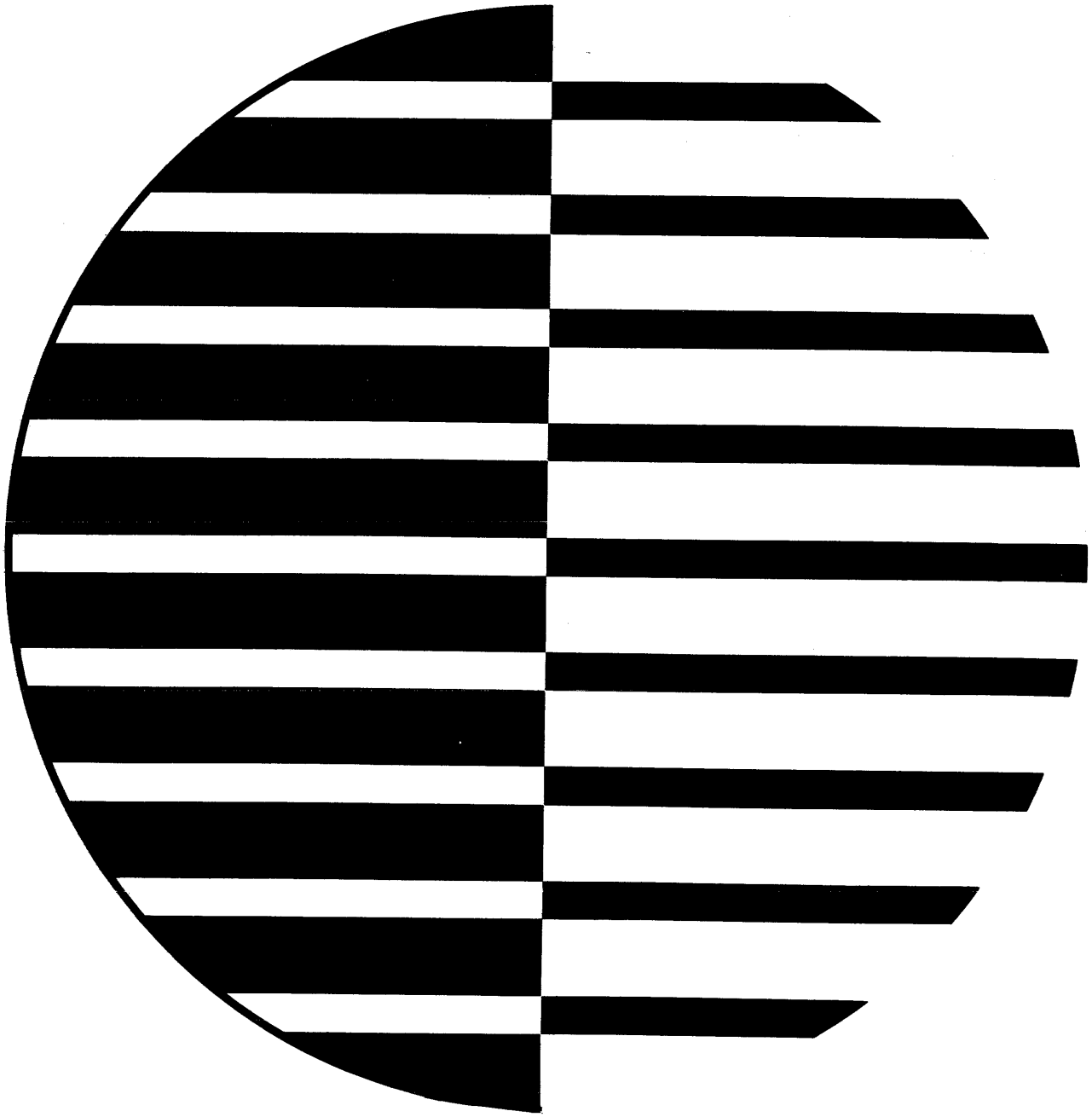
CONTROL DATA® 6400/6600 COMPUTER SYSTEMS

SCOPE Reference Manual



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

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

September, 1966
Pub. No. 60173800

PREFACE

SCOPE, an operating system which directs the execution sequence of programs on the CONTROL DATA® 6400/6600 computers, is an extension of the Chippewa Operating System. In addition to performing input/output, compilation, and storage assignment tasks, the SCOPE system includes a linking loader with segment and overlay loading capabilities.

Familiarity with 6000 hardware and related software manuals describing the ASCENT, ASPER, and FORTRAN languages is assumed.

This manual is intended to assist the user in programming on the central processor only.

CONTENTS

SCOPE TERMINOLOGY

byte	A 12-bit group of bits operated upon as a unit.
control point	A number, 1 through 7, appearing on the display scope which is associated with status information about jobs in central memory.
common files	Files that are not discarded upon job completion and may be used by other jobs.
dayfile	A disk file containing a running account of all control cards, equipment assignments, error diagnostics, central and peripheral processor time used, and I/O packages (e.g., PRINT, READ) used by the jobs in central memory.
dead start	Initial loading of the system tape by manual toggle switching after setting the panel switches.
display code	A code in which alphanumeric files are stored for console display. Each line of alphanumeric characters begins at the first 12-bit byte of a central word and continues two characters per byte to the end of the line.
equipment number	A unique two-digit octal number which identifies the equipment for job assignments.
file name table	A table containing the name of all files. (FNT)
idle program	Monitor transfers control to this program when no other program is ready for execution.
input files	Job files which have not been assigned a control point (listed in the job backlog (H) display with associated priority). A stored form of a job on disk, or logical file of cards separated by a record separator or file separator card.
job display	The program that displays only data pertaining to the particular job. (DIS)
local files	Files accessible at specified control points; they may be initiated by a job and discarded at the end of a job.

output files	A list of job files which have not been printed (appearing on the job backlog (H) display). These jobs are processed on a priority basis. If equal priorities are encountered, the last one in is the first one out.
pseudo control point	A number which does not appear on the display scope; 0 is associated with the monitor program in PP0, and the display program in PP9.
reference address	The address serving as a starting point for subsequent central resident address modification. (RA)
system display	The program that provides an overall status display for all currently running jobs. (DSD)

INTRODUCTION 1

Control Data 6400/6600 SCOPE controls the execution of a wide variety of jobs, assures optimum use of I/O equipment and priority processing, and performs such functions as loading, storage allocation, job scheduling, accounting, and operator communication.

6400/6600 computers are composed of eleven processors. Ten of these processors perform peripheral and operating system functions. The eleventh, the central processor, performs computation and processing at very high speeds. The eleven processors have separate memories and operate concurrently under SCOPE control. The large central memory is accessible to all processors.

1.1 SYSTEM COMPONENTS

SCOPE is initially loaded from the system tape by means of keyed settings on the Dead Start panel. During this operation, SCOPE components are distributed among the central memory, the ten peripheral memories, and the magnetic disk unit.

The central resident portion of SCOPE includes system control parameters and pointers, communication linkage, and frequently used programs and subroutines for both the central and peripheral processors.

The peripheral resident portion consists of a resident program stored in each peripheral memory, the system monitor (MTR) assigned permanently to processor 0, and the system display program (DSD) assigned permanently to processor 9.

The remainder of SCOPE is stored on the magnetic disk unit or in central memory to be called as needed.

1.2 CENTRAL PROCESSOR

The central processor executes computation and production jobs including operational user programs and compilation and assembly of user source language programs. These programs are stored in central memory along with the data necessary for execution and control. Central memory, which is accessible to both central and peripheral processors, serves as the communication link between them.

A number of programs may be in operation concurrently in the central processor. Central resident contains status information for each running program which SCOPE uses for orderly control and sequencing of the programs.

1.3 PERIPHERAL PROCESSORS

The ten peripheral processors (PP) are used by SCOPE to perform all I/O functions required by the system or the operational programs. They also perform certain auxiliary system functions connected with job sequencing and control.

PP programs are stored either in central memory or in a magnetic disk file. When called, they are transferred to an available PP for execution.

Identical resident programs in each PP sense a location in central memory for a control word calling for some action. The resident program locates the required program and loads it into its own memory for execution. A peripheral program may call additional peripheral programs into its memory, or call for action by the system monitor.

A common PP pool is available for assignment as needed by the system. Except for MTR, DSD and the resident programs, peripheral programs have no fixed processor assignments, and they are loaded each time they are called. To maintain a maximum I/O transfer rate all processors operate concurrently.

1.3.1 MONITOR

SCOPE functions under the direction of the system monitor (MTR) which is permanently assigned to PP0. MTR repeatedly scans a set of central memory locations which are set by transient peripheral programs to call for monitor action. MTR also senses the status of the running central program to determine program terminations and requests for monitor action.

MTR is used in the assignment and release of all peripheral processors, data channels, disk storage and I/O equipment. It maintains constant surveillance of all processing in the central and peripheral processors.

1.3.2 SYSTEM DISPLAY

The system display program (DSD), permanently assigned to PP9, serves as the communication linkage between the system and the operator. Two console screens provide system monitoring information and displays of central memory, selectable by the operator. Through the console keyboard, the operator can modify central memory contents and request system functions.

1.4 DISK STORAGE

The magnetic disk unit contains the non-resident portion of SCOPE including both peripheral and central library programs. Central resident contains directories which define the disk location for each library program.

The disk also holds the job stack and the data files for jobs in process. Output from job execution is collected on the disk for printing or punching by a peripheral program. In this sense, the disk serves as a large capacity buffer between the I/O devices and the processor complex.

The basic unit of storage on the disk is a half-track consisting of either the odd or the even numbered sectors in a physical track.

A logical file on the disk is defined as a named half-track followed by any number of continuation half-tracks in a single cabinet. This named file must begin in the first sector of a half-track and may be of any length up to the capacity of the cabinet.

1.5 OPERATION

Once SCOPE is loaded, all eleven processors execute idling programs while waiting to be called into operation. Processing is initiated when the operator selects the system input unit from the console keyboard; the system display program alerts the monitor to enter information in the system tables and calls the requested program into operation in a PP. The input program loads job decks from the system input unit into disk storage. As each job is filed on the disk, the input program makes an entry in a file name table (FNT) from which jobs are selected on a priority basis.

As many as seven programs may be active in the central processor at one time. Each active program is assigned a control point area which contains all information necessary to control the program and resume operation after interrupts. MTR continually senses the status of each PP and the running central program. When a call for action is detected, MTR either performs the operation itself or passes the request to one of the other PP's.

MTR interrupts the running central program to pass control to the next active central program on a priority basis if I/O requests are made. When program completion is sensed, MTR assigns a PP to search the job stack for the next program to be assigned to the control point and loaded into central memory.

A program operating in the central processor may call peripheral library programs simply by entering the name of the required program in its reference address location plus one (RA + 1). It can call directly any peripheral program with a name that begins with a letter. Peripheral programs whose names begin with numbers are called only by other peripheral programs.

2.1 CONTROL POINTS

The control points, numbered 1 to 7 in central resident are used to record associated activities. The monitor program in PP0 and the display program in PP9 are attached to a pseudo control point, numbered 0. Any other central or PP activity must be attached to control points 1 to 7.

An activity attached to a control point can reserve an area of central memory and may request use of the central memory and of the central processor. PP's may also be attached temporarily to a control point. During normal operation, some control points, used for system operations such as loading jobs from a card reader, may never require the central processor.

2.2 EXCHANGE JUMP AREA

Central processor operation is initiated or interrupted by an exchange jump command from MTR. MTR furnishes the central processor with the first word address of a 16-word area in central memory, the exchange jump area, which contains all necessary information about the program to be started or resumed in the central processor. The structure of the 16-word block is shown below:

		Words
Program Address (P)	A0(Address Registers)	0
Reference Address (RA)	A1	B1(Increment Register) 1
Field Length (FL)	A2	B2 2
Exit Mode (EM)	A3	B3 3
	A4	B4 4
	A5	B5 5
	A6	B6 6
	A7	B7 7
X0 (Operand Registers)		10
X1		11
X2		12
X3		13
X4		14
X5		15
X6		16
X7		17

EXCHANGE PACKAGE

The central processor enters the information about a new program into the appropriate registers and stores the current information of the interrupted program in the same 16-word block in central memory, thereby exchanging two programs. During this exchange, the normal functions of the central processor are not in effect.

All central processor reference addresses to central memory instructions or data are relative to the reference address (RA). The RA and field length (FL) define the central memory limits of a program (RA plus FL); field length is the total program length. The program address register (P) defines the location of a program step within the limits described. Each reference to memory is made to the address specified by P+RA. Therefore, relocation of a central memory program is easily performed by moving the program in memory and resetting RA to the new address in the exchange jump area.

Each central memory program is assigned to a control point, within which the first 16 words are reserved for the exchange jump area. When a central program is ready for execution, the initial values of P, RA and FL are entered into its exchange jump area by a PP. If this program's priority is greater than the currently operating central program, MTR initiates this program with an exchange jump command which interrupts the currently running program. Otherwise execution will not begin until the running program terminates or issues a recall request (RCL), or until its priority becomes higher than that of the running program.

2.3 STORAGE ALLOCATION AND MOVEMENT

Control point storage in central memory is allocated contiguously beginning with control point 1.

Example:

Assuming central resident and library programs occupy from 0 to 13777_8 a possible arrangement is: (octal addresses)

Control Point Number	Reference Address (RA)	Field Length (FL)
1	14000	4000
2	20000	10000
3	30000	100000
4	130000	0

Control Point Number	Reference Address (RA)	Field Length (FL)
5	130000	0
6	130000	40000
7	170000	40000

The area of unoccupied central memory starts at RA+FL of control point 7, 340000.

A PP attached to a control point can request or release storage via the monitor program. This commonly takes place when a new job with a different requirement than the previous job is brought to the control point.

When storage allocated to a control point is to be changed, the monitor moves the storage that has been allocated to control points with larger numbers.

2.4 CENTRAL/MTR COMMUNICATION

A central program may request MTR action by entering the name of a routine in display code left-justified in location RA+1. MTR periodically scans RA+1 of the running program for such requests. When RA+1 is non-zero, MTR passes the value to a PP for action. Requests such as input/output (CIO) are processed in this manner. When RA+1 is cleared, the running program may assume that the request has been honored, though not necessarily completed. Any parameters associated with the request must be put in the lower 36 bits of location RA+1 by the calling program. The format of the parameter list is dependent upon the program called.

The RA+1 entries take the following forms:

Call Peripheral	PP program name in display code in upper 18 bits of RA+1. Lower 36 bits contain the parameters for the peripheral program. MTR clears RA+1 as soon as requested program is passed to PP for execution.
Recall	RCL in display code in upper 18 bits of RA+1. MTR exchanges to next program waiting on a priority basis. This request should be used whenever the program cannot continue processing until an outside function is complete. All system peripheral programs recall the central program at completion of requested function.

End	END in display code in upper 18 bits of RA+1. MTR calls a peripheral program to advance to next control statement.
Abort	ABT in display code in upper 18 bits of RA+1. MTR calls a peripheral program to terminate the job and release the central processor.

MTR also performs the following operations with respect to central program:

Higher priority	MTR interrupts a running program in favor of a higher priority program.
Arithmetic exit	MTR exchanges to next equal or lower priority program when central program address (P) becomes zero. Arithmetic exit mode flag is set.
Time limit	MTR exchanges to next central program on a priority basis. When the time limit of the running program is reached, a flag is set which causes the program to be terminated.

2.5 CIRCULAR BUFFER I/O

The circular buffer I/O (CIO) program may be called to a PP to perform input and output between a file and a circular central memory buffer. The user specifies a file name and operation code, plus information about the buffer, then CIO performs the operation.

Before calling CIO, circular buffer parameters must be set in five central memory words as follows:

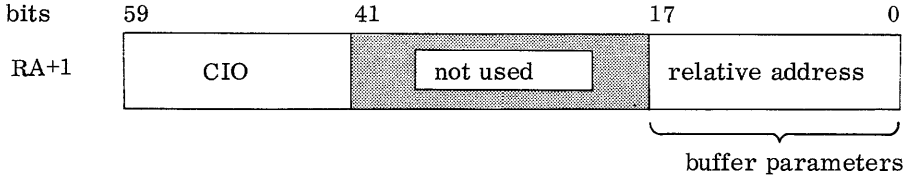
bits 59 17 6 0

file name	not used	code	Name (display code left adjusted) and operation
not used	FIRST		Beginning address
not used	IN		Current input address
not used	OUT		Current output address
not used	LIMIT		Last address + 1

} of circular buffer

The buffer and buffer parameter area must be within the field length of the job, and addresses are relative (address 0 for the job in absolute word RA).

A central program can call on CIO by entering in its word 1 (absolute RA+1) the code CIO and the (relative) address of the circular buffer parameters.



Example:

CBP is the symbolic address of the parameters:

SX6	031117B	CIO in display code
LX6	42	To top of X6
SX5	CBP	
IX6	X6+X5	Add in address
SA6	1	Write to (RA+1)

PP calls in RA+1 of a running central program are detected by the monitor in PP0. The monitor finds a free PP to perform the task, then clears RA+1 to indicate that the task is started, not completed. The operation code in the first parameter word is even, and one is added to it by the PP when it has performed the operation. PP also updates IN and OUT in the parameter area, according to the function performed.

The central program continues after setting RA+1 but must not use RA+1 again until it is cleared by the monitor. This is programmed either by looping on non-zero RA+1 during the short time taken by the monitor to detect a call, or by checking that RA+1 is zero before making a further call.

The central program should inform the monitor with an RCL call if it is unable to proceed further for the time being, then the monitor will switch to another program. A central program may buffer input and output in order to proceed to a certain stage before being delayed.

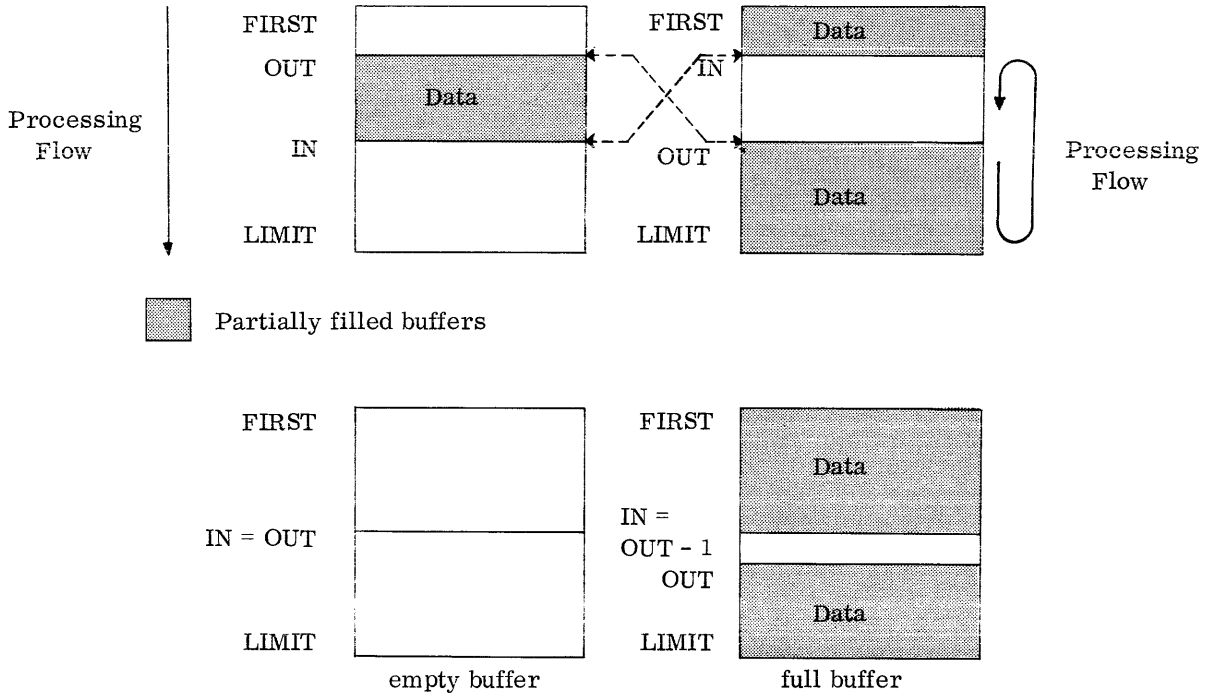
Continuing the last example, if the central program chooses to wait until the PP has finished:

```
L1    SA1   1          Read (RA+1) to X1
      NZ    X1, L1     Wait till clear
      SA1   CBP        Get first parameter word
      LX1   59         Determine if odd (PP finished)
      NG    OK         Continue to OK if PP finished
      SX6   220314B    RCL code
      LX6   42         To top of X6
      SA6   1          Recall code to (RA+1)
      JP    L1         Loop
OK                     Continue processing
```

The recall code causes control to be taken from the program. Periodically, control is returned to the program at intervals of approximately 500 milliseconds (depending on priority) or when a PP completing an operation tells the monitor to recall the corresponding central program. The monitor clears RA+1 after receiving the RCL. The loop at L1 holds up the program until the monitor switches control; it allows the program to continue when control is returned.

Most users need not be concerned with CIO and RCL since their FORTRAN programs are compiled to include the necessary calls. However, users programming input and output in machine instructions must remember to explicitly drop the central processor with a RCL when awaiting progress of a PP task. A PP call does not in itself cause the central processor to be switched; progress of a CIO operation is determined by examination of the parameter area.

The current data in a circular buffer starts at OUT and continues (possibly round the end of the buffer) to IN-1.



When a buffer is filled to capacity, the unused word between IN and OUT distinguishes it from an empty buffer for which IN = OUT. The capacity of a buffer is LIMIT - FIRST - 1. A buffer is generally initialized with IN = OUT = FIRST, then IN and OUT circle the buffer as data is inserted and extracted.

IN defines the address for insertion of data into a buffer. As data is inserted, IN is stepped round the buffer but never so as to catch up with OUT to avoid overstepping buffer capacity.

OUT defines the address for extraction of data from a buffer. As data is extracted, OUT is stepped round the buffer but never beyond IN since the buffer is empty by the time OUT = IN.

Commonly CIO, moving IN, reads data from a file to the buffer, and the data is extracted by a central program moving OUT; or a central program, moving IN, inserts data into the buffer, and the data is written to a file by CIO, moving OUT.

File operations include read and write (coded or binary), backspace, write end record, and write end file mark. Files are stored serially; read and write refer to the next position of the file. Equivalence is preserved subject to the limitations of the equipment; for instance, a card or printer file may not be backspaced; also a file is considered to extend only as far as the last record written. A file on disk may have only one file mark after the last record; therefore if an object program is to preserve equivalence between tape and disk it should not write more than one file mark.

In a CIO write operation, only data sufficient to write physical records for the file unit is extracted from the circular buffer. For a disk write, for example, CIO takes only 64-word sectors from the buffer, unless an end-of-record or end-of-file is requested when a shorter sector of data can be written to empty the buffer. Similarly, on a read operation, CIO transfers a physical record to the buffer only if there is room for it. At the end of a read or write call, the positions of IN and OUT show how much data is left in the circular buffer.

In writing to the disk, the circular buffer pointers are advanced at the end of the operation; in reading from the disk, the IN pointer is advanced after each sector has been read.

The operation code given to CIO via the last 7 bits of the first buffer parameter word is also known as the buffer status since CIO returns a code to these 7 bits when a call has been completed. CIO is given an even code and adds one to the code after completing a called operation. For read operations, the code returned also indicates whether end-of-record or end-of-file terminated the read.

The two fields of the buffer status bits have the following meanings:

Value	First Field (4 bits)	Second Field (3 bits)
00	Not used	Request coded read
01	Buffer I/O	Coded read completed
02	End Record	Request binary read
03	File Mark Forward	Binary read completed
13	File Mark Backward	Not applicable
04	Backspace	Request coded write
05	Rewind	Coded write completed
06	Rewind unload	Request binary write
07	Not used	Coded write completed

A command given to CIO is even, the first field specifies the type of operation, and the second specifies the direction (read/write) and mode (coded/binary). For buffer I/O, as many physical records as possible are transferred between file and buffer. In writing, End Record or End File are used to empty the buffer and write end-of-logical-record or end-of-file for the particular device.

One is added to the code to inform the program when a PP has finished a CIO call. For reading, the first field may be changed by CIO to indicate that the transfer was terminated when end-of-logical-record (EOR) or end-of-file (EOF) was encountered. A programmer should make certain that a buffer is emptied of all information to be read, before it is reused.

A backspace operation (040 = coded, 042 = binary) sets the parameters so data of the new file position can be extracted from OUT. The amount of information in the buffer (up to IN) depends upon the device and the previous values of IN and OUT.

Only the Output file buffer is emptied at the end of the job.

2.5.1 BUFFER CODES

In the following discussion, the expressions in parentheses are the binary values which correspond to the buffer status field, x may be either 0 or 1.

Input (0001, 0x0)

File is read into circular buffer until buffer is filled (0001, 0x1) or until end record (0010, 0x1) or file mark (0011, 0x1). The mode bit (binary/BCD) is ignored only when reading from disk or one-inch tape. A file mark response should never occur with data.

Output (0001, 1x0)

Data in circular buffer is recorded on file for as many complete physical records as available data. No partial end record is made. (0001, 1x1)

End Record (0010, 1x0)

Data in circular buffer is recorded on file including a short physical record to mark end of logical record. The last physical record may be of zero length. IN=OUT=FIRST. (0010, 1x1)

File Mark (0011, 1x0)

Data in circular buffer; or
Last buffer status (0001, 1xx)

Data in circular buffer is recorded on file including a short physical record to mark end of logical record. Then a file mark is recorded. IN=OUT=FIRST (0011, 1x1)

All others

A file mark is recorded. IN=OUT=FIRST. (0011, 1x1)

Backspace File (1011, 000)

Magnetic tape only; backspace to previous file mark. IN=OUT=FIRST (1011, 0x1)

Skip File (0011, 000)

Magnetic tape only; skip forward to next file. IN=OUT=FIRST (0011, 0x1)

Backspace Binary (0100, x10)

Backspace to end of last record. A file mark is considered a record in this case. IN=OUT=FIRST. (0100, 011)

Backspace Coded (0100, x00)

Backspace one coded line. A file mark is considered a coded line in this case. The last physical record will be left in the buffer beginning at FIRST. IN and OUT will be adjusted for a one line backspace. (0100, 001)

Rewind (0101, xx0)

Rewind the file. IN=OUT=FIRST. (0101, 0x1)

Rewind Unload (0110, xx0)

Rewind and unload the file. IN=OUT=FIRST. (0110, 0x1)

2.5.2 OPERATION

All parameter values (IN, OUT, FIRST, LIMIT) must be set by the calling program. For input, CIO reads data into the buffer beginning at IN. CIO continues reading as long as storage is available or until the data is depleted. IN is advanced by one for each data word read. When the value of IN=OUT-1, the buffer is full and the operation complete.

If IN≥OUT, IN is advanced to LIMIT-1; CIO automatically resets IN to FIRST and continues in the same read operation until IN≤OUT-1. The maximum buffer capacity for any read is LIMIT-FIRST-1.

For output, CIO writes data from the buffer beginning at OUT. Only complete data blocks are written to the output file; no partial end record is written unless the end of record or end of file function is given in the CIO call.

If EOR or EOF is not selected, CIO continues writing until there is not room in the buffer for a full block. If EOR or EOF is selected, CIO writes until OUT=IN and then sets IN=OUT=FIRST. OUT is advanced by one for each word written. When OUT=LIMIT-1, CIO automatically resets OUT to FIRST and continues the write operation.

The central program must complete writing a record before requesting a backspace, rewind, or mode change but it need not complete writing a record before requesting a file mark.

The central program must complete reading a record before beginning output data to the buffer.

Binary tapes and coded one-inch tapes record 1000 octal word physical records in odd parity: no special control words are added. A zero length physical record is generated by recording a partial word (4 bytes). Coded one-inch tapes use packed display code with short word separators. Coded half-inch tapes record 120 character BCD code in even parity.

A one-inch tape with mixed binary and coded records presents problems if a backspace crosses a mode boundary. A problem exists in mode change from coded input to output on one-inch tape if the file is positioned between two lines of code. No problem exists if the file is positioned before or after a file mark. No problem exists in mode change on binary files.

A disk file with multiple file marks or data recorded after a file mark cannot be substituted for a tape file.

2.6 CONTROL CARD PERIPHERAL PROGRAMS

The following peripheral programs may be requested by the programmer with control cards.

2.6.1 DUMP STORAGE

This program may be called with a control card or from a display console in any of the forms shown below: An octal jump is entered in the output file with the central storage address and one data word per line.

DMP.

dumps the exchange area into the output file.

DMP, 3400.

dumps from the reference address to the parameter address.

DMP (4000, 6000)

dumps from the first specified address to the second.

2.6.2 LOAD BINARY CORRECTIONS

This program may be called with a control card or from a display console. Binary corrections are read from the input file and entered in central storage. If a parameter is specified in the program call, binary cards are loaded beginning with that address; otherwise, loading begins at the reference address. Only one record is read from the input file. A call must be made for each block of data to be loaded. This program may be called with either of the following formats:

LBC.

LBC, 2300.

This program is intended for loading cards punched through PBC (section 2.6.4).

2.6.3 LOAD OCTAL CORRECTIONS

This program may be called with a control card or at a display console. Octal corrections are read from the input file and entered in central storage. The octal correction cards must be in the following format:

```
1       7
23001   45020  04000  00042  00044
```

Address begins in column 1; leading zeros may be dropped in the address. The data word begins in column 7; spacing in the data word is not important but the word must contain 20 digits.

LOC.

reads all correction cards in the next input file record and modifies central storage accordingly.

LOC, 1000.

clears central storage from the reference address to the specified address; correction cards are then read from the input file.

LOC (2022, 3465)

clears central storage from the first specified address to the second; correction cards are then read from the input file. This program may be called to clear storage by providing an empty record in the input file.

2.6.4 PUNCH BINARY CARDS

Only a control card may be used to call this program which punches a deck of binary cards directly from central storage. Storage is not modified by this operation.

PBC, 2000.

a binary deck is punched from the reference address to the specified address.

PBC (2000, 3000)

a binary deck is punched from the first specified address to the second.

PBC.

punches a binary deck using the first word in central storage as a control word for deck length. The deck always begins at the reference address and terminates one address less than that indicated in the lower 18 bits of the first word. This call may be used for punching any central or peripheral program in standard format.

2.6.5 READ BINARY RECORD

One binary record may be loaded from a file specified by the user.

RBR, n.

n specifies the fifth character of the file name, 1-7. (The first four characters are TAPE.)

Loading begins at RA+0. If the record cannot fit into central memory, a dayfile message RECORD TOO LONG appears. RBR uses central memory locations FL-5 through FL-1 for buffer parameters; the original contents of these locations are destroyed.

2.6.6 REQUEST FIELD LENGTH

The field length for the execution of a program may be changed.

RFL, nfl.

nfl new field length (octal)

This routine is also used internally by the compiler (RUN). For a short 5000_8 word program, storage would be used most efficiently by specifying RFL.

Example:

```
JOB, 5, 3000, 40000.
RUN(S, 5000)
RFL, 5000.
SAM. (Execute program with FL=5000)
```

2.6.7 SOS

A program may be inserted in the disk portion of the peripheral library. The program must begin at RA+100_8. After SOS has stored the program on the disk, the control point is released.

SOS.

Example:

```
JOB,1,500,1000.
LBC,100.
SOS.
record separator card (7,8,9 in column 1)
(PPU binary deck from an ASPER assembly)
file separator card (6,7,8,9 in column 1)
```

SOS does not erase a similarly named program on the disk. If two identical names exist on the disk the system will reference the first, or original, program.

2.6.8 WRITE BINARY RECORD

A binary record may be written from central memory to a file specified by the user.

WBR,n,rl.

- n — the fifth character of the file name, 1-7. (The first four characters of the file name are TAPE.)
- rl — record length in words. If omitted, the length of the record is taken from the lower 18 bits of RA+0. When rl is omitted, the period is specified after n.

WBR begins writing from RA+0. The contents of central memory locations FL-5 through FL-1 are used by WBR for buffer parameters; the original contents of these locations are destroyed.

Example: To write a program on tape after patching it:

```
REQUEST TAPE5.
REQUEST TAPE2.
REWIND(TAPE5)
REWIND(TAPE2)
RBR,5.
LOC.
WBR,2.
```

2.6.9 HLP

A program may be inserted in the resident portion peripheral library.

HLP.

After the last word of the program added to the resident library, the HLP routine inserts a word in the following format:

0000 0000 0000 0000 0001

A similarly named routine will be removed from the library and replaced by the new routine.

2.6.10 JOB DISPLAY-DIS

The job display program (DIS) is similar to DSD†, but it is used for information more relevant to a single job. With DIS, the B display reveals the exchange jump area of the job; central memory addresses relative to the job's reference address are used for data and program displays.

This display can be called either from a control card (DIS.) or by a command to DSD, and it may be called at any time during the execution of job. The job display package stops further automatic advance of the job control cards. The display covers only data pertaining to the particular job. The keyboard is used to advance the job control cards and to provide any two of the following displays in the same manner as for the DSD display.

The B display shows only the condition of the control point to which DIS is attached; it includes the next control statement and a picture of the job's exchange package. The exchange package is displayed only while the job is in W, X or blank status. The operator may change priorities and suspend job execution with DIS.

† DSD (System Display program) is described in the 6000 Chippewa Operating System Operating Guide, Publication No. 60172400.

JOB DISPLAY CODES

Codes		
A	Dayfile	
B	Job Status	
C	Data Storage	5 groups of 4 octal digits per group
D	Data Storage	
E	Data Storage	
F	Program Storage	4 groups of 5 octal digits per group
G	Program Storage	

CHANGING PRIORITY

All jobs are assigned priorities (0-17) from the job control card; a zero priority job is ignored. The operator may change the priority of any job through the keyboard, the range is $0\text{-}77_8$ with 77_8 being highest priority.

Operator procedure:

1. Type in: ENPR,dd.

   dd two-digit octal priority number

2. Press the carriage return key. If a priority change is attempted during a run, the program will stop (normal stop) and the operator may type in RCP. to resume central processor operation. A priority change may be made only when the job is assigned to a control point.

SUSPENDING EXECUTION

Suspending Job

To temporarily suspend execution of a job, the operator may type the following entries:

DCP. and carriage return

This temporarily suspends the central processor and displays the exchange jump area.

RCP. and carriage return

This resumes central processor execution at the next program address for a job suspended by a DCP.

Multiprocessing Termination

When a control point is associated with more than one piece of equipment, subpoints are specified for the additional equipment used.

Operator procedure to terminate a single unit during multi-unit processing:

1. Type in n. ENDx.

   n control point

   x subpoint

2. Press the carriage return key.

The following keyboard entries to DIS refer to the control point to which it is attached. Some of the entries cause the job to be switched away from the central processor. Execution can be resumed with RCP. or BKP. Numbers are in octal.

ENTRIES FOR CHANGING JOB DISPLAY CONTENTS

ENP,12345.	Set P = 12345. (next instruction address, in exchange jump area).
ENA3,665000.	Set A3=665000 in exchange jump area.
ENB2,44.	Set B2=44 in exchange jump area.
ENX5,2223 4000 0000 0000 0200. (Spacing is unimportant)	Set X5=2223400000000000200 in exchange package.
ENEM,7.	Set Exit Mode = 7 in exchange jump area.
ENFL,10000.	Set FL=10000 in exchange jump area. (storage moved if necessary).
ENTL,200.	Set central processor time limit = 200_8 seconds.
ENPR,5.	Set job priority = 5.

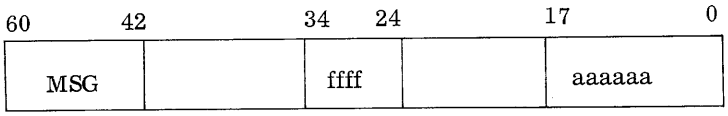
DCP.	Drop central processor and display exchange jump area (in display B). When DIS is used, the exchange jump area is displayed in any case if the job does not have status A, B etc.
RCP.	Request central processor. This puts the job in W status, and it will use the central processor if its priority is sufficient. The register settings of the exchange jump area will be used.
BKP, 44300.	Breakpoint to address 44300 in the program. Central processor execution begins at the current value of P and stops when P = 44300. DIS clears 44300 to stop the program at that point, and restores the original word when the stop occurs.
RNS.	Read and execute next control statement.
RSS.	Read next control statement and stop prior to execution.
ENS. xxxxxxxxxxxxxx.	Allows the entry of any control statement xxxxxxxxxxxxxx as if it had been entered on a control card. The statement can then be processed using RNS. or RSS.
GO.	Restarts a program which has paused.
ONSW3.	Set sense switch 3 for the job.
OFFSW4.	Turn off sense switch 4 for the job.
HOLD.	DIS relinquishes the display console, but the job is held at the present status. A console must be reassigned to continue use of DIS.
DROP.	DIS is dropped and normal execution of the job is continued; it does not drop the job.
DMP(200, 300)	Dump storage from 200 to 277 in the output file.
DMP(400)	Dump storage from the job reference address to 377.
DMP.	Dump exchange jump area to output file. (DMP formats are the same as if used on control cards).

2.7 USER-REQUESTED PERIPHERAL PROGRAMS

The following peripheral programs may be requested by the user from the central processor.

2.7.1 MSG

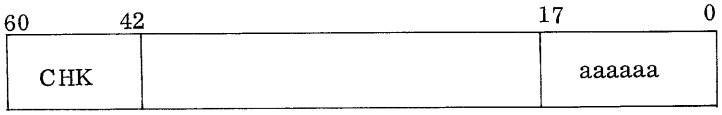
With this routine, messages may be sent to the console (displayed at the third line of the control point) and optionally to the dayfile. The request to MSG is stored in RA+1 as follows:



MSG	Characters in display code
ffff	Console/dayfile flag
	0 message to console and dayfile
	1 message only to console
aaaaaa	Address where message begins

2.7.2 CHK

CHK allows the user to determine whether anything has been written on OUTPUT. The request to CHK is stored in RA+1 as follows:



CHK	Characters in display code
aaaaaa	Address of a word in central memory to be used as an indicator. If this word equals 0, OUTPUT has not been written; if it equals 1, OUTPUT has been written.

EQUIPMENT USE AND FILE STRUCTURE 3

3.1 FILE NAME TABLE

SCOPE maintains a directory of all files known to the system. This directory is called the File Name Table (FNT).

3.2 FILE FORMAT

Files may be transferred from one device to another since equivalent formats are used for card, printer, disk, and magnetic tape files. The information in a file is stored serially. An object program may operate on named files and the device can be specified on control cards; disk storage is assumed if no assignment is made.

Except for a tape file which may have more than one file mark, files consist of a single physical file divided into logical records. A logical record consists of a number of 60-bit words containing either coded or binary information. The form of storage and method of separating logical records depend upon the equipment.

The concept of logical records makes it possible to have equivalent forms of a file on several devices, without losing the advantages of each form of storage. For example, cards constituting a logical record can be transferred to an equivalent form on disk storage where they are blocked in sectors.

3.3 FILE NAMES

Input and output operations of a central program involve a named file - a disk, magnetic tape, punched card, or printer file. The physical unit associated with a file name is controlled by the job control cards and is not a function of the central program coding. The operating system provides a common interface between the central program and the peripheral programs which drive the equipment.

File names must begin with an alphabetic character and may have a maximum of seven alphanumeric characters.

Five special file names, INPUT, OUTPUT, DAYFILE, PUNCH, PUNCHB are implied with each job. These names must not be used for temporary files and other I/O files.

INPUT - the file from which the job cards are read. Each job file is assigned the name appearing on the job card when loaded. As the job file is picked up for processing, the job file name is changed to INPUT.

OUTPUT - the file which ends in printer copy at the end of the job. During job processing, this file name (OUTPUT) is used to collect all records to be printed. When the job is completed, this file name is changed from OUTPUT to the name which appeared on the job card.

DAYFILE - a disk file which contains the system history (dayfile) messages for all jobs.

PUNCH - a disk file which is punched in Hollerith when the job is completed.

PUNCHB - a disk file which is punched in binary when the job is completed.

Example:

The entire dayfile may be dumped with the following job deck structure:

```
JOBn,p,t,fl.
COMMON DAYFILE.
BKSP (DAYFILE)
ASSIGN52,A.
COPYBRS (DAYFILE,A)
BKSP (DAYFILE)
```

The dump will be on magnetic tape 52 in binary format with each line shifted one character position to the right and a leading blank added. Other formats may be selected with other copy routines. The dayfile is not updated while this job is being executed.

TAPEnn.

The characters, TAPE, are added by the FORTRAN compiler whenever a program I/O statement refers to unit nn. If unit nn is to be designated as other than a disk file, the name TAPEnn must be used in control cards, such as COMMON, REQUEST, etc.

Example:

READ(5,10) list

This FORTRAN statement would generate a reference to file TAPE5. The control card necessary for the file to be a tape would be:

ASSIGN MT, TAPE5.

or

ASSIGN 50, TAPE5.

REQUEST TAPE5.

or

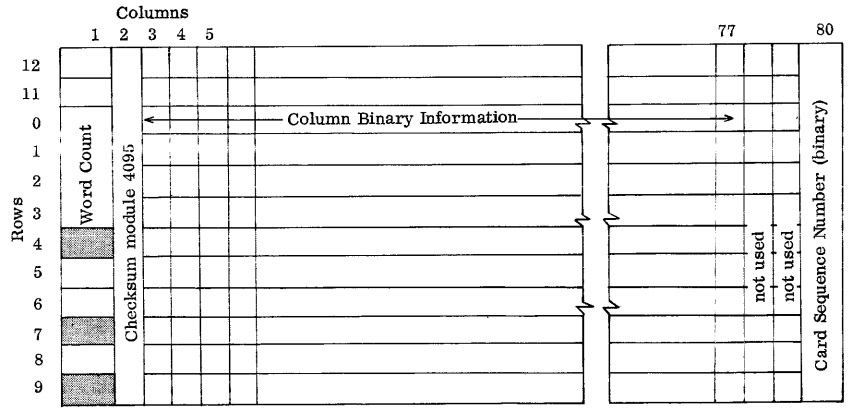
with subsequent operator assignment.

3.4 CARD FILES

The format of a card is as follows:

Column 1

7,8,9	End of logical record
6,7,8,9	End of file
7,9	Binary card
7 and 9 not both in column 1	Coded card



Standard Binary Card Format

A binary card can contain up to 15 central memory words starting at column 3. Column 1 also contains a central memory word count in rows 0, 1, 2 and 3 plus a check indicator in row 4. If row 4 of column 1 is zero, column 2 is used as a checksum for the card on input; if column 4 is one, no check is performed on input.

Columns 78 and 79 of a binary card are not used, and column 80 contains a binary serial number. If a logical record is output on the card punch, each card has a checksum in column 2 and a serial number in column 80, which orders it within the logical record.

Coded cards are translated on input from Hollerith to display code, and packed 10 columns per central memory word. A central memory word with a lowest byte of zero marks the end of a coded card (it is a coded record), and the full length of the card is not stored if it has trailing blanks. This produces a compact form if coded cards are transferred to another device.

3.5 DISK FILES

Storage for a disk file is reserved by the monitor in half-tracks as needed. A particular disk file is stored on a number of half-tracks within the same disk cabinet. A half-track consists of either the even or the odd numbered sectors of a track. With this format, data can be efficiently streamed between disk and central memory without the need for large buffers in peripheral memory.

Each sector of a file contains up to 64 central memory words of data, plus two control bytes. The first control byte identifies the location of the next file sector; it contains a sector number if the file is continued on the same half-track, a logical half-track number if the file is continued on a different track, or zero if there is no further information in the file.

A logical half-track number is specified as follows:

(1 bit)	Always set (indicates byte is a half-track number and not a sector number)
(7 bits)	Physical track number
(1 bit)	Track of odd or even sectors
(3 bits)	Head group number

The second control byte specifies the number of central memory words of information in a sector. End-of-logical-record is indicated if this number is less than 64. Both control bytes are zero for end-of-file.

Each disk file must start at the first sector of a logical half-track. When a half-track is full, continuation is to the first sector of another half-track. The forward linkage is constructed when writing and followed when reading. A sector with two zero control bytes is always generated after a write operation. The status word of a disk file records the current position. End-of-file can be generated explicitly by an operation code, or implicitly after a write operation.

A binary record on disk storage is the same as a logical record. A logical record may contain coded records, the end of each is indicated by a central memory word in which the lowest byte is zero. In this sense, a logical record of coded records is equivalent on disk storage to a binary record, and its use depends upon context.

3.6 BINARY AND CODED MODES

The code for an operation on a named file specifies whether it is to be performed in binary or coded mode. For some devices, the stored form of a file is not dependent upon the mode. When writing to disk, for example, a number of central memory words is transferred from a buffer, and there is no physical difference between a binary write and a coded write, as the distinction lies in the contents. This does not destroy equivalence between devices within their limitations, for instance, a printer always assumes coded information. For a program copying coded information from disk to 1/2" tape, the mode in reading from the disk to a central memory buffer is ignored; but the PP writing on tape will accept coded records (ending in a central memory word with zero lowest byte) from the buffer, and write each as a separate physical record on tape in BCD mode. A central program may extract a coded record from a named file with a coded input, and a search for a blank byte since the PP software will have taken care of any special hardware action if the device was mode dependent.

3.7 MAGNETIC TAPE FILES

The stored form of a logical record is independent of the mode when written on 1" tape. A logical record is divided into 1000_8 word (central) physical blocks; a shorter block marks end-of-logical-record. If the length of a logical record is not a multiple of 1000_8 words the data is exactly contained in the physical blocks; if the length is a multiple of 1000_8 words, an additional shorter block of four 12-bit bytes is necessary.

A logical record can have zero length (corresponding to successive 7, 8, 9 cards).

Any coded records within a logical record end in a word with a lowest byte of zero. Thus, coded information sent to 1" tape is blocked in display code.

Exactly the same information is written on 1/2" as on 1" tape in binary mode. On a coded write to 1/2" tape BCD records of 120 characters are written. Each coded record is translated from display code to BCD (IBM external code), padded with spaces to 120 characters, and written on tape with even parity. On input from 1/2" tape in coded mode, characters are translated from BCD to display code, trailing spaces are discarded, and the record is stored internally in the normal form. End-of-file for 1/2" and 1" tapes is written as the usual file mark.

In reading 1" or binary 1/2" tape, blocks of less than 4 bytes are ignored as noise. Blocks of less than 6 bytes (12 characters) are ignored when reading 1/2" tape in coded (even parity) mode.

Disk storage cannot be substituted for tape if a program writes more than one end-of-file or writes information beyond end-of-file. Also there is no equivalent to end-of-logical-record on 1/2" tape in coded mode.

CONTROL CARDS AND JOB PROCESSING 4

A job consists of one or more central programs executed with data files. Control cards are the first logical record; they identify the programs and data files, and sequence program execution. The control cards specify how a job is to be processed; operations performed upon other records of a job file depend upon the control cards.

Each job must begin with a job card and end with a file separator card, and all control cards must appear between the job card and the first record separator. The end of the control cards is signified by an end-of-record card (7,8,9 punch) or an end-of-file card (6,7,8,9) if the job consists of control cards only. No special multipunches are used for control cards and the information on a control card can start in column 1 (with no imbedded blanks). The card is free field thereafter. Cards terminate with a period or, if a parenthesized list appears, a closing parenthesis. The order of cards is described in Chapter 6.

Except for program call cards and job cards, the control card formats are unique to the system and must not be used for the names of any programs.

4.1 JOB CARD

The first control card of a job indicates the job name, priority, central processor time limit, and central memory requirement. If only a job name is specified, priority 1, time limit 1 minute, field length 40000_8 is assumed. Fields are separated by commas and the last field is terminated by a period.

```
name,priority,time limit,field length.
```

name — Alphanumeric job name; 1-7 characters, must begin with a letter.

SCOPE replaces characters 5,6, and 7 of the job name with a unique sequence number, which indicates the number of jobs run through the system since dead start.

priority — 1 through 17 (octal)

The highest priority job on disk which will fit in memory is the next to be brought to a free control point for processing.

The central processor is always given the highest priority control point that can use it.

Completed job output files are printed in order of priority.

If a job which is completely compute-bound has the highest priority (17_8), and another job with a great deal of I/O processing issues a recall instruction and is waiting, the operator may enter a higher priority for the I/O job (the compute-bound program will still retain the associated control point).

time limit — Total time limit for the job in seconds (central processing time); a maximum of 5 octal digits. The octal value in hundreds is approximately the time in minutes; the time limit is rounded up to a multiple of 10_8 by the system.

Time and space limits must suffice for the whole job, including all compilation and execution.

field length — Total field length of the job, in octal, maximum of 6 octal digits. The length is equal to the total storage of the system less that used by SCOPE for permanent resident tables. This varies among installations. The field length is rounded up to a multiple of 100_8 by the system.

If the RUN compiler is used and a listing requested, it prints out the amount of storage that was not needed, both for itself and the compiled program, so that future runs may request a lesser amount. A common trial storage request for compilation is 100000_8 (32K). The standard amount, 40000_8, is sometimes sufficient for compilation, and usually adequate for utility jobs such as file copying.

Example:

```
JOB765,3,600,40000.
```

Statements on a job card may use different separators, for example:

```
JOB765(2,350,100000)
```

4.2 LOADING AND EXECUTING

4.2.1 LOAD

The load card directs the system to load a file.

```
LOAD,fn.
```

fn File name

The system searches the File Name Table (FNT) for the file, and if it does not appear there, the system searches the system library. The file is rewound before it is loaded.

All types of loading, normal, segment, or overlay, may be loaded by LOAD cards but they may not be mixed in one operation. One job may contain any number of LOAD cards. The type of loading for all LOAD cards within a job is determined from the first record of the first named file.

4.2.2 EXECUTE

The execute card completes loading and transfers control to a program.

```
EXECUTE(name,p1,p2,...,pi)
```

name Entry point of the program. If name is absent, control will be transmitted to a name in a table generated either by the compiler (from a FORTRAN PROGRAM card) or by the assembler (from an ASCENT END card).

p_i Parameter that is passed to the program to be executed.

Completion of normal loading includes filling all unsatisfied references with entry points from the system library or with out-of-bounds references. Program execution then begins at the entry point named on EXECUTE or in the table generated by the assembler or compiler.

When segments are loaded, unsatisfied references are not filled and execution begins in the first segment. Subsequent segments may be loaded by user calls in the program.

When overlays are loaded, execution begins in the main overlay. Subsequent overlays may be loaded by user calls in the program.

4.2.3 PROGRAM CALL

```
name(name1,name2,...namen)
```

name	Name of a system or user program being called.
namen	Names of all files referenced in this job, or parameters to the program. If a called program has no parameters the line must terminate with a period. A closing parenthesis terminates a line with parameters.

Examples:
```
RUN(G,300 000,100000,3000)
COPYCR(TAPE2,TAPE3,1000)
LINNEY.
```

The program call card names a program to be loaded and transfers control to that program's entry point in the same way as LOAD and EXECUTE cards; it may be substituted for a LOAD and EXECUTE.

Example:
```
JOHN.
```

is equivalent to:

```
LOAD (JOHN)
EXECUTE.
```

The system searches for the file in the same way as it does for a LOAD card. The file is rewound before it is loaded. Any subprogram already loaded by LOAD cards will be bypassed and a message will be issued.

Thus, in the case of:

```
LOAD (NEWSUB)
JOHN.
```

The subprograms will be loaded from the file NEWSUB and those with identical names on file JOHN will be bypassed by the program call card.

4.2.4 NOGO

This card closes out the loading operation, bypassing execution.

```
NOGO.
```

When a NOGO card appears, the loader completes loading of the present program by satisfying external references wherever possible and writing the memory map on the output file. The loader then clears out all load tables, link tables, and availability tables and resets the job to the initial, unloaded status. Since execution is bypassed, the NOGO card is generally used to produce a memory map for the purpose of finding loading errors.

NOGO may follow a LOAD card.

4.3 EQUIPMENT ASSIGNMENT

Any file not specifically assigned on control cards is assigned by the system to storage on a disk unit. A job need not request card reader and printer for normal input/output since its cards are already stored in the job input file on disk, and output for a printer is sent to the job output file on disk.

Since control cards of a job are processed in order, equipment assignment must be made before the corresponding file is referenced.

4.3.1
ASSIGN u,f,dd

This control card assigns an available peripheral unit of type u to a file named f. The type u may be any of the equipment listed below or it may be an equipment number, in which case, operator action is not required.

DA	disk cabinet, channel 0	CP	card punch
DB	disk cabinet, channel 1	CR	card reader
DC	disk cabinet, channel 2	LP	line printer
DD	disk cabinet, channel 3	MT	607 magnetic tape (1/2")
DE	disk cabinet, channel 4	WT	626 magnetic tape (1")
DS	display console		

This must be the first appearance of the name f in the job file. The file name f is alphanumeric, begins with a letter, and contains a maximum of seven characters. Multiple file names are not allowed.

The optional density parameter, dd, indicates LO, HI, or HY density (200, 556, or 800 bpi). If dd is omitted, the terminating period directly follows f, and density is set by the operator.

Examples:

ASSIGN50, TAPE6, LO.	Assign equipment number 50, LO density, to TAPE6. The job will be held up if this tape is presently assigned to another job.
ASSIGNMT, TAPE2.	Operator to assign 1/2" tape to file TAPE2.
ASSIGN WT, TAPE3.	Operator to assign 1" tape to file TAPE3.

For MT and WT, a message for the operator is displayed under the number of the job's control point:

WAITING FOR MT (or WT)

To assign tape 61 to control point 6, the operator would key 6.ASSIGN61.

ASSIGN01, TAPE9. Use disk unit 1 to store file TAPE9.

For tape, the ASSIGN statement is intended for scratch tapes only; the operator may assign any free tape of the specified type.

A user supplied tape should be assigned with a REQUEST statement.

4.3.2 REQUEST f,dd

This control card requests the operator at the system display console to assign to this job the peripheral equipment specified by f. This must be the first appearance of the name f. The job waits for operator action before proceeding.

The optional density parameter, dd, indicates LO, HI, or HY density (200, 556, or 800 bpi). If dd is omitted, the terminating period directly follows f, and density is set by the operator.

Example:

```
REQUEST TAPE4,HI.
```

Operator to assign an equipment for file TAPE 4, HI density.

In this case, the message REQUEST TAPE4. is displayed under the number of the job's control point, and the operator can key the number of the equipment on which the user's tape is mounted. For control point 4: 4.ASSIGN71.

The equipment number is related to specific equipment through an equipment table unique to the installation.

4.4 COMMON FILES

Common files are not discarded upon job completion. Normally, input files used by a running job (type local) are dropped; disk space is freed or equipment released. Output files are printed and discarded after printing. A job may declare a file to be common so as to make it available to other jobs. However, a job to which a common file is attached, can change it to local so it may be discarded.

4.4.1 COMMON f.

This control card has two effects:

1. If the file name, f, has common status in the FNT and is not being used by another job, it is assigned to this job until dropped. If the file is being used by another job or does not have common status, this job must wait until the file is available.

2. If the file name, f, already appears as a local file name for the job, the file will be assigned common status in the FNT and becomes available to any succeeding job after it is dropped by this job.

A file generated by a job may not be declared in a COMMON card until the job has been completed.

Example: COMMON BFILE.

4.4.2 RELEASE f.

With this control card, the common file named f currently assigned to this job will be dropped from common status and assigned local status in the FNT.

Example:

RELEASE BFILE.

The common file, named BFILE, attached to this job is changed to type local so that it will be dropped at the end of the job.

4.5 SWITCH, MODE, EXIT

SWITCH n.

This control card sets pseudo sense switches for reference by a subsequent FORTRAN program; n = 1-6. The settings are preserved at the control point and copied to RA for use by the central program. Switches may be changed by a console command.

Example:

SWITCH 6.

MODE n.

This control card may be used to change the arithmetic exit mode. n is a single octal digit. (See Exchange Jump Information, Section 2.3) The exit mode is set to 7 unless otherwise specified.

Example:

MODE 3.

```
EXIT.
```

The EXIT card can be used to separate the control cards associated with the normal execution of a job from a group of control cards to be executed in the event of an error exit as listed below:

1	TIME LIMIT.	Job has used all the central processor time it requested.
2	ARITHMETIC ERROR.	Central processor error exit has occurred. A dump of the exchange jump package is automatically written on OUTPUT.
3	PPU ABORT.	PP has discovered an illegal request, such as an illegal file name or request to write outside job field length.
4	CPU ABORT.	Central program has requested that the job be terminated.
5	PP CALL ERROR.	Monitor has discovered an error in the format of a PP call entered in RA+1 by a central program (can occur if a program accidentally writes in RA+1, as can condition 3).
6	OPERATOR DROP.	Operator has requested the job be dropped.
7	TRACK LIMIT.	The limit of 1000_8 half tracks assigned to a control point has been exceeded.

When one of these conditions occurs, an error flag (numbered as above) is set at the control point. In cases 1, 2, 5, 6, 7, a dayfile message is issued; and in case 3, the fault-finding PP issues a message (BUFFER ARG. ERROR from CIO, or NOT IN PPLIB).

When an error flag is set, a search is made for the next EXIT control card; and if it is not found, the job is terminated. If an EXIT card is found, the error flag is cleared and succeeding control cards are processed. If an EXIT card is encountered when no error flag is set, the job is terminated normally at that point.

Example:

MYJOB,1,400,100000.	Job card
REQUEST TAPE1.	Request scratch tape
RUN.	Compile and execute
EXIT.	
DMP.	Dump exchange package
DMP,1000.	Dump first 1000_8 words of storage
7,8,9	End of control cards
(Program)	
7,8,9	
(Data)	
6,7,8,9	

The dumps are made only if an error condition occurs.

Record Separator

This card, consisting of a 7,8,9 punch in column 1, separates the different types of records (control cards, source language cards, data cards) within a job.

File Separator

This card, consisting of a 6,7,8,9 punch in column 1, must be the last card of each job deck. No job may use information beyond this card.

4.6 COMMENT

```
COMMENT. comments
```

This control card allows comments to be listed in the dayfile. The period following the T is mandatory.

4.7 COMPILER AND PROGRAM CALLS

After the job card, any control card other than ASSIGN, REQUEST, COMMON, RELEASE, MODE, EXIT, or SWITCH, is a call for a central or PP program to be executed. A program may be in the library, or stored on a file used by the job. The control cards of a job are processed in order, and a number of programs may be executed in one job. A job is a set of programs using the same data files.

Parameters of a central program call follow on the same control card, for example:

RUN(P)	Compile, punch, and do not execute next record of INPUT file.
COPYBF(TAPE1,DISK2C)	Copy binary file from file TAPE1 to file DISK2C.
RUN.	Compile, execute, and do not list program on next logical record of input file. (Assumed by RUN if no compile mode parameter given.)

A program, compiled by RUN, is written as a binary record on a disk file with the same name as the program. The program can thus be executed separately, for example:

RUN(S)	Compile, list, and do not execute program.
PG8C.	Execute program called PG8C.

A program call control card is interpreted as follows:

1. The names of files attached to the job's control point are searched for the named program, and it is read to central memory from the next record of the file.

2. The library of central programs on disk 0 is searched for the named program; if found, the program is read to central storage.

3. The library of peripheral programs on disk 0 is searched for the named program, and the program is assigned to a PP.

The parameters of a central program are entered beginning at RA+2, left adjusted in display code, before execution. Parameters are normally compiled into a program, and overridden only if new parameters are specified by a control card call. A peripheral program may have two numerical parameters of at most 6 octal digits. These are entered in the input register of the PP which executes the program.

The first card of a program compiled by RUN is not a control card. It is part of the program and must include a list of files used by the program, for example:

```
PROGRAM SAM3 (INPUT, OUTPUT, TAPE1)
```

Such a statement supplies to RUN a list of files used by the program, which RUN enters from RA+2 as parameters in the binary form of the program. If the program is compiled and executed directly, those files will be used; but a separate call for the program can specify other files to be used instead. The binary form of SAM3 in which the parameters have been compiled could be used from the INPUT file:

```
JOB6,,,100000.
REQUEST FRED.
INPUT(,,FRED)
7,8,9
(SAM3 on binary cards)
7,8,9
(Data)
6,7,8,9
```

Here, any reference to TAPE1 in the source code of SAM3 would actually use FRED. As the first two file names were not overridden by the INPUT card, they would be used as in the source code.

The file names INPUT and OUTPUT are reserved for the job deck and output file for printing.

4.8 SYSTEM ACTION ON CONTROL CARDS

When a job is brought to a control point, the first record of the input file is copied to a 96-word buffer in central memory attached to the control point. If the control cards will not fit in this buffer, the message TOO MANY CONTROL CARDS appears in the dayfile and the job is terminated. Since cards are compactly stored 10 columns to a word without trailing blanks, a large number of control cards (about 40) are allowed; and the error usually arises from the omission of the 7,8,9 card following the control cards.

When a job is neither using nor awaiting the central processor or PP's, the monitor processes the next control card. If there are no more control statements, the job is terminated.

LOADER 5

The loader performs the following functions:

- Loads absolute and relocatable binary programs
- Links separately compiled or assembled programs
- Loads library subprograms and links them to user programs
- Detects errors and provides diagnostics
- Outputs a memory map
- Generates and loads overlays
- Loads segments

When data is to be transferred from any input or storage device to central memory, SCOPE calls the loader. Initially the call to the loader is generated by control cards but a running program may call it with user requests. All errors detected are written as diagnostics on OUTPUT. During the loading process the loader links subprograms together and generates overlays as directed. The memory map is created for all programs other than the main programs loaded from the system library. At completion of loading, the subprograms are ready for execution. The loader operates in both the central and peripheral processors. The central processor portion is loaded into the user's job area. The loader performs three types of loading: normal, segment and overlay; but all loading within one operation must be of the same type. The loader determines the type of loading from the file to be loaded.

5.1 LOADING TYPES

With segments and overlays, programs which exceed storage may be organized so that portions or groups of programs may be called, executed and delinked as needed. Since segments and overlays represent distinct types of loading, they may not be intermixed. To facilitate loading and delinking, both segments and overlays are identified by relative priorities called levels. A segment has one level number (1). An overlay has 2 levels, one primary and one secondary (1,1).

5.1.1 NORMAL LOADING

Relocatable binary subprograms may be compiled or assembled independently and brought together for execution. The loader links the subprograms by associating external symbols with entry points in other subprograms. If some external symbols remain unsatisfied, the loader attempts to satisfy them from the system library. If unsatisfied externals still remain, they will be satisfied with out-of-bounds references; but if this is done and the instruction is executed, the job is terminated and a message is issued. When loading is completed, control cards instruct the loader to proceed with execution or to bypass it and output the memory map.

5.1.2 SEGMENT LOADING

A segment is a group of relocatable subprograms or sections loaded and delinked as a unit. A section is a collection of relocatable programs with one section name; it is included in the loader scheme to reduce the number of program names in segment calls. The user defines the programs and sections which are to be included in a given segment. Segments allow the user to dynamically select programs which he requires in memory. Segment loading proceeds like normal loading. However, when additional segments are called, they may destroy existing segments.

5.1.3 OVERLAY LOADING

An overlay is a portion of a program written on a file in absolute form and loaded at execution time without relocation. Therefore the resident loader for overlays is substantially reduced in size. The user defines the overlay.

Loading an overlay may destroy previously loaded overlays in much the same way as segments operate.

5.2 USER REQUESTS

A program may call the loader with the following calling sequence:

```
RJ LOADER
CON param
```

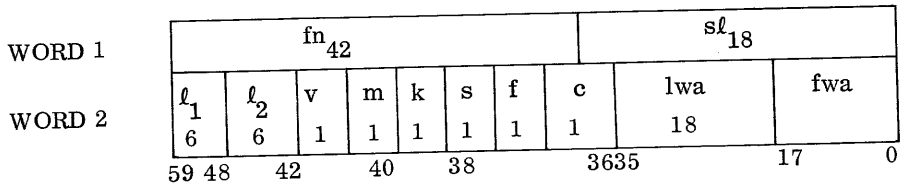
param is the location at which the user has established a parameter list for the desired load sequence.

When a job area is initially loaded with program data a small resident is placed within the user's field length. LOADER is an external symbol which will be satisfied by the loader and which will ultimately reference an entry point in this small resident.

Files are not rewound at the beginning of a loading operation initiated by a loader request. It is the user's responsibility to position his file properly. Loading is performed from the file in an end-around fashion from current position until the required programs are found or until the end-of-file is reached for the second time. In the latter case a fatal error occurs. All loading operations search in the same end-around fashion.

5.2.1 PARAMETER LIST FOR CALLING LOADER

The parameter list is composed of 2-word entries:



The parameter list is terminated by 60 bits of zero in word 1. Names may not exceed seven characters.

Word 1

fn is one of the following:

1 name of the file from which programs are to be loaded

2 name of an entry point in a program

3 subprogram name

4 system library if fn = 0

sℓ

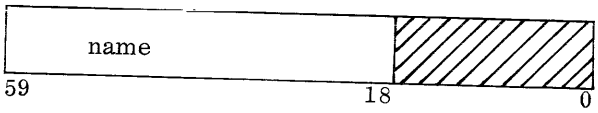
For normal loading:

list of subprograms or entry points to be loaded from file fn. If sℓ is blank, the entire file is loaded.

For segment loading:

the location of a segment or a list of sections and/or subprograms to be loaded as a segment.

Each entry in the sℓ list has the following format where name is the subprogram word.



name Subprogram name

The sℓ list is terminated by a zero word.

Word 2

ℓ_1	Segment level (1-63) if s = 1 and v = 0
ℓ_1	Primary overlay level if s = 0 and v = 1
ℓ_2	Secondary overlay level.
v	Overlay flag; if v = 1 an overlay load operation is requested.
m	NOMAP flag. If m = 1 all maps of segment or overlay loads will be suppressed. Otherwise a map will be written on the OUTPUT file.
k	Search key; if k = 1, fn is the name of an entry point. The search key is used to find the address of a previously loaded entry point; no loading is performed.
s	Segment flag; if s = 1, a segment loading operation is requested.
c	Complete flag. If c = 1, loading is to be completed by loading subroutines from the resident subroutine library and/or the central disk library.
f	Fill flag. If f = 1, unsatisfied external symbols will be filled with out-of-bounds references.
lwa	Last location, relative to RA, available for the loading operation. If lwa = 0, RA+FL (FL = user's field length) will be used. LOADER will place its tables at lwa-1.
fwa	Initial location, relative to RA, to begin loading. If fwa = 0, loading begins at the next available location as determined by the current state of the loading operation.

The following are examples of parameter lists which might be processed by the loader.

1. Load From File

fn = name of file

sℓ = 0

v = 0

k = 0

s = 0

All subprograms are loaded from fn until end-of-file is encountered.

ℓ is ignored.

If c = 1 loading will be completed.

2. Load Named Entry

fn is one of the following:

fn 1 name of entry point in a subprogram

2 name of a subprogram

sℓ = 0

v = 0

k = 0

s = 0

The named routine will be loaded from the system library.

ℓ is ignored.

If c = 1 loading will be completed.

3. Load Segment From File

The segment defined by the list at sℓ will be loaded from fn at level ℓ. If ℓ_1 > current segment level, the segment will be loaded at the current level +1. If $\ell_1 \leq$ current level, segments at a higher level will be removed. If a subprogram specified in the segment list is not located on fn, the system library is searched. fn is not rewound prior to loading.

fn = name of file

sℓ = address of list containing a segment name or section names and subprogram names only.

ℓ = desired level

v = 0

m = 1

k = 0

s = 1

If c = 0 loading will be completed by establishing the origin and length of blank COMMON.

If c = 1 loading will be completed normally.

If f = 1, unsatisfied external references will be set to out-of-bounds references.

No memory map is output.

4. Load Named Subprograms From File

The list of subprograms specified by the list at sℓ will be loaded from fn and the system library.

fn = name of file

sℓ = address of list

v = 0

k = 0

s = 0

ℓ is ignored

If c = 1 loading will be completed.

If fn = 0, only the system library is searched.

If fn = 0 and sℓ contains only 1 entry name, it is the same as sℓ = 0 and fn = entry name.

5. Load Overlays

fn = name of file

sℓ = zero

ℓ_1 = primary level

ℓ_2 = secondary level

v = 1

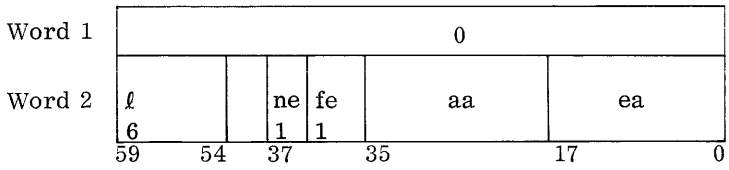
s, f, c and k are ignored.

The overlay file built during the initial load from overlay cards and binary text is searched for the unique identifier ℓ_1, ℓ_2.

The overlay is then loaded into its absolute locations. The absence of such an overlay will cause the loader to set the fatal error flag.

5.2.2. PARAMETER LIST FOR REPLY FROM LOADER

When LOADER has completed the requested operation (loading not necessarily completed) LOADER signals the caller by setting the parameter list as follows:



ℓ Level at which the segment was loaded. ℓ = 0 if segment loading was not requested.

ne Non-fatal error flag. ne = 1 if the following loading errors were detected:

Unsatisfied externals if c = 1

Duplicate occurrence of a named program; all but the first occurrence will be ignored.

fe Fatal error flag. fe = 1 if the following loader errors were detected by LOADER.

Improper deck structure

Improper parameter specification

Requested file name, program name, or entry point not found.

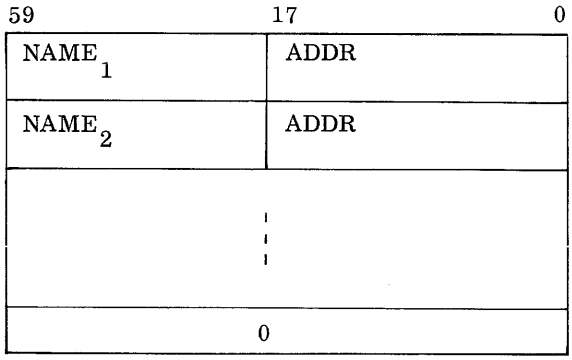
ea Entry address of named entries, specified in a transfer table. If f = 0, ea is the location (relative to RA) of last encountered named entry. If there is more than one entry named in the table, the last encountered entry will be in ea, and the previous entry will be in aa.

If k = 1, ea is the location (relative to RA) of the named-entry fn. If v = 0, ea is the entry point to the overlay. If ea = 0, no name was found.

aa Additional entry address of named transfers. aa = 0 if less than two named transfers were encountered. aa = address of next to last name if more than one named transfer was encountered.

If sℓ was non-zero, fn=0, and s=0 in the parameter list, the list of entry points and/or subroutines to be loaded from the library will contain the address at which each name is loaded. If the name was not loaded the address will be zero.

The list will then have the form:



5.3 RELOCATABLE SUBROUTINE TABLES

The loader requires certain information about each subprogram in order to load, relocate, and link it. This information, output by ASCENT and the various compilers in the form of tables, forms part of the logical record of each subroutine.

5.4 MEMORY ALLOCATION

5.4.1 SYSTEM USAGE

Storage areas are allocated within the user's declared field length in contiguous memory locations. The first 100_8 locations of the area are automatically assigned as follows:

RA+0 RA+1	reserved for use during execution
RA+2 : RA+63	Parameters from the program call card which are available to the user during execution
RA+64	Number of parameters in lower 12 bits
RA+65	Zero
RA+66	Address of the origin of user's program area
RA+67	Loading control words
RA+70 : RA+77	Upon initial entry from a named routine call or an EXECUTE card, these locations will contain the card image, in display code, of the card which called for execution.

After location $RA+77_8$ the storage is allocated as follows:

Usually the user's first loaded subprogram will be originated at $RA+100_8$. However, if a section card appears prior to an initial loading operation, a section definition table (SDT) will be originated at $RA+100_8$. The actual origin of the user's program area can be found in the lower 18 bits of RA+66.

The system establishes loader tables at the high end of the user's field length area. These tables, which are present during the loading operation, are discarded when normal or overlay loading is complete.

If a segment was loaded, the tables are moved to a point immediately following the last loaded subprogram common block. Blank common may overlay the loader and its tables. Conversely, if the loader is called again, it may overlay blank common. The user must allow for the loader, its tables, and blank common within his field length definition if he wants to preserve his data.

There is no protection against the programmer destroying the loader or loader tables. Both of these areas are checksummed and this checksum is verified upon initial entry into LOADER. If the loader is destroyed, the next RJ LOADER will cause the verification routine to call it again. But if this initial verification routine is destroyed the results of RJ LOADER may be meaningless.

5.4.2 USER ALLOCATIONS

Subprograms and their associated labeled common blocks are assigned memory area as they are encountered. The initial declaration of a labeled common block establishes the maximum length for that block. Declarations in subsequent programs must have a length which is less than or equal to the original declaration; otherwise, a diagnostic will be issued.

Blank common relocation information is preserved until loading is completed, at which time it is allocated to the area following the last loaded program and/or labeled common block. Declarations of blank common may vary between subprograms; the largest declaration determines the memory allocation.

5.5 MEMORY MAP

After completion of loading, a map of the user's area is provided on the OUTPUT file. The map is optional and can be suppressed by setting the NOMAP bit in the LOADER parameter to 1. The map includes:

- Names, lengths, and locations of loaded programs
- Names and locations of entry points with a sublist of all programs referencing the entry point
- Name and locations of common blocks
- Total length of all loaded programs and common blocks, both labeled and blank.
- Length of the loader and its tables
- Unsatisfied external references

During execution of a segmented or overlay job, a record of a new segment or overlay load is provided each time a call is made to LOADER.

5.6 SEGMENTATION

The user defines a segment with a SEGMENT card, which is a loader card described in section 5.9. Segments are loaded by the loader during initial load. A running program may load a segment with a user request.

5.6.1 LEVELS

Each segment is assigned one level number ($0\text{-}77_8$) by the user. The level serves as a programmer's tool for rapid delinking of segments. Level zero is reserved for the initial or main segment which remains in memory during segment execution; subsequent segments may be loaded at any level. The number of segments in central memory at one time is limited only by the amount of memory available.

The loader indicates the level at which the segment was loaded in the parameter list (5.3.2).

5.6.2 LOADING SEGMENTS

When a segment is loaded, its external references will be linked to their corresponding entry points in subprograms and common blocks of previously loaded segments at lower levels. Unsatisfied references in the segments will remain unsatisfied. Subsequent segments loaded may include entry points to satisfy them; or the user may specify that they be satisfied from the system library. If execution is attempted and unsatisfied externals exist, the job will be terminated and a message issued.

Levels are used to delink segments that are no longer needed. If a segment is loaded at a requested level which is less than or equal to the level of the last loaded segment, all segments at levels down to and including the requested level will be delinked and removed. If a segment is loaded at level 6, any segments previously loaded at level 6 or 7, 8, and upward will be delinked and removed. When a segment is delinked, the linkage of its entry points to external references in lower levels are destroyed and the externals are unsatisfied once again.

Example:

> A SINE routine is loaded in a segment at level 2. If any external symbols refer to entry points in level 1, they are linked. To try an experimental version of SINE, the user loads a segment containing new SINE at level 3. The original SINE remains at level 2 and so do its links to level 1; but, any new segments loaded at higher levels will link to the new SINE at level 2. The linkage will remain until a new level 3 is loaded, in which case the SINE at level 3 would be wiped out and any references to it left unsatisfied. If a new SINE were loaded at a level higher than 3, any segments loaded afterward would be linked to it.

5.7 OVERLAYS

Overlays are generated by the loader via an overlay card and written on a file to be called as needed for execution. The overlay card, which is a loader card, is described in 5.9.

5.7.1 LEVELS

Each overlay in a program must have a unique identifier, consisting of a pair of numbers or levels ($0\text{-}77_8$). The first number is the primary level, the second is the secondary level. An overlay with a non-zero primary level and a zero secondary level (1,0) is a primary overlay. Any overlay with the same primary level and a non-zero secondary level (1,1) is associated subordinate to it and is a secondary overlay. This difference is significant when overlays are loaded.

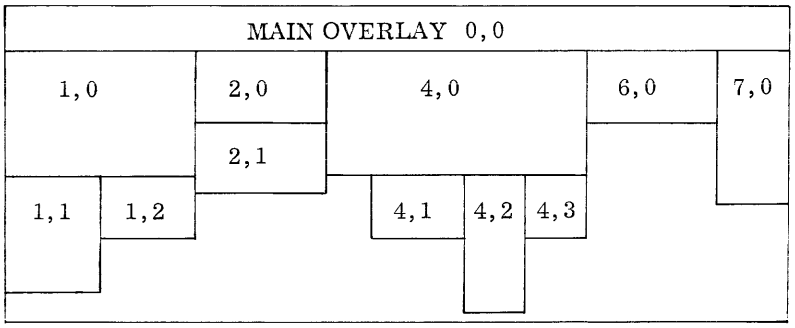
Level 0,0 is reserved for the initial, or main overlay which is neither primary nor secondary but a special case which remains in memory during overlay execution. Overlay numbers (0,1) to (0,77) are illegal.

5.7.2 LOADING OVERLAYS

The main overlay (0,0) is loaded first. All primary overlays are loaded at the same point immediately following the main overlay. Secondary overlays are loaded immediately following the primary overlay. Loading the next primary overlay destroys the first loaded primary overlay and any associated overlays. Likewise, the loading of a secondary overlay destroys a previously loaded secondary overlay.

Two levels of overlay, one primary and one secondary, are available to the programmer.

Example:



0,0 is the main overlay which remains in memory during execution. 1,0 is a primary overlay. When 1,1, a secondary overlay, is loaded, it is subordinate to 1,0 and there are two levels of overlay. Loading 1,2 destroys 1,1 and 1,2 becomes subordinate to 1,0 but there are still two levels of overlay. When 2,0 is loaded, it destroys 1,0 and its secondary overlay; and there is only one level of overlay until 2,1 is loaded. Similarly, when 4,0 is loaded, it destroys 2,0 etc.

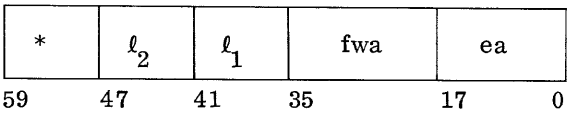
All external references in overlays must be directed upward. For example, the primary overlay may contain references to the main overlay but not to the secondary overlay. The secondary overlay may contain references to the main overlay or to the primary overlay. The primary overlay calls the secondary overlay through the loader.

When the loader detects illegal overlays (identification or size error), loading of all overlays is completed and memory maps recorded on output; but an abort flag is set which causes the system to bypass the next execution.

5.7.3 OVERLAY FORMAT

Each overlay consists of a logical record in the following format:

Word 1



*	Unused
ℓ_1	Primary overlay level
ℓ_2	Secondary overlay level
ea	Entry point to the overlay
fwa	First word address of overlay (overlay is loaded at fwa)

Word 2 through end of record: 60-bit data words.

5.8 LOADER CARDS

Loader cards are processed directly by the loader rather than by the monitor. They provide the loader with information necessary for generating overlays and segments. All loader cards must precede the subprogram text to be loaded. Formats are the same as for SCOPE control cards.

5.8.1 SEGMENT CARDS

All subprograms named in a segment must reside in the same file.

SEGZERO

All programs requiring segment loading must have a SEGZERO card defining the first segment. There may be only one SEGZERO card in the initial load. If the user wishes to load a new segment level zero, it must be defined in the user's parameter list and called in by the user's program.

SEGZERO(sn, pn_1, pn_2, . . . pn_i)

sn	Segment name
pn_i	Names of subprograms or sections

SEGMENT

Segments other than segment zero may be defined by a segment card or in the user's program.

SEGMENT (sn, pn_1, pn_2, . . . , pn_i)

sn and pn_i are defined as in SEGZERO

SECTION

This card defines a section, or group of programs within a SEGMENT.

```
SECTION (sname, pn1, pn2, . . . , pni)
```

sname	Name of the section
pn_i	Name of a subprogram belonging to the section

If more than one card is necessary to define a section, consecutive cards with the same sname may follow. Whenever the named section is loaded, all subprograms within a section will be loaded.

All section cards must appear prior to the SEGMENT cards which refer to the named sections.

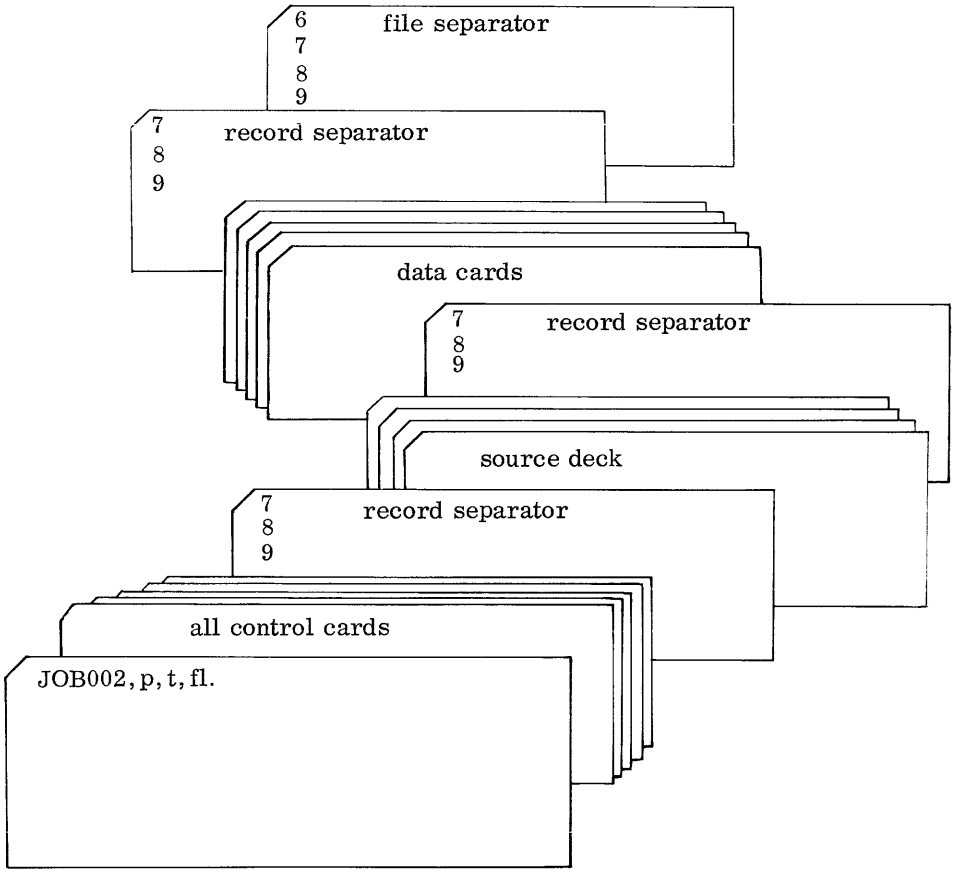
5.8.2 OVERLAY CARDS

```
OVERLAY (fn, ℓ1, ℓ2, Cnnnnnn)
```

fn	File name onto which the generated overlay is to be written	
ℓ_1	Primary level number	must be 0, 0 for first overlay card
ℓ_2	Secondary level number	must be 0, 0 for first overlay card
Cnnnnnn	optional; nnnnnn is 6-octal digits. If absent, overlay is loaded normally. If present, overlay is loaded nnnnnn words from the start of blank common. This provides a method for changing the size of blank common at execution time.	

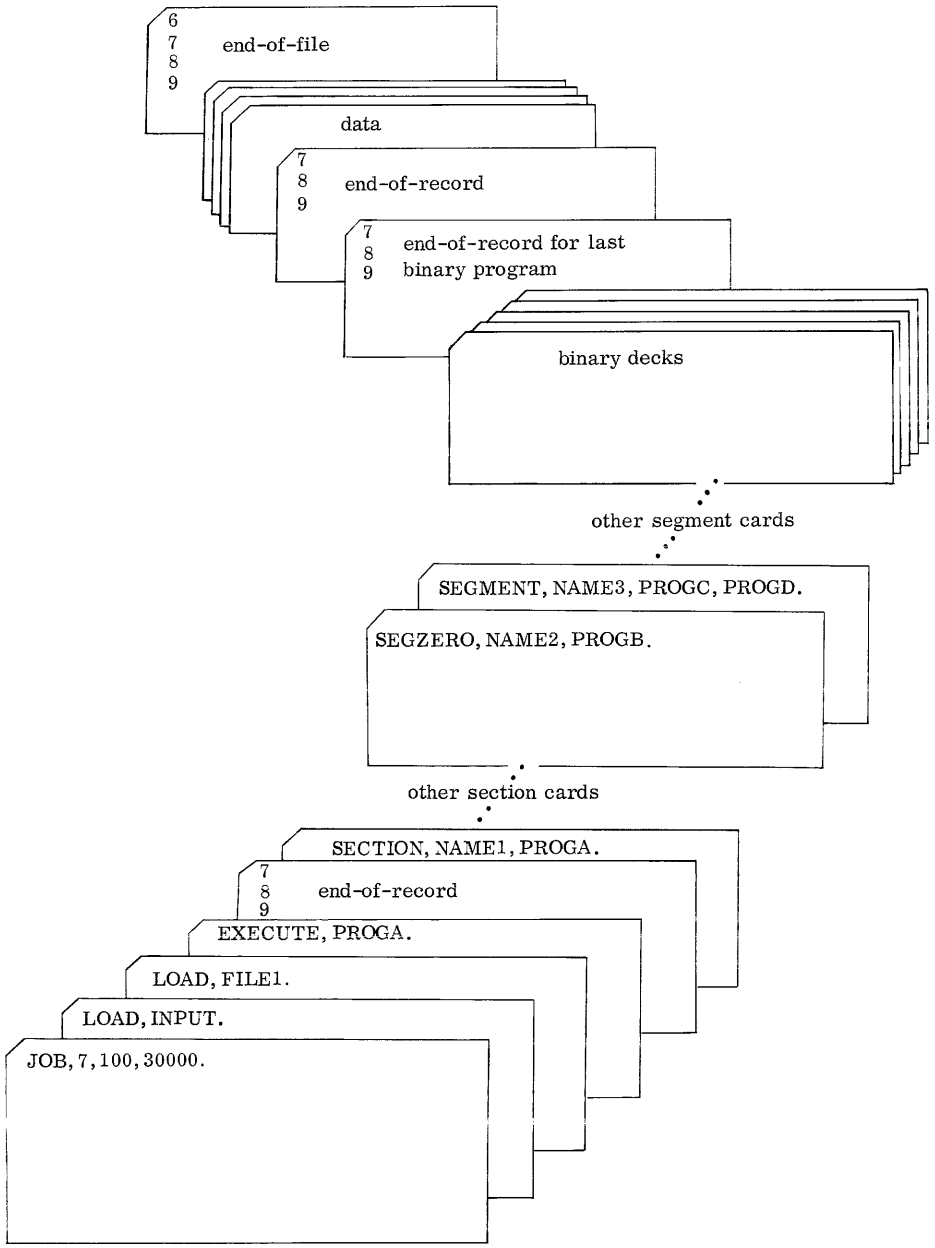
The first overlay card must have an fn. Subsequent cards may omit fn, and the overlay is written on the same fn.

The following card deck indicates the arrangement of control cards to begin a job, separate job records, and terminate a job. The record separator which must be used between different types of cards has 7, 8, 9 punches in column one. The file separator which terminates a job has 6, 7, 8, 9 punches in column 1.



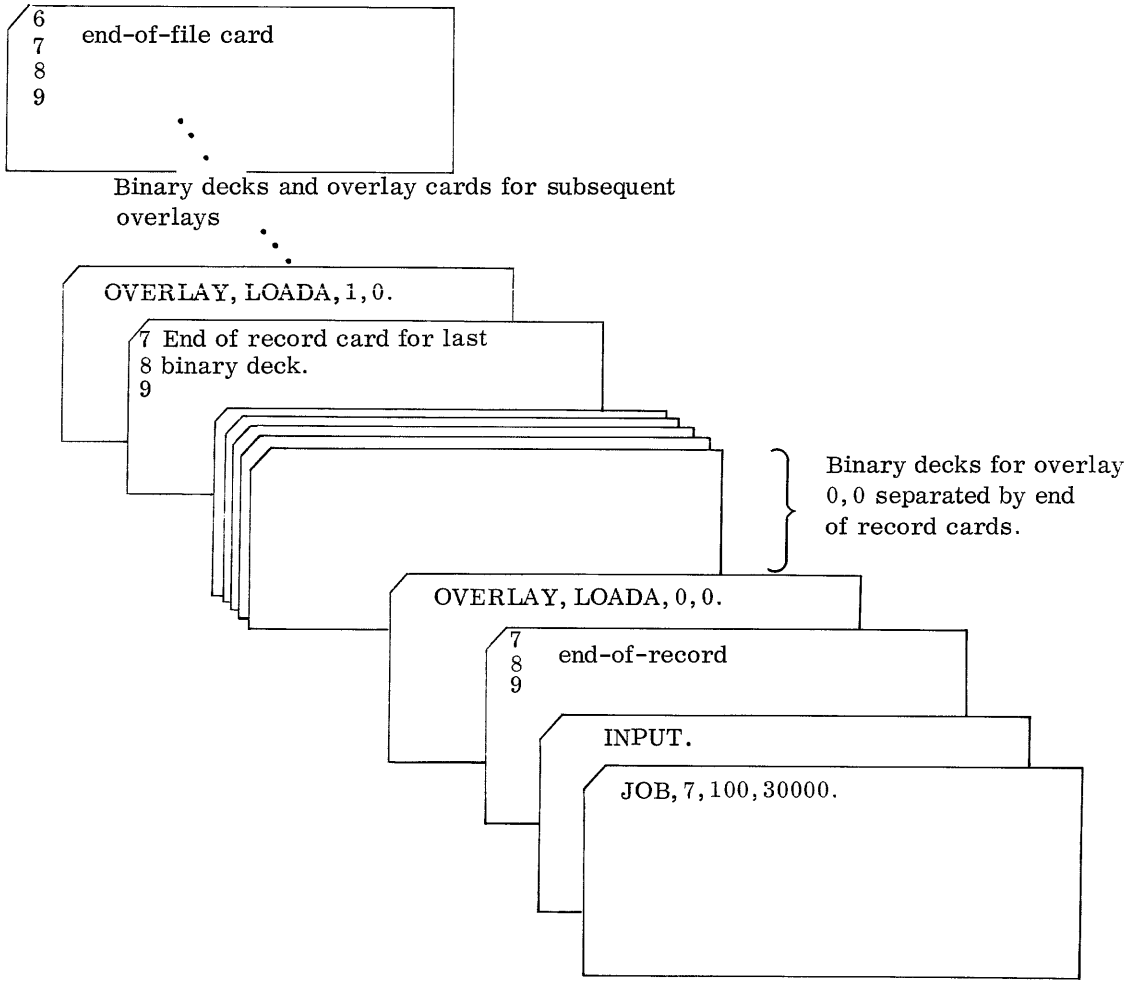
SEGMENT LOADING

Load segments from INPUT and FILE1,
and execute program PROGA.

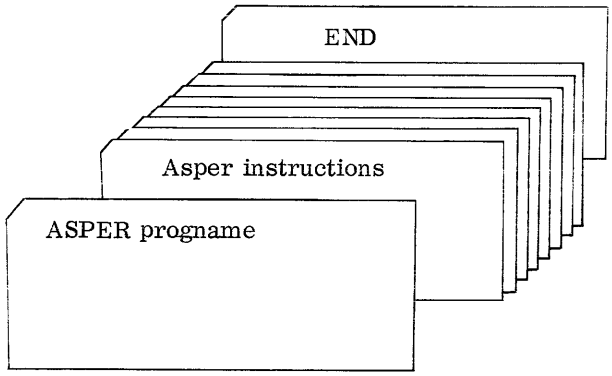


OVERLAY LOADING

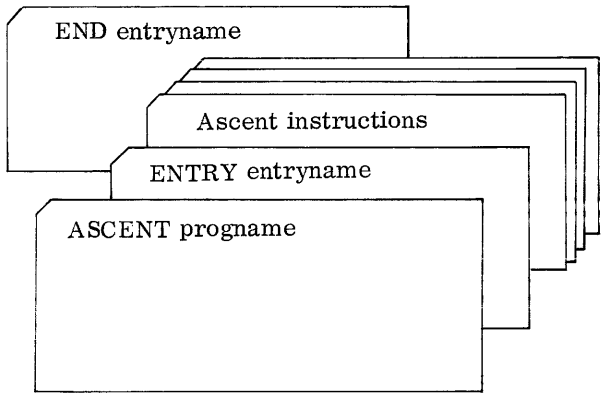
Generate overlays on file LOADA and execute overlay 0, 0.



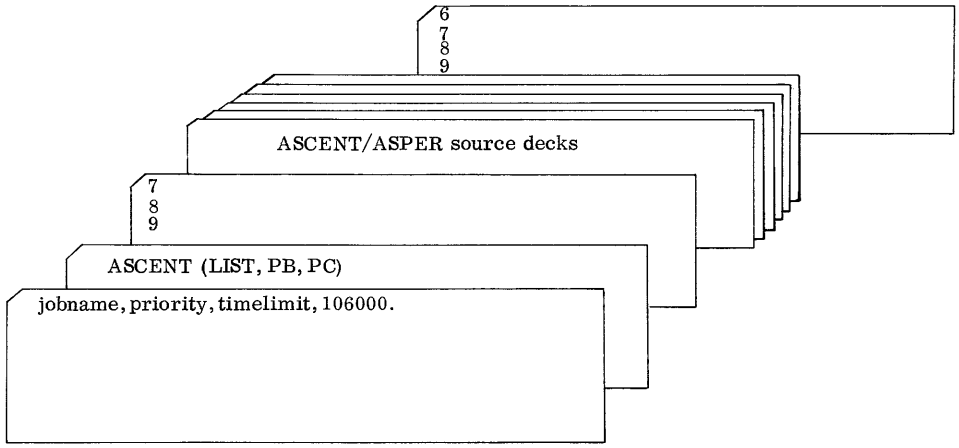
ASPER SOURCE DECK



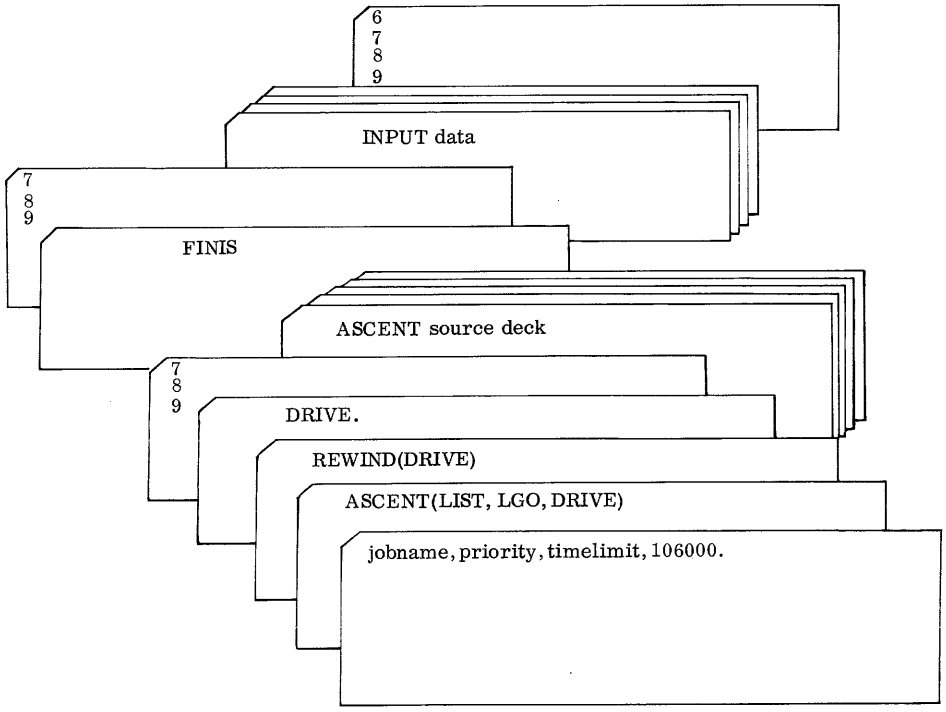
ASCENT SOURCE DECK



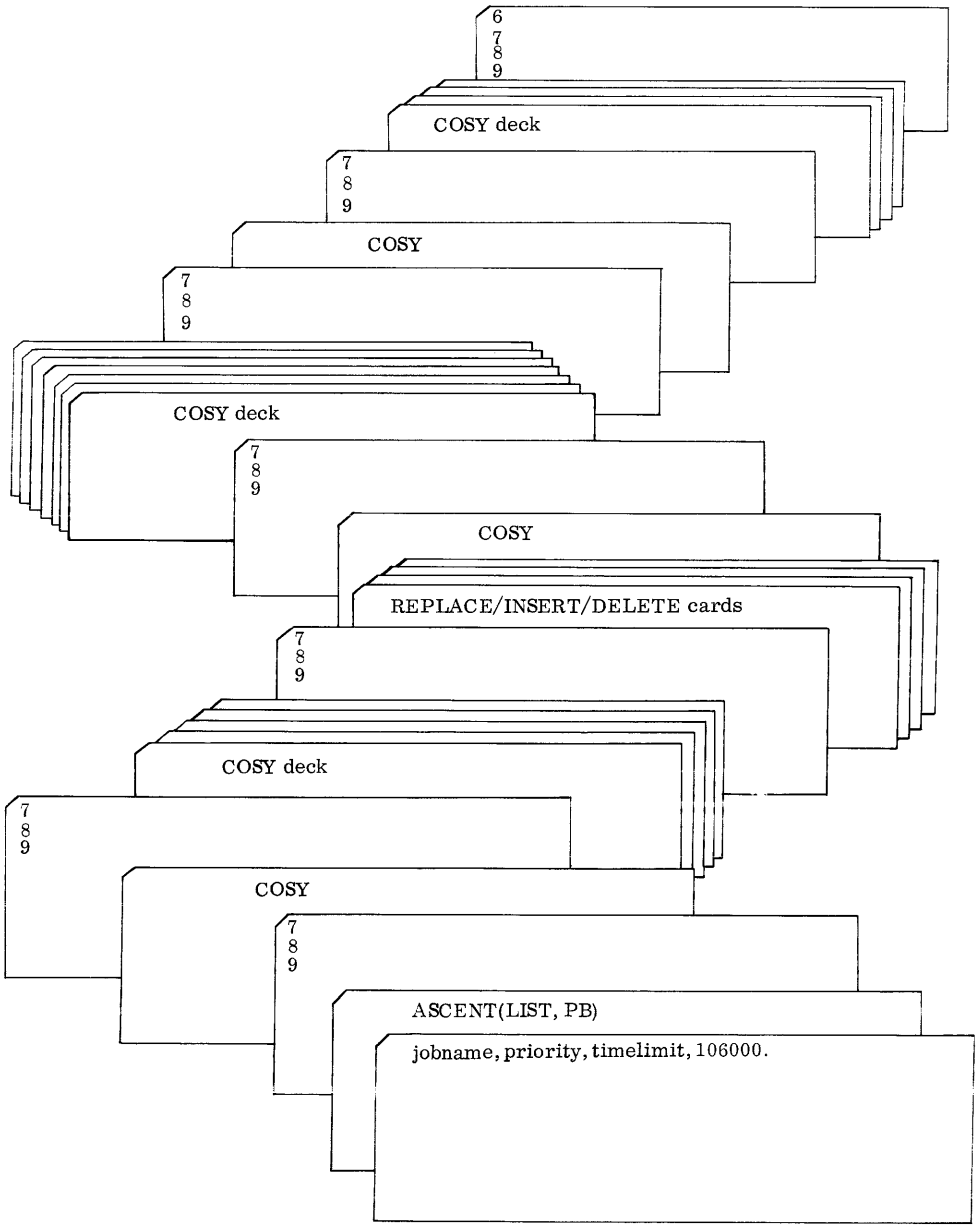
SIMPLE ASCENT/ASPER ASSEMBLY; output listing, binary decks, and COSY decks



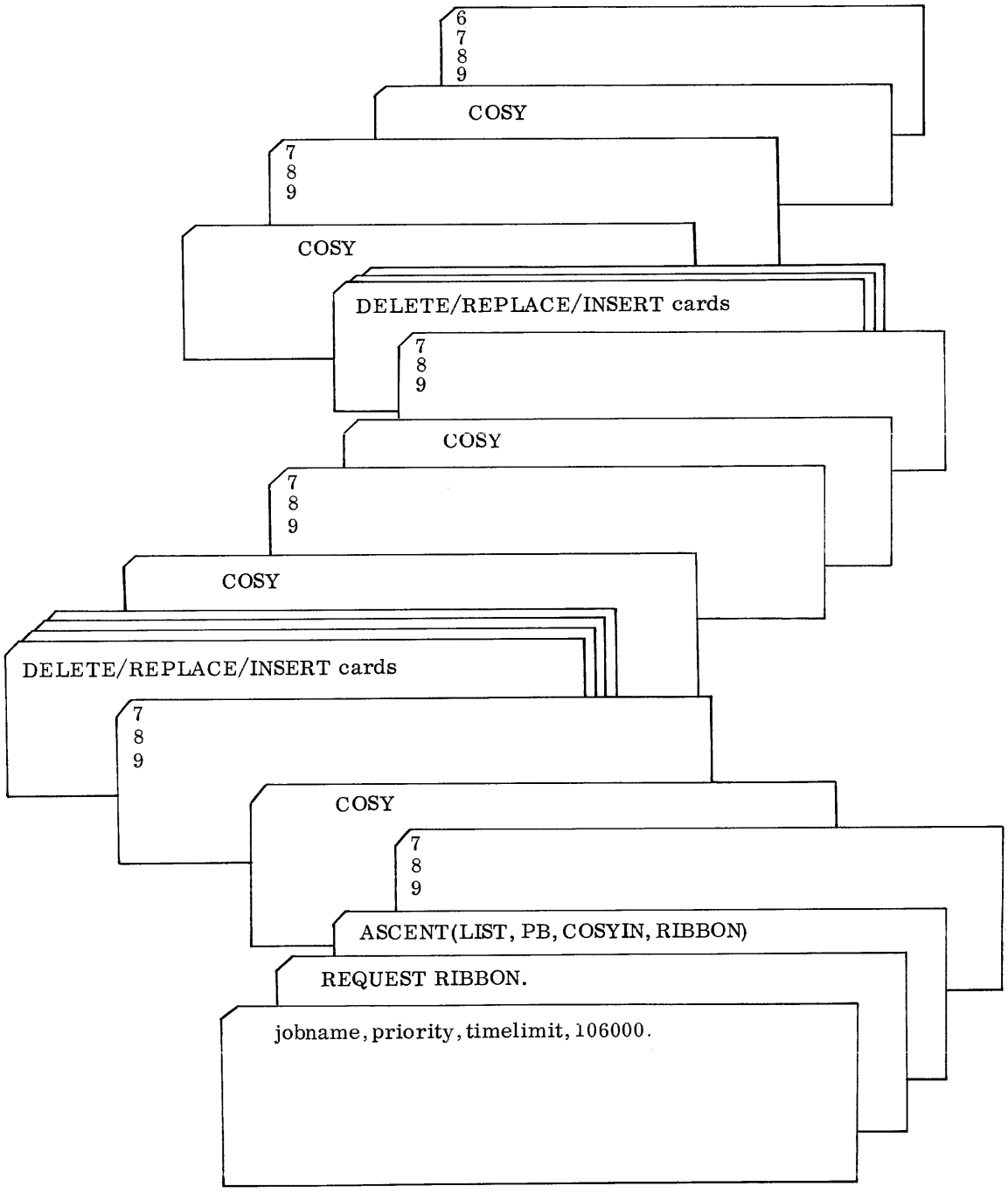
SIMPLE ASCENT ASSEMBLY FOR LOAD-AND-GO; output listing only



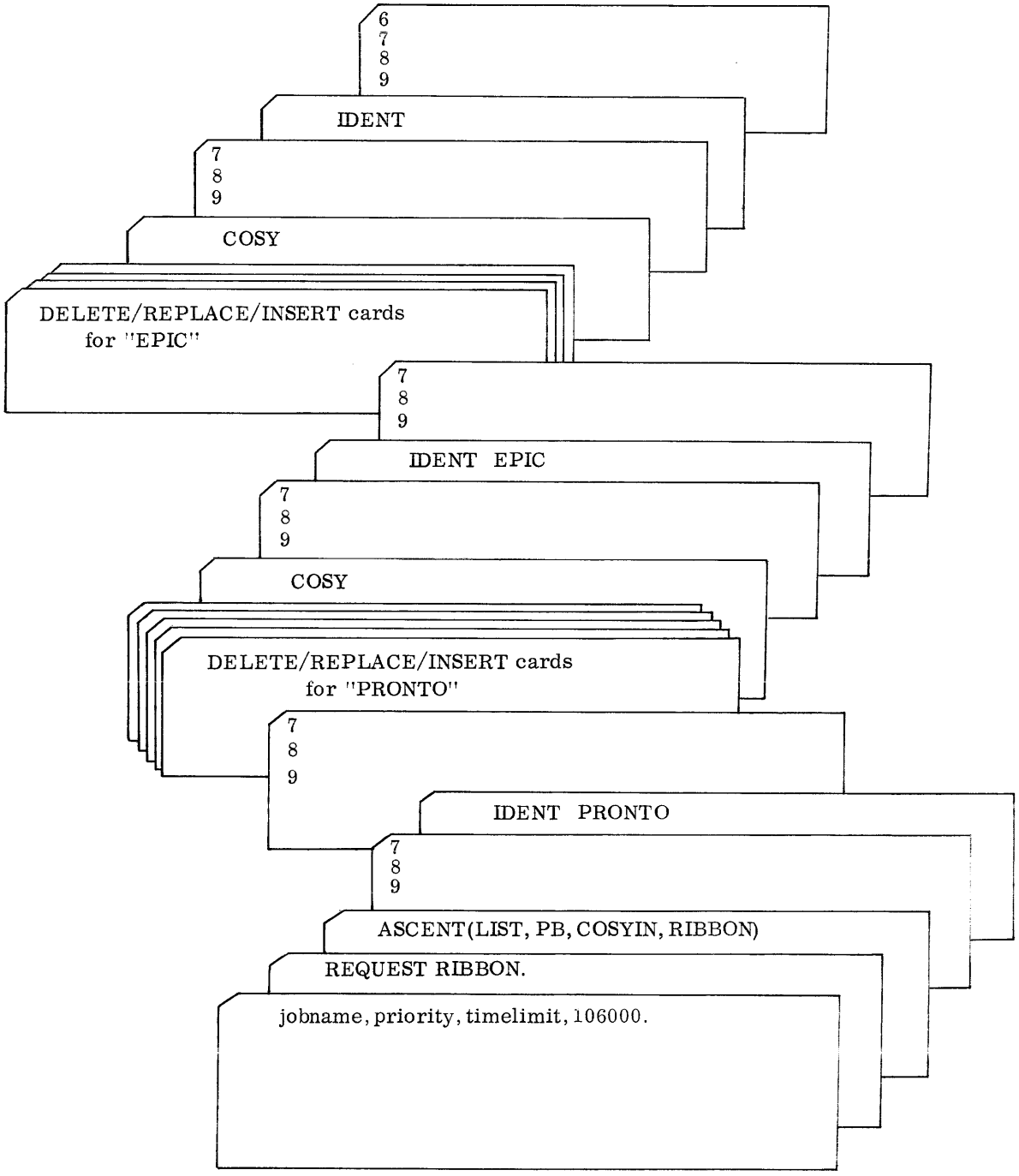
ASCENT/ASPER ASSEMBLY FROM COSY DECKS with modifications; output listing, and binary decks. Since no REQUEST card is included, the input file is on disk.



ASCENT/ASPER ASSEMBLY FROM COSY DECKS with five COSY decks on a tape named RIBBON.



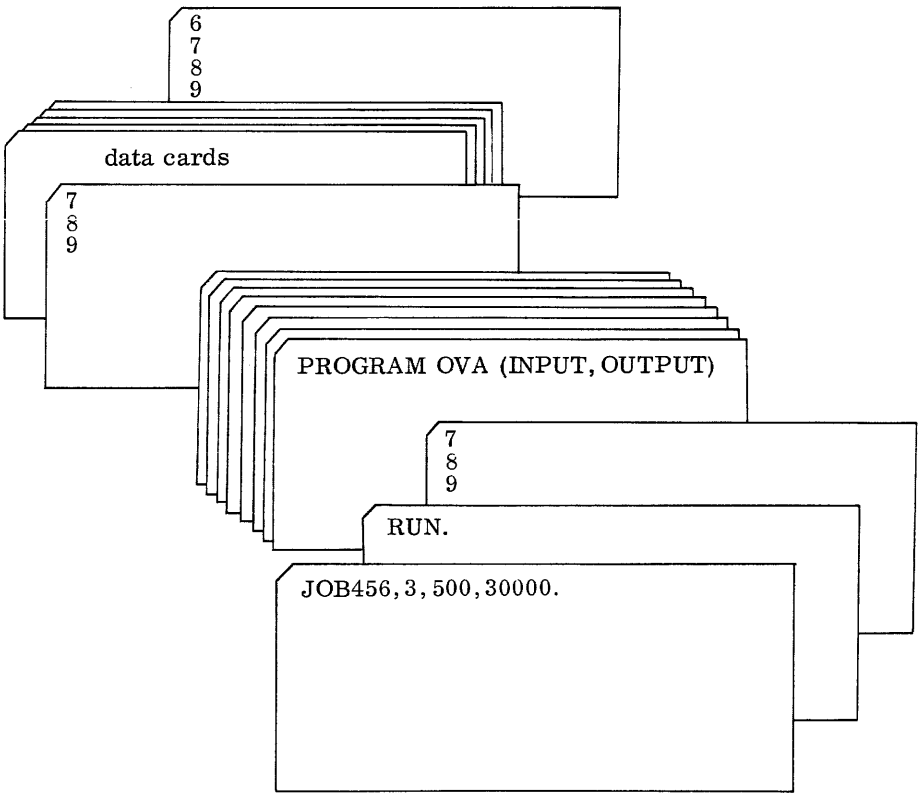
ASCENT/ASPER ASSEMBLY FROM COSY DECKS; with five COSY decks on a tape that will be named RIBBON; the second program to be named PRONTO and the fourth EPIC. Modifications are specified for PRONTO and EPIC only.



FORTRAN Load and Run

Job 1

INPUT and OUTPUT are the only I/O files used; no special control cards.



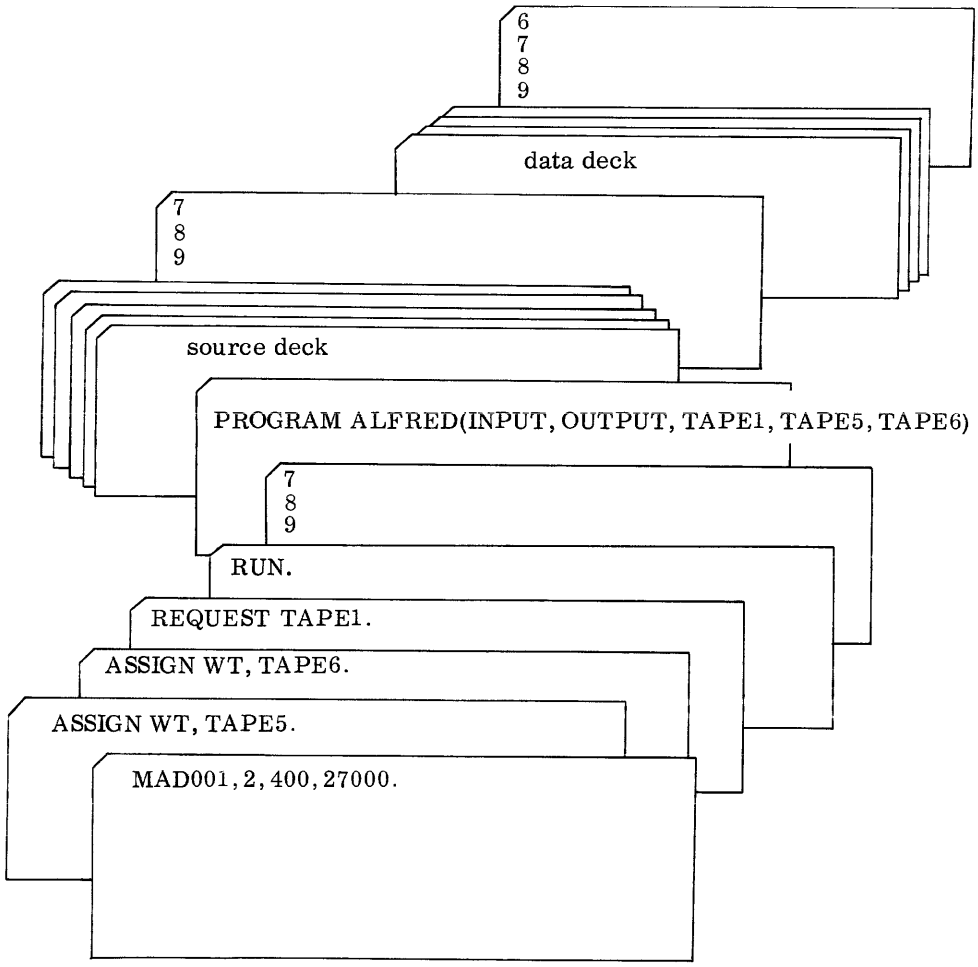
FORTRAN Load and Run

Job 2

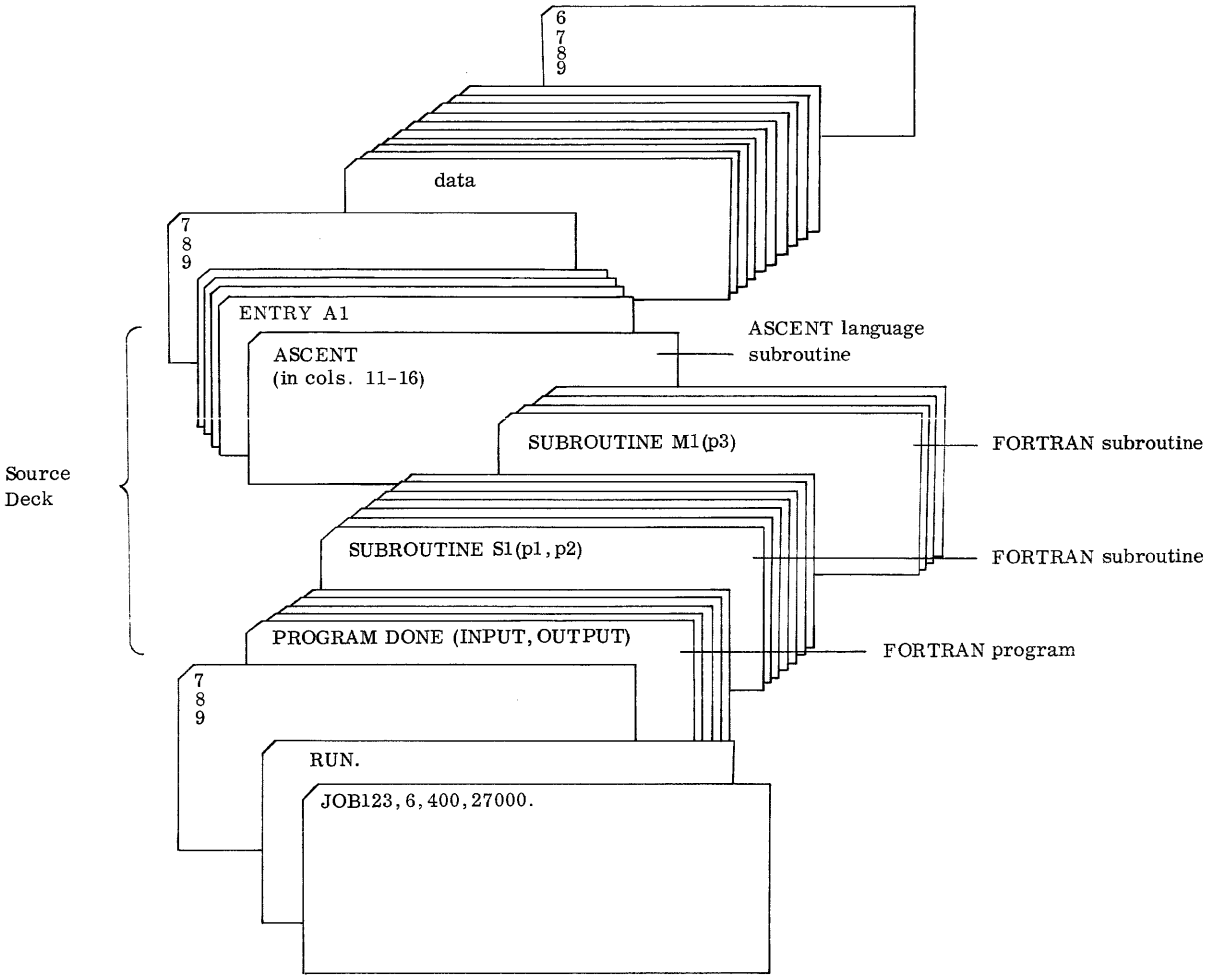
Three tape references:

TAPE1 — assumed input tape which operator loads on a particular unit

TAPE5
TAPE6 output scratch tapes drawn from tape pool

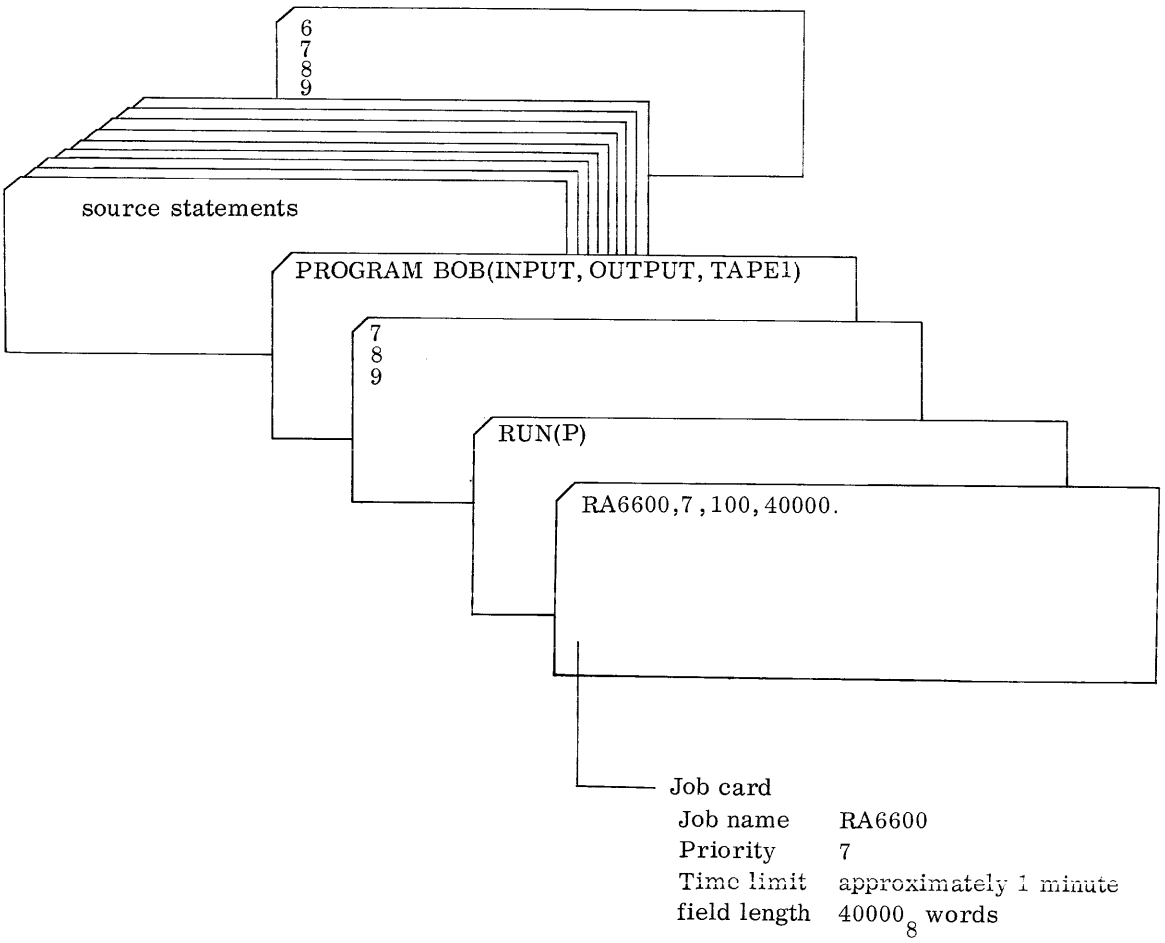


FORTRAN Compile and Execute with Mixed Deck

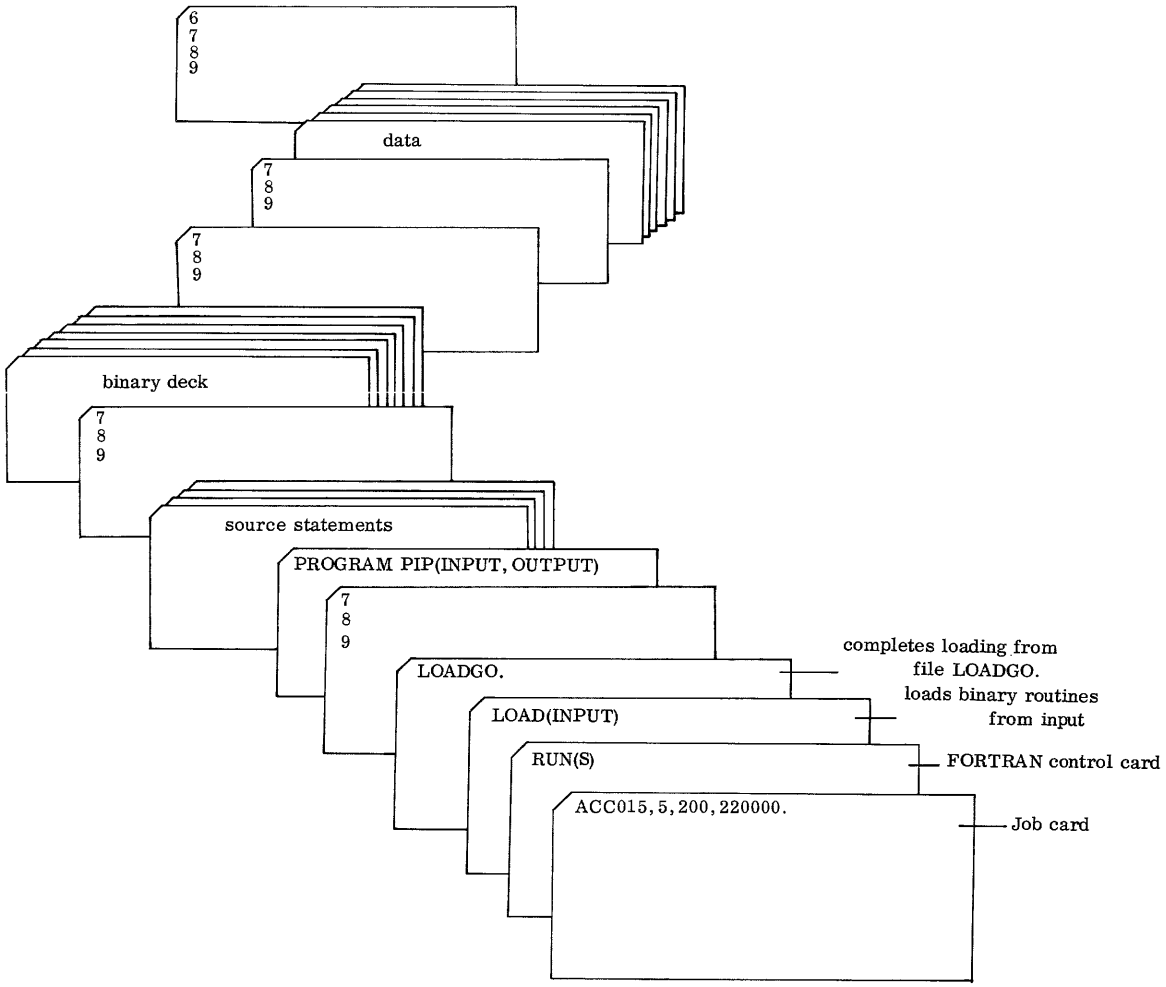


FORTRAN Compile and Produce Binary Cards; do not execute.

Three files of I/O - INPUT, OUTPUT and TAPE1

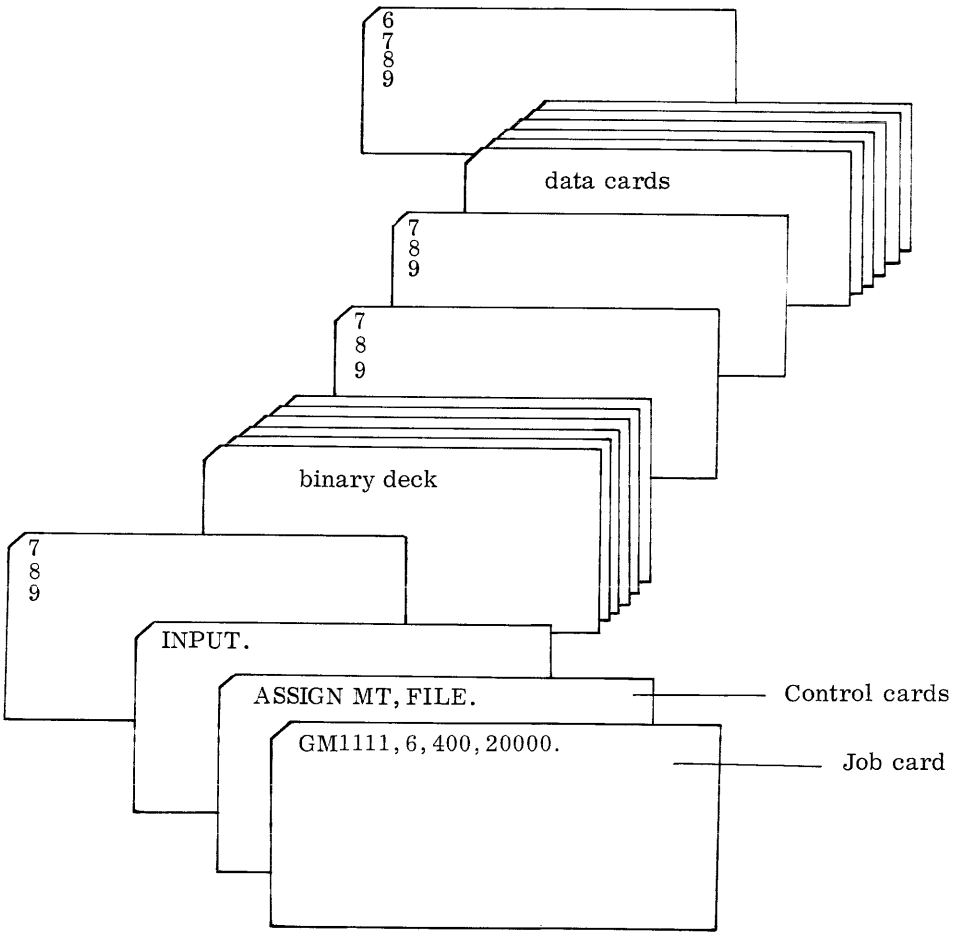


FORTRAN Compile and Execute (plus a prepunched binary subroutine deck)

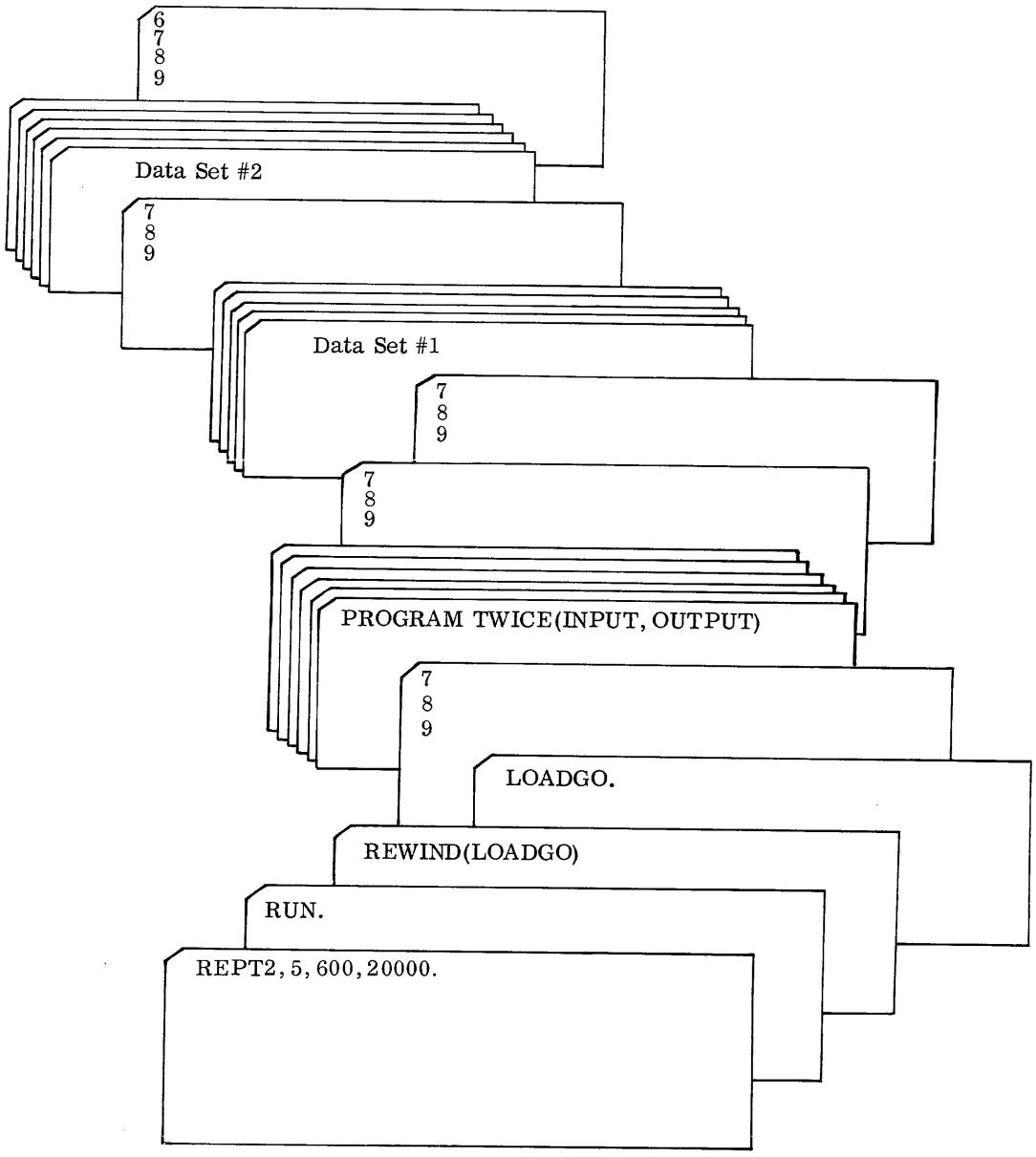


Load and Execute a Prepunched Binary Program

The binary cards in the input file following the record separator are loaded into central memory when the program call card INPUT is encountered.



Compile Once and Execute Twice† with Different Data Decks



† Program TWICE must read the end-of-record card.

SCOPE UTILITY PROGRAMS 7

The SCOPE library contains a set of utility programs which may be called by control cards during the execution of any job. These programs include card-to-tape, tape-to-tape, tape-to-print, and general file manipulation as well as creation of new files.

All utility operations are performed with named files, each of which designates a specific peripheral device, such as a card reader, tape unit, printer, card punch or disk file.

Before the first reference to any named file, an equipment must be assigned to it by the operator or programmer through the ASSIGN statement; otherwise the system assigns the file to a disk unit. Therefore all files, except disk, specify a unique peripheral equipment and all references to a specific equipment are made through the file name.

Utility jobs conform to the normal deck structure. The utility job deck contains the following:

Job Card	First control card
Request cards	Equipment assignment
Program cards	Data operations
6,7,8,9	End of job

The job card includes job name, job priority, time limit and field lengths.

If only utility programs are to be executed a short field may be specified (2000). In all copy operations, the central memory buffer is automatically set up to use the entire field length of the job. Some operations between high speed devices may be accelerated with a larger field length.

All necessary files which do not reside on the disk should request the operator to assign equipment. Tapes can be rewound and positioned upon request.

The program cards name the utility routines. The copy routines are used for positioning files by copying unassigned temporary files to the position point.

Examples:

To print the third and fourth coded files from a tape:

TAPEFRT,17,1000,2200. (Job card)

Assign unique file names, PRINTER and MAGTAPE, with REQUEST control card to a printer and tape unit.

REQUEST PRINTER. (Operator would assign available printer i.e.,3.ASSIGN20)

REQUEST MAGTAPE. (Operator would assign specific tape unit i.e.,3.ASSIGN53.)

Rewind tape unit to be sure of actual position.

REWIND(MAGTAPE)

Skip tape to beginning of third file by copying first two files to an unused dummy file XXX.

COPYCF(MAGNTAPE,XX,2)

Copy the two coded files to the printer.

COPYCF(MAGTAPE,PRINTER,2)

An end-of-file card completes the job.

```
6
7
8
9
```

The following routines can be called by control cards with the entry beginning in column one.

7.1 BACKSPACE LOGICAL RECORD

```
BKSP(file1,n)
```

This program allows backspacing of multiple logical records as specified by the decimal n. Backspacing will terminate if file becomes rewound

7.2 CATALOG SYSTEM TAPE FILE

```
CATALOG(file1,file2)
```

This routine provides a convenient means for identifying the contents of a system tape. It may include information as to record and routine lengths in addition to checksum information which will uniquely identify a particular version of a system component on that system tape.

The information is taken from file1 and listed on file2. If parameters are omitted, LIBRARY,OUTPUT are assumed.

CATALOG output is formatted as follows:

RECORD	Number of the logical record with respect to its position on tape. Zero-length records produce a record number.
LENGTH	Length of the logical record as determined from the total number of central memory words contained in the binary cards.
PACKAGE	Routine name. A dash appears whenever a legitimate name does not appear in bits 59-18 of the first word of the logical record.
CKSUM	A 12-bit checksum of the logical record or routine.

A field length of at least 40000 (octal) should be used for any job using CATALOG.

The following program will catalog the system tape.

```
6
7
8
9
REWIND (LIBRARY)
CATALOG.
REWIND(LIBRARY)
REQUEST LIBRARY.
JOB,10,100,40000.
```

(operator assigns a specific tape unit)

7.3 COPYN

```
COPYN (p1, out, in1, in2, . . . , in10
```

Logical records from up to ten binary input files (in_1-in_{10}) may be extracted and written on an output file (out). Input and output files not defined by REQUEST cards, are assumed to be disk files. The record format is indicated by p_1; a non-zero value indicates the identification field (ID)† of the logical records is to be omitted from the output file, zero indicates the records are to be copied verbatim. If the records do not contain an ID, or if the ID is in Chippewa 1.1 format, the records are copied verbatim.

Text cards associated with the COPYN routine determine the order of the final tape. A routine may be selected from a composite tape, temporarily written on a scratch tape and transmitted as input to a translator, assembler, or programmer routine, eliminating the need for tape manipulation by the second program. Several tapes may be merged with COPYN to create a composite COSY or library tape. In its most basic form, COPYN can perform a tape copy.

The file names (in_1-in_{10}) reference binary files on tape, disk, or cards. A binary tape file is defined as the information contained between load point and a double end-of-file; the tape file may contain any number of single end-of-file marks. A disk file ends in one file mark, and a card deck must be terminated by a record separator (7,8,9 punch in column one). The output file name may reference a disk, tape, or card punch. A file mark for an output tape is written by a WEOF card or may be copied in a range of records and be counted as a record (7.3.5).

The records being copied may or may not have an ID prefix control number (12 bits), number of words in the prefix (12 bits), and the name associated with the logical record. A record ID in Chippewa 1.1 format consists of the first seven characters of the first word of each logical record. If logical records of the input file are not prefixed, all record identification cards must use the record number — the position of the logical record from the current position of the file.

† The ID referred to is produced automatically by FORTRAN 2.0 and ASCENT 2.0.

REWIND, SKIPF, SKIPR, WEOF (write End-Of-File), and record identification cards may be used in conjunction with COPYN: these text cards are read from INPUT and are terminated by a record separator (7, 8, 9 punch in column 1). The text cards are free field; they may contain blanks but must include the separators indicated in each card description.

7.3.1 REWIND

```
REWIND (p)
```

This card generates a rewind of file p which must be one of the input or output file names given on the COPYN control card. File p may not be the system INPUT file.

7.3.2 SKIPF

```
SKIPF (p, ±n)
```

With this card, n file marks on file p may be skipped. File p must be a tape; requests for other types of files will be ignored. The skip may be forward (+n) or backward (-n). There is no indication when SKIPF causes a tape to go beyond the double end-of-file or when the tape is at load point.

7.3.3 SKIPR

```
SKIPR (p, ±n)
```

With this card, n records may be skipped on file p. File p must be a tape; requests for other types of files will be ignored. The skip may be forward (+n) or backward (-n). Zero length records and file marks must be included in n.

7.3.4 WEOF

```
WEOF (p)
```

This card writes a file mark on file p, which must be one of the input or output file names on the COPYN control card.

7.3.5 RECORD IDENTIFICATION CARD

```
p1, p2, p3
```

The parameters on this card identify a record or set of records to be copied from a given file.

p_1 Record to be copied or the beginning record of a set of records to be copied. The name associated with the record or a number giving the position of the record from the current position of the file may be specified.

p_2 Last record to be copied in a set of records.

name	logical records p_1 through p_2 are copied.
decimal integer n	n logical records are copied, beginning with p_1. Zero length records and file marks are counted.
*	p_1 through an end-of-file mark are copied.
**	p_1 through a double end-of-file mark are copied.
/	p_1 through a zero length record are copied.
0 or blank	only p_1 is copied.

p_3 Input file to be searched. If p_1 is a name, and p_3 is omitted, all input files declared on the COPYN card are searched until the p_1 record is found. If it is not located, a diagnostic is issued. If p_1 is a number and p_3 is omitted, the last input file referenced is assumed. If this is the first text card, the first input file on the COPYN card is used.

Examples:

SIN, TAN, INPUTA	Copies all logical records from SIN through TAN from file INPUTA.
SIN, 10, INPUTA	Copies 10 logical records from file INPUTA, from SIN through SIN+9.
SIN, TAN	Searches all input files beginning with current file. (If this is the first text card, the first input file named on the COPYN card is used). When SIN is encountered, all logical records from SIN through TAN are copied.
SIN, , INPUTA	Copies logical record SIN from file INPUTA.
1, TAN, INPUTA	Copies the current logical record through TAN from file INPUTA.
1, 10, INPUTA	Copies ten logical records, beginning with the current logical record on file INPUTA.
1, *, INPUTA	Copies the current logical record through the first file mark encountered on file INPUTA.

7.3.6 FILE POSITIONING

The files manipulated during a COPYN operation are left in the position indicated by the previously executed text card, they are moved only during a search. If the file name (p_3) is omitted from the record identification card, all files mentioned on the COPYN card will be searched end-around. The end of a file is determined by a double end-of-file if tape, or a single end-of-file if disk. The first input file declared is searched until either p_1 or the original position of the file is reached, whereupon a search of the second input file begins. In this manner, all files remain effectively in the same position except the file containing p_1, which is positioned at p_2+1.

If the system INPUT is declared as one of the input files, it is not searched; the first record encountered must be p_1 or an error diagnostic is issued.

The output file is not repositioned after a search so that the COPYN routine may be re-entered, if desired. Therefore, the programmer is responsible for any REWIND, SKIP, or WEOF requests referencing the output file that may be necessary prior to exiting the job.

Example 1: Record identification card: REC,,INPUT1

Input file INPUT1:

ABLE	BAKER	...	REC	SIN	TAN	ZEE	EE OO FF

If INPUT1 were positioned at TAN, TAN and ZEE would be examined for REC. The double end-of-file would cause ABLE to be the next logical record examined, continuing until REC is read and copied to the output file. INPUT1 would then be positioned at SIN.

Example 2: Record identification card: RECA

Input file INPUT,1 positioned at B1:

A1	B1	...	Z1	EE OO FF

Input file INPUT2, positioned at loadpoint

A2	RECA	D2	EE OO FF

Input file INPUT3, positioned at loadpoint

A3	B3	C3	...	Z3	EE OO FF

All routines from B1 through A1 are compared to RECA and INPUT1 is repositioned at B1. A2 is compared, then RECA is copied to the output file and INPUT2 is positioned at D2. INPUT3 is not searched.

Example 3:

Record Identification cards and binary logical records on INPUT file.

REC,,INPUT

JOB1,JOB3,INPUT

ABLE,,IN2

Record Separator (7,8,9 punch in column 1)

REC (binary)

Record Separator

JOB1 (binary)

Record Separator

JOB2 (binary)

Record Separator

JOB3 (binary)

Record Separator

Since there is no end-around search of the INPUT file, REC and JOB1-JOB3 must directly follow the requesting record identification cards in the order specified by those cards. An incorrect request for an INPUT record terminates the job.

7.3.7 SAMPLE JOB

In the following example, the SIN routine is being updated and placed on the library tape. Tape files COSYLIB, OLDLIB, and LIBRARY must be defined with REQUEST cards. The COSY tape COSYLIB is searched for logical record SIN; this routine will be written verbatim on the disk file COSY. ASCENT reads COSY and the newly modified SIN is written on PUNCH. Next, COPYN copies the OLDLIB tape to LIBRARY, replacing SIN from PUNCH.

```
JOB, 5, 500, 40000.
REQUEST COSYLIB.
COPYN(, COSY, COSYLIB)
REWIND(COSY)
ASCENT (L, COSYOUT, PUNCH, COSYIN, COSY)
REQUEST OLDLIB.
REQUEST LIBRARY.
COPYN(X, LIBRARY, PUNCH, OLDLIB)
Record Separator card (7, 8, 9 punch in column 1)
SIN, , COSYLIB
Record Separator card
(COSY corrections)
COSY
Record Separator card
REWIND(PUNCH)
REWIND(LIBRARY)
REWIND(LIBRARY)
REWIND(OLDLIB)
1, COS, OLDLIB
SIN, , PUNCH
TAN, **, OLDLIB
WEOF(LIBRARY)
WEOF(LIBRARY)
REWIND(LIBRARY)
```

7.3.8 ERROR MESSAGES

The text cards are written on the system OUTPUT as they are read and processed. When an error occurs, the abort flag is set, and a message from the following list is printed on OUTPUT followed by the card in error. This card is not processed and an attempt is made to process the next text card. When the last text card is processed, the abort flag is checked; if it is set, the job is terminated. Otherwise, control is given to the next control card.

NO INPUT FILE ON THE COPYN CONTROL CARD
At least one input file must be specified on the COPYN card.

CONTROL CARD REWIND(INPUT) IS ILLEGAL
The system INPUT cannot be rewound.

P2 IS NOT IN INPUT FILE P3
Logical record p_2 must be in file p_3; if p_3 is omitted, p_2 must be in the same file as p_1.

ID NAME (P1) IS REQUIRED ON ALL TEXT CARDS
The first parameter on any of the text cards is required.

P1 IS GREATER THAN SEVEN CHARACTERS
Record identification card names must not exceed seven characters.

TEXT CARD CONTAINS AN ILLEGAL SEPARATOR
The only acceptable separators are + - , ( ) or blanks.

NO OUTPUT FILES ON THE COPYN CONTROL CARD
An output file must be specified in the COPYN control card.

ID NAME NOT IN INPUT FILE(S) BEING SEARCHED
Either p_1 or p_2 cannot be located by the COPYN routine.

BINARY RECORD MISSING FROM INPUT
Logical records requested from the system INPUT file must begin with the next logical record on the INPUT file.

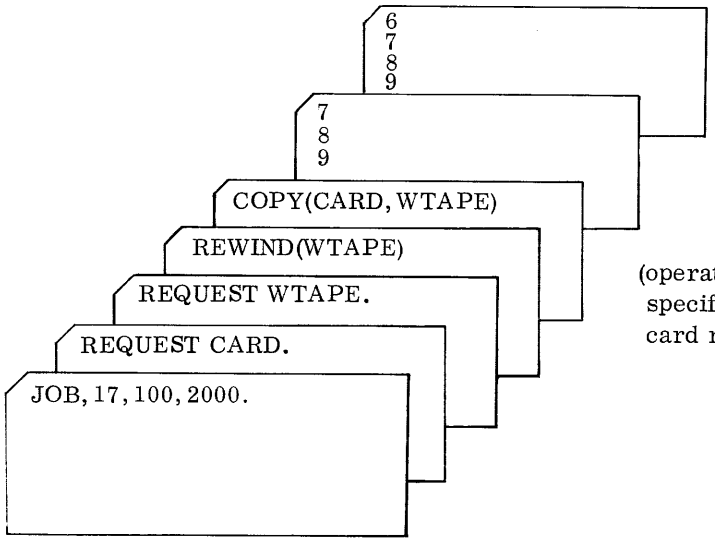
P2 NUMERIC EXTENDS BEYOND DOUBLE EOF
A numeric p_2 is specified on the record identification card which indicates a logical record located past the double end-of-file mark that terminates the file.

7.4 COPY TO DOUBLE FILE MARK

```
COPY(file1, file2)
```

The named file1 is copied onto file2 until a double file mark is detected on the first file. Both files are then backspaced over the last file mark. If parameters are omitted, INPUT, OUTPUT are assumed.

This routine is used in creating a new input job tape. For example, if input is from cards and output to tape a sample deck structure would be:



It may be necessary for the operator to drop a control point read operation to make a card reader available.

7.5 COPY BINARY FILE

```
COPYBF(file1,file2,n)
```

The number of binary files specified by n (decimal) are copied from file1 to file2. If parameters are omitted INPUT, OUTPUT, 1 are assumed.

When multiple files are copied onto a disk file, only one file is created.

7.6 COPY BINARY RECORD

```
COPYBR(file1,file2,n)
```

The number of binary records specified by n (decimal) are copied from file1 to file2. If parameters are omitted INPUT, OUTPUT, 1 are assumed.

This operation terminates on reading a file mark from file1 or when the required number of records has been read. A file mark is not written on file2.

7.7 COPY CODED (BCD) FILE

```
COPYCF(file1,file2,n)
```

The number of coded (BCD) files specified by n (decimal) are copied from file1 to file2. If parameters are omitted, INPUT, OUTPUT, 1 are assumed.

7.8 COPY CODED RECORD

```
COPYCR(file1,file2,n)
```

The number of coded records specified by n (decimal) are copied from file1 to file2. If parameters are omitted, INPUT, OUTPUT, 1 are assumed.

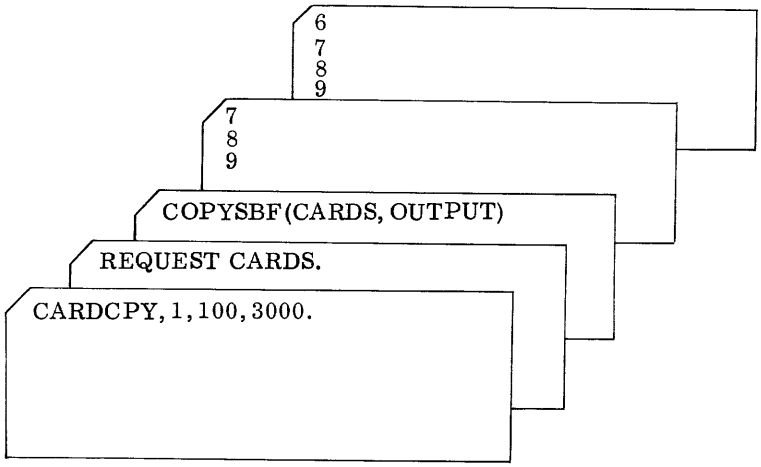
7.9 COPY SHIFTED BINARY FILE

```
COPYSBF(file1,file2)
```

A single binary file of coded information is copied from file1 to file2, shifting each line one character and adding a leading space. If parameters are omitted, INPUT,OUTPUT are assumed.

This routine is used in formatting a print file where the first character of every line is not a control character and is to be printed. The space character added will result in single line spacing when the file is printed.

Example: Control cards to print a Hollerith card file.



The Hollerith card file read into operator assigned card reader will be printed on OUTPUT file of job CARDCPY.

7.10 UNLOAD FILE

```
UNLOAD(file1)
```

File1 is rewound and unloaded. This does not release the file assignment to the control point. If parameters are omitted, the current file is assumed.

7.11 REWIND FILE

```
REWIND(file1)
```

This central program rewinds file1. If parameters are omitted, the current file is assumed.

7.12 VERIFY TWO FILES

```
VERIFY(file1,file2,p1,p2)
```

This routine compares the contents of two binary files. Comments regarding the success of the comparison are provided. File1 and file2 are the files to be compared; p_1 is the number of comment lines of type 5 to be printed in the event of an unsuccessful comparison. p_2 is the destination file for the comments (normally assigned to OUTPUT).

The field length required to verify two files is approximately 4000_8. Since the buffers are used in a circular fashion, no additional allowance is needed for large records.

The parameters may be truncated at any point from right to left; the routine assumes the following values:

file1 = FILE1

file2 = FILE2

p_1 = 1

p_2 = OUTPUT

The following diagnostics are issued with VERIFY:

1. VERIFY OK

   This comment indicates a successful comparison of the two files. This message is also placed in dayfile.

2. VERIFY FAILURE

   This comment which is also placed in dayfile indicates a failure to compare for one of the following reasons.

   The number of logical records does not agree

   The number of words in a record does not correspond

   A word in a record does not compare with the corresponding word

3. n(10) m(8) EXCESS RECORDS IN FILE (f)

   The named file, f, contained n more records than the other file.

4. n(10) m(8) EXCESS WORDS IN RECORD p(10) q(8) OF FILE (f)

   Record p of file f contains n words more than the corresponding record of the other file.

5. nnnnnn word 1 word 2

   This comment is printed for each pair of words not in agreement. The location n is octal. Each line is identified by a record number.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

6400/6600 Computer Systems
SCOPE Reference Manual

Pub. No. 60173800 September, 1966

THIS FORM IS NOT INTENDED TO BE USED AS AN ORDER BLANK. YOUR EVALUATION OF THIS MANUAL WILL BE WELCOMED BY CONTROL DATA CORPORATION. ANY ERRORS, SUGGESTED ADDITIONS OR DELETIONS, OR GENERAL COMMENTS MAY BE MADE BELOW. PLEASE INCLUDE PAGE NUMBER REFERENCE.

FROM NAME: _____

BUSINESS ADDRESS: _____

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE

STAPLE

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
Documentation Department
3145 PORTER DRIVE
PALO ALTO, CALIFORNIA

FOLD

FOLD

STAPLE

STAPLE

CONTROL DATA SALES OFFICES

ALAMOGORDO, NEW MEXICO
ALBUQUERQUE, NEW MEXICO
ATLANTA, GEORGIA
AUSTIN, TEXAS
BILLINGS, MONTANA
BIRMINGHAM, ALABAMA
BOSTON, MASSACHUSETTS
BOULDER, COLORADO
CAPE CANAVERAL, FLORIDA
CEDAR RAPIDS, IOWA
CHICAGO, ILLINOIS
CINCINNATI, OHIO
CLEVELAND, OHIO
COLORADO SPRINGS, COLORADO
DALLAS, TEXAS
DAYTON, OHIO
DENVER, COLORADO
DETROIT, MICHIGAN
DOWNEY, CALIFORNIA
GREENSBORO, NORTH CAROLINA
HARTFORD, CONNECTICUT
HONOLULU, HAWAII
HOUSTON, TEXAS
HUNTSVILLE, ALABAMA
IDAHO FALLS, IDAHO
INDIANAPOLIS, INDIANA
KANSAS CITY, KANSAS
LAS VEGAS, NEVADA
LIVERMORE, CALIFORNIA
LOS ANGELES, CALIFORNIA
MADISON, WISCONSIN
MIAMI, FLORIDA
MILWAUKEE, WISCONSIN
MINNEAPOLIS, MINNESOTA
MONTEREY, CALIFORNIA
NEWARK, NEW JERSEY
NEW ORLEANS, LOUISIANA
NEW YORK, NEW YORK
OAKLAND, CALIFORNIA
OMAHA, NEBRASKA
PALO ALTO, CALIFORNIA
PHILADELPHIA, PENNSYLVANIA
PHOENIX, ARIZONA
PITTSBURGH, PENNSYLVANIA
PORTLAND, OREGON
ROCHESTER, NEW YORK
SACRAMENTO, CALIFORNIA
ST. LOUIS, MISSOURI
SALT LAKE CITY, UTAH
SAN BERNARDINO, CALIFORNIA
SAN DIEGO, CALIFORNIA
SAN FRANCISCO, CALIFORNIA
SAN JUAN, PUERTO RICO
SANTA BARBARA, CALIFORNIA
SEATTLE, WASHINGTON
TULSA, OKLAHOMA
VIRGINIA BEACH, VIRGINIA
WASHINGTON, D. C.

ADELAIDE, AUSTRALIA
AMERSFOORT, THE NETHERLANDS
AMSTERDAM, THE NETHERLANDS
ATHENS, GREECE
BOMBAY, INDIA
CALGARY, ALBERTA, CANADA
CANBERRA, AUSTRALIA
DUSSELDORF, GERMANY
FRANKFURT, GERMANY
GENEVA, SWITZERLAND
HAMBURG, GERMANY
JOHANNESBURG, SOUTH AFRICA
KASTRUP, DENMARK
LONDON, ENGLAND
LUCERNE, SWITZERLAND
MELBOURNE, AUSTRALIA
MEXICO CITY, MEXICO
MONTREAL, QUEBEC, CANADA
MUNICH, GERMANY
OSLO, NORWAY
OTTAWA, ONTARIO, CANADA
PARIS, FRANCE
ROME, ITALY
STOCKHOLM, SWEDEN
STUTTGART, GERMANY
SYDNEY, AUSTRALIA
TEHERAN, IRAN
TEL AVIV, ISRAEL
TOKYO, JAPAN (C. ITOH ELECTRONIC COMPUTING SERVICE CO. LTD.)
TORONTO, ONTARIO, CANADA
VANCOUVER, BRITISH COLUMBIA, CANADA
ZURICH, SWITZERLAND

CONTROL DATA CORPORATION
8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

Pub. No. 60173800

Litho in U.S.A.