery is possible. Error recovery declares the error as irrecoverable and notifies the operator.

---

† Detectable on all 512 printers and on 501 printers with optional error checking hardware.

† † Applicable only to train printers. The character code does not match a code in the image memory.

PRINTER CODES FOR 501/505 PRINTERS AND 512 PRINT TRAINS

Int. BCD Code	501/505 and 595-1 Train (501)	595-2 Train ("AN")	595-3 Train ("HN")	595-4 Train (ASCII)	Int. BCD Code	501/505 and 595-1 Train (501)	595-2 Train ("AN")	595-3 Train ("HN")	595-4 Train (ASCII)
00	0	0	0	0	40	-	-	-	-
01	1	1	1	1	41	J	J	J	J
02	2	2	2	2	42	K	K	K	K
03	3	3	3	3	43	L	L	L	L
04	4	4	4	4	44	M	M	M	M
05	5	5	5	5	45	N	N	N	N
06	6	6	6	6	46	O	O	O	O
07	7	7	7	7	47	P	P	P	P
10	8	8	8	8	50	Q	Q	Q	Q
11	9	9	9	9	51	R	R	R	R
12	:			:	52				]
13	=	#	=		53	$	$	$	$
14	≠	@	'	"	54	*	*	*	*
15	≤			<	55				@
16	%			%	56				?
17	[			,	57	>			>
20	+	+	+	+	60	Blank	Blank	Blank	Blank
21	A	A	A	A	61	/	/	/	/
22	B	B	B	B	62	S	S	S	S
23	C	C	C	C	63	T	T	T	T
24	D	D	D	D	64	U	U	U	U
25	E	E	E	E	65	V	V	V	V
26	F	F	F	F	66	W	W	W	W
27	G	G	G	G	67	X	X	X	X
30	H	H	H	H	70	Y	Y	Y	Y
31	I	I	I	I	71	Z	Z	Z	Z
32	<				72	]	&	&	!
33	.	.	.	.	73	,	,	,	,
34	)		)	)	74	(	%	(	(
35	≥				75				-†
36					76				#
37	;			;	77				&

† Underline

CONSOLE TYPEWRITER CODES

Internal BCD Codes	Typewriter Characters
00	0 (zero)
01	1
02	2
03	3
04	4
05	5
06	6
07	7
10	8
11	9
12	±
13	=
14	"
15	:
16	;
17	?
20	+
21	A
22	B
23	C
24	D
25	E
26	F
27	G
30	H
31	I
32	(Shift to lower case)
33	. (period)
34	)
35	' (apostrophe)
36	@
37	!

Internal BCD Codes	Typewriter Characters
40	- (minus)
41	J
42	K
43	L
44	M
45	N
46	O
47	P
50	Q
51	R
52	(degree)
53	$
54	*
55	#
56	%
57	(Shift to upper case)
60	(space)
61	/
62	S
63	T
64	U
65	V
66	W
67	X
70	Y
71	Z
72	&
73	, (comma)
74	(
75	(tab)
76	(backspace)
77	(carriage return)

CHARACTER SETS

BCD/Hollerith/ASCII Codes

BCD Internal Code	BCD Tape Code	ASCII Internal Code	Hollerith Printer Character	Hollerith Card Punches	ASCII Printer Character	ASCII Card Punches	BCD Internal Code	BCD Tape Code	ASCII Internal Code	Hollerith Printer Character	Hollerith Card Punches	ASCII Printer Character	ASCII Card Punches
60	20	20	BLANK	BLANK	BLANK	BLANK	27	67	47	G	12,7	G	12,7
†12	†00	3A	:	8,2	:	8,2	30	70	48	H	12,8	H	12,8
15	15	3C	≤	8,5	<	12,8,4	31	71	49	I	12,9	I	12,9
16	16	25	%	8,6	%	0,8,4	52	52	7D	v	11,0	}	11,8,2 ††††
17	17††	27	[	7,8	'	8,5	41	41	4A	J	11,1	J	11,1
75	35	5F	→	0,8,5	_	0,8,5	42	42	4B	K	11,2	K	11,2
76	36	23	≡	0,8,6	#	8,3	43	43	4C	L	11,3	L	11,3
77	37	26		0,8,7	&	12	44	44	4D	M	11,4	M	11,4
55	55	40	↑	11,8,5	@	8,4	45	45	4E	N	11,5	N	11,5
56	56	3F	↓	11,8,6	?	0,8,7	46	46	4F	O	11,6	O	11,6
57	57	3E	>	11,8,7	>	0,8,6	47	47	50	P	11,7	P	11,7
35	75	5C	≥	12,8,5	\	0,8,2	50	50	51	Q	11,8	Q	11,8
36	76	5E	¬	12,8,6	^	11,8,7	51	51	52	R	11,9	R	11,9
33	73	2E	.	12,8,3	.	12,8,3	72	32	21	I	0,8,2	!	12,8,7
34	74	29	)	12,8,4	)	11,8,5	62	22	53	S	0,2	S	0,2
37	77	3B	;	12,8,7	;	11,8,6	63	23	54	T	0,3	T	0,3
20	60	2B	+	12	+	12,8,6	64	24	55	U	0,4	U	0,4
53	53	24	$	11,8,3	$	11,8,3	65	25	56	V	0,5	V	0,5
54	54	2A	*	11,8,4	*	11,8,4	66	26	57	W	0,6	W	0,6
40	40	2D	-	11	-	11	67	27	58	X	0,7	X	0,7
61	21	2F	/	0,1	/	0,1	70	30	59	Y	0,8	Y	0,8
73	33	2C	,	0,8,3	,	0,8,3	71	31	5A	Z	0,9	Z	0,9
74	34	28	(	0,8,4	(	12,8,5	00	12	30	0	0	0	0
13	13	3D	=	8,3	=	8,6	01	01	31	1	1	1	1
14	14	22	≠	8,4	"	8,7	02	02	32	2	2	2	2
32	72	7B	<	12,0	{	12,8,2 ††††	03	03	33	3	3	3	3
21	61	41	A	12,1	A	12,1	04	04	34	4	4	4	4
22	62	42	B	12,2	B	12,2	05	05	35	5	5	5	5
23	63	43	C	12,3	C	12,3	06	06	36	6	6	6	6
24	64	44	D	12,4	D	12,4	07	07	37	7	7	7	7
25	65	45	E	12,5	E	12,5	10	10	38	8	8	8	8
26	66	46	F	12,6	F	12,6	11	11	39	9	9	9	9

† A 00 code cannot be written on tapes using even parity. Therefore, all 00 codes are automatically converted to 12's before writing them on tape. All 12 codes read from tape are automatically converted back to 00. As a result, if a 12 code (a : character) is written on tape, the : will be converted to a 00 when it is read back.

†† Filemark on tape when bracketed with inter block gaps.

††† 11,8,2 and 11,0 are equivalent

†††† 12,8,2 and 12, 0 are equivalent

6-LEVEL FLEXOWRITER AND PAPER TAPE CODES

OUTPUT CONVERSION

Internal BCD	6-Level Flexowriter	Character
00	L. C. 56	0
01	74	1
02	70	2
03	64	3
04	62	4
05	66	5
06	72	6
07	60	7
10	33	8
11	37	9
12	43	Stop
13	42	-
14	77	Illegal
15	77	Illegal
16	77	Illegal
17	77	Illegal
20	46	+
21	30	A
22	23	B
23	16	C
24	22	D
25	20	E
26	26	F
27	13	G
30	05	H
31	14	I
32	56	0
33	L. C. 42	.
34	L. C. 54	)
35	77	Illegal
36	77	Illegal
37	77	Illegal
40	L. C. 52	--
41	32	J
42	36	K
43	11	L
44	07	M
45	06	N
46	03	O
47	15	P

Internal BCD	6-Level Flexowriter	Character
50	35	Q
51	12	R
52	52	-
53	50	$
54	44	*
55	77	Illegal
56	77	Illegal
57	77	Illegal
60	04	Blank
61	L. C. 44	/
62	24	S
63	01	T
64	34	U
65	17	V
66	31	W
67	27	X
70	25	Y
71	21	Z
72	45	C/R
73	L. C. 46	,
74	54	(
75	77	Illegal
76	77	Illegal
77	77	Illegal

NOTE

77_8 results in a delete

-- means character deleted and no substitution is made internally.

INPUT CONVERSION

6-Level Flexowriter Code	Internal BCD U. C. - L. C.	Character
00	EOR	Feed
01	63 63	T
02	-- --	Color Shift
03	46 46	0
04	60 60	Space
05	30 30	H
06	45 45	N
07	44 44	M
10	60 60	Illegal
11	43 43	L
12	51 51	R
13	27 27	G
14	31 31	I
15	47 47	P
16	23 23	C
17	65 65	V
20	25 25	E
21	71 71	Z
22	24 24	D
23	22 22	B
24	62 62	S
25	70 70	Y
26	26 26	F
27	67 67	X
30	21 21	A
31	66 66	W
32	41 41	J
33	10 10	8
34	64 64	U
35	50 50	Q
36	42 42	K
37	11 11	9
40	60 60	Illegal
41	60 60	Illegal
42	13 33	=
43	-- --	STOP
44	54 61	* /
45	EOR	CR
46	20 73	+ ,
47		U.C.

6-Level Flexowriter Code	Internal BCD U. C. - L. C.	Character
50	53 53	$ $
51	75 75	TAB
52	52 40	- -
53	60 60	Illegal
54	74 34	( )
55	60 60	Illegal
56	32 00	0
57		L. C.
60	07 07	7
61	-- --	B. S.
62	04 04	4
63	60 60	Illegal
64	03 03	3
65	60 60	Illegal
66	05 05	5
67	60 60	Illegal
70	02 02	2
71	60 60	Illegal
72	06 06	6
73	60 60	Illegal
74	01 01	1
75	60 60	Illegal
76	60 60	Illegal
77	-- --	Delete

8-LEVEL (ASCII) FLEXOWRITER AND PAPER TAPE CODES

OUTPUT CONVERSION

Internal BCD	ASCII Code	ASCII Graphic
00	011 0000	0
01	011 0001	1
02	011 0010	2
03	011 0011	3
04	011 0100	4
05	011 0101	5
06	011 0110	6
07	011 0111	7
10	011 1000	8
11	011 1001	9
12	011 1010	: (COLON)
13	011 1101	= (EQUALS)
14	010 0010	" (QUOTATION MARKS)
15	011 1100	< (LESS THAN)
16	010 0101	% (PERCENT)
17	010 0111	' (APOSTROPHE)
20	010 1011	+ (PLUS)
21	100 0001	A
22	100 0010	B
23	100 0011	C
24	100 0100	D
25	100 0101	E
26	100 0110	F
27	100 0111	G
30	100 1000	H
31	100 1001	I
32	111 1011	[(OPENING BRACE)
33	010 1110	. (PERIOD)
34	010 1001	) (CLOSING PARENTHESIS)
35	101 1100	\ (REVERSE SLANT)
36	101 1110	^ (CIRCUMFLEX)
37	011 1011	; (SEMICOLON)
40	010 1101	- (HYPHEN)
41	100 1010	J
42	100 1011	K
43	100 1100	L
44	100 1101	M
45	100 1110	N
46	100 1111	O
47	101 0000	P

Internal BCD	ASCII Code	ASCII Graphic
50	101 0001	Q
51	101 0010	R
52	111 1101	] (CLOSING BRACE)
53	010 0100	$ (DOLLAR SIGN)
54	010 1010	* (ASTERISK)
55	100 0000	@ (COMMERCIAL AT)
56	011 1111	? (QUESTION MARK)
57	011 1110	> (GREATER THAN)
60	010 0000	SP (SPACE)
61	010 1111	/ (SLANT)
62	101 0011	S
63	101 0100	T
64	101 0101	U
65	101 0110	V
66	101 0111	W
67	101 1000	X
70	101 1001	Y
71	101 1010	Z
72	010 0001	! (EXCLAMATION POINT)
73	010 1100	, (COMMA)
74	010 1000	((OPENING PARENTHESIS)
75	101 1111	_ (UNDERLINE)
76	010 0011	# (NUMBER SIGN)
77	010 0110	&

INPUT CONVERSION

ASCII Code	CDC Internal BCD	ASCII Graphic	Control Character
000 0000	77		NUL
000 0001	77		SOH
000 0010	77		STX
000 0011	77		ETX
000 0100	77		EOT
000 0101	77		ENQ
000 0110	77		ACK
000 0111	77		BEL
000 1000	77		BS
000 1001	77		HT
000 1010	77		LF
000 1011	77		VT
000 1100	77		FF
000 1101	77		CR
000 1110	77		SO
000 1111	77		SI
001 0000	77		DLE
001 0001	77		DC1
001 0010	77		DC2
001 0011	77		DC3
001 0100	77		DC4
001 0101	77		NAK
001 0110	77		SYN
001 0111	77		ETB
001 1000	77		CAN
001 1001	77		EM
001 1010	77		SUB
001 1011	77		ESC
001 1100	77		FC
001 1101	77		GS
001 1110	77		RS
001 1111	77		US

ASCII Code	CDC Internal BCD	ASCII Graphic	Control Character
010 0000	60	SP	
010 0001	72	!	
010 0010	14	"	
010 0011	76	#	
010 0100	53	$	
010 0101	16	%	
010 0110	77	&	
010 0111	17	'	
010 1000	74	(	
010 1001	34	)	
010 1010	54	*	
010 1011	20	+	
010 1100	73	,	
010 1101	40	-	
010 1110	33	.	
010 1111	61	/	
011 0000	00	0	
011 0001	01	1	
011 0010	02	2	
011 0011	03	3	
011 0100	04	4	
011 0101	05	5	
011 0110	06	6	
011 0111	07	7	
011 1000	10	8	
011 1001	11	9	
011 1010	12	:	
011 1011	37	;	
011 1100	15	<	
011 1101	13	=	
011 1110	57	>	
011 1111	56	?	
100 0000	55	@	
100 0001	21	A	
100 0010	22	B	

ASCII Code	CDC Internal BCD	ASCII Graphic	Control Character
100 0011	23	C	
100 0100	24	D	
100 0101	25	E	
100 0110	26	F	
100 0111	27	G	
100 1000	30	H	
100 1001	31	I	
100 1010	41	J	
100 1011	42	K	
100 1100	43	L	
100 1101	44	M	
100 1110	45	N	
100 1111	46	O	
101 0000	47	P	
101 0001	50	Q	
101 0010	51	R	
101 0011	62	S	
101 0100	63	T	
101 0101	64	U	
101 0110	65	V	
101 0111	66	W	
101 1000	67	X	
101 1001	70	Y	
101 1010	71	Z	
101 1011	77	[	
101 1100	35	\	
101 1101	77	]	
101 1110	36	^	
101 1111	75	—	
110 0000	77	\	
110 0001	77	a	
110 0010	77	b	
110 0011	77	c	
110 0100	77	d	
110 0101	77	e	
110 0110	77	f	
110 0111	77	g	
110 1000	77	h	
110 1001	77	i	
110 1010	77	j	

ASCII Code	CDC Internal BCD	ASCII Graphic	Control Character
110 1011	77	k	
110 1100	77	l	
110 1101	77	m	
110 1110	77	n	
110 1111	77	o	
111 0000	77	p	
111 0001	77	q	
111 0010	77	r	
111 0011	77	s	
111 0100	77	t	
111 0101	77	u	
111 0110	77	v	
111 0111	77	w	
111 1000	77	x	
111 1001	77	y	
111 1010	77	z	
111 1011	32	[	
111 1100	77	:	
111 1101	52	]	
111 1110	77	~	
111 1111	77		DEL

PRINT CONTROL CHARACTERS B

In each output buffer for a 501, 505, or 512 line printer, the first character (character 0) is a print control character. This character controls line spacing and page positioning, both before and after printing a line. The control character is not printed.

Page positioning is in accordance with a prepunched format tape mounted in the printer. Selecting a level with a control character positions the page to the position punched for that level in the format tape.

Control Character	Preprinting Function	Postprinting Function
+	No spacing	No spacing
blank	Space one line	No spacing
0	Space two lines	No spacing
1	Space to top line of next page†	No spacing
2	Space to last line on page††	No spacing
3	Space to level 6	No spacing
4	Space to level 5	No spacing
5	Space to level 4	No spacing
6	Space to level 3	No spacing
7	Space to level 2	No spacing
8	Space to level 1	No spacing
9 (512/580 printer only)	Space to level 7	No spacing
- (minus)	Space three lines	No spacing
A	Space one line	Space to top line of next page†
B	Space one line	Space to last line on page††
C	Space one line	Space to level 6
D	Space one line	Space to level 5
E	Space one line	Space to level 4
F	Space one line	Space to level 3
G	Space one line	Space to level 2
H	Space one line	Space to level 1

†Top line is level 8 punch for 501/505 printers and a level 1 punch for 512/580 printers.

††Last line is level 7 punch for 501/505 printer and level 12 punch for 512/580 printers.

Control Character	Preprinting Function	Postprinting Function
I (512/580 printer only)	Space one line	Space to level 7
J (512/580 printer only)	Space one line	Space to level 8
K (512/580 printer only)	Space one line	Space to level 9
L (512/580 printer only)	Space one line	Space to level 10
Q	Set auto eject†	Printing suppressed
R	Clear auto eject†	Printing suppressed
S	For 512/580, set 6 line/inch print density; for 501/505 no action	Printing suppressed
T	For 512/580 set 8 line/inch print density; for 501/505 no action	Printing suppressed
Z (512/580 printer only)	Space to level 8	No spacing
W (512/580 printer only)	Space to level 9	No spacing
X (512/580 printer only)	Space to level 10	No spacing

†Auto eject automatically spaces to top of next page when last line on page is sensed.

HARDWARE CODES C

HARDWARE TYPE CODES

The following hardware codes are used in the system tables to identify hardware types.

Code	Hardware Type
1	Magnetic tape drive
2	Card reader
3	Printer
4	Card punch
5	Console typewriter
6	Paper tape reader
7	Paper tape punch
10	Typewriter station
11	Incremental plotter
12	Satellite controller
13	Disk drive
14	Disk file
15	Drum
16	Optical character reader
17	Seismic processor
20	Display entry station
21 through 31	Reserved for future use

DEVICE TYPE CODES

The following codes are used in the system tables to identify controller and device types.

Code	Hardware Type	Controller Number	Device Type
01	Card reader	3447, 3649	405
02	Card reader	3248	405
05	Card punch	3446, 3659	415
06	Card punch	3245	415
11	Printer	3256, 3659	501
12	Printer	3254	501
13	Printer	3256, 3659	505
14	Printer	3555	512
15	Printer	580 †	580
20	Display/entry station	3290	211
21	Incremental plotter	3293	3293
22	Paper tape stations	3691	3691
30	Magnetic tape drive	362X, 3423, 322X	607
31	Magnetic tape drive	362X, 3423, 322X	606
32	Magnetic tape drive	3423, 322X	604
33	Magnetic tape drive	3423, 322X	603
34	Magnetic tape drive	3518, 3528	659
35	Magnetic tape drive	3518, 3528	657
36	Magnetic tape drive	3127, 3128	602, 608
37	Magnetic tape drive	3127, 3128	601
50	Disk drive	3234	853
51	Disk drive	3234	854
52	Disk drive	3553	841
60	Disk file	3234	813/814
70	Drum	3637	863

† Controller is in printer.

ILLEGAL COMPASS INSTRUCTIONS D

MEMORY PROTECT VARIANT OF MSOS

The following COMPASS instructions are illegal for use in priority 3, priority 4, and batch programs, or whenever the system is operating in the program state. † Illegal use of one of these instructions causes an executive interrupt which terminates the program.

ACR	A to CR
CILO	Channel interrupt lockout
CINS	Copy internal status
CLCA	Clear channel activity
CON	Connect an I/O unit
COPY	Copy I/O channel status and contents of interrupt mask register into register A
CRA	CR to A
CTI	Set console typewriter input
CTO	Set console typewriter output
DINT	Disable interrupt control
EINT	Enable interrupt control
EXS	Sense external equipment status
HLT	Halt
IAPR	Interrupt associated processor
INAC	Input character to A
INAW	Input word to A
INCL	Clear interrupt
INPC	Character addressed input to memory
INPW	Word addressed input to memory
OTAC	Output character from A
OTAW	Output word from A
OUTC	Character addressed output from memory
OUTW	Word addressed output from memory
PRP	Priority pause
SCIM	Selectively clear interrupt mask register
SEL	Select I/O unit function
SLS	Selective stop
SSIM	Selectively set interrupt mask register
UCS	Unconditional stop
UMP	Set upper memory protect††
LMP	Set lower memory protect††

† All instructions are legal in priority 1 and priority 2 programs.
†† These instructions are available only with programmable memory protection.

Only limited use of the following instructions is allowed in priority 3, priority 4, and batch programs, or whenever the system is operating in the program state.† Use of these instructions beyond their limits results in an executive interrupt and termination of the program.

INS	Sense internal status. It cannot be used to sense channel status. It can be used to sense internal fault status when bits 11 through 08 are set.
INTS	Sense interrupt. INTS cannot be used to sense external equipment interrupt lines. It can be used to sense fault interrupts.
IOCL	Clear I/O, typewriter, or search/move. IOCL cannot be used to clear I/O channel, equipment, or the console typewriter. It can be used to clear a search/move function.
MOVE	Move n characters from field R to field S. MOVE instruction cannot be used to move data to a protected memory area.
PAUS	Pause if condition selected by pause mask is sensed. It cannot be used to sense I/O, channel, or typewriter. It can be used only to sense a search/move function.
TAM	It writes the contents of register A in register file n. TAM cannot be used to write in registers 00 through 37_8. Bits 05 through 00 must be greater than 37_8.
TIM	It writes the contents of register B_b in register file n. TIM cannot be used to write in registers 00 through 37_8. Bits 05 through 00 must be greater than 37_8.
TQM	It writes the contents of register Q in register file n. TQM cannot be used to write in registers 00 through 37_8. Bits 05 through 00 must be greater than 37.

†In priority 1 and priority 2 programs, use of these instructions is not limited.

STANDARD MSOS

The standard version of MSOS operates in the nonexecutive mode. In this mode, memory protection is disabled and all instructions are legal for all priority and batch programs. The following executive mode instructions are executed as no-operation (NOP) instructions. These instructions should be avoided in batch and priority programs because the programs do not run under all variants of MSOS.

ACI	(A) ⟶ CIR
ACR	(A) ⟶ CR
AIS	(A) ⟶ ISR
AOS	(A) ⟶ OSR
APF	(A_{00-11}) ⟶ PIF
CIA	(CIR) ⟶ A
CRA	(CR) ⟶ A
ISA	(ISR) ⟶ A
JAA	Jump address ⟶ A
OSA	(OSR) ⟶ A
PFA	(PIF) ⟶ A
RIS	Relocate to instruction state
ROS	Relocate to operand state
SBJP	Set boundary jump
SDL	Set destructive load
TMAV	Test memory available

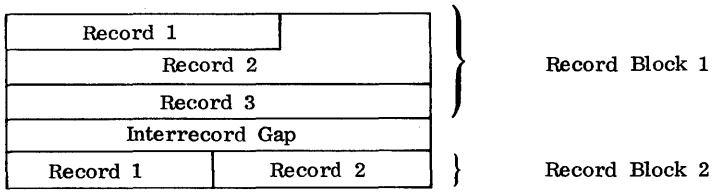
L-MSIO RECORD FORMATS E

Logical MSIO processes the following combinations of logical records and record block formats for both magnetic tape and mass storage.

VARIABLE LENGTH RECORD BLOCKS

In variable length blocks the number of records and corresponding words within each block of a file vary in numbers.

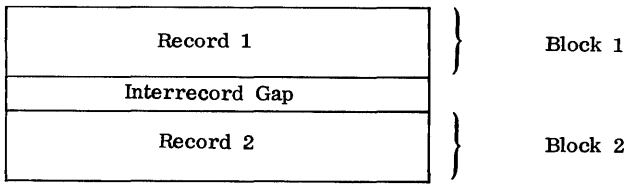
Sample:



UNBLOCKED RECORDS

In unblocked records (variable or fixed length) the record and the record block are one in the same.

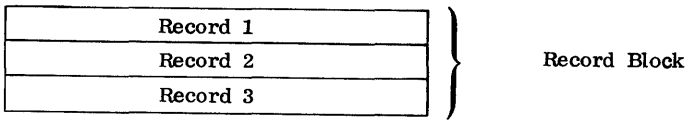
Sample:



FIXED LENGTH RECORDS

Fixed length records contain the same number of characters for each record within a record block.

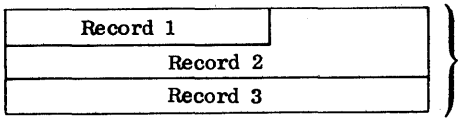
Sample:



VARIABLE LENGTH RECORDS

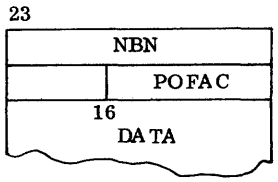
Variable length records contain an unequal number of characters for each record within the record block.

Sample:



TWO-WORD PREAMBLE

The two-word preamble used exclusively with mass storage files that indicates the next sequential block and the position of first available character within the block. The format of the two-word preamble is as follows:

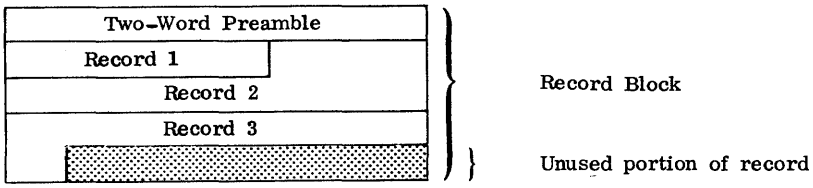


NBN - next block number to be used when file is accessed sequentially.

POFAC - position of first available (unused) data character in the block (that is, the last character written plus one). †

The two-word preamble can be used with both fixed and variable length records located on block files. If records on mass storage are unblocked the preamble is not provided.

MASS STORAGE



NOTE

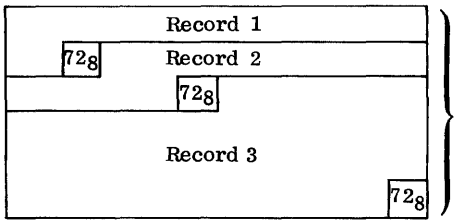
The unused portion of the record block is due to available block space not fully utilized by the existing record data on the file. When record blocks are allocated, the block size should be specified as a multiple of the standard 256 or 640 characters per sector. Therefore, to optimize use of mass storage blocks, the total number of characters for all records in a block should approximate the files block size.

† Relative to first character of the NBN which is character 0.

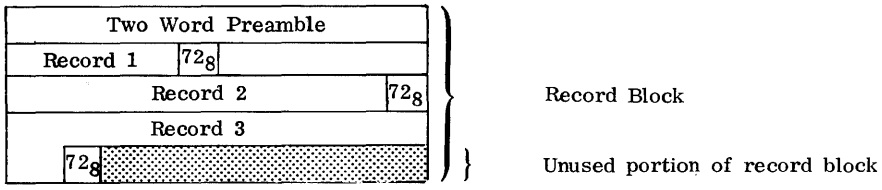
RECORD DELIMITER FORMAT

A special delimiting character terminates each record within a file. The record delimiter can be specified with the default value (72_8) used in the illustration below or the user can specify his own delimiting character. The record delimiter can be used with fixed length or variable length records on blocked files.

MAGNETIC TAPE

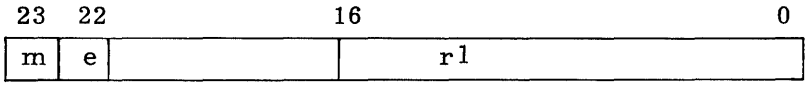


MASS STORAGE



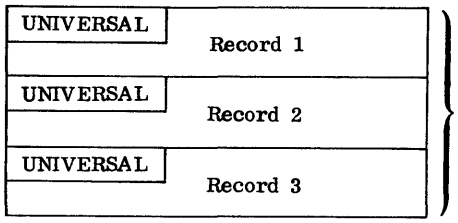
UNIVERSAL HEADER FORMAT

The universal header format consists of a 24-bit binary control field as the first word of each record that specifies the character length and record type of the record in which it appears. The universal format can be used with fixed or variable length records located on blocked files. The universal header format is as follows.

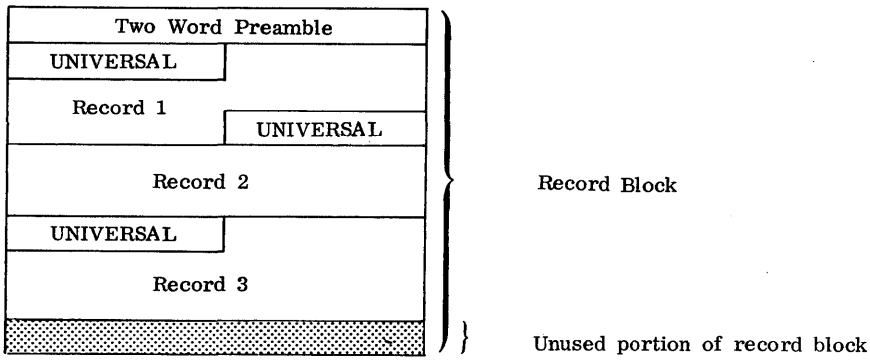


m	1	Binary
	0	BCD
e	1	End-of-file record.
	0	Data record.
rl	Character length of record, excluding the four characters of the record header.	

MAGNETIC TAPE



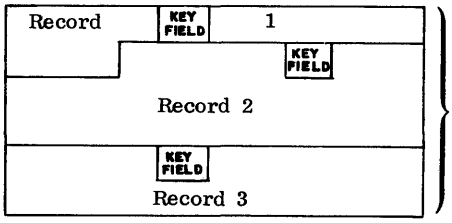
MASS STORAGE



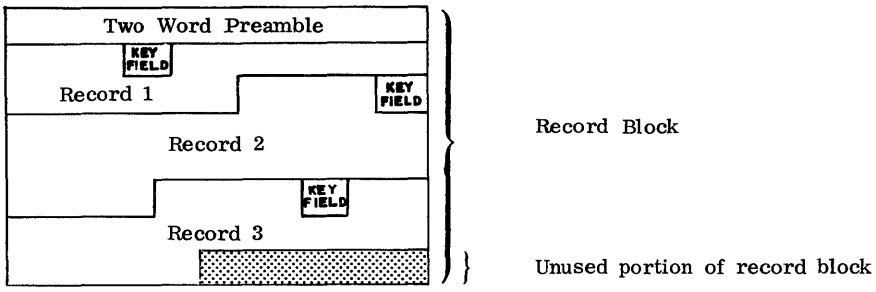
KEY FIELD FORMAT

The key field (1 through 4095 character) set in BCD format by the user indicates the length in characters of the record in which it appears. The key field can be located in the same character position relative to the first character position of each record in a file or can be located outside the records. Key fields are used only with variable length records located on blocked files.

MAGNETIC TAPE



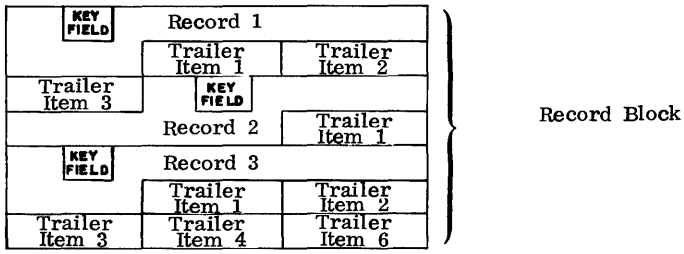
MASS STORAGE



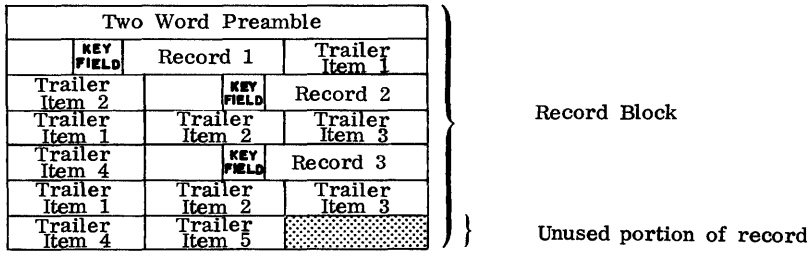
TRAILER ITEM/KEY FORMAT

There are a variable number of fixed length trailer items that can be attached to a fixed length base in logical records. If trailer items are specified a key field within the fixed length base of each record or located outside the record is set by the user in BCD format. The key is used to indicate the number of trailer items within each record.

MAGNETIC TAPE



MASS STORAGE



SYSTEM TABLES F

The MSOS executive tables are located in the executive resident area of memory. Many of these tables can be read from both batch and priority COMPASS programs to obtain information related to the current batch jobs status, I/O hardware and file status, and system operating status. Refer to executive tables in section 8 for methods of referencing these tables. The tables that can be referenced from COMPASS programs are grouped in this appendix as follows:

SYSTEM AND JOB STATUS TABLES

- Resident parameter table (RPT)
- System accounts table (ACCOUNTS)
- Memory limits table (UMEM and LMEM)

I/O HARDWARE TABLES

- File ordinal table (FOT)
- File description table (FDT)
- Available unit table (AUT)

MASS STORAGE RELATED TABLES

- Mass storage table (MST)
- Mass storage parameter table (MSPT)
- Resident allocation table (RAT)
- File environment table (FET)

The following tables cannot be referenced directly from a program, but the contents of the tables may be inspected in a core dump listing as an aid in locating program errors.

- Equipment status table (EST)
- Channel status table (CST)
- Register save table (RSA)

RESIDENT PARAMETER TABLE (RPT)

The RPT contains mass storage file parameters and the symbolic memory locations of the MST, MSPT, RAT, and AUT tables. The RPT can be referenced by indexing symbolic address RPT. All entries in the RPT are in octal and are right-justified.

RPT Entries

Location	Contents
RPT	First word address of the MSPT table
RPT+1	Number of words in the MSPT table
RPT+2	First word address of the MST table
RPT+3	Number of words in the MST table
RPT+4	First word address of the RAT table
RPT+5	Number of words in the RAT table
RPT+6	Ordinal of the AUT entry for the system device
-	Reserved for system use
-	Reserved for system use
RPT+9	Maximum mass storage file size (tracks)
RPT+10	Maximum number of segments in any mass storage file
RPT+11	Maximum number of mass storage files that can exist in the system
RPT+12	Current number of mass storage files allocated in the system
-	Reserved for system use
-	Reserved for system use
RPT+15	Number of words in the AUT table
RPT+16	RAT class. 0 specifies entries in RAT table are for non-class-R devices. 2 specifies entries in RAT table are for class-R devices.

SYSTEM ACCOUNTS TABLE

In COMPASS programs the system accounts table can be referenced to obtain the following information.

Date
Time of day
Library edition in use
Sequence number of current batch job
Time current batch job started execution
Account number for current batch job
Name of current batch job
Estimated run time for current batch job (from JOB statement)
Maximum number of lines the current batch job can print
Number of lines currently printed by the batch job
Maximum number cards that the current batch job can punch
Maximum number of cards currently punched by the batch job

Refer to system accounts table in section 16 for the format of the table. The table can be read by indexing the symbolic address ACCOUNTS.

MEMORY LIMITS TABLE

The user may reference the memory limits table to determine the absolute memory addresses in which his program was loaded. In batch programs, the user may also obtain the upper and lower addresses of the area assigned to common. The memory between the upper common address and the lowest batch program address can be used to expand the batch program into, or to load additional subprograms or routines from a library with the loader. If common is not used in a batch program, the upper and lower memory limits for common are the same.

Refer to memory limits table in section 8 for the format of the table. The table can be read by indexing the symbolic addresses UMEMV50, which is the highest program address, and LMEMV50, which is the lowest program address.

FILE ORDINAL TABLE (FOT)

Each program mode, batch and priorities 1 through 4, has an FOT. The FOT links each file that was opened or equipped to the file description table (FDT) for the file.

The first word address of each FOT can be obtained from the FOTADV50 table by indexing the symbolic address FOTADV50.

FOTADV50

	23 ... 17	11 ... 05 ... 00
FOTADV50	Zero	Reserved for future use
+1	Zero	fgfot
+2	Zero	p4fot
+3	Zero	p3fot
+4	Zero	Reserved for future use
+5	Zero	p2fot
+6	Zero	p1fot
+7	Zero	Reserved for future use

fgfot	Address of first entry of the FOT for batch programs
p4fot	Address of first entry of the FOT for priority 4 programs
p3fot	Address of first entry of the FOT for priority 3 programs
p2fot	Address of first entry of the FOT for priority 2 programs
p1fot	Address of first entry of the FOT for priority 1 programs

The format of the FOT for each program mode is identical, except for the number of entries (69 entries for batch, 63 entries for each priority level).

FOT Entry

	23		14 00
Word 1	r	Reserved	Current size of the FDT table
Word 2	r	Reserved	fdt for file 01
Word 3	r	Reserved	fdt for file 02
Word 4	r	Reserved	fdt for file 03
Word 5	r	Reserved	fdt for file 04
⋮			⋮
Word 66	r	Reserved	fdt for file 65
Word 67	r	Reserved	fdt for file 66
Word 68	r	Reserved	fdt for file 67
Word 69	r	Reserved	fdt for file 68

r FDT release flag (refer to parameter in CLOSE macro in section 13)

0 Releases the FDT for a file when the file is closed.

1 Does not release the FDT for a file when the file is closed.

fdt First word address of the FDT for the file. The FDT is zero if the file is not open or equipped.

FILE DESCRIPTION TABLE (FDT)

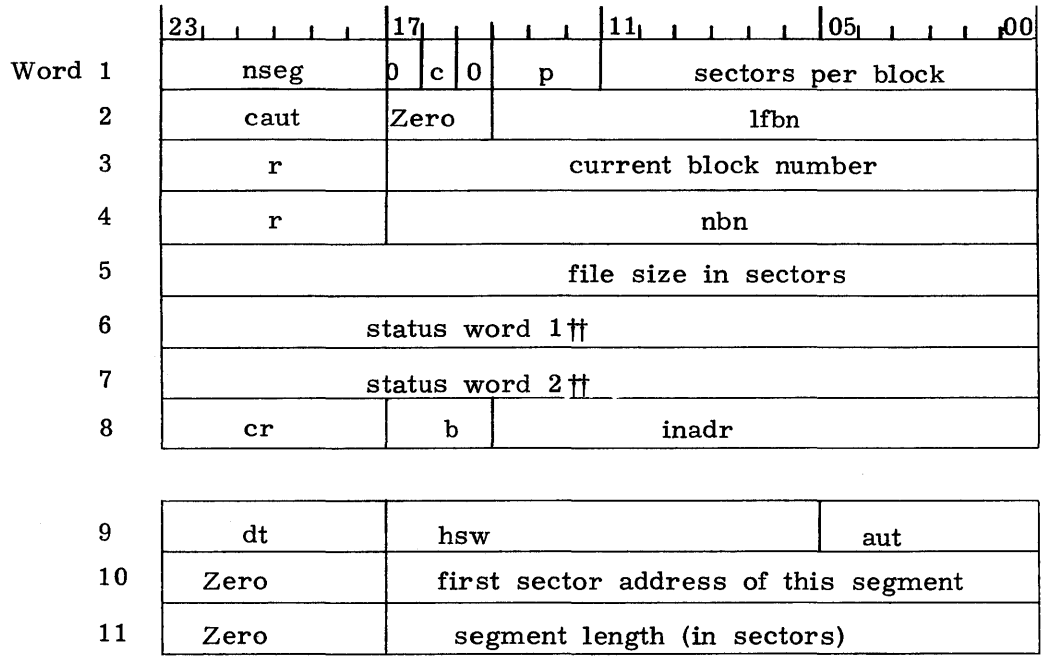
Each time a file is equipped or opened, the EQUIP or OPEN statement creates an FDT for the file. The FDT for a file is released from core when the file is closed, when an EOJ statement is read in an associated batch program, or when the associated priority program terminates. The first word address of a file FDT table can be obtained from the file FOT.

FDT For Unit Record Devices

	23 17 11 05 00			
Word 1	Zero		cpb	
2	caut	Zero		
3	Zero		Current record number	
4	a	Zero	hsw2	
5	Zero		mbc	
6	status word 1†			
7	status word 2†			
8	cr	b	inadr	
9	dt	hsw		aut

† Refer to Unit Status Request in section 11 for the format of these words.

FDT For Mass Storage Files



Parameter	Description	Use†	Word	Bits
a	ASCII flag. Applicable only to 3446, 3447, and 3691 controllers, card reader, card punch, and paper tape station. 0 Non-ASCII I/O function 1 ASCII I/O function	UR	4	23
aut	AUT entry for the file.	B	9	05 through 00
b	Memory bank number that the file is located in.	B	8	17 through 15
c	File label update flag 0 update file's label if file closed 1 Do not update file's label when file is closed	MS	1	16

† MS = mass storage, UR = unit record, B = both mass storage and unit record
†† Refer to Unit Status Request in section 11 for the format of these words.

Parameter	Description	Use	Word	Bits
caut	Ordinal of AUT entry used for last I/O function on this file (that is, AUT for file segments on different mass storage drives, etc.).	B	2	23 through 18
cpb	Maximum number of characters allowed per block (that is, max buffer size).	UR	1	14 through 00
cr	Condition register value to be used if an I/O interrupt routine for the file is entered. The values may be: 60 Program state, interrupt enabled. 20 Monitor state, interrupt enabled.	B	8	23 through 18
dt	Device type (refer to appendix C).	B	9	23 through 18
hsw	Hardware status word obtained from the equipment upon receiving an interrupt.	B	9	17 through 6
hsw2	Second hardware status word obtained from 657 or 659 tape drives.	UR	4	14 through 11
inadr	First word address of a user I/O interrupt processing routine.	B	8	14 through 00
lfbn	Number of the block containing the label for the file in LABEL-FILE.	MS	2	14 through 00
mbc	Maximum number of blocks that can be read or written. This is an installation parameter which is applicable only to files 60, 61, and 62.	UR	5	14 through 00
nbn	Next available block number (last block written plus one).	MS	4	14 through 00
nseg	Number of segments in this file.	MS	1	23 through 21
p	Protection flag 0 Read or write 1 Read only file	MS	1	14 through 12
r	Reserved for future use.			

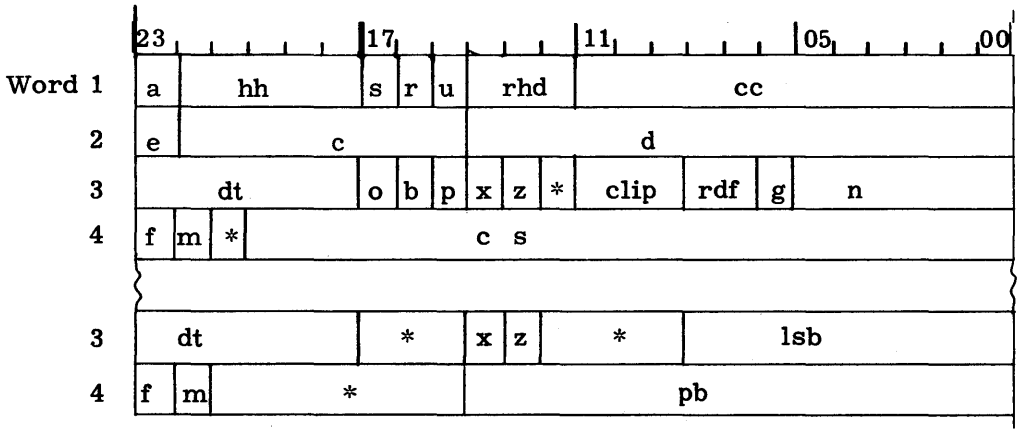
AVAILABLE UNIT TABLE (AUT)

The AUT describes each I/O unit known to the system. It contains one four-word entry for each unit. CIO cannot reference or use I/O units that do not have an AUT entry. These units can be used only by COMPASS priority 1 and 2 programs. These programs do their own selects, connects, input, and output instruction.

The AUT starts at symbolic address AUTV50. The starting address of any AUT entry for a file may be obtained as follows:

1. Obtain the AUT ordinal from the file FDT.
2. Subtract one from the ordinal and multiply the result by four.
3. Add the result from step 2 to the starting address of the AUT.

AUT Entry



These two words replace the preceding words 3 and 4 for card punch units

Parameter	Description	Word	Bits
a	Unit assigned flag	1	23
	0 No file number assigned to the unit		
	1 A file number is assigned to the unit		
b	Controlled backspace hardware flag.	3	16
	0 Controlled backspace hardware available		
	1 Controlled backspace hardware not available		

Parameter	Description	Word	Bits
c	Channel number flags. A one indicates the unit controller is attached to the channel. A zero indicates the unit controller is not attached to the channel. Channel / Bit 0 / 22 1 / 21 2 / 20 3 / 19 4 / 18 5 / 17 6 / 16 7 / 15	2	22 through 15
cc	Twelve-bit connect code for the unit.	1	11 through 00
clip	Value of read clipping level bias to be used for reading. Applicable only for tape drives with programmable read clipping-level hardware.	3	11 through 9
cs	Checksum from last record written.	4	20 through 00
d	The entry point of the driver for this unit.	2	14 through 00
dt	Device type code (refer to appendix C).	3	23 through 18
e	Assignment in process flag. 0 File number assignment not in process 1 File number assignment in process (FDT is being constructed)	2	23
f	End of file flag 0 Last record written was not an EOF 1 Last record written was an EOF	4	23
g	System noise record flag (only for tape drives) 0 System noise records are used 1 System noise records are not used	3	06
hh	Hardware type code (refer to appendix C).	1	22 through 18

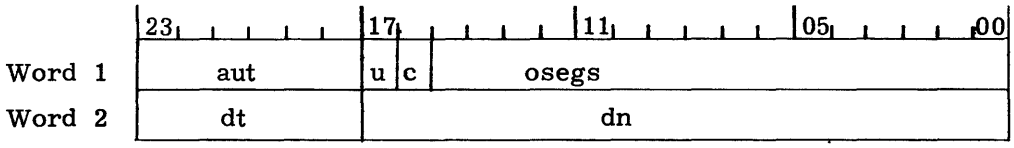
Parameter	Description	Word	Bits
lsb	Number of words punched in the last card.	3	08 through 00
m	Binary record flag	4	22
	0 Last record was not written in binary mode		
	1 Last record was written in binary mode		
n	Maximum noise record size in characters. For tape drivers only.	3	05 through 00
o	ASCII hardware flag	3	17
	0 No BCD to ASCII conversion hardware		
	1 Controller has BCD to ASCII conversion hardware		
p	Programmable read clipping level hardware flag	3	15
	0 Hardware not available		
	1 Hardware available		
pb	First word address of a 40-word punch save area address. Used for punch error recovery.	4	14 through 00
r	Reserved flag	1	16
	0 Unit not reserved		
	1 Unit reserved for use by a second CPU		
rdf	Read recovery flag for tape drives	3	08, 07
	00 Use standard read recovery		
	01 Suppress opposite direction read recovery		
	10 Force opposite direction read recovery		
rhd	Reserved for special hardware driver flags	1	12 through 10
s	Unit inoperable flag	1	17
	0 Unit operable		
	1 Unit inoperable		
u	ASCII mode default flag. Specifies default value if Hollerith/ ASCII parameter is omitted from JOB statement.	1	15
	0 Unit is Hollerith mode		
	1 Unit is ASCII mode		

Parameter	Description	Word	Bits
x	Error recovery flag	3	13
	0 Normal recovery processing		
	1 Operator requested recovery be abandoned		
z	Operator recovery action flag	3	12
	0 Not waiting for operator action for reject or error recovery		
	1 Waiting for operator action for error or reject recovery		
*	Reserved for future use.		

MASS STORAGE TABLE (MST)

The MST has one two-word entry for each disk drive and disk file in the system. The first word address of the table and the table length are supplied in the RPT.

MST Entry



aut The ordinal of the AUT table entry for the unit.

u Unit inoperable flag. Flag (bit 17) is set and cleared by system operator.

c Device class. 1 = class R; 0 = nonclass R.

osegs Number of file segments on the device that are currently open.

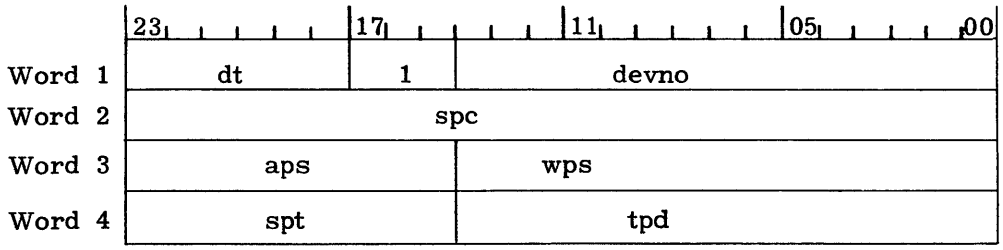
dt Device type code (refer to appendix C).

dn Device number assigned to the disk pack currently mounted on-line.

MASS STORAGE PARAMETER TABLE (MSPT)

The MSPT describes the characteristics of each type of mass storage device used by MSOS. The table contains one four-word entry for each type of device. The first word address of the table and the table length are supplied in the RPT table.

MSPT Entry



dt — Device type code (refer to appendix C).

devno — Device number (in octal) corresponding to the device type code. For example, if dt equals 50, devno equals 853.

spc — Number of sectors per cylinder.

aps — Number of addresses per sector (128 for drum; 1 for all disk units).

wps — Number of words per sector.

spt — Sectors per track.

tpd — Tracks per device.

RESIDENT ALLOCATION TABLE (RAT and NRAT)

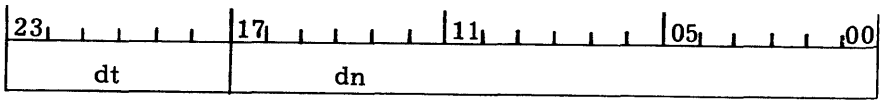
The RAT contains a 1-word entry for each mass storage device that OCAREM can use to allocate or expand a mass storage file. The user makes and clears entries in the RAT with RAT and RRAT statements. Each RAT or RRAT statement clears the old entries and enters the devices specified on the RAT or RRAT statement. A RAT or RRAT statement without parameters clears the RAT. The EOJ statement at the end of each job also clears the RAT.

The NRAT is a table similar to the RAT. Just prior to each allocate or expand function, MSOS clears the NRAT and copies the entry or entries in the RAT into the NRAT. If the RAT has been cleared (no entries), MSOS makes an entry in the NRAT for each on-line mass storage device.

OCAREM uses the NRAT entries and the RAT class entry in the RPT table to determine which devices it can use to allocate or expand a file. OCAREM uses only those devices in the NRAT which are of the class specified by the RAT class entry. The remaining entries are ignored.

The first word address and the number of words in the RAT can be obtained from the RPT.

RAT and NRAT Entries



dt Mass storage device type code (refer to appendix C).

dn Device number assigned when the device was entered in the system with the ENTER statement.

FILE ENVIRONMENT TABLE (FET)

MSOS has two types of FETs. One FET is constructed by OCAREM to identify mass storage files. The other FET is constructed by L-MSIO for both tape and mass storage files.

The OCAREM FET must be constructed to identify a mass storage file before the file can be allocated, opened, modified, or released. Either a FET statement or a FILEID macro can be used to construct the FET. The OCAREM FET contains the following file identification information:

Owner's name

File name

File edition number

File access security code

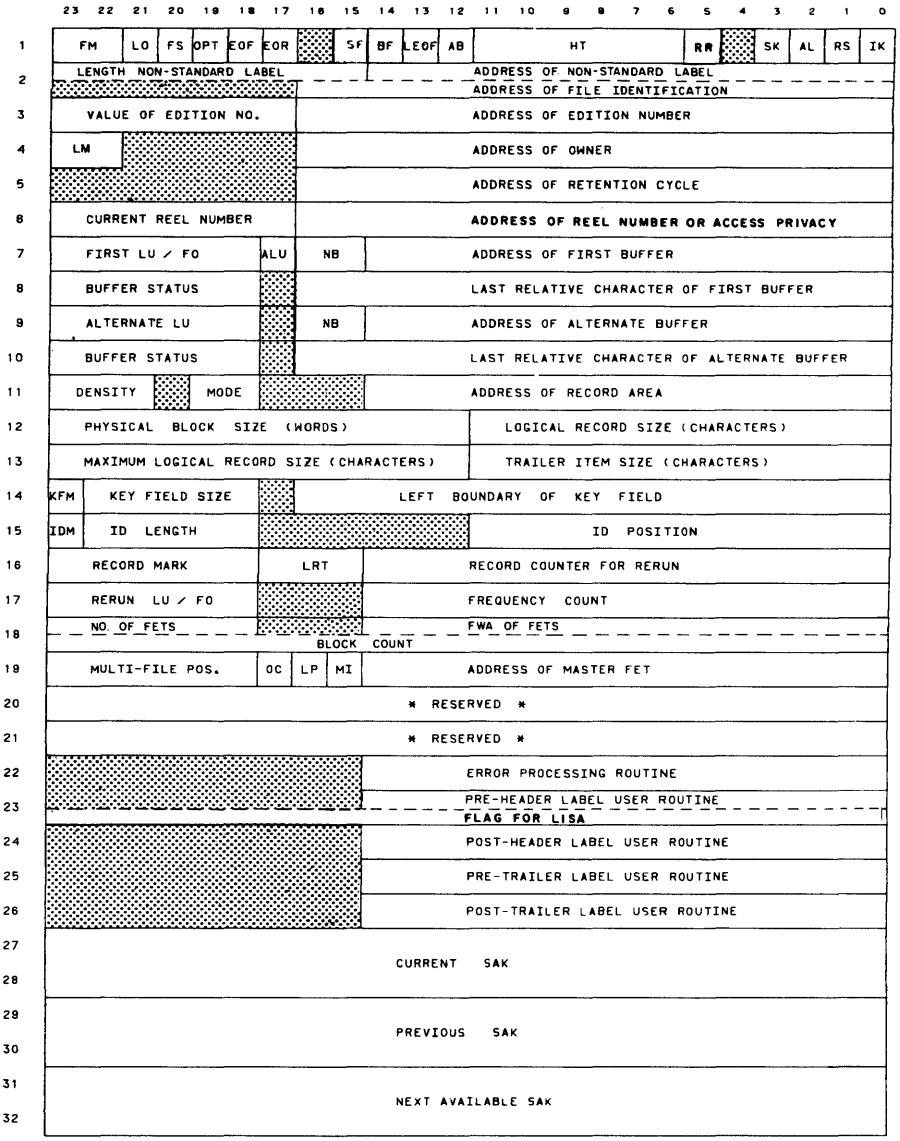
File modification security code

File block size

If a FILEID macro was used to construct the FET, the symbolic address for the FET location that was used in the FILEID macro may be used in COMPASS programs to read the FET. If a FET control statement was used to build the FET, it cannot be referenced in a program.

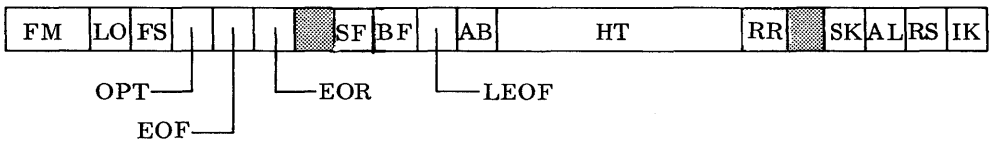
Refer to the FILEID macro in section 13 for the format of the OCAREM FET. An L-MSIO FET must be constructed for each tape and mass file to be processed with the L-MSIO routines. The L-MSIO FET must be constructed with a FILEDESC macro in a COMPASS program. The symbolic location address used in the FILEDESC macro may be used to read L-MSIO FET in COMPASS programs.

FILE ENVIRONMENT TABLE



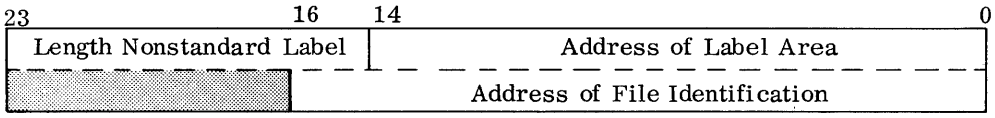
Codes in the column headed Set By represent the following:

F	FILEDESC macro
V	VARIABLE macro
L	LABELING macro
R	RERUN macro
S	STOPOPEN macro
M	Set during execution
C	MS COBOL



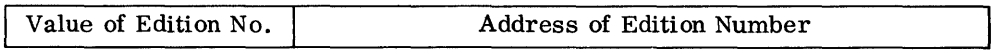
Word	Bit	Set By	Description
1	23-22	M	file mode 01 input 10 output 11 input/output
	21	M	last operation on this file 0 GET 1 PUT
	20	M	file status 1 open 0 closed
	19	F	1 file is optional 0 file is mandatory
	18	M	1 EOF trailer label on this file
	17	M	1 EOT trailer label on this file
	16		not used
	15	C	used by COBOL: mass storage scratch file flag
	14	F	block format 1 unblocked, one logical record per physical record 0 records are blocked to capacity of block size
	13	M	1 EOF mark on this file
	12	M	0 first buffer is active 1 alternate buffer is active

Word	Bit	Set By	Description
1	11-6	M	hardware type 01 magnetic tape 02 card reader 03 printer 04 card punch 05 console typewriter 10 channel typewriter 40 1311 disk packs 41 852 disk packs 50 853 disk packs 51 854 disk packs 52 841 disk packs 60 813/814 disk file 70 863 drum
	5	M	read reverse 1 file open for reverse read
	3	M	used by COBOL 1 last operation was SEEK
	2	M	1 active buffer has been altered
	1	F	access mode 0 sequential access 1 random access
	0	M	1 invalid key condition resulted from last operation

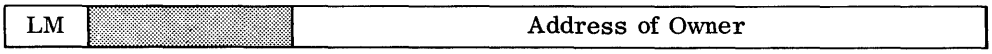


2	23-15	L	length of nonstandard label in characters
	14-0	L	word address of the nonstandard label area
	16-0	L	for standard labeling, address of the file identification

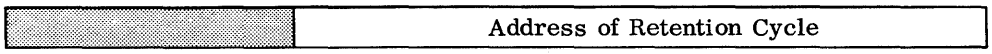
If word 2 is zero and LM in word 4 specifies nonstandard labels, the record area is used as the label area. For an output file with fixed length records, MSIO writes all characters stored in the record area for each reel of the file opened. If the records are variable length, the user must specify the length of the label by the presence of a key field or record mark.



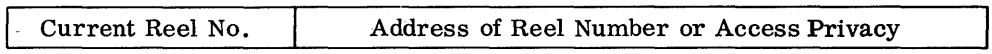
Word	Bit	Set By	Description
3	23-17	M	binary value of the edition number
	16-0	L	character address of two-digit BCD (00-99) edition number



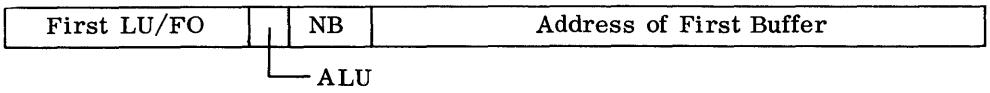
4	23-22	L	labeling mode 00 omitted labels 01 nonstandard labels 11 standard labels
	21-17		not used
	16-0	L	address of an eight-character owner identification; not used for nonmass storage files



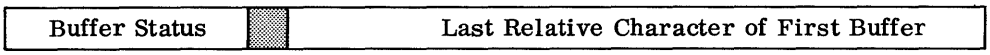
5	16-0	L	address of three-digit BCD retention cycle field (000-999; 999 indicates permanent retention); not used for mass storage



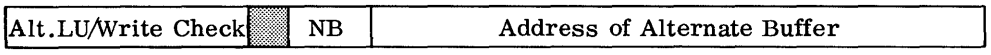
6	23-17	M	current value of reel number; initially contains specified value of the reel number; incremented by one for each new reel of file
	16-0	L	address of a two-digit BCD reel number. If zero, reel numbers are not checked on input. For mass storage files this is the address of four-character access security code.



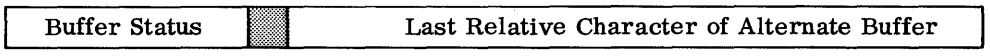
Word	Bit	Set By	Description
7	23-18	F	first logical unit or file ordinal; for mass storage, the file ordinal is assigned by MSIO when the file is opened
	17	M	active logical unit 0 first logical unit active 1 alternate logical unit active
	16-15	F	number of buffers (00, 01, or 10)
	14-0	F	address of the first buffer



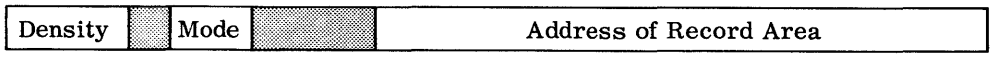
8	23-18	M	buffer status of first buffer
	17		not used
	16-0	M	last relative character in current buffer



9	23-18	F	alternate logical unit for nonmass storage files. If nonzero for mass storage files, write check operations are issued for all writes.
	17		not used
	16-15	M	number of buffers (00, 01, or 10)
	14-0	F	address of the alternate buffer



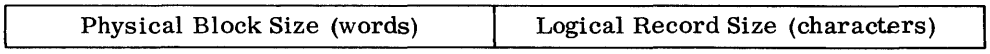
10	23-18	M	buffer status of alternate buffer
	17-15		not used
	16-0	M	last relative character in current buffer



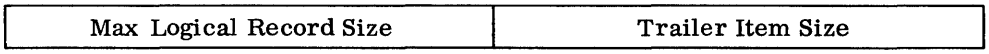
Word	Bit	Set By	Description
11	23-2	F	recording density

		7 Track	9 Track
1	low	200 bpi	800 cpi
2	med	556 bpi	800 cpi
3	high	800 bpi	1600 cpi

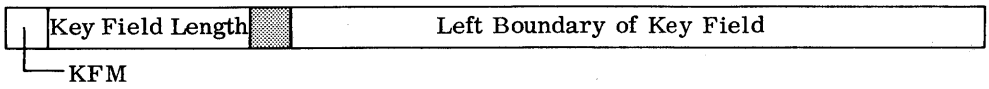
	20	not used
	19-18	F recording mode 00 even parity (BCD) 01 odd parity (binary)
	17-15	not used
	14-0	F address of the record area



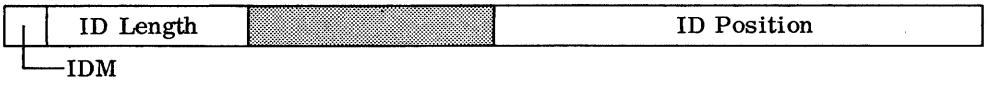
12	23-12	F	physical block size; number of words per physical record
	11-0	F	logical record size, in characters, of fixed records; size of fixed portion of variable records which have trailer items. This is zero if variable records vary by key field or record mark.



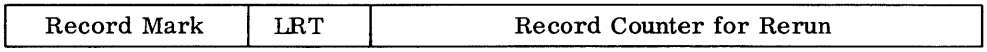
13	23-12	V	maximum size in characters of the variable portion of logical records. For variable records with trailers, this is the size of the trailer item times the maximum number of occurrences. For all others, this is the maximum size of a record within the file.
	11-0	V	trailer item size in characters



Word	Bit	Set By	Description
14	23	V	mode of key field address 0 key field is within the record 1 key field is outside the record
	22-18	V	number of characters in key field
	17		not used
	16-0	V	character position of the key field relative to the beginning of the record if the key field is within the record. This is the character address of the key field if the field is outside the record area.

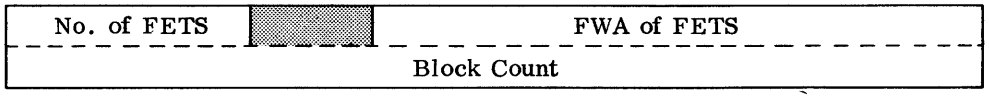


15	23	V	mode of identification field 0 numeric 1 alphanumeric
	22-18	V	character size of identification field
	17-12		not used
	11-0	V	position of left character of identification field within the record

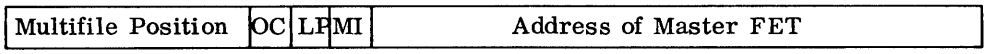


16	23-18	V	record mark to be used with record delimiter format
	17-15	V	logical record type 0 fixed length records 1 key field contains total number of characters 2 key field contains the number of occurrences of a fixed length trailer item 3 universal format 4 record mark terminates each record
	14-0	M	record counter for check point dumps

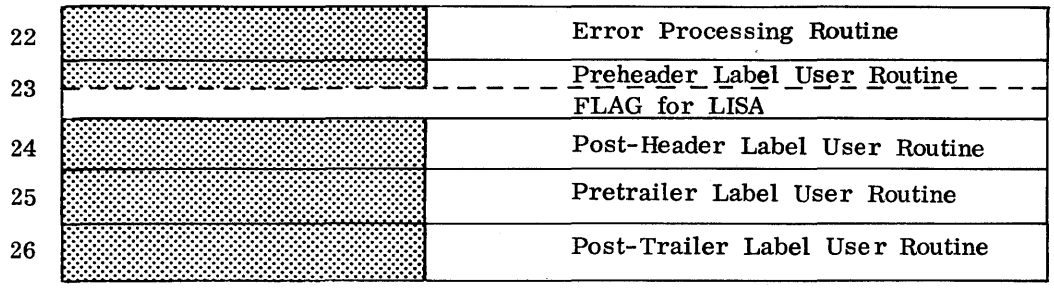
Word	Bit	Set By	Description
17	23-18	R	logical unit or file ordinal on which check point dumps are written
	17-15		not used
	14-0	R	frequency count at which check point dumps are to be taken. If zero, dumps are taken at end of reel.



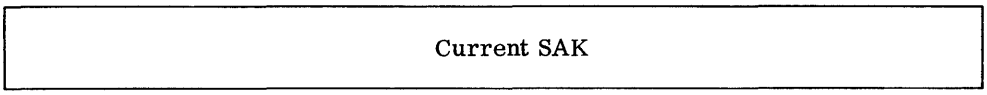
18	23-18	R	number of FET's within this program
	14-0	R	first word address of FET's
	23-0	M	count of the number of physical blocks read or written; not applicable to random access files



19	23-18	S	position of file on a multifile reel
	17	M	1 file on this reel is already open
	16	M	0 reel is at load point
	15	S	1 master FET indicator 0 address below points to the master FET
	14-0	S	address of master FET or, if bit 15 is 1, address of the last opened FET

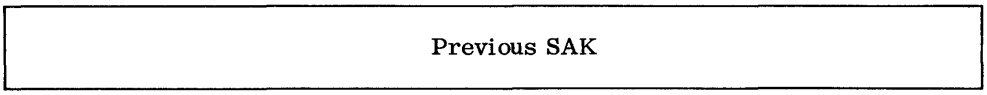


Word	Bit	Set By	Description
22	14-0	M	address of error processing routine to be entered after standard error recovery procedure is executed
23	14-0	M	address of preheader label processing routine or
	23-0		flag used for LISA
24	14-0	M	address of post-header label processing routine
25	14-0	M	address of pretrailer label processing routine
26	14-0	M	address of post-trailer label processing routine



27	23-0	M	block number for the current SAK
28	15-0	M	relative character position of the current SAK

Words 27 and 28 combine to form the current SAK.



29	23-0	M	block number for the previous SAK
30	14-0	M	relative character position for the previous SAK

Words 29 and 30 combine to form the previous SAK.

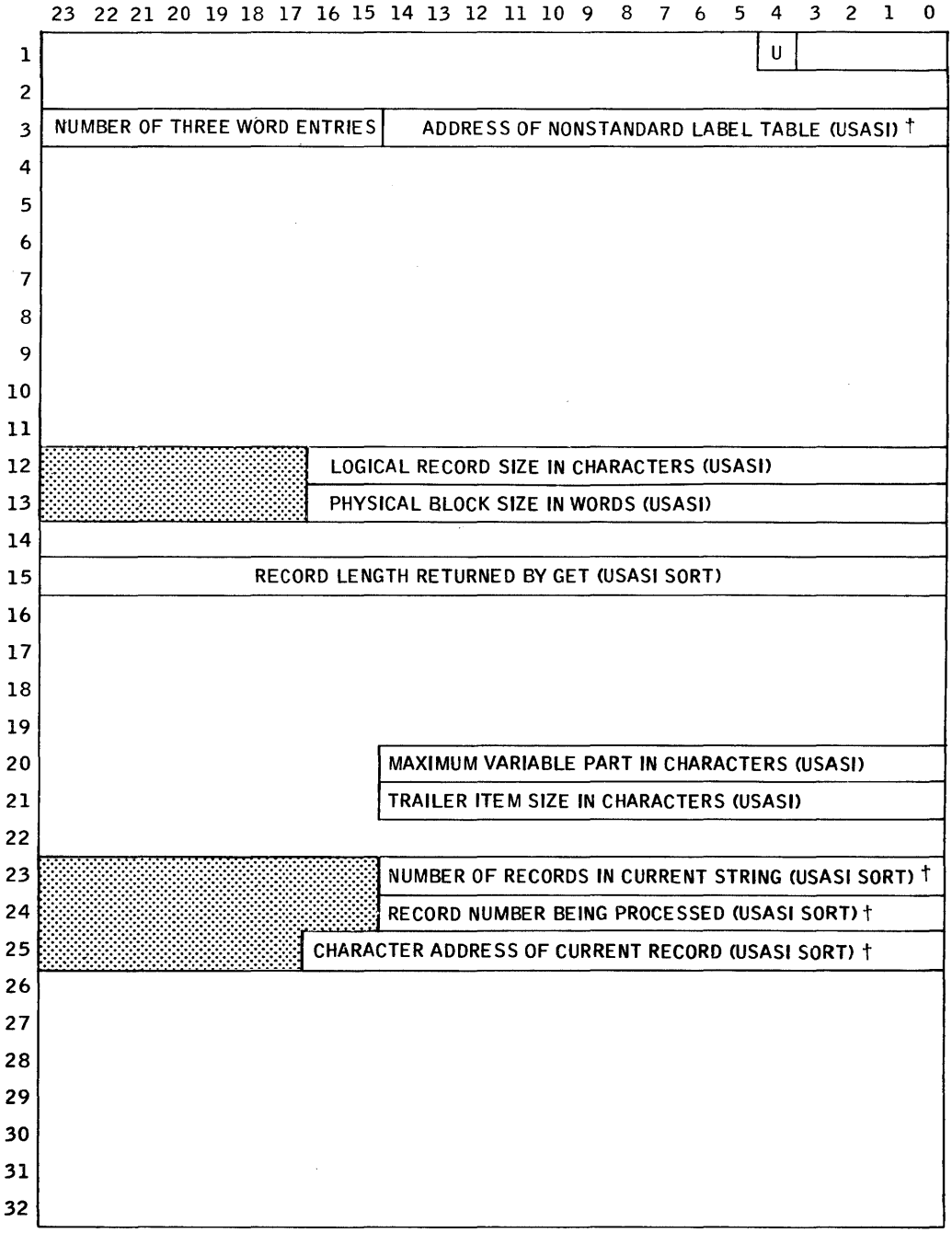
Next Available SAK

Word	Bit	Set By	Description
31	23-0	M	block number where the next record can be written
32	14-0	M	relative character position within the above block where the next record can begin

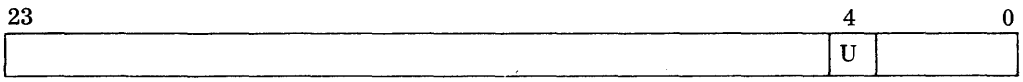
Words 31 and 32 combine to form the SAK of the next available record. This is not maintained for random access files.

MSOS FILE ENVIRONMENT TABLE WITH USASI OPTIONS

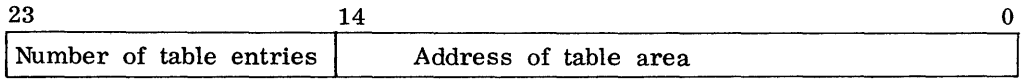
If the USASI option is set (bit 4 of word 1) the following words of the FET table are redefined as follows. The remaining words in the table retain the same word definitions specified in the first section of this appendix.



† These words can have variable definitions. Even though the USASI option is selected, the program can select not only the new word definition listed above but the alternate word definitions listed in the first section of this appendix.

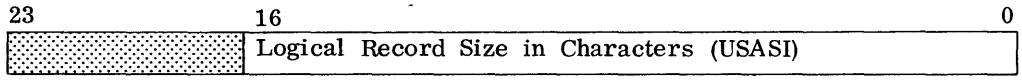


Word	Bit	Description
1	23-6	unused
	4	USASI FET set by USASI COBOL compiler

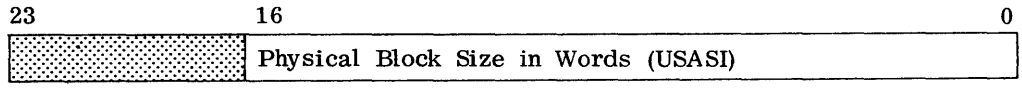


†

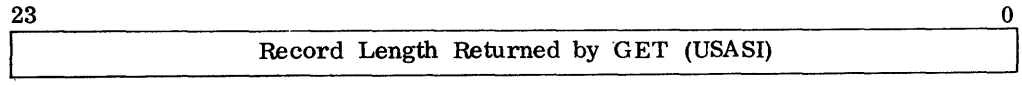
3	23-15	number of three word table entries used to make comparisons on nonstandard labels for USASI COBOL.
	14-0	address of table area



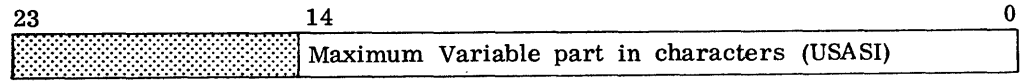
12	23-17	not used
	16-0	logical record size in characters. For variable length records the logical record size specified in this word refers to the fixed length section found in each variable length record.



13	23-17	not used
	16-0	physical block size in words

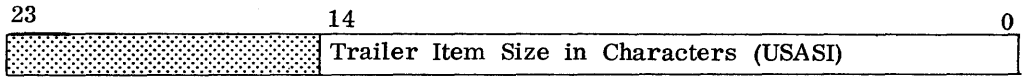


15	23-0	record length returned by GET. This record length is used by USASI SORT.

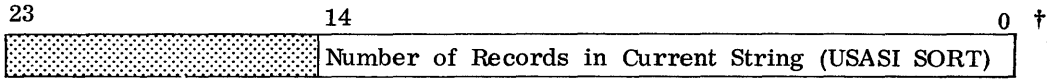


† This word can have variable definitions. Even though the USASI option is selected, the program can select not only the new word definition listed above but the alternate word definition listed in the first section of this appendix.

Word	Bit	Description
20	23-15	not used
	14-0	maximum size of variable sections of logical records

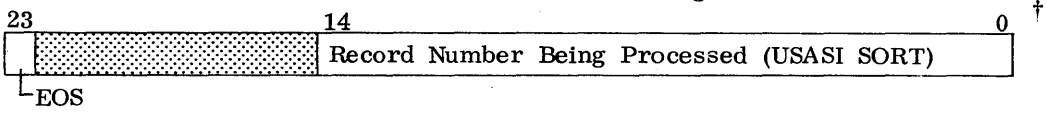


21	23-15	not used
	14-0	trailer item size in characters



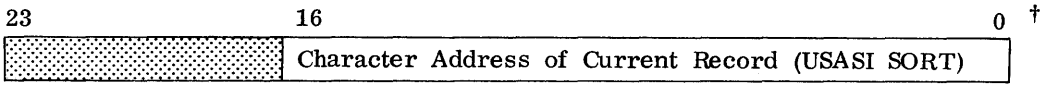
†

23	23-15	not used
	14-0	number of records in current string



†

24	23	end of string indicator
	22-15	not used
	14-0	record number being processed



†

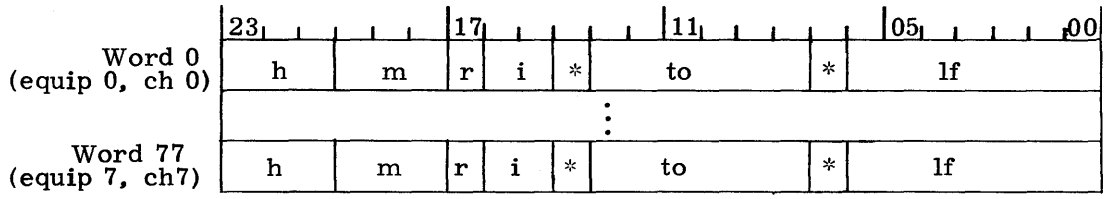
25	23-17	not used
	16-0	character address of current record

† These words can have variable definitions. Even though the USASI option is selected, the program can select not only the new word definition listed above but the alternate word definitions listed in the first section of this appendix.

EQUIPMENT STATUS TABLE (EST)

The EST is a 64-word table that contains one entry for each possible I/O equipment (controller) in the system. MSOS uses the EST for lost interrupt detection and to identify equipment dedicated to real-time applications. The EST cannot be referenced within a program. However, the EST may be inspected in a core dump listing by obtaining the first word address from a PRELIB listing.

EST Entry



h — Always equal to one.

m — Mode of program using the unit

- 0 unit not in use
- 1 batch
- 2 P4
- 3 P3
- 5 P2
- 6 P1

r — Real-time flag

- 0 Unit is not a real-time equipment
- 1 Unit is a real-time equipment

i — Interrupt flag

- 0 No user interrupt selected for the unit
- 1 Abnormal termination interrupt was selected for the unit
- 2,3 Normal completion or abnormal termination interrupt selected for the unit

* — Reserved for future use.

to — Time out flag. Contains the number of system-check clock interrupts that occurred since I/O was initiated on this unit. This parameter is used for lost interrupt detection.

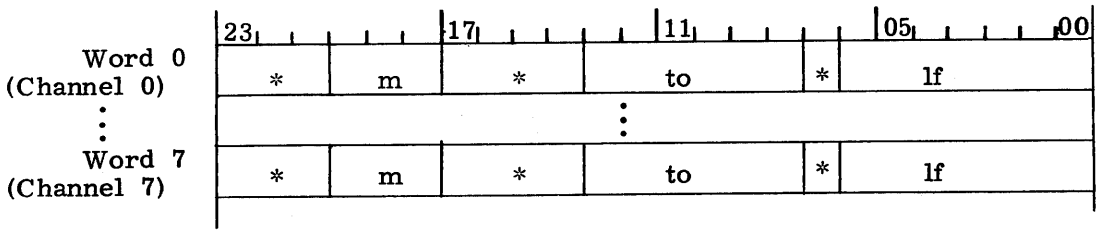
lf — Logical file number assigned to this unit (FOT index for this unit).

CHANNEL STATUS TABLE (CST)

The CST is an 8-word table that contains one entry for each possible I/O channel in the system. The CST follows immediately after the EST (that is, first word address of EST+100_8 equals first word address of CST).

The CST is used for lost interrupt detection, identifying the unit which is currently active on the channel, and the mode of the program using the channel for I/O. The CST cannot be directly referenced in a program. However, the contents of the CST may be inspected in a core dump listing by obtaining the first word address from a PRELIB listing or by adding 100_8 to the first word address of EST.

CST Entry



* Reserved for future use.

m Mode of the program using the channel

0	Channel unused
1	Batch
2	P4
3	P3
5	P2
6	P1

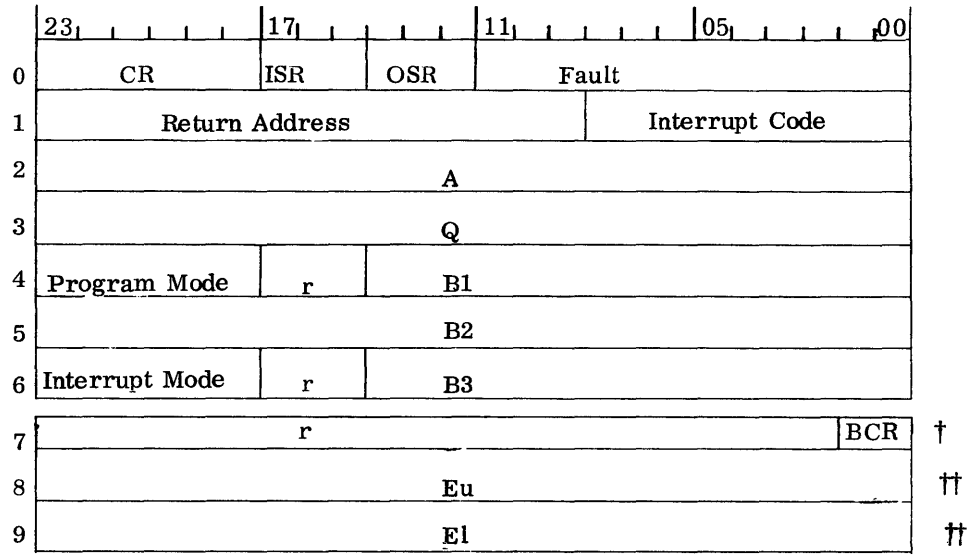
to Time out flag. Contains the number of system-check clock interrupts that occurred since I/O was initiated on the channel. This parameter is used for lost interrupt detection.

lf Logical file number currently active on this channel.

The CST can be used to determine the equipment and unit number of an I/O unit that caused a channel interrupt. The interrupt code appearing in the lower 12 bits of address five when the I/O interrupt occurred can be used to locate the CST entry for the channel that caused the interrupt. This can be done by adding the interrupt code to the first word address of the EST. Once the CST entry for the channel is located, the interrupting unit FOT entry can be found by using the lf and m parameters in the CST entry. Then the FOT can be used to locate the unit FDT table and AUT entry. The interrupting unit equipment and unit numbers may be obtained from the connect code in the unit AUT entry.

REGISTER SAVE TABLE

MSOS reserves an area for 12 register save tables which CIC uses to save the system operating conditions when a batch or priority program is interrupted. The table may be referenced from a priority 1 real-time program. The format of the table is dependent upon the absence or presence of floating point and/or BDP hardware in the system.



CR	=	Contents of the condition register
ISR	=	Contents of the instruction state register
OSR	=	Contents of the operand state register

† This word is included only if the system has either BDP or floating point hardware. The word contains zero if the system has floating point but no BDP hardware.

†† These words are included only if the system has floating point hardware.

Fault	=	Contents of interrupt fault register
Return Address	=	Address of next instruction in the program that was interrupted
Interrupt Code	=	Interrupt code received when interrupt occurred
A	=	Contents of register A
Q	=	Contents of register Q
B1-B3	=	Contents of B registers
Program Mode	=	Level of program that was interrupted 1 = Batch 5 = P2 2 = P4 6 = P1 3 = P3 7 = CIO processing
Interrupt Mode	=	Interrupt mode; lower three bits are the same as program mode and upper three bits are interrupt hierarchy 1 = User selected interrupt 2 = Manual or program fault interrupt 3 = Real-time interrupt †
r	=	Reserved
BCR	=	Contents of BDP condition register
Eu	=	Upper 24 bits of E register
E1	=	Lower 24 bits of E register

† For real-time interrupts, the program mode is 6 for P2 real-time and 7 for P1 real-time.

FILE LABELS G

TAPE FILE LABELS

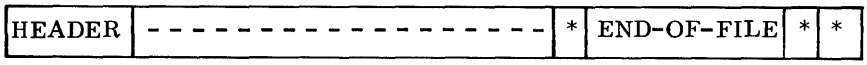
L-MSIO, COBOL, and SORT use 3000/1700 Standard Tape Labels. The standard tape labels can be used with the following types of file sets:

Single file, single reel	A user data file which can be contained on a single reel of magnetic tape
Single file, multireel	A user data file which cannot be contained on a single reel of magnetic tape
Multifile, single reel	Related user data files which can be contained on a single reel of magnetic tape. All files must be recorded at the same density, but recording parity may vary.

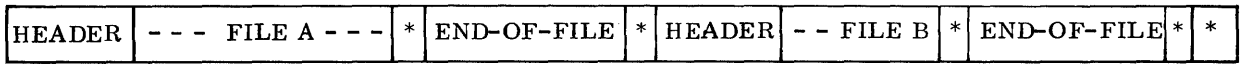
Every reel has a header label as the first block of the reel. The last block of a file is followed by an end-of-file label. One tape mark† immediately precedes, and two tape marks (or one tape mark and one header label) immediately follow this end-of-file label. Whenever a reel ends within a file, the last block of the file on that reel is followed by an end-of-tape label. One tape mark immediately precedes and two tape marks immediately follow this end-of-tape label. The following examples show the use of these labels in the file structure.

Examples:

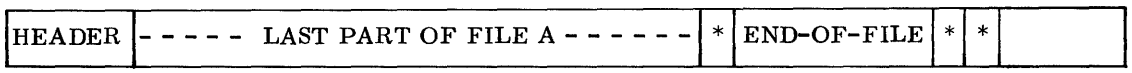
1. Single File, Single Reel



2. Multifile, Single Reel



3. Single File, Multireel



* Tape mark

\- User data

†Tape marks (end-of-file marks) are not prohibited as data on 3000/1700 standard labeled tapes, but because they are used to detect labels, they may produce erroneous block counts in the trailer label and inhibit the EOF feature. Tapes containing such extra tape marks should not be multifile and generally should not be used for information interchange.

HEADER LABEL

All header label records are 80 characters (480 bits) long and are unblocked. They are recorded in the same density as the remainder of the data file and are always recorded in even parity mode. Header records are separated from succeeding data records by an interrecord gap only. The following table shows the format and content of a header label.

Field Name	Starting Character Position	Length in Characters	Defined Values (BCD Characters Only)	Function
Density	1	1	2 5 8	7-track / 9-track 200 bpi / 800 cpi 556 bpi / 800 cpi 800 bpi / 1600 cpi
Header label identifier	2-3	2	( )	Identifies record as header label record
Logical unit number	4-5	2	As applicable	Specifies logical unit to which file is assigned
Retention code	6-8	3	000-999	Specifies, in days, retention period of file
File name	9-22	14	Any combination of legal BCD characters	Identifies file
Reel number	23-24	2	01-99	Identifies sequence of reels for multireel files
Date written	25-30	6	Any legal numeric date, expressed as mmddyy	Identifies date written; used with retention period to determine release date of file
Edition number	31-32	2	00-99	Identifies a single file set
User-supplied information	33	48	Any combination of legal BCD characters	User comments field

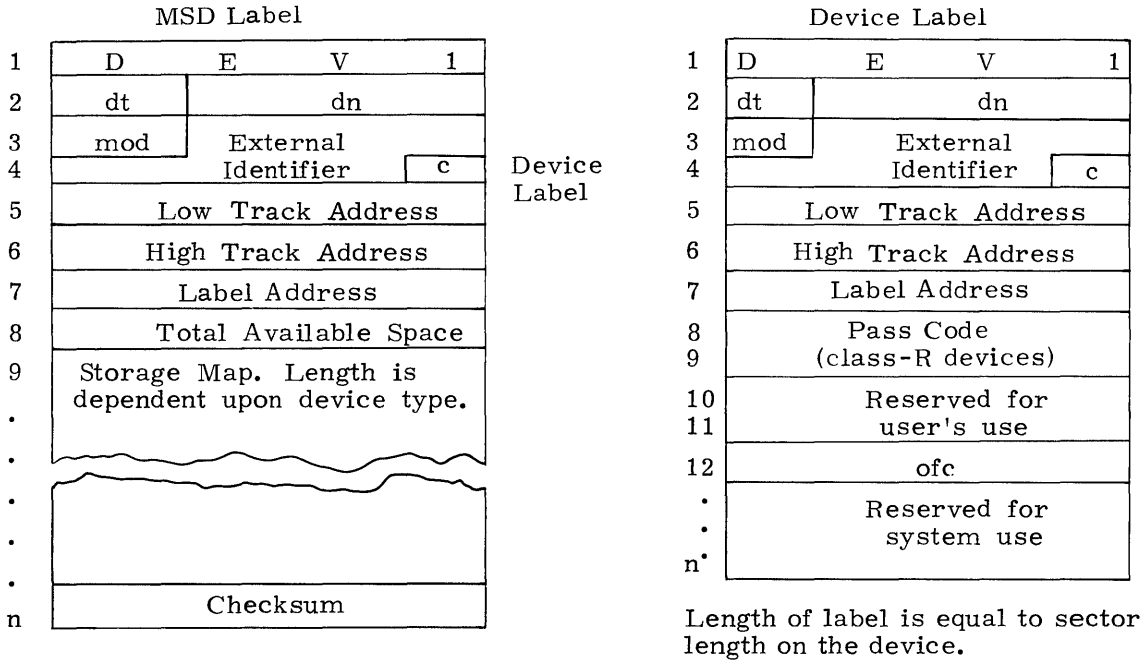
TRAILER LABEL

All trailer labels are 80 characters (480 bits) long, recorded in even parity in the same density as the remainder of the data file. Trailer labels are separated from preceding records (either data or label records) by a single tape mark, and are always followed by either a double tape mark or a tape mark and a header label. The format and content of the three types of trailer labels, end-of-tape, end-of-set, and end-of-file, are shown in the following table.

Field Name	Starting Character Position	Length in Characters	Defined Values (BCD Characters Only)	Function
Trailer label identifier	1	3	EOT	End-of-tape for intermediate reel
			EOS	End-of-set for final tape
			EOF	End-of-file for final tape
Record count	4	5	00000-99999	If nonblank, the value is the count of the physical records between the last reader label record and this trailer record.
User field	9	72	Any combination of legal BCD characters	User comments field

MASS STORAGE DEVICE LABEL

Each mass storage device that has been entered in the system has a device label written on track 0, and an MSD label written in the mass storage director file (MSD). The format of the labels is as follows:



In the MSD label, word 8 contains the number of unassigned tracks. Words 9 through n contain a bit mapping of the tracks on the device. A bit set to one indicates the corresponding track is assigned. A bit set to zero indicates the correspondent track is available. The correspondence between bits and tracks is:

Word Number	Bit Number	Track Number
9	0	0
9	1	1
:	:	:
9	23	23
10	0	24
10	1	25
:	:	:
j	i	24(j-9)+i

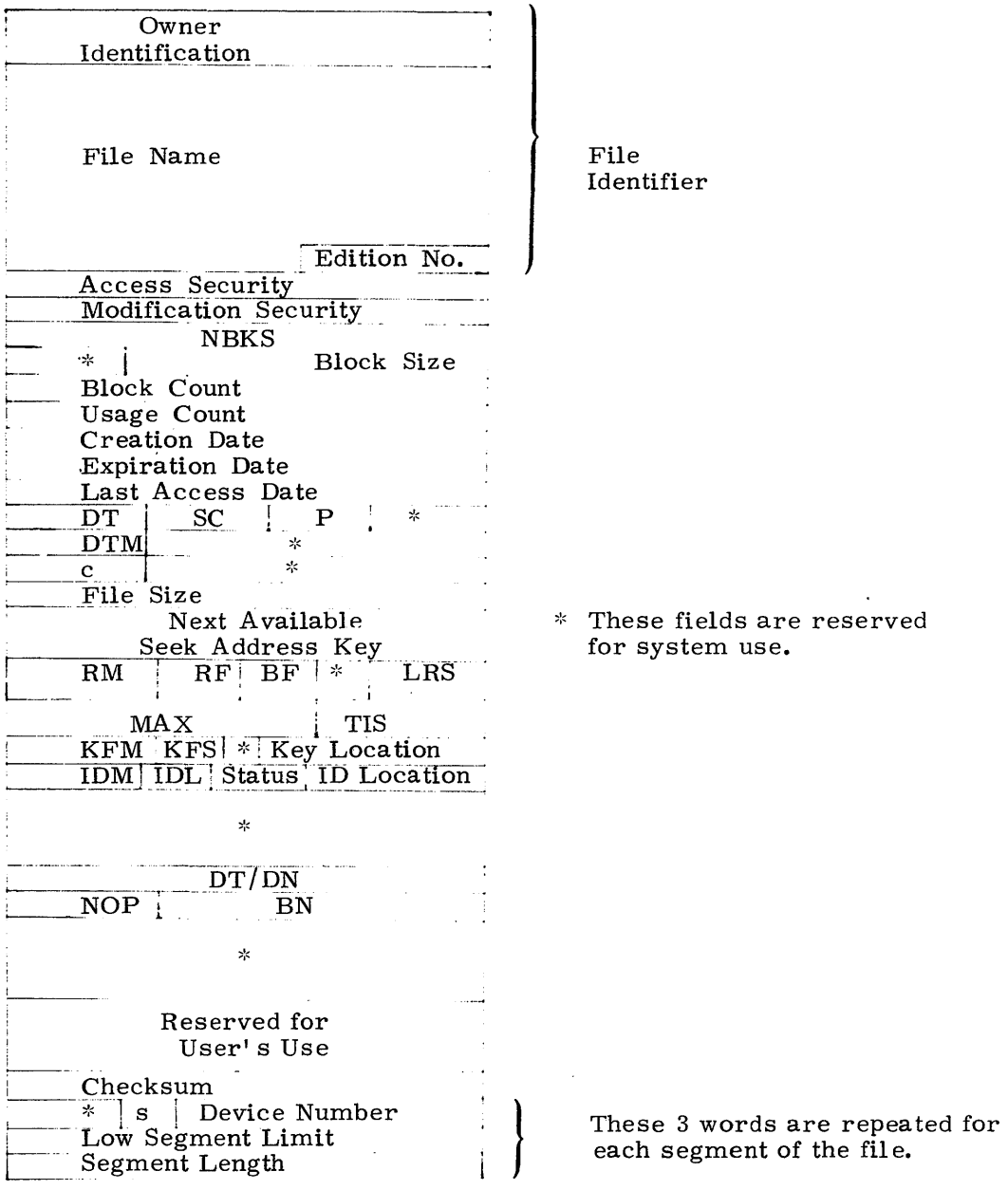
The length of a label for a device is 8+n/24 where n is the number of allocatable tracks on the device. When the number of bits in the map exceeds the number of tracks (number of tracks not divisible by 24), the unused bits in the last word are set to one.

DEVICE AND MSD LABEL PARAMETERS

Field Name	Number of Characters	Description
DEV1	4	Standard identifier prefixed to device labels
dt	1	6-bit device type code (appendix C)
dn	3	18-bit device number that matches an external number on each device
mod	1	6-bit device type modifier. Value is always 1.
External identifier	6	Alphanumeric characters corresponding to an external identifier on each device
c	1	Device class. 0 = nonclass-R, 2 = class-R
Low track address	4	Lowest track number (binary) accessible by MSIO
High track address	4	Highest track number (binary) accessible by MSIO
Label address	4	For class-R devices: bits 23 through 18 contain the number of tracks (octal) reserved for the R label on the device. Bits 17 through 00 contain the first track address of the R label. For a device containing IDFILE: first sector address of the IDFILE. For all other devices: not used, zeros.
Pass code	8	Class-R pass code specified on the ENTER statement. Not used on nonclass-R devices.
ofc	4	This field indicates whether or not the current OCAREM function has completed. A new OCAREM function cannot be started until the current one has completed. Appears on label of system device only. Zero for all other devices.

MASS STORAGE FILE LABEL

A mass storage file label is composed of a fixed 50-word base plus three additional words for each segment of the file. MSIO is capable of processing files which contain up to 63 segments although installations may set the maximum allowable segment count to some value less than 63 if they wish. The labels for all files are stored in a central label directory (called LABELFILE) which is contained on the system disk pack.



† OCAREM sets these fields to zero for all nonclass-R files.

FILE LABEL FIELD DESCRIPTION

Field Name	Size	Description
File Identifier	40 Char.	Used to uniquely identify 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 in other ways if an installation chooses to do so.
Access Security	4 Char.	This field is supplied when the file is allocated and must be supplied for each succeeding OPEN request.
Modification Security	4 Char.	This field is supplied when the file is allocated and must be supplied for each RELEASE, EXPAND, and MODIFY request.
NBKS	24 Bits	Number of blocks (binary) allocated to a file.
Block Size	3 Char. (binary)	This field contains the number of 6-bit characters in each record block.
Block Count	4 Char. (binary)	This field contains the highest block number that has been written. If the file is processed sequentially, this is actually the number of blocks written into the file.
Usage Count	4 Char. (binary)	Number of times the file has been opened.
Creation Date	4 Char. (binary)	Date, in the form YYMMDD, that the file was allocated.
Expiration Date	4 Char. (binary)	A date, in the form YYMMDD, supplied by the user when the file was allocated. This field determines when a file may be deleted.
Last Access Date	4 Char. (binary)	A date, in the form YYMMDD, supplied by the I/O system each time the file is opened or changed.
DT (Device Type)	1 Char.	A 6-bit code to indicate the type of mass storage device on which the file is contained.
SC (Segment Count)	1 Char. (binary)	This field contains the number of segments in the file.
P (Protection)	1 Char.	This field contains protection flags for use by the I/O system. The only values currently defined are: 0 The file may be read or written 1 The file may not be written

FILE LABEL FIELD DESCRIPTION (Cont'd)

Field Name	Size	Description
DTM	1 Char.	Always a one.
c	1 Char.	Device class 0 Nonclass R 2 Class R
File size	4 Char. (binary)	This field contains the number of tracks assigned to the file.
Next available seek address key	8 Char. (binary)	Number of the next available block that can be written on.
RM (record mark)	1 Char.	Character that terminates each record when variable record sizes are used with a record delimiting character.
RF (record format)	3 Bits	Denotes type of record format used in the file 0 Fixed length records 1 Key field containing number of characters in the record 2 Variable number of fixed length trailers 3 Universal 4 Record terminated by a delimiting character
BF (block format)	1 Bit	Denotes type of block format 1 Logical record per block 0 More than one logical record per block. Each block contains a two-word preamble (refer to appendix E).
LRS (logical record size)	12 Bits (binary)	Logical record size in characters for fixed length records, or size of fixed length trailer. Zero for all other variable size records.
MAX (maximum logical record size)	12 Bits (binary)	Maximum size in characters of variable logical records. For records with variable number of fixed length trailers, MAX is size of the trailer times maximum number of trailers. For all others, maximum size of records.
TIS (trailer item size)	12 Bits (binary)	Size of trailers in characters, if the RF field equals two. Otherwise, TIS is zero.
KFM (key field mode)	1 Bit	Mode of key field address 0 Key field within each record 1 Key field outside of the record or not in the file
KFW (key field size)	5 Bits (binary)	Number of characters in the key field of each record.
KEY LOCATION	17 Bits (binary)	Location of first character of key field in all records. If key field is not in the record, key location contains the character address in core of the key field.

FILE LABEL FIELD DESCRIPTION (Cont'd)

Field Name	Size	Description
IDM (Id mode)	1 Bit	Type of record identification 0 Alphanumeric 1 Numeric
IDL (Id length)	5 Bits (binary)	Length in characters of record identification field in each record
STATUS	6 Bits	Current status of the file
ID location	12 Bits (binary)	First character of identification field in each record
DT/DN	4 Char.	DT is a 6-bit device type number (octal, bits 23 through 18). DN is a device number (octal, bits 17 through 00). If the file is on a nonclass-R device, OCAREM sets DT/DN field to zero.
NOP	1 Bit	Nonopenable flag. A 1 indicates file is a class-R device and is not usable because allocation parameters (from different systems) exceeded value allowable on this system (that is, too many segments, too large file size, etc.). A 0 indicates the file can be used.
BN	23 Bits	Block number (octal) in RLABEL that contains this file's label. OCAREM sets BN to 0 for all nonclass-R files.
Checksum	4 Char. (binary)	Checksum of the entire label. This field is checked by the I/O system to detect accidental modification of the label.
S (segment down)	1 Bit	Segment down bit. Set by a CLOSE function for all segments downed in the FDT. 0 Segment up 1 Segment downed
Device number	3 Char. (binary)	The number of the device on which this file segment is stored. This field is checked against the device label to ensure that proper packs are mounted.
Low segment limit	4 Char. (binary)	The hardware address at which this file segment begins
Segment length	4 Char. (binary)	The number of allocatable tracks in this segment
* (Reserved)	--	These fields are reserved for future use by the system

SYSTEM FILES H

The MSOS system files, other than system scratch described in Table 2-1, cannot be directly referenced by a user. This appendix provides general background information about these files. These files are allocated at system installation time and are opened automatically each time the system is autoloaded. The system files are grouped in this appendix as follows:

MSOS LIBRARY FILES

- Resident file (RES)
- Absolute binary file (ABS)
- System library file (LIB)
- Directory of relocatable subprograms (DRS)

MASS STORAGE FILES

- Mass storage label file (LABELFILE)
- Mass storage label ID file (IDFILE)
- Mass storage devices label file (MSDFILE)
- Bad track file (BADTRACK)

RESIDENT FILE (RES)

RES contains the resident executive routines in absolute binary format. The autoload routine loads the RES file into the executive resident area of core when the system is autoloaded. The executive resident is described in section 8.

ABSOLUTE BINARY FILE (ABS)

ABS consists of all the absolute binary subprograms in the library that are not a part of the resident executive. ABS contains the nonresident executive routines which are loaded or overlayed in the variable resident area of core as they are needed by the system executive. These routines include the loader, error recovery, and job control routines used to process MSOS control statements.

In addition, the ABS contains subprograms which are part of L-MSIO, COMPASS, FORTRAN, COBOL, and ALGOL. These subprograms are absolutized to decrease compiler and assembler loading time. Each of these subprograms has a system entry point so that it can be called by a user or another library subprogram.

All routines in ABS are loaded in the variable resident area of core by RDCKF1 which is a resident routine. RDCKF1 uses a resident directory table (RDT) to locate and load subprograms from the ABS. The format of each entry in the RDT is as follows:

word 1 Subprogram name
word 2 Subprogram name
word 3 Location in ABS (first block address)
word 4 First absolute address in memory
word 5 Subprogram length (in words)

For special applications where fast loading is needed, user supplied subprograms or data can be placed on the ABS at system installation time.

SYSTEM LIBRARY FILE (LIB)

The LIB contains all the relocatable binary programs, subprograms, and COMPASS macros that can be called and used in a user program. This includes COBOL and FORTRAN object-time routines, special data processing packages such as LISA, SORT, the MSOS utilities, etc.

When a user calls a subprogram or macro from the library, MSOS uses RDCKF1 to load the loader from ABS. The loader absolutizes, links, and loads the subprograms that were called from the library.

DIRECTORY OF RELOCATABLE SUBPROGRAMS (DRS)

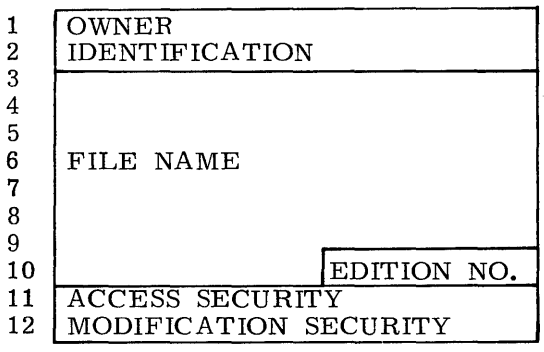
MSOS uses the DRS to locate subprograms and macros in the LIB. The DRS contains a three-word entry for each subprogram and COMPASS macro on the LIB. Each entry contains a subprogram or macro name and the block number in LIB on which the subprogram or macro starts. The format of the DRS is the same as the AUX library directory described in section 22.

MASS STORAGE LABEL FILE (LABELFILE)

The LABELFILE contains the label for each mass storage file allocated in the system. The mass storage file label is described in appendix G. The first label in LABELFILE is the label for LABELFILE itself.

MASS STORAGE LABEL ID FILE (IDFILE)

The IDFILE consists of one 12-word entry for each label in the label file. The 12 words are identical to the first 12 words in the LABELFILE entry for the same file. The 12-word entries are blocked in 120-word blocks.



ID File
Entry

To locate a label in LABELFILE, OCAREM searches the IDFILE to find an entry that matches the file FET (appendix F). Since all labels in LABELFILE are the same size and in the same order as in the IDFILE, OCAREM can calculate the block number in the LABELFILE that contains the first word entry of the label.

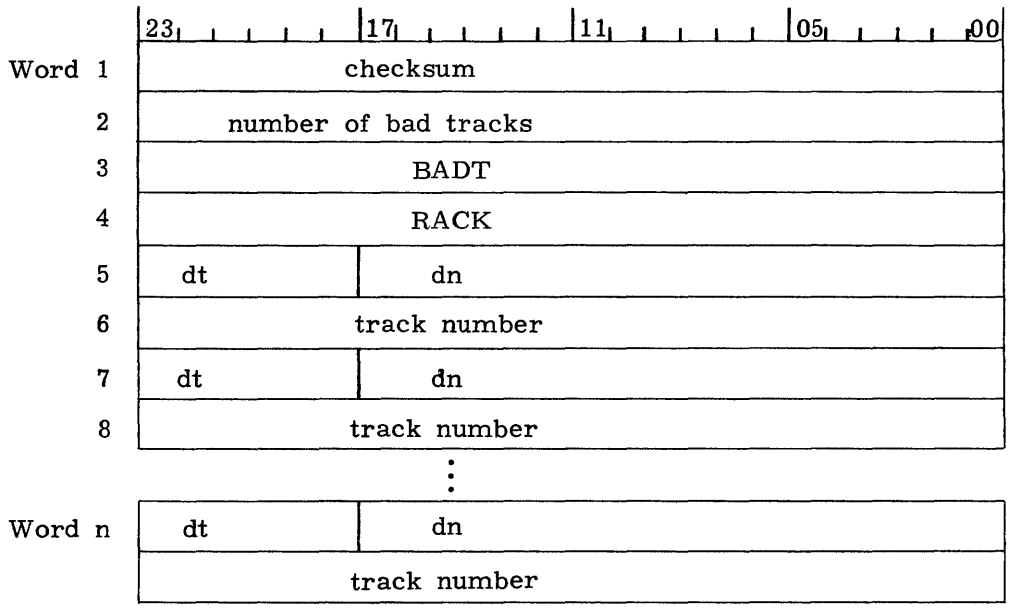
MASS STORAGE DEVICE LABEL FILE (MSDFILE)

The MSD contains the labels for each mass storage device (disk pack or disk file) that has been entered in the system with an ENTER statement. The mass storage device label is described in appendix G.

BAD TRACK FILE

The bad track file is a file listing each track on a mass storage device that has been downed by the system operator.† This file must be allocated by the system analyst before the bad track utilities can be used by the system operator.

The format of the bad track file is as follows:



dt — Device type (refer to appendix C)

dn — Device number assigned to the device when it was entered in the system

track number — Track number of the bad track

All unused entries are zero.

†When a track is downed by operator command, the track bit in the device track map in the MSD is set to one.

OCR I/O CODES I

I/O functions for the optical character reader (OCR) differ in certain particulars from the standard CIO requests. The five types of functions for the OCR are:

1. Data transfer

   Read from OCR into CONTROL DATA® 3195 Page Reader Controller buffer

   Read from 3195 buffer into core

2. Mirror control

   Read current mirror status (mirror status is the current horizontal position of the mirror)

   Position mirror and/or advance page

3. Other control functions

   Zero mirror

   Locate line

   Advance counter 1, 2, or 3

   Clear counter 1, 2, or 3

   Stop read

   Primary or secondary sort

   Alarm 1 or 2

   Mark document

4. Status

   The OCR function is identical to the standard CIO status function and uses the same calling sequence.

5. Format

   The OCR function is identical to the standard CIO format function and uses the same calling sequence.

Legal functions, format codes for the format calling sequence, and status bits for the OCR driver are described in the OCR driver table.

OCR PARAMETERS

The following parameters appear in OCR calling sequences and macros.

u	Logical unit number of OCR
jump	RTJ or UJP; in the macros, zero or blank = UJP
raddr	Reject address
i	Interrupt selections:
	0 No interrupt
	1 Interrupt on abnormal end-of-operation
	2,3 Interrupt on end-of-operation, normal or abnormal

Other parameters are defined as they appear.

OCR DATA TRANSFER

To read from OCR into 3195 buffer:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
p	RTJ	CIO	Call CIO
p+1	02	u,i	
p+2	jump	raddr	} Function code and parameters
p+3	m	p_1,0	
p+4		p_2	
p+5	normal return if i = 0	iaddr	
p+6	normal return if i ≠ 0		Continue program

m Mode change

00	No change
40	Scan 3 character heights, alphanumeric
41	Scan 3, alphabetic
42	Scan 3, numeric
43	Scan 3, mark sense (read zeros and filled zeros only)
44	Scan 2 character heights, alphanumeric
45	Scan 2, alphabetic
46	Scan 2, numeric
47	Scan 2, mark sense (read zeros and filled zeros only)

p_1 Initial mirror coordinate of data block; three octal digits, $0 \leq p_1 \leq 377_8$

p_2 Final mirror coordinate of data block; three octal digits, $p_1 < p_2 \leq 377_8$

The OCR macro for this function is:

```
LOCATION  | OPERATION, MODIFIERS  ADDRESS FIELD                    | COMMENTS
1       8 | 10          20                                         | 41
          | READOCR     (u,icoor,fcoor,m,s,raddr,jump,i,iaddr)
```

icoor Location containing initial mirror coordinate of the data block

fcoor Location containing final mirror coordinate of the data block

m Mode change

	0	No change
	ANM	Alphanumeric
	APH	Alphabetic
	NUM	Numeric
	MKS	Mark sense
s	2	Scan 2 character heights
	other symbol	Scan 3 character heights

In processing this function, the OCR driver determines the position of the mirror. If the mirror is to the right of the requested initial mirror coordinate, the driver sets the mirror position error bit in the status entry and rejects the request. If the mirror is to the left of the requested initial coordinate, the read operation occurs, starting at the requested initial coordinate. The function code is changed from 02 to 04 to prevent a channel from being declared busy.

To read from 3195 buffer into core:

```
LOCATION  | OPERATION, MODIFIERS  ADDRESS FIELD | COMMENTS
1       8 | 10          20                      | 41
p         | RTJ         CIO                     | Call CIO
p+1       | 01          u,i                     |
p+2       | jump        raddr                   |
p+3       | 00          fadd,1                  |  Function code
p+4       | 00          1                       |  and parameters
p+5       | normal      iaddr                   |
          | return      |                       |
          | if i = 0    |                       |
p+6       | normal return if i ≠ 0              | Continue program
```

fwaddr Address of first word in user's buffer

n Octal digits specifying number of words to read

Character addressing is not allowed since data input is in 12-bit bytes.

The OCR macro for this function is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	INPUT	(u,fwaddr,r,raddr,jump,i,iaddr)	

Parameters are as described previously.

OCR MIRROR

To read current mirror status:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
p	RTJ	CIO	Call CIO
p+1	01	u,i	
p+2	jump	raddr	
p+3	00	fwaddr,0	Function code and parameters
p+4	00	n	
p+5	normal return if i = 0	iaddr	
p+6	normal return if i ≠ 0		Continue program

fadd Address of location in which mirror status is to be stored

The OCR macro for this function is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	MIRSTAT	(u,mstat,raddr,jump)	

mstat Word address of location where mirror status is to be stored

To position mirror and/or advance page:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
p	RTJ	CIO	Call CIO
p+1	02	u,i	
p+2	jump	raddr	Function code and parameters
p+3		p_1,1	
p+4		p_2	
p+5	normal return if i = 0	iaddr	
p+6	normal return if i ≠ 0		Continue program

p_1 Four octal digits of the form Dxxx

D = 2 Forward mirror movement

3 Backward mirror movement

xxx Desired mirror coordinate; $0 \leq xxx \leq 377_8$

p_2 Four octal digits of the form xxx

xxx Number of lines to advance page

The OCR macro for this function is:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	POSITION	(u,nl,mc,raddr,jump,i,iaddr)	

nl Location containing the number of lines to advance; $0 \leq (nl) \leq 177_8$

mc Location containing the desired mirror coordinate; $0 \leq (mc) \leq 377_8$

When positioning the mirror for a read request, the user should place the mirror 10 to 20 coordinates to the left of the initial coordinate of the data block. This position compensates for document drift, hardware tolerances, and acceleration time.

If the direction specified for motion of the mirror is not the direction in which the mirror must move to reach the desired coordinate, the OCR driver rejects the request. The mirror position error bit is set in the status word for this type of reject.

When the request format is legal, the driver changes the function code to 04 to prevent a channel from being declared busy.

OTHER OCR CONTROL FUNCTIONS

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
p	RTJ	CIO	Call CIO
p+1	02	u,i	
p+2	jump	raddr	} Function code and parameters
p+3		p_1,2	
p+4		0	
p+5	normal return if i = 0	iaddr	
p+6	normal return if i ≠ 0		Continue program

p_1 Hardware control function code

Code	Meaning	
05	Zero mirror	These codes generate an external interrupt
07	Line locate	
47	Stop read	
50	Primary sort	
51	Secondary sort	
57	Mark document	
30	Advance counter 1	These codes do not generate an external interrupt. In the calling sequence, i should be zero and iaddr should be blank. The normal return is to p+5.
31	Clear counter 1	
32	Advance counter 2	
33	Clear counter 2	
34	Advance counter 3	
35	Clear counter 3	
52	Alarm 1	
53	Alarm 2	

The OCR driver rejects a request for an illegal hardware function. For legal requests which generate external interrupt, the driver changes the dummy function code 02 to 04 to prevent a channel from being declared busy. For legal requests for functions which do not generate external interrupt, 02 is changed to 14 (FORMAT) so that neither the channel nor the equipment is declared busy.

OCR macros for these functions are as follows:

Control functions which generate external interrupt:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ZMIRR	(u,raddr,jump,i,iaddr)	
	LINELOC	(u,raddr,jump,i,iaddr)	
	MARK	(u,raddr,jump,i,iaddr)	
	STPREAD	(u,raddr,jump,i,iaddr)	

ZMIRR	Zero mirror
LINELOC	Line locate
MARK	Mark document
STPREAD	Stop read

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	SORT	(u,S,raddr,jump,i,iaddr)	

S	When present, eject page to secondary hopper; when omitted, eject page to primary hopper

Control functions which do not generate external interrupt:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ADVANCC	(u,cn,raddr,jump)	
	CLEARC	(u,cn,raddr,jump)	

ADVANCC	Advance counter	
CLEARC	Clear counter	
cn	1, 2, or 3	Advance counter 1, 2, or 3
	Other symbol	Advance counter 1

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	ALARM	(u,a,raddr,jump)	

a	1 or blank	Alarm 1
	2	Alarm 2
	Other symbol	Alarm 1

OCR STATUS AND FORMAT CALLS

Calling sequences for status and format requests for the OCR are identical to the sequences for other peripheral equipment.

The OCR macros for these functions are:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	STATUS	(u,d)	

d	1	Dynamic status request
	0	Static status request

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 8	10	20	41
	FRMTOCR	(u,m,s,raddr,jump)	

m	Mode change:	
	0	No change
	ANM	Alphanumeric
	APH	Alphabetic
	NUM	Numeric
	MKS	Mark sense (read only zeros or filled zeros)
s	2	Scan 2 character heights
	other symbol	Scan 3 character heights

OCR REJECT AND ABORT

The normal CIO reject conditions and abort conditions apply in calls for the OCR. The following conditions result in reject by the OCR driver.

Function code in p+1 is not 01, 02, 13, or 14.

Mirror position error occurs; mirror position error status bit is set to 1.

Request contains an illegal hardware function code.

CIO passes the reject to the user program.

OPTICAL CHARACTER READER

Operation Performed	Permissible Function Code	Calling Sequence	Macro Name	Logical Unit (u)						
				1-57	58 CFO	59 CTO	60 INP	61 OUT	62 PUN	63 LIB
Read from OCR to 3195 buffer	02†	data transfer	READ OCR	yes						
Read from 3195 buffer into core	01	data transfer	INPUT	yes						
Read present mirror status	01	mirror control	MIRSTAT	yes						
Position mirror and/or advance Page	02†	mirror control	POSITION	yes						
Select hard-ware function 1	02†	other control	1	yes						
Check unit Status	13	status	STATUS	yes						
Select Format 2	14	format	FORMAT OCR	yes						

B=batch P=priority

† Dummy function code

1. Hardware function codes and macro names:

	Code	Meaning	Macro Name
Generate external interrupt	05	Zero mirror	ZMIRR
	07	Line locate	LINELOC
	47	Stop read	STPREAD
	50	Primary sort	SORT
	51	Secondary sort	SORT
	57	Mark document	MARK
Do not generate external interrupt	30	Advance counter 1	ADVANCC
	31	Clear counter 1	CLEARC
	32	Advance counter 2	ADVANCC
	33	Clear counter 2	CLEARC
	34	Advance counter 3	ADVANCC
	35	Clear counter 3	CLEARC
	52	Alarm 1	ALARM
	53	Alarm 2	ALARM

2. Format codes:

00	Scan 2, alphanumeric
01	Scan 3, alphabetic
02	Scan 3, numeric
03	Scan 3, mark sense
04	Scan 2, alphanumeric
05	Scan 2, alphabetic
06	Scan 2, numeric
07	Scan 2, mark sense

OCTAL/DECIMAL CONVERSIONS

TABLE OF POWERS OF TWO

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5
2 199 023 255 552	41	0.000 000 000 000 454 747 350 886 464 118 957 519 531 25
4 398 046 511 104	42	0.000 000 000 000 227 373 675 443 232 059 478 759 765 625
8 796 093 022 208	43	0.000 000 000 000 113 686 837 721 616 029 739 379 882 812 5
17 592 186 044 416	44	0.000 000 000 000 056 843 418 860 808 014 869 689 941 406 25
35 184 372 088 832	45	0.000 000 000 000 028 421 709 430 404 007 434 844 970 703 125
70 368 744 177 664	46	0.000 000 000 000 014 210 854 715 202 003 717 422 485 351 562 5
140 737 488 355 328	47	0.000 000 000 000 007 105 427 357 601 001 858 711 242 675 781 25
281 474 976 710 656	48	0.000 000 000 000 003 552 713 678 800 500 929 355 621 337 890 625
562 949 953 421 312	49	0.000 000 000 000 001 776 356 839 400 250 464 677 810 668 945 312 5
1 125 899 906 842 624	50	0.000 000 000 000 000 888 178 419 700 125 232 338 905 334 472 656 25
2 251 799 813 685 248	51	0.000 000 000 000 000 444 089 209 850 062 616 169 452 667 236 328 125
4 503 599 627 370 496	52	0.000 000 000 000 000 222 044 604 925 031 308 084 726 333 618 164 062 5
9 007 199 254 740 992	53	0.000 000 000 000 000 111 022 302 462 515 654 042 363 166 809 082 031 25
18 014 398 509 481 984	54	0.000 000 000 000 000 055 511 151 231 257 827 021 181 583 404 541 015 625
36 028 797 018 963 968	55	0.000 000 000 000 000 027 755 575 615 628 913 510 590 791 702 270 507 812 5
72 057 594 037 927 936	56	0.000 000 000 000 000 013 877 787 807 814 456 755 295 395 851 135 253 906 25
144 115 188 075 855 872	57	0.000 000 000 000 000 006 938 893 903 907 228 377 647 697 925 567 626 953 125
288 230 376 151 711 744	58	0.000 000 000 000 000 003 469 446 951 953 614 188 823 848 962 783 813 476 562 5
576 460 752 303 423 488	59	0.000 000 000 000 000 001 734 723 475 976 807 094 411 924 481 391 906 738 281 25
1 152 921 504 606 846 976	60	0.000 000 000 000 000 000 867 361 737 988 403 547 205 962 240 695 953 369 140 625

DECIMAL/BINARY POSITION TABLE

Largest Decimal Integer	Decimal Digits Req'd*	Number of Binary Digits	Largest Decimal Fraction
1		1	.5
3		2	.75
7		3	.875
15	1	4	.937 5
31		5	.968 75
63		6	.984 375
127	2	7	.992 187 5
255		8	.996 093 75
511		9	.998 046 875
1 023	3	10	.999 023 437 5
2 047		11	.999 511 718 75
4 095		12	.999 755 859 375
8 191		13	.999 877 929 687 5
16 383	4	14	.999 938 964 843 75
32 767		15	.999 969 482 421 875
65 535		16	.999 984 741 210 937 5
131 071	5	17	.999 992 370 605 468 75
262 143		18	.999 996 185 302 734 375
524 287		19	.999 998 092 651 367 187 5
1 048 575	6	20	.999 999 046 325 683 593 75
2 097 151		21	.999 999 523 162 841 796 875
4 194 303		22	.999 999 761 581 420 898 437 5
8 388 607		23	.999 999 880 790 710 449 218 75
16 777 215	7	24	.999 999 940 395 355 244 609 375
33 554 431		25	.999 999 970 197 677 612 304 687 5
67 108 863		26	.999 999 985 098 838 806 152 343 75
134 217 727	8	27	.999 999 992 549 419 403 076 171 875
268 435 455		28	.999 999 996 274 709 701 538 085 937 5
536 870 911		29	.999 999 998 137 354 850 769 042 968 75
1 073 741 823	9	30	.999 999 999 068 677 425 384 521 484 375
2 147 483 647		31	.999 999 999 534 338 712 692 260 742 187 5
4 294 967 295		32	.999 999 999 767 169 356 346 130 371 093 75
8 589 934 591		33	.999 999 999 883 584 678 173 065 185 546 875
17 179 869 183	10	34	.999 999 999 941 792 339 086 532 592 773 437 5
34 359 738 367		35	.999 999 999 970 896 169 543 266 296 386 718 75
68 719 476 735		36	.999 999 999 985 448 034 771 633 148 193 359 375
137 438 953 471	11	37	.999 999 999 992 724 042 385 816 574 096 679 687 5
274 877 906 943		38	.999 999 999 996 362 021 192 908 287 048 339 843 75
549 755 813 887		39	.999 999 999 998 181 010 596 454 143 524 169 921 875
1 099 511 627 775	12	40	.999 999 999 999 090 505 298 227 071 762 084 960 937 5
2 199 023 255 551		41	.999 999 999 999 545 252 649 113 535 881 042 480 468 75
4 398 046 511 103		42	.999 999 999 999 772 626 324 556 767 940 521 240 234 375
8 796 093 022 207		43	.999 999 999 999 886 313 162 278 383 970 260 620 117 187 5
17 592 186 044 415	13	44	.999 999 999 999 943 156 581 139 191 985 130 310 058 593 75
35 184 372 088 831		45	.999 999 999 999 971 578 290 569 595 992 565 155 029 296 875
70 368 744 177 663		46	.999 999 999 999 985 789 145 284 797 996 282 577 514 648 437 5
140 737 488 355 327	14	47	.999 999 999 999 992 894 572 642 398 998 141 288 757 324 218 75

*Larger numbers within a digit group should be checked for exact number of decimal digits required.

Examples of use:

1. *Q.* What is the largest decimal value that can be expressed by 36 binary digits?
   A. 68,719,476,735.
2. *Q.* How many decimal digits will be required to express a 22-bit number?
   A. 7 decimal digits.

OCTAL ARITHMETIC MATRICES

ADDITION-SUBTRACTION

0	1	2	3	4	5	6	7
1	2	3	4	5	6	7	10
2	3	4	5	6	7	10	11
3	4	5	6	7	10	11	12
4	5	6	7	10	11	12	13
5	6	7	10	11	12	13	14
6	7	10	11	12	13	14	15
7	10	11	12	13	14	15	16

MULTIPLICATION-DIVISION

0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7
2	2	4	6	10	12	14	16
3	3	6	11	14	17	22	25
4	4	10	14	20	24	30	34
5	5	12	17	24	31	36	43
6	6	14	22	30	36	44	52
7	7	16	25	34	43	52	61

CONSTANTS

π	=	3.14159 26535 89793 23846 26433 83279 50
$\sqrt{3}$	=	1.732 050 807 569
$\sqrt{10}$	=	3.162 277 660 1683
e	=	2.71828 18284 59045 23536
ln 2	=	0.69314 71805 599453
ln 10	=	2.30258 50929 94045 68402
$\log_{10} 2$	=	0.30102 99956 63981
$\log_{10} e$	=	0.43429 44819 03251 82765
$\log_{10} \log_{10} e$	=	9.63778 43113 **00537-10**
$\log_{10} \pi$	=	0.49714 98726 94133 85435
1 degree	=	0.01745 32925 11943 radians
1 radian	=	57.29577 95131 degrees
$\log_{10}(5)$	=	0.69897 00043 36019
7!	=	5040
8!	=	40320
9!	=	362,880
10!	=	3,628,800
11!	=	39,916,800
12!	=	479,001,600
13!	=	6,227,020,800
14!	=	87,178,291,200
15!	=	1,307,674,368,000
16!	=	20,922,789,888,000
$\frac{\pi}{180}$	=	0.01745 32925 19943 29576 92369 07684 9
$\left(\frac{\pi}{2}\right)^2$	=	2.4674 01100 27233 96
$\left(\frac{\pi}{2}\right)^3$	=	3.8757 84585 03747 74
$\left(\frac{\pi}{2}\right)^4$	=	6.0880 68189 62515 20
$\left(\frac{\pi}{2}\right)^5$	=	9.5631 15149 54004 49
$\left(\frac{\pi}{2}\right)^6$	=	15.0217 06149 61413 07
$\left(\frac{\pi}{2}\right)^7$	=	23.5960 40842 00618 62
$\left(\frac{\pi}{2}\right)^8$	=	37.0645 72481 52567 57
$\left(\frac{\pi}{2}\right)^9$	=	58.2208 97135 63712 59
$\left(\frac{\pi}{2}\right)^{10}$	=	91.4531 71363 36231 53
$\left(\frac{\pi}{2}\right)^{11}$	=	143.6543 05651 31374 95
$\left(\frac{\pi}{2}\right)^{12}$	=	225.6516 55645 350
$\left(\frac{\pi}{2}\right)^{13}$	=	354.4527 91822 91051 47
$\left(\frac{\pi}{2}\right)^{14}$	=	556.7731 43417 624

π^2	=	9.86960 44010 89358 61883 43909 9988
$2\pi^2$	=	19.73920 88021 78717 23766 87819 9976
$3\pi^2$	=	29.60881 32032 68075 85680 31729 9964
$4\pi^2$	=	39.47841 76043 57434 47533 75639 9952
$5\pi^2$	=	49.34802 20054 46793 09417 19549 9940
$6\pi^2$	=	59.21762 64065 36151 71300 63459 9928
$7\pi^2$	=	69.08723 08076 25510 33184 07369 9916
$8\pi^2$	=	78.95683 52087 14868 95067 51279 9904
$9\pi^2$	=	88.82643 96098 04227 56950 95189 9892

$\sqrt{2}$	=	1.414 213 562 373 095 048 801 688
$1 + \sqrt{2}$	=	2.414 213 562 373 095 048 801 688
$(1 + \sqrt{2})^2$	=	5.828 427 124 746 18
$(1 + \sqrt{2})^4$	=	33.970 562 748 477 08
$(1 + \sqrt{2})^6$	=	197.994 949 366 116 30
$(1 + \sqrt{2})^8$	=	1153.999 133 448 220 72
$(1 + \sqrt{2})^{10}$	=	6725.999 851 323 208 02
$(1 + \sqrt{2})^{12}$	=	39201.999 974 491 027 40
$(1 + \sqrt{2})^{14}$	=	228485.999 995 622 956 38
$(1 + \sqrt{2})^{16}$	=	1331713.999 999 246 711
$(1 + \sqrt{2})^{18}$	=	7761797.999 999 884 751

Sin .5	=	0.47942 55386 04203
Cos .5	=	0.87758 25618 90373
Tan .5	=	0.54630 24898 43790
Sin 1	=	0.84147 09848 07896
Cos 1	=	0.54030 23058 68140
Tan 1	=	1.55740 77246 5490
Sin 1.5	=	0.99749 49866 04054
Cos 1.5	=	0.07073 72016 67708
Tan 1.5	=	14.10141 99471 707

OCTAL-DECIMAL INTEGER CONVERSION TABLE

	0	1	2	3	4	5	6	7
0000	0000	0001	0002	0003	0004	0005	0006	0007
0010	0008	0009	0010	0011	0012	0013	0014	0015
0020	0016	0017	0018	0019	0020	0021	0022	0023
0030	0024	0025	0026	0027	0028	0029	0030	0031
0040	0032	0033	0034	0035	0036	0037	0038	0039
0050	0040	0041	0042	0043	0044	0045	0046	0047
0060	0048	0049	0050	0051	0052	0053	0054	0055
0070	0056	0057	0058	0059	0060	0061	0062	0063
0100	0064	0065	0066	0067	0068	0069	0070	0071
0110	0072	0073	0074	0075	0076	0077	0078	0079
0120	0080	0081	0082	0083	0084	0085	0086	0087
0130	0088	0089	0090	0091	0092	0093	0094	0095
0140	0096	0097	0098	0099	0100	0101	0102	0103
0150	0104	0105	0106	0107	0108	0109	0110	0111
0160	0112	0113	0114	0115	0116	0117	0118	0119
0170	0120	0121	0122	0123	0124	0125	0126	0127
0200	0128	0129	0130	0131	0132	0133	0134	0135
0210	0136	0137	0138	0139	0140	0141	0142	0143
0220	0144	0145	0146	0147	0148	0149	0150	0151
0230	0152	0153	0154	0155	0156	0157	0158	0159
0240	0160	0161	0162	0163	0164	0165	0166	0167
0250	0168	0169	0170	0171	0172	0173	0174	0175
0260	0176	0177	0178	0179	0180	0181	0182	0183
0270	0184	0185	0186	0187	0188	0189	0190	0191
0300	0192	0193	0194	0195	0196	0197	0198	0199
0310	0200	0201	0202	0203	0204	0205	0206	0207
0320	0208	0209	0210	0211	0212	0213	0214	0215
0330	0216	0217	0218	0219	0220	0221	0222	0223
0340	0224	0225	0226	0227	0228	0229	0230	0231
0350	0232	0233	0234	0235	0236	0237	0238	0239
0360	0240	0241	0242	0243	0244	0245	0246	0247
0370	0248	0249	0250	0251	0252	0253	0254	0255

	0	1	2	3	4	5	6	7
0400	0256	0257	0258	0259	0260	0261	0262	0263
0410	0264	0265	0266	0267	0268	0269	0270	0271
0420	0272	0273	0274	0275	0276	0277	0278	0279
0430	0280	0281	0282	0283	0284	0285	0286	0287
0440	0288	0289	0290	0291	0292	0293	0294	0295
0450	0296	0297	0298	0299	0300	0301	0302	0303
0460	0304	0305	0306	0307	0308	0309	0310	0311
0470	0312	0313	0314	0315	0316	0317	0318	0319
0500	0320	0321	0322	0323	0324	0325	0326	0327
0510	0328	0329	0330	0331	0332	0333	0334	0335
0520	0336	0337	0338	0339	0340	0341	0342	0343
0530	0344	0345	0346	0347	0348	0349	0350	0351
0540	0352	0353	0354	0355	0356	0357	0358	0359
0550	0360	0361	0362	0363	0364	0365	0366	0367
0560	0368	0369	0370	0371	0372	0373	0374	0375
0570	0376	0377	0378	0379	0380	0381	0382	0383
0600	0384	0385	0386	0387	0388	0389	0390	0391
0610	0392	0393	0394	0395	0396	0397	0398	0399
0620	0400	0401	0402	0403	0404	0405	0406	0407
0630	0408	0409	0410	0411	0412	0413	0414	0415
0640	0416	0417	0418	0419	0420	0421	0422	0423
0650	0424	0425	0426	0427	0428	0429	0430	0431
0660	0432	0433	0434	0435	0436	0437	0438	0439
0670	0440	0441	0442	0443	0444	0445	0446	0447
0700	0448	0449	0450	0451	0452	0453	0454	0455
0710	0456	0457	0458	0459	0460	0461	0462	0463
0720	0464	0465	0466	0467	0468	0469	0470	0471
0730	0472	0473	0474	0475	0476	0477	0478	0479
0740	0480	0481	0482	0483	0484	0485	0486	0487
0750	0488	0489	0490	0491	0492	0493	0494	0495
0760	0496	0497	0498	0499	0500	0501	0502	0503
0770	0504	0505	0506	0507	0508	0509	0510	0511

0000	0000
to	to
0777	0511
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
1000	0512	0513	0514	0515	0516	0517	0518	0519
1010	0520	0521	0522	0523	0524	0525	0526	0527
1020	0528	0529	0530	0531	0532	0533	0534	0535
1030	0536	0537	0538	0539	0540	0541	0542	0543
1040	0544	0545	0546	0547	0548	0549	0550	0551
1050	0552	0553	0554	0555	0556	0557	0558	0559
1060	0560	0561	0562	0563	0564	0565	0566	0567
1070	0568	0569	0570	0571	0572	0573	0574	0575
1100	0576	0577	0578	0579	0580	0581	0582	0583
1110	0584	0585	0586	0587	0588	0589	0590	0591
1120	0592	0593	0594	0595	0596	0597	0598	0599
1130	0600	0601	0602	0603	0604	0605	0606	0607
1140	0608	0609	0610	0611	0612	0613	0614	0615
1150	0616	0617	0618	0619	0620	0621	0622	0623
1160	0624	0625	0626	0627	0628	0629	0630	0631
1170	0632	0633	0634	0635	0636	0637	0638	0639
1200	0640	0641	0642	0643	0644	0645	0646	0647
1210	0648	0649	0650	0651	0652	0653	0654	0655
1220	0656	0657	0658	0659	0660	0661	0662	0663
1230	0664	0665	0666	0667	0668	0669	0670	0671
1240	0672	0673	0674	0675	0676	0677	0678	0679
1250	0680	0681	0682	0683	0684	0685	0686	0687
1260	0688	0689	0690	0691	0692	0693	0694	0695
1270	0696	0697	0698	0699	0700	0701	0702	0703
1300	0704	0705	0706	0707	0708	0709	0710	0711
1310	0712	0713	0714	0715	0716	0717	0718	0719
1320	0720	0721	0722	0723	0724	0725	0726	0727
1330	0728	0729	0730	0731	0732	0733	0734	0735
1340	0736	0737	0738	0739	0740	0741	0742	0743
1350	0744	0745	0746	0747	0748	0749	0750	0751
1360	0752	0753	0754	0755	0756	0757	0758	0759
1370	0760	0761	0762	0763	0764	0765	0766	0767

	0	1	2	3	4	5	6	7
1400	0768	0769	0770	0771	0772	0773	0774	0775
1410	0776	0777	0778	0779	0780	0781	0782	0783
1420	0784	0785	0786	0787	0788	0789	0790	0791
1430	0792	0793	0794	0795	0796	0797	0798	0799
1440	0800	0801	0802	0803	0804	0805	0806	0807
1450	0808	0809	0810	0811	0812	0813	0814	0815
1460	0816	0817	0818	0819	0820	0821	0822	0823
1470	0824	0825	0826	0827	0828	0829	0830	0831
1500	0832	0833	0834	0835	0836	0837	0838	0839
1510	0840	0841	0842	0843	0844	0845	0846	0847
1520	0848	0849	0850	0851	0852	0853	0854	0855
1530	0856	0857	0858	0859	0860	0861	0862	0863
1540	0864	0865	0866	0867	0868	0869	0870	0871
1550	0872	0873	0874	0875	0876	0877	0878	0879
1560	0880	0881	0882	0883	0884	0885	0886	0887
1570	0888	0889	0890	0891	0892	0893	0894	0895
1600	0896	0897	0898	0899	0900	0901	0902	0903
1610	0904	0905	0906	0907	0908	0909	0910	0911
1620	0912	0913	0914	0915	0916	0917	0918	0919
1630	0920	0921	0922	0923	0924	0925	0926	0927
1640	0928	0929	0930	0931	0932	0933	0934	0935
1650	0936	0937	0938	0939	0940	0941	0942	0943
1660	0944	0945	0946	0947	0948	0949	0950	0951
1670	0952	0953	0954	0955	0956	0957	0958	0959
1700	0960	0961	0962	0963	0964	0965	0966	0967
1710	0968	0969	0970	0971	0972	0973	0974	0975
1720	0976	0977	0978	0979	0980	0981	0982	0983
1730	0984	0985	0986	0987	0988	0989	0990	0991
1740	0992	0993	0994	0995	0996	0997	0998	0999
1750	1000	1001	1002	1003	1004	1005	1006	1007
1760	1008	1009	1010	1011	1012	1013	1014	1015
1770	1016	1017	1018	1019	1020	1021	1022	1023

1000	0512
to	to
1777	1023
(Octal)	(Decimal)

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

2000 (Octal)	1024 (Decimal)
to	to
2777 (Octal)	1535 (Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
2000	1024	1025	1026	1027	1028	1029	1030	1031
2010	1032	1033	1034	1035	1036	1037	1038	1039
2020	1040	1041	1042	1043	1044	1045	1046	1047
2030	1048	1049	1050	1051	1052	1053	1054	1055
2040	1056	1057	1058	1059	1060	1061	1062	1063
2050	1064	1065	1066	1067	1068	1069	1070	1071
2060	1072	1073	1074	1075	1076	1077	1078	1079
2070	1080	1081	1082	1083	1084	1085	1086	1087
2100	1088	1089	1090	1091	1092	1093	1094	1095
2110	1096	1097	1098	1099	1100	1101	1102	1103
2120	1104	1105	1106	1107	1108	1109	1110	1111
2130	1112	1113	1114	1115	1116	1117	1118	1119
2140	1120	1121	1122	1123	1124	1125	1126	1127
2150	1128	1129	1130	1131	1132	1133	1134	1135
2160	1136	1137	1138	1139	1140	1141	1142	1143
2170	1144	1145	1146	1147	1148	1149	1150	1151
2200	1152	1153	1154	1155	1156	1157	1158	1159
2210	1160	1161	1162	1163	1164	1165	1166	1167
2220	1168	1169	1170	1171	1172	1173	1174	1175
2230	1176	1177	1178	1179	1180	1181	1182	1183
2240	1184	1185	1186	1187	1188	1189	1190	1191
2250	1192	1193	1194	1195	1196	1197	1198	1199
2260	1200	1201	1202	1203	1204	1205	1206	1207
2270	1208	1209	1210	1211	1212	1213	1214	1215
2300	1216	1217	1218	1219	1220	1221	1222	1223
2310	1224	1225	1226	1227	1228	1229	1230	1231
2320	1232	1233	1234	1235	1236	1237	1238	1239
2330	1240	1241	1242	1243	1244	1245	1246	1247
2340	1248	1249	1250	1251	1252	1253	1254	1255
2350	1256	1257	1258	1259	1260	1261	1262	1263
2360	1264	1265	1266	1267	1268	1269	1270	1271
2370	1272	1273	1274	1275	1276	1277	1278	1279

	0	1	2	3	4	5	6	7
2400	1280	1281	1282	1283	1284	1285	1286	1287
2410	1288	1289	1290	1291	1292	1293	1294	1295
2420	1296	1297	1298	1299	1300	1301	1302	1303
2430	1304	1305	1306	1307	1308	1309	1310	1311
2440	1312	1313	1314	1315	1316	1317	1318	1319
2450	1320	1321	1322	1323	1324	1325	1326	1327
2460	1328	1329	1330	1331	1332	1333	1334	1335
2470	1336	1337	1338	1339	1340	1341	1342	1343
2500	1344	1345	1346	1347	1348	1349	1350	1351
2510	1352	1353	1354	1355	1356	1357	1358	1359
2520	1360	1361	1362	1363	1364	1365	1366	1367
2530	1368	1369	1370	1371	1372	1373	1374	1375
2540	1376	1377	1378	1379	1380	1381	1382	1383
2550	1384	1385	1386	1387	1388	1389	1390	1391
2560	1392	1393	1394	1395	1396	1397	1398	1399
2570	1400	1401	1402	1403	1404	1405	1406	1407
2600	1408	1409	1410	1411	1412	1413	1414	1415
2610	1416	1417	1418	1419	1420	1421	1422	1423
2620	1424	1425	1426	1427	1428	1429	1430	1431
2630	1432	1433	1434	1435	1436	1437	1438	1439
2640	1440	1441	1442	1443	1444	1445	1446	1447
2650	1448	1449	1450	1451	1452	1453	1454	1455
2660	1456	1457	1458	1459	1460	1461	1462	1463
2670	1464	1465	1466	1467	1468	1469	1470	1471
2700	1472	1473	1474	1475	1476	1477	1478	1479
2710	1480	1481	1482	1483	1484	1485	1486	1487
2720	1488	1489	1490	1491	1492	1493	1494	1495
2730	1496	1497	1498	1499	1500	1501	1502	1503
2740	1504	1505	1506	1507	1508	1519	1510	1511
2750	1512	1513	1514	1515	1516	1517	1518	1519
2760	1520	1521	1522	1523	1524	1525	1526	1527
2770	1528	1529	1530	1531	1532	1533	1534	1535

3000 (Octal)	1536 (Decimal)
to	to
3777 (Octal)	2047 (Decimal)

	0	1	2	3	4	5	6	7
3000	1536	1537	1538	1539	1540	1541	1542	1543
3010	1544	1545	1546	1547	1548	1549	1550	1551
3020	1552	1553	1554	1555	1556	1557	1558	1559
3030	1560	1561	1562	1563	1564	1565	1566	1567
3040	1568	1569	1570	1571	1572	1573	1574	1575
3050	1576	1577	1578	1579	1580	1581	1582	1583
3060	1584	1585	1586	1587	1588	1589	1590	1591
3070	1592	1593	1594	1595	1596	1597	1598	1599
3100	1600	1601	1602	1603	1604	1605	1606	1607
3110	1608	1609	1610	1611	1612	1613	1614	1615
3120	1616	1617	1618	1619	1620	1621	1622	1623
3130	1624	1625	1626	1627	1628	1629	1630	1631
3140	1632	1633	1634	1635	1636	1637	1638	1639
3150	1640	1641	1642	1643	1644	1645	1646	1647
3160	1648	1649	1650	1651	1652	1653	1654	1655
3170	1656	1657	1658	1659	1660	1661	1662	1663
3200	1664	1665	1666	1667	1668	1669	1670	1671
3210	1672	1673	1674	1675	1676	1677	1678	1679
3220	1680	1681	1682	1683	1684	1685	1686	1687
3230	1688	1689	1690	1691	1692	1693	1694	1695
3240	1696	1697	1698	1699	1700	1701	1702	1703
3250	1704	1705	1706	1707	1708	1709	1710	1711
3260	1712	1713	1714	1715	1716	1717	1718	1719
3270	1720	1721	1722	1723	1724	1725	1726	1727
3300	1728	1729	1730	1731	1732	1733	1734	1735
3310	1736	1737	1738	1739	1740	1741	1742	1743
3320	1744	1745	1746	1747	1748	1749	1750	1751
3330	1752	1753	1754	1755	1756	1757	1758	1759
3340	1760	1761	1762	1763	1764	1765	1766	1767
3350	1768	1769	1770	1771	1772	1773	1774	1775
3360	1776	1777	1778	1779	1780	1781	1782	1783
3370	1784	1785	1786	1787	1788	1789	1790	1791

	0	1	2	3	4	5	6	7
3400	1792	1793	1794	1795	1796	1797	1798	1799
3410	1800	1801	1802	1803	1804	1805	1806	1807
3420	1808	1809	1810	1811	1812	1813	1814	1815
3430	1816	1817	1818	1819	1820	1821	1822	1823
3440	1824	1825	1826	1827	1828	1829	1830	1831
3450	1832	1833	1834	1835	1836	1837	1838	1839
3460	1840	1841	1842	1843	1844	1845	1846	1847
3470	1848	1849	1850	1851	1852	1853	1854	1855
3500	1856	1857	1858	1859	1860	1861	1862	1863
3510	1864	1865	1866	1867	1868	1869	1870	1871
3520	1872	1873	1874	1875	1876	1877	1878	1879
3530	1880	1881	1882	1883	1884	1885	1886	1887
3540	1888	1889	1890	1891	1892	1893	1894	1895
3550	1896	1897	1898	1899	1900	1901	1902	1903
3560	1904	1905	1906	1907	1908	1909	1910	1911
3570	1912	1913	1914	1915	1916	1917	1918	1919
3600	1920	1921	1922	1923	1924	1925	1926	1927
3610	1928	1929	1930	1931	1932	1933	1934	1935
3620	1936	1937	1938	1939	1940	1941	1942	1943
3630	1944	1945	1946	1947	1948	1949	1950	1951
3640	1952	1953	1954	1955	1956	1957	1958	1959
3650	1960	1961	1962	1963	1964	1965	1966	1967
3660	1968	1969	1970	1971	1972	1973	1974	1975
3670	1976	1977	1978	1979	1980	1981	1982	1983
3700	1984	1985	1986	1987	1988	1989	1990	1991
3710	1992	1993	1994	1995	1996	1997	1998	1999
3720	2000	2001	2002	2003	2004	2005	2006	2007
3730	2008	2009	2010	2011	2012	2013	2014	2015
3740	2016	2017	2018	2019	2020	2021	2022	2023
3750	2024	2025	2026	2027	2028	2029	2030	2031
3760	2032	2033	2034	2035	2036	2037	2038	2039
3770	2040	2041	2042	2043	2044	2045	2046	2047

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

	0	1	2	3	4	5	6	7
4000	2048	2049	2050	2051	2052	2053	2054	2055
4010	2056	2057	2058	2059	2060	2061	2062	2063
4020	2064	2065	2066	2067	2068	2069	2070	2071
4030	2072	2073	2074	2075	2076	2077	2078	2079
4040	2080	2081	2082	2083	2084	2085	2086	2087
4050	2088	2089	2090	2091	2092	2093	2094	2095
4060	2096	2097	2098	2099	2100	2101	2102	2103
4070	2104	2105	2106	2107	2108	2109	2110	2111
4100	2112	2113	2114	2115	2116	2117	2118	2119
4110	2120	2121	2122	2123	2124	2125	2126	2127
4120	2128	2129	2130	2131	2132	2133	2134	2135
4130	2136	2137	2138	2139	2140	2141	2142	2143
4140	2144	2145	2146	2147	2148	2149	2150	2151
4150	2152	2153	2154	2155	2156	2157	2158	2159
4160	2160	2161	2162	2163	2164	2165	2166	2167
4170	2168	2169	2170	2171	2172	2173	2174	2175
4200	2176	2177	2178	2179	2180	2181	2182	2183
4210	2184	2185	2186	2187	2188	2189	2190	2191
4220	2192	2193	2194	2195	2196	2197	2198	2199
4230	2200	2201	2202	2203	2204	2205	2206	2207
4240	2208	2209	2210	2211	2212	2213	2214	2215
4250	2216	2217	2218	2219	2220	2221	2222	2223
4260	2224	2225	2226	2227	2228	2229	2230	2231
4270	2232	2233	2234	2235	2236	2237	2238	2239
4300	2240	2241	2242	2243	2244	2245	2246	2247
4310	2248	2249	2250	2251	2252	2253	2254	2255
4320	2256	2257	2258	2259	2260	2261	2262	2263
4330	2264	2265	2266	2267	2268	2269	2270	2271
4340	2272	2273	2274	2275	2276	2277	2278	2279
4350	2280	2281	2282	2283	2284	2285	2286	2287
4360	2288	2289	2290	2291	2292	2293	2294	2295
4370	2296	2297	2298	2299	2300	2301	2302	2303

	0	1	2	3	4	5	6	7
4400	2304	2305	2306	2307	2308	2309	2310	2311
4410	2312	2313	2314	2315	2316	2317	2318	2319
4420	2320	2321	2322	2323	2324	2325	2326	2327
4430	2328	2329	2330	2331	2332	2333	2334	2335
4440	2336	2337	2338	2339	2340	2341	2342	2343
4450	2344	2345	2346	2347	2348	2349	2350	2351
4460	2352	2353	2354	2355	2356	2357	2358	2359
4470	2360	2361	2362	2363	2364	2365	2366	2367
4500	2368	2369	2370	2371	2372	2373	2374	2375
4510	2376	2377	2378	2379	2380	2381	2382	2383
4520	2384	2385	2386	2387	2388	2389	2390	2391
4530	2392	2393	2394	2395	2396	2397	2398	2399
4540	2400	2401	2402	2403	2404	2405	2406	2407
4550	2408	2409	2410	2411	2412	2413	2414	2415
4560	2416	2417	2418	2419	2420	2421	2422	2423
4570	2424	2425	2426	2427	2428	2429	2430	2431
4600	2432	2433	2434	2435	2436	2437	2438	2439
4610	2440	2441	2442	2443	2444	2445	2446	2447
4620	2448	2449	2450	2451	2452	2453	2454	2455
4630	2456	2457	2458	2459	2460	2461	2462	2463
4640	2464	2465	2466	2467	2468	2469	2470	2471
4650	2472	2473	2474	2475	2476	2477	2478	2479
4660	2480	2481	2482	2483	2484	2485	2486	2487
4670	2488	2489	2490	2491	2492	2493	2494	2495
4700	2496	2497	2498	2499	2500	2501	2502	2503
4710	2504	2505	2506	2507	2508	2509	2510	2511
4720	2512	2513	2514	2515	2516	2517	2518	2519
4730	2520	2521	2522	2523	2524	2525	2526	2527
4740	2528	2529	2530	2531	2532	2533	2534	2535
4750	2536	2537	2538	2539	2540	2541	2542	2543
4760	2544	2545	2546	2547	2548	2549	2550	2551
4770	2552	2553	2554	2555	2556	2557	2558	2559

4000	2048
to	to
4777	2559
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
5000	2560	2561	2562	2563	2564	2565	2566	2567
5010	2568	2569	2570	2571	2572	2573	2574	2575
5020	2576	2577	2578	2579	2580	2581	2582	2583
5030	2584	2585	2586	2587	2588	2589	2590	2591
5040	2592	2593	2594	2595	2596	2597	2598	2599
5050	2600	2601	2602	2603	2604	2605	2606	2607
5060	2608	2609	2610	2611	2612	2613	2614	2615
5070	2616	2617	2618	2619	2620	2621	2622	2623
5100	2624	2625	2626	2627	2628	2629	2630	2631
5110	2632	2633	2634	2635	2636	2637	2638	2639
5120	2640	2641	2642	2643	2644	2645	2646	2647
5130	2648	2649	2650	2651	2652	2653	2654	2655
5140	2656	2657	2658	2659	2660	2661	2662	2663
5150	2664	2665	2666	2667	2668	2669	2670	2671
5160	2672	2673	2674	2675	2676	2677	2678	2679
5170	2680	2681	2682	2683	2684	2685	2686	2687
5200	2688	2689	2690	2691	2692	2693	2694	2695
5210	2696	2697	2698	2699	2700	2701	2702	2703
5220	2704	2705	2706	2707	2708	2709	2710	2711
5230	2712	2713	2714	2715	2716	2717	2718	2719
5240	2720	2721	2722	2723	2724	2725	2726	2727
5250	2728	2729	2730	2731	2732	2733	2734	2735
5260	2736	2737	2738	2739	2740	2741	2742	2743
5270	2744	2745	2746	2747	2748	2749	2750	2751
5300	2752	2753	2754	2755	2756	2757	2758	2759
5310	2760	2761	2762	2763	2764	2765	2766	2767
5320	2768	2769	2770	2771	2772	2773	2774	2775
5330	2776	2777	2778	2779	2780	2781	2782	2783
5340	2784	2785	2786	2787	2788	2789	2790	2791
5350	2792	2793	2794	2795	2796	2797	2798	2799
5360	2800	2801	2802	2803	2804	2805	2806	2807
5370	2808	2809	2810	2811	2812	2813	2814	2815

	0	1	2	3	4	5	6	7
5400	2816	2817	2818	2819	2820	2821	2822	2823
5410	2824	2825	2826	2827	2828	2829	2830	2831
5420	2832	2833	2834	2835	2836	2837	2838	2839
5430	2840	2841	2842	2843	2844	2845	2846	2847
5440	2848	2849	2850	2851	2852	2853	2854	2855
5450	2856	2857	2858	2859	2860	2861	2862	2863
5460	2864	2865	2866	2867	2868	2869	2870	2871
5470	2872	2873	2874	2875	2876	2877	2878	2879
5500	2880	2881	2882	2883	2884	2885	2886	2887
5510	2888	2889	2890	2891	2892	2893	2894	2895
5520	2896	2897	2898	2899	2900	2901	2902	2903
5530	2904	2905	2906	2907	2908	2909	2910	2911
5540	2912	2913	2914	2915	2916	2917	2918	2919
5550	2920	2921	2922	2923	2924	2925	2926	2927
5560	2928	2929	2930	2931	2932	2933	2934	2935
5570	2936	2937	2938	2939	2940	2941	2942	2943
5600	2944	2945	2946	2947	2948	2949	2950	2951
5610	2952	2953	2954	2955	2956	2957	2958	2959
5620	2960	2961	2962	2963	2964	2965	2966	2967
5630	2968	2969	2970	2971	2972	2973	2974	2975
5640	2976	2977	2978	2979	2980	2981	2982	2983
5650	2984	2985	2986	2987	2988	2989	2990	2991
5660	2992	2993	2994	2995	2996	2997	2998	2999
5670	3000	3001	3002	3003	3004	3005	3006	3007
5700	3008	3009	3010	3011	3012	3013	3014	3015
5710	3016	3017	3018	3019	3020	3021	3022	3023
5720	3024	3025	3026	3027	3028	3029	3030	3031
5730	3032	3033	3034	3035	3036	3037	3038	3039
5740	3040	3041	3042	3043	3044	3045	3046	3047
5750	3048	3049	3050	3051	3052	3053	3054	3055
5760	3056	3057	3058	3059	3060	3061	3062	3063
5770	3064	3065	3066	3067	3068	3069	3070	3071

5000	2560
to	to
5777	3071
(Octal)	(Decimal)

OCTAL-DECIMAL INTEGER CONVERSION TABLE (Cont'd)

6000	3072
to	to
6777	3583
(Octal)	(Decimal)

Octal	Decimal
10000 -	4096
20000 -	8192
30000 -	12288
40000 -	16384
50000 -	20480
60000 -	24576
70000 -	28672

	0	1	2	3	4	5	6	7
6000	3072	3073	3074	3075	3076	3077	3078	3079
6010	3080	3081	3082	3083	3084	3085	3086	3087
6020	3088	3089	3090	3091	3092	3093	3094	3095
6030	3096	3097	3098	3099	3100	3101	3102	3103
6040	3104	3105	3106	3107	3108	3109	3110	3111
6050	3112	3113	3114	3115	3116	3117	3118	3119
6060	3120	3121	3122	3123	3124	3125	3126	3127
6070	3128	3129	3130	3131	3132	3133	3134	3135
6100	3136	3137	3138	3139	3140	3141	3142	3143
6110	3144	3145	3146	3147	3148	3149	3150	3151
6120	3152	3153	3154	3155	3156	3157	3158	3159
6130	3160	3161	3162	3163	3164	3165	3166	3167
6140	3168	3169	3170	3171	3172	3173	3174	3175
6150	3176	3177	3178	3179	3180	3181	3182	3183
6160	3184	3185	3186	3187	3188	3189	3190	3191
6170	3192	3193	3194	3195	3196	3197	3198	3199
6200	3200	3201	3202	3203	3204	3205	3206	3207
6210	3208	3209	3210	3211	3212	3213	3214	3215
6220	3216	3217	3218	3219	3220	3221	3222	3223
6230	3224	3225	3226	3227	3228	3229	3230	3231
6240	3232	3233	3234	3235	3236	3237	3238	3239
6250	3240	3241	3242	3243	3244	3245	3246	3247
6260	3248	3249	3250	3251	3252	3253	3254	3255
6270	3256	3257	3258	3259	3260	3261	3262	3263
6300	3264	3265	3266	3267	3268	3269	3270	3271
6310	3272	3273	3274	3275	3276	3277	3278	3279
6320	3280	3281	3282	3283	3284	3285	3286	3287
6330	3288	3289	3290	3291	3292	3293	3294	3295
6340	3296	3297	3298	3299	3300	3301	3302	3303
6350	3304	3305	3306	3307	3308	3309	3310	3311
6360	3312	3313	3314	3315	3316	3317	3318	3319
6370	3320	3321	3322	3323	3324	3325	3326	3327

	0	1	2	3	4	5	6	7
6400	3328	3329	3330	3331	3332	3333	3334	3335
6410	3336	3337	3338	3339	3340	3341	3342	3343
6420	3344	3345	3346	3347	3348	3349	3350	3351
6430	3352	3353	3354	3355	3356	3357	3358	3359
6440	3360	3361	3362	3363	3364	3365	3366	3367
6450	3368	3369	3370	3371	3372	3373	3374	3375
6460	3376	3377	3378	3379	3380	3381	3382	3383
6470	3384	3385	3386	3387	3388	3389	3390	3391
6500	3392	3393	3394	3395	3396	3397	3398	3399
6510	3400	3401	3402	3403	3404	3405	3406	3407
6520	3408	3409	3410	3411	3412	3413	3414	3415
6530	3416	3417	3418	3419	3420	3421	3422	3423
6540	3424	3425	3426	3427	3428	3429	3430	3431
6550	3432	3433	3434	3435	3436	3437	3438	3439
6560	3440	3441	3442	3443	3444	3445	3446	3447
6570	3448	3449	3450	3451	3452	3453	3454	3455
6600	3456	3457	3458	3459	3460	3461	3462	3463
6610	3464	3465	3466	3467	3468	3469	3470	3471
6620	3472	3473	3474	3475	3476	3477	3478	3479
6630	3480	3481	3482	3483	3484	3485	3486	3487
6640	3488	3489	3490	3491	3492	3493	3494	3495
6650	3496	3497	3498	3499	3500	3501	3502	3503
6660	3504	3505	3506	3507	3508	3509	3510	3511
6670	3512	3513	3514	3515	3516	3517	3518	3519
6700	3520	3521	3522	3523	3524	3525	3526	3527
6710	3528	3529	3530	3531	3532	3533	3534	3535
6720	3536	3537	3538	3539	3540	3541	3542	3543
6730	3544	3545	3546	3547	3548	3549	3550	3551
6740	3552	3553	3554	3555	3556	3557	3558	3559
6750	3560	3561	3562	3563	3564	3565	3566	3567
6760	3568	3569	3570	3571	3572	3573	3574	3575
6770	3576	3577	3578	3579	3580	3581	3582	3583

7000	3584
to	to
7777	4095
(Octal)	(Decimal)

	0	1	2	3	4	5	6	7
7000	3584	3585	3586	3587	3588	3589	3590	3591
7010	3592	3593	3594	3595	3496	3497	3598	3599
7020	3600	3601	3602	3603	3604	3605	3606	3607
7030	3608	3609	3610	3611	3612	3613	3614	3615
7040	3616	3617	3618	3619	3620	3621	3622	3623
7050	3624	3625	3626	3627	3628	3629	3630	3631
7060	3632	3633	3634	3635	3636	3637	3638	3639
7070	3640	3641	3642	3643	3644	3645	3646	3647
7100	3648	3649	3650	3651	3652	3653	3654	3655
7110	3656	3657	3658	3659	3660	3661	3662	3663
7120	3664	3665	3666	3667	3668	3669	3670	3671
7130	3672	3673	3674	3675	3676	3677	3678	3679
7140	3680	3681	3682	3683	3684	3685	3686	3687
7150	3688	3689	3690	3691	3692	3693	3694	3695
7160	3696	3697	3698	3699	3700	3701	3702	3703
7170	3704	3705	3706	3707	3708	3709	3710	3711
7200	3712	3713	3714	3715	3716	3717	3718	3719
7210	3720	3721	3722	3723	3724	3725	3726	3727
7220	3728	3729	3730	3731	3732	3733	3734	3735
7230	3736	3737	3738	3739	3740	3741	3742	3743
7240	3744	3745	3746	3747	3748	3749	3750	3751
7250	3752	3753	3754	3755	3756	3757	3758	3759
7260	3760	3761	3762	3763	3764	3765	3766	3767
7270	3768	3769	3770	3771	3772	3773	3774	3775
7300	3776	3777	3778	3779	3780	3781	3782	3783
7310	3784	3785	3786	3787	3788	3789	3790	3791
7320	3792	3793	3794	3795	3796	3797	3798	3799
7330	3800	3801	3802	3803	3804	3805	3806	3807
7340	3808	3809	3810	3811	3812	3813	3814	3815
7350	3816	3817	3818	3819	3820	3821	3822	3823
7360	3824	3825	3826	3827	3828	3829	3830	3831
7370	3832	3833	3834	3835	3836	3837	3838	3839

	0	1	2	3	4	5	6	7
7400	3840	3841	3842	3843	3844	3845	3846	3847
7410	3848	3849	3850	3851	3852	3853	3854	3855
7420	3856	3857	3858	3859	3860	3861	3862	3863
7430	3864	3865	3866	3867	3868	3869	3870	3871
7440	3872	3873	3874	3875	3876	3877	3878	3879
7450	3880	3881	3882	3883	3884	3885	3886	3887
7460	3888	3889	3890	3891	3892	3893	3894	3895
7470	3896	3897	3898	3899	3900	3901	3902	3903
7500	3904	3905	3906	3907	3908	3909	3910	3911
7510	3912	3913	3914	3915	3916	3917	3918	3919
7520	3920	3921	3922	3923	3924	3925	3926	3927
7530	3928	3929	3930	3931	3932	3933	3934	3935
7540	3936	3937	3938	3939	3940	3941	3942	3943
7550	3944	3945	3946	3947	3948	3949	3950	3951
7560	3952	3953	3954	3955	3956	3957	3958	3959
7570	3960	3961	3962	3963	3964	3965	3966	3967
7600	3968	3969	3970	3971	3972	3973	3974	3975
7610	3976	3977	3978	3979	3980	3981	3982	3983
7620	3984	3985	3986	3987	3988	3989	3990	3991
7630	3992	3993	3994	3995	3996	3997	3998	3999
7640	4000	4001	4002	4003	4004	4005	4006	4007
7650	4008	4009	4010	4011	4012	4013	4014	4015
7660	4016	4017	4018	4019	4020	4021	4022	4023
7670	4024	4025	4026	4027	4028	4029	4030	4031
7700	4032	4033	4034	4035	4036	4037	4038	4039
7710	4040	4041	4042	4043	4044	4045	4046	4047
7720	4048	4049	4050	4051	4052	4053	4054	4055
7730	4056	4057	4058	4059	4060	4061	4062	4063
7740	4064	4065	4066	4067	4068	4069	4070	4071
7750	4072	4073	4074	4075	4076	4077	4078	4079
7760	4080	4081	4082	4083	4084	4085	4086	4087
7770	4088	4089	4090	4091	4092	4093	4094	4095

OCTAL-DECIMAL FRACTION CONVERSION TABLE

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

OCTAL-DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

OCTAL-DECIMAL FRACTION CONVERSION TABLE (Cont'd)

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

ABBREVIATIONS AND ACRONYMS

The list of abbreviations does not include COMPASS pseudo instructions, machine language instructions, programmer control cards, operator control statements, or diagnostics.

ACC	Standard accounting unit
ABS	Library file containing absolutized subprograms and routine.
ALGOL	Algorithmic language compiler
AP	Alternate processor
APC	Automatic peripheral control
AUT	Available unit table
BCD	Binary coded decimal
CFO	Comment-from-operator unit; console typewriter
CIC	Central interrupt control routine
CIO	Central input/output routine
COBOL	Common business oriented language
COMPASS	Comprehensive assembly system
COSY	Compressed symbolic library program
CPU	Central processing unit
CR	Card reader
CST	Channel status table
CTO	Comment-to-operator unit; console typewriter
DINT.	Disable interrupt routine
DRS	Library directory file
EINT.	Enable interrupt routine
ELD	End-loader-declaration card
EOF	End-of-file
EOT	End-of-tape mark
EPT	Loader entry point name card
EST	Equipment status table
fca	First character address
FDT	File description table
FET	File environment table
FORTRAN	Formula translation compiler
FOT	File ordinal table
fwa	First word address

iaddr	Interrupt address
IDC	Subprogram identification card
IDFILE	Directory for LABELFILE entries
IMR	Interrupt mask register
INP	Standard input unit; typically a card reader
I/O	Input/output
LABELFILE	A file containing the label of each allocated mass storage file
lca	Last character address
LED	Loader equipment declaration card
LGO	Load-and-go unit
LIB	Standard system library file. Contains relocatable subprograms and routines.
L-MSIO	Logical MSIO
LRL	Local reference list loader card
lwa	Last word address
MAP	Listing the subprogram entry points and locations in core
MSD	Mass storage device/pack
MSDFILE	A file containing the mass storage device labels
MT	Magnetic tape
OCAREM	Mass storage file management routine
OCR	Optical character reader
OUT	Standard output unit; typically a printer
PERT	Programmed evaluation and review technique
PRELIB	Program to prepare library
PUN	Standard punch unit
RES	Library file containing the executive resident routine
RIF	Relocatable information loader card
RAT	Resident allocation table for mass storage files
RNI	Read next instruction
TRA	Transfer address loader card
XNL	External name loader card

GLOSSARY

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 PROGRAM
A program that must always be loaded in the same memory addresses.

ABSOLUTE CODE
A code using absolute operators and addresses; that is, 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 such as allocating space on a disk for a mass storage file).

ALPHANUMERIC
Pertaining to a 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 instruction 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.

AUTOLOAD
To place the resident routines of the operating system in core storage.

BATCH
In MSOS, an object program running in a stacked job manner. Shares the central processing unit with the priority program when a priority program is present and executes only when the priority program is not in control of the processor. Batch interrupts have lowest priority in the interrupt processing priority scheme.

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 written or read as one unit.

BLOCKING
Combining two or more numbers (records) into one block to reduce the number of physical I/O operations.

BLOCK LENGTH
Number of records, words, or characters in one block.

BUFFER
An area in core used to store a block of words for output, or to receive a block of words from an input unit. The entire buffer is written when an output function is performed. When an input function is performed, one block of data from the input unit is written in the buffer area.

CARD COLUMN
A vertical line of punching positions on a card.

CARD IMAGE
A representation in storage of the holes punched in a card such that holes are represented by ones 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 card.

COMMON
An area of memory that may be shared between batch subprograms. Common may not be preset with data. Priority programs may not have a common area.

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.

CONTROL CARD, CONTROL STATEMENT
An instruction recognized by the operating system.

DATA AREA
An area of memory which may be preset with data at load time and shared between subprograms. Both batch and priority programs may have data areas.

DECK
A collection of punched cards that has a definite service or purpose, structured to represent a processing unit to the operating system.

DRIVER
A program that operates 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 copy, or the resulting document.

END-OF-FILE
Information designating the termination point of data or of a program.

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.

EQUIPMENT
An interface between a data channel and a unit; a channel controller.

ERROR
Any deviation of a computed or a measured quantity from the theoretically correct value.

EXECUTE
To carry out an instruction or perform a routine.

EXECUTION
The process whereby the instructions contained in a program direct the activities of the central processing unit.

EXTERNAL INTERRUPT
An interrupt occurring as a result of conditions within peripheral devices or their immediate interfaces. 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; that is, 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; that is, 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
A collection of related records treated as a unit; that is, 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.

FILE LABEL
For mass storage devices, an information set in a standard format, including entries for file identification, structure, location, and use. The file label appears in the file label directory (LABELFILE).

FILE ORDINAL
A number equated to a mass storage file for the duration of a job.

FLAG
1. Any of various types of indicators used for identification; that is, 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 initiate) 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.

INPUT
Information or data transferred from an external storage device into computer memory.

INPUT/OUTPUT
The bidirectional transmission of information between computer memory and peripheral devices.

INPUT/OUTPUT SYSTEM
The portion of the monitor that handles I/O; includes CIO, CIC, and I/O drivers.

INTERNAL INTERRUPT
An interrupt occurring as a result of conditions within computer mainframe.

INTERRUPT
1. A break in the normal flow of a system or routines 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 statements, programs, and data; presented serially in a job stack to MSOS through the standard input unit.

LIBRARY
An organized collection of standard subprograms and routines as a file or files.

LIBRARY FILES
In MSOS, the library consists of RES, ABS, LIB, and DRS files.

LINKAGE
The interconnections between subprograms or between a main routine and closed subroutines; that is, the entry into a closed routine and the exit back to the main routine.

LOAD-AND-GO FILE
The MSOS file designated by the file ordinal 56 when MSOS is initiated. This is automatically positioned to its origin when the user requests loading from the file. When loading is complete, MSOS again positions the load-and-go file to its origin to make it available for other output. Unless the user specifies otherwise, assembly and compilation output is written on the load-and-go file.

LOADING
The process of transferring a program from external devices to memory. In MSOS the relocatable loader transfers a relocatable program to the first sequential available positions in core; the absolute loader RDCKF1 transfers programs which must be loaded into specific locations.

LOCATION
A position in storage where one computer word can be stored and which is usually identified by an address.

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.

MAIN
An element of a program prepared in overlays. The main element typically is a controlling program which calls overlay elements into core in succession.

MASS-STORAGE CAPABILITY
The 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 permanent data and program files on mass storage devices with full facilities for modification and manipulation of these files. Privacy access codes prevent unauthorized use.

MASS STORAGE DEVICE
A disk or drum capable of storing large quantities of information (2 million to 200 million characters).

MASS STORAGE (ON-LINE)
High-capacity data storage accessible to the central processing unit.

MEMORY PROTECT
Hardware and software that prevent batch program from writing in core assigned to priority programs or the executive resident.

MSIO FILES
System files used by OCAREM to manage the mass storage files. These files are LABELFILE, IDFILE, and MSDFILE.

MULTIPROGRAMMING
In MSOS, a technique for processing two programs simultaneously by overlapping or interleaving their execution. In MSOS, multiprogramming is achieved by allowing the priority program to gain control of the processor periodically through interrupts.

OBJECT LANGUAGE
The language that is the output of a given translation process; that is, the language into which as assembler or compiler translates a source language.

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.

OVERLAY

In overlay processing, an element called by the main element.

OVERLAY PROCESSING

A technique for processing a program whose total storage requirements for instructions exceed available memory. The user divides the program into elements which are stored in mass storage and brought into core at different points of processing. An element of an overlay program, when brought into core memory, may occupy the same storage locations as another element which executed previously.

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.

PRIMARY ENTRY POINT

An entry point named on a EPT card that is used to call a subprogram from a library.

PRIORITY

A scheme for determining that a routine or job is to be executed before another. In MSOS priority distinctions are applicable in the following:

1. Multiprogramming. The priority program may gain control of the processor from the batch program through interrupts; the batch program receives control of the processor only when the priority program relinquishes control.
2. Interrupts. Real-time program interrupts have highest priority processing under MSOS; that is, when an interrupt generated by a real-time program occurs, MSOS immediately gives control to the real-time interrupt processing routine. Non-real-time priority programs have next highest priority; MSOS gives control to the non-real-time priority program when an interrupt generated by that program occurs, except when this interrupt occurs during MSOS input/output processing. In this case, MSOS waits for the input/output routine to complete execution and then gives control to the non-real-time priority program. Batch program interrupts have the lowest priority. The priority program, real-time or non-real-time, may gain control of the processor through an interrupt after a batch interrupt has occurred. MSOS waits until the priority program has relinquished control of the processor before routing control to the batch program interrupt routine.
3. Job stack processing. Under MSOS, the priority program may submit a batch job stack, and this stack has a processing priority over all batch jobs except the one currently in execution. MSOS waits for completion of this job and then initiates processing of the priority-submitted batch job stack.

PRIORITY PROGRAM

A specially prepared program requiring control for discrete intervals and that resides in core during batch runs.

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 transfer 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.
2. To put data into a storage device.

RE-ENTRANT

A subprogram or routine that does not modify itself while in execution. The subprogram or routine does not have to be reloaded to reenter it.

RELOCATABLE BINARY SUBPROGRAM

A subprogram that can be loaded into any available area in memory which is large enough to contain the subprogram.

RESIDENT

That part of the operating system residing in core memory at all times.

RETURN

To transfer control back to the point in a program or subprogram from which a call was issued.

ROUTINE

A set of instructions arranged in a sequence such that the computer performs a desired task.

SEGMENT

1. An element of a program prepared in overlays. A segment element is called into core by an overlay element.
2. A section of a mass storage file that is not contiguous with the rest of the file. A segment may be on a different device or pack.

SNAPSHOT DUMP

A selective dynamic dump performed at various points in a machine run.

SOURCE LANGUAGE

Input language for a given translation process.

STATUS

A state or condition of hardware or task; that is, busy or not busy.

SUBPROGRAM

A part of a larger program which can be converted into machine language independently.

SPOOLING

Buffering the system input, output, and punch files on mass storage so the CPU does not have to wait for the current job output to be printed and punched before starting the next job.

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.

SYSTEM DEVICE

The mass storage device containing system library files and the MSIO files.

SYSTEM FILES

The operating system files on the system device.

TRANSFER ADDRESS

The entry point address for a program supplied on the TRA card.

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 its operand address is changed by a stated amount each time the instruction is performed.

USER INTERRUPT
An interrupt selected by the user program through a CIC call.

USER PROGRAM
An object program loaded and entered under MSOS control; includes batch and priority programs.

UTILITY ROUTINE
A routine in general support of the operation of a computer; that is, an input/output, diagnostic, tracing, or monitoring routine.

WRITE
To transfer information, usually from internal storage, to an output device.

INDEX

COMMENT SHEET

MANUAL TITLE CONTROL DATA® MSOS Version 5 Operating System

Reference Manual

PUBLICATION NO. 60410600 REVISION C

FROM: NAME: _____

BUSINESS ADDRESS: _____

COMMENTS:

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

CUT ALONG LINE

AA3419 REV. 11/69 PRINTED IN U.S.A.

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

FOLD

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
Technical Publications Department
4201 North Lexington Ave.
Arden Hills, Minnesota 55112

MD 220

FOLD

FOLD

CUT ALONG LINE

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

LITHO IN U.S.A.