3600 3800

COMPUTER SYSTEMS DRUM SCOPE/MSIO

OPERATING GUIDE

CONTROL DATA
CORPORATION

3600 3800

COMPUTER SYSTEMS
DRUM SCOPE
REFERENCE MANUAL

CONTROL DATA
CORPORATION

60059200

REVISION RECORD

REVISION	NOTES
	This printing includes the most current revision level.
(11-64)	Original printing.
A	
(11-65)	Reprint with revision.
B	
(7-67)	Reprint with revision. Errata sheets and chapter 8 have been added and corrected.

Additional copies of this manual may be
obtained from the nearest Control Data
Corporation sales office.

Pub. No. **60059200**

Address comments concerning this manual to:

Control Data Corporation
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

or use Comment Sheet in the back of this manual.

CONTENTS

INTRODUCTION

Drum SCOPE, a comprehensive operating system for the Control Data® 3600 computer system, uses the drum storage system as the principal input/output device. Drum SCOPE provides economy of operation, higher throughput capacity, automatic priority job scheduling, and a full realization of the total computing power of the 3600 system.

The Drum SCOPE monitor system facilitates job processing and simplifies programming and operating by providing:

- Job Processing
  - Maintenance of job accounting information
  - Compilations, assemblies, and executions
    - ALGOL
    - COBOL
    - FORTRAN
    - COMPASS
    - PERT
    - SORT
  - Dynamic assignment of programmer units to available hardware units
  - Loading, linking, and execution of object subprograms
  - Program, data, and common storage allocation
  - Program overlay and segment processing
  - Loading of absolute production programs from drum
- Debugging Aids
  - Execution time diagnostics
  - Octal corrections
  - Selective dumps during program execution (SNAP and TRACE)
  - Memory maps of program elements
  - Post-mortem dumps for abnormal program termination
- Program Requests
  - Standard and non-standard I/O processing by background programs
  - Tape handling, including labeling and continuation reels
  - Drum handling
  - Random access processing
  - External and internal interrupt control
  - Sampling of time, date, equipment status, and available memory

- Library Preparation and Maintenance

  Preparation of a new library tape

  Editing of an external library tape

- Background Processors

  Card-to-drum

  Card-to-tape

  Card-to-printer

  Drum-to-printer

  Drum-to-punch

  User programs, such as, file updating, information retrieval, and processing of input from remote stations

EQUIPMENT CONFIGURATION

Drum SCOPE can operate with varying equipment configurations. Certain minimal capabilities must be provided as follows:

3603 Storage Module, designated bank 0

3604 Computation Module

3602 Communication Module

3606 Data Channel

Drum Storage System with two drums

Card Punch System

Card Reader System

Line Printer System

Magnetic Tape System with two tape units

Additional 3603 or 3609 storage modules and 3606 data channels may be added as desired. If MSIO is to be used, two 3603 storage modules are required. Each peripheral system mentioned above may be satisfied by various devices:

Drum Storage System	Standard	Non-Standard
Drum Storage Controller	3436 or 3637	
Two Drum Storage Units (up to eight may be accommodated)	861	
Direct Data Channel	3816	3806/3602 or 3606/3602
Card Reader System		
Card Reader Controller	3447 or 3649	
Card Reader	405	
Card Punch System		
Card Punch Controller	3446 or 3644	
Card Punch	415 or IBM 544 or 523	
Line Printer System		
Printer Controller	3256 or 3659	3655
Line Printer	501	1612
Magnetic Tape System		
Tape Controller	3228 or 3229 3421, 3422, or 3423	
	3622, 3623, 3624, 3625, or 3626	
Two Tape Units	604, 607, or 601	603 or 606

Addenda and Errata February 1968

3600/3800 Computer System DRUM Scope Reference Manual

CONTROL DATA
Pub. No. 60059200B

These addenda and errata are composed of two sections — general corrections within various parts of the text, and a revised Chapter 8 which completely replaces the old Chapter 8 in the manual.

GENERAL CORRECTIONS

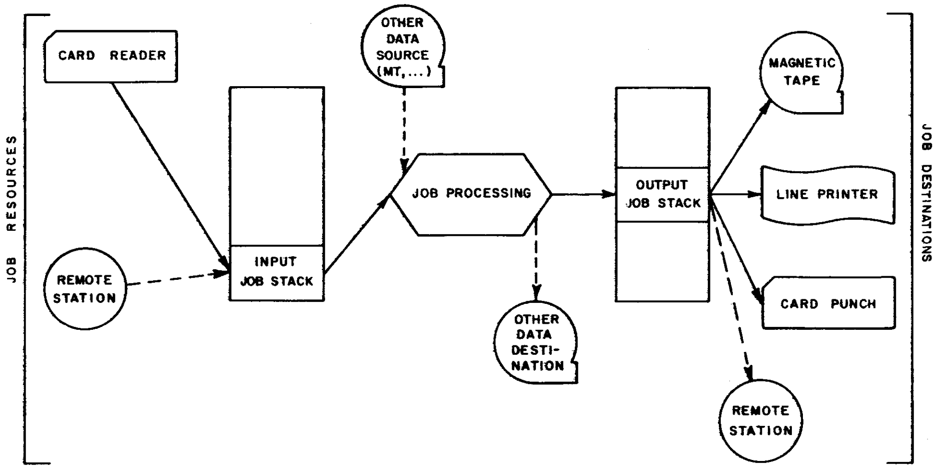
Page	Location	Error	Should Read
2-12	Top line	The LOCATE request (Sec. 3.1.2)...	The LOCATE request (Sec. 4.1.2)...
2-13	Sec. 2.7	...loader is in Chapter 5. The LOADER request (Sec. 3.3)...	...loader is in Chapter 7. The LOADER request (Sec. 4.3)...
2-17	Sec. 2.7.2, Line	...program (Overlay Processing, Chapter 6).	...program (Overlay Processing, Chapter 8).
2-17	Immediately above heading 2.8 LIBRARY STATEMENT	Statement to be added	d Phase termination dump key for error-free run. If non-blank, an octal dump of all memory assigned to the job will be written on standard output at the end of an error-free run.
2-17	Card illustration	LOADMAIN,u,t,p,r	LOADMAIN,u,t,p,r,d
4-3	Line 17	RELEASE u,ra,s	RELEASE u,ra,p
4-3	Lines 21-22	s Any character, meaningless unless u is magnetic tape. If s is non-zero, the unload is suppressed.	p Meaningless unless u is magnetic tape. If p is a non-zero digit, the unload is suppressed.
4-9	Line 9	...otherwise, the assignment is kept. (Sec. 3.1.4)	...otherwise, the assignment is kept. (Sec. 2.5.7)
4-12	Chart, Table 4-1, last column, second block from tape	next card binary	(This block should be blank.)
4-23	Footnote	† See OVERLAY PROCESSING, Chapter 6.	† See OVERLAY PROCESSING, Chapter 8.

Page	Location	Error	Should Read
4-24	Line 15	...placed in the A and Q registers (section 5.2).	...placed in the A and Q registers (Sec. 7.13).
6-5	Sec. 6.3, Top line	On the RUN control statement (Sec. 2.3)...	On the RUN control statement (Sec. 2.7.2)...
7-1	Next to last paragraph, second sentence	...through the BANK control card (Sec. 7.12)...	...through the BANK control card Sec. 7.12.1)...
7-8	Sec. 7.7, last paragraph	...(See IDC card, sec. 7.1.1).	...(See IDC card, sec. 7.3)
B-5	Line 7	RELEASE u, ra, s	RELEASE u, ra, p

1.1 JOB PROCESSING

Data to be processed by Drum SCOPE is submitted in units called jobs. The extent, purpose, equipment requirements, and priority of the job are controlled by the programmer with Drum SCOPE control statements (Chapter 2).

A job submitted to the system on any applicable input device is placed along with other user-specified jobs in a job stack on the drum. Drum SCOPE then selects from the stack the job with the highest priority and processes it in memory. The job output is placed on the drum. Similarly, on output a stack is formed; jobs are selected from the drum for output according to their priority.



The solid lines in the diagram above represent the typical data flow of a job; however, variations are easily permissible, these are represented by dotted lines.

The Drum SCOPE system includes routines for such tasks as card to drum, card to tape, card to printer, drum to printer, and drum to punch. The installation may also code its own. These routines, called background programs, may be performed concurrently with the user's job processing. The user's job being processed in memory is called the central program.

Background programs, the central program, and the Drum SCOPE system share the compute module on a priority basis achieved through the computer interrupt system. Background programs, which perform extensive input/output operations, are coded so that control returns to Drum SCOPE while time-consuming I/O operations are being completed. Drum SCOPE then gives control to the waiting program with the next highest priority; it may be another background program or the central program. This procedure continues as control passes down the program priority hierarchy.

When a program with priority higher than the one currently being executed completes an I/O operation and is ready to continue executing, Drum SCOPE returns control to that program. This method of dynamically controlled execution, in conjunction with the high character transfer rate of the 861 drum, utilizes the full computing power of the 3600 system.

	63	Reserved by Drum SCOPE
	62	Reserved by Drum SCOPE
	61	Background processor
	60	Background processor
priority	⋮	⋮
	3	Reserved by Drum SCOPE
	2	Reserved by Drum SCOPE
	1	Central program interrupt subroutine
	0	Central program

1.2 JOB STRUCTURE

Jobs submitted by a programmer typically consist of one or more assemblies, compilations, and executions; the programmer defines the extent of a job with Drum SCOPE control cards. A job is specifically defined as all operations which follow a JOB card. End-of-file cards separate jobs.

A job may be divided into runs. Within each run, phase control cards specify the operations to be performed. A run consists of one or more library phases, and/or execution phases. Runs are separated by end-of-file cards.

During a library phase, a source program, coded in a source language, is prepared for execution. The Drum SCOPE library includes ALGOL, COBOL, FORTRAN, COMPASS, PERT, and SORT. The installation may modify or add to the Drum SCOPE library with LIBEDIT (Chapter 9). An object program is produced from a library phase, and the program may then enter the execution phase.

The first request for an execution phase encountered after an error is detected in any library phase causes the remainder of the run to be deleted.

1.3 PROGRAMMER REQUESTS

Programmer requests, which may be included in assembly language programs, specify operations for input/output control, internal interrupt, and system requests. They are written as system macros and assembled into a calling sequence to Drum SCOPE.

1.4 LOGICAL UNITS

All I/O references to equipment within a program are in terms of logical units. All units are on the drum, unless specifically assigned to some other equipment. Logical units designated by the programmer are divided into four classes: programmer, scratch, system, and background.

1.4.1 PROGRAMMER UNITS

These units are assigned throughout the job for reference by the program. Assignment is released at the end of the job, except for unit 69, load-and-go, which is normally released when an execution phase is initiated.

Standard Input — INP (logical unit 60)

Control cards for all Drum SCOPE jobs are read from this unit. Frequently, programs and data to be processed are also on INP.

Standard Output — OUT (logical unit 61)

Drum SCOPE control statements, diagnostics, dumps, and loader control cards are written in BCD mode on this unit. Program output may also be written on OUT.

Standard Punch — PUN (logical unit 62)

Program and Drum SCOPE output for punching is recorded on this unit. All information on PUN is punched.

Load-and-Go — LGO (logical unit 69)

Binary object programs transferred from the standard input unit or produced by compilation or assembly may be stored here prior to loading and executing. The programmer may save this unit by assigning it to other equipment with an EQUIP statement. If saved, it will be released at the end of the job; otherwise, it will be released at the beginning of the run.

Auxiliary Libraries — (logical units 71-79)

Auxiliary libraries are used for library preparation and editing and for augmenting the system library.

Other Programmer Units — (logical units 1-49)

These units may be assigned throughout the job for reference by the program.

1.4.2 SCRATCH UNITS

Scratch units may be referenced at any time by the programmer, but they are released after each execution phase.

1.4.3 SYSTEM UNITS

The system units, assigned by Drum SCOPE, are used by the programmer and Drum SCOPE; they are never released.

System Input Comment — ICM (logical unit 63)

Comments from the operator to the monitor system are made on this unit, usually the console typewriter. The programmer may also use ICM for certain input directions.

System Output Comment — OCM (logical unit 64)

Statements from the monitor system to the operator are made on this unit, usually the console typewriter. The programmer may also list information on this unit.

System Library — LIB (logical unit 70)

The library contains the monitor system and all programs and subroutines which operate under Drum SCOPE, such as FORTRAN, COBOL, COMPASS, SORT, and ALGOL.

Other System Units — (logical units 65-68)

These numbers are reserved for use by the system.

1.4.4 BACKGROUND UNITS

These units, assigned by request to background programs, use the designation 1 to n; n is an integer specified by a parameter of the IDENT card of the background program (Appendix A).

1.4.5 SYSTEM ACCOUNTING UNIT

Accounting information is written on the drum with a disposition code of AC. The SYSIO request is used to write the accounting information.

CONTROL STATEMENTS 2

2.1 FORMAT

Drum SCOPE control statements have the following format: a 7,9 punch in column 1 followed by a statement name beginning in or to the right of column 2. Parameters are separated from the statement name and each other by commas. If a parameter is omitted, the commas surrounding it must be included; however, if the last parameter or group of parameters is omitted, no commas are required.

Control cards containing a fixed number of parameters† may contain comments after the last parameter; in this case, all commas must be punched, with an additional comma separating the parameters from the comments.

Control statements are free-field, but must be contained on a single 80-column card. No terminating character is needed.

2.2 PRIORITY STATEMENT

This statement defines job priority.

$^{7}_{9}$PRIORITY,p

p   priority number, 7-0 (7, most urgent; 0, least urgent)

A priority applies only to the ensuing data. Jobs having the same priority number are executed in the order of input. As jobs are recognized by the card reader processor, they are assigned a sequence number. When jobs have the same priority, they are executed sequentially beginning with the lowest sequence number. If no PRIORITY card appears, 0 is assumed. The PRIORITY card must be followed immediately by a JOB or DISPOSE control card.

2.3 JOB STATEMENT

All programs to be processed under Drum SCOPE begin with a JOB card which indicates the beginning of a job, provides accounting information, identifies the programmer, and sets a time limit.

†JOB, LIBRARY, CORRECT, LOAD, RUN, and LOADMAIN

Option 1

$^{7}_{9}$JOB,c,i,t

c charge number; alphanumeric characters of any length. Drum SCOPE truncates to the first 16 non-blank characters for accounting identification.

i programmer identification; may be any length and appears as given in the control card listing. It is truncated to the first 16 non-blank characters for operator identification and accounting.

t maximum time limit (m. s) in minutes and seconds for the entire job. If t = 0 or blank, a standard time determined by the installation is used.

Option 2

$^{7}_{9}$JOB(n),c,i,t

n is a decimal number which indicates how many extra logical units are required by the job in addition to the number allotted by the installation. Option 1 is equivalent to JOB (0). If the total is greater than 75, 75 units will be allotted.

Each extra logical unit requested by the programmer will require five locations in bank 0. Drum SCOPE will employ this number (5 x n) when allocating memory.

2.4 DEMAND STATEMENT

The DEMAND card permits the user to reserve sufficient memory and hardware requirements for successful job execution. No cards other than COMMENT and PAUSE may appear between the JOB and DEMAND cards.

$^{7}_{9}$DEMAND,$r_1,r_2,\ldots,r_k$

r_i memory or hardware requirement. A decimal number (or octal ending with B) declares a memory requirement in bank 0. The hardware mnemonics, listed below, may be preceded by a decimal number to indicate the need for more than one device.

CR	card reader
CP	card punch
LP	line printer
DR	drum
MT	magnetic tape
CT	console typewriter

Example:

DEMAND, MT, 2LP, 1MT, 62000B, CP

declares a requirement of two magnetic tapes, two line printers, one card punch, and 62000_8 locations in bank 0.

If the hardware is currently assigned, and demands cannot be met immediately, the operator is informed and is permitted to make hardware available, postpone, or cancel the job. If the hardware is not attached, demands cannot be met, and the job is abandoned without operator intervention.

If the system is unable to provide the bank 0 memory requirements, the job is abandoned. The system will attempt to interrupt non-resident background processing to provide the memory requirements, if necessary.

2.5 EQUIP STATEMENT

EQUIP statements provide information concerning the hardware type, usage, random access properties, logical unit equivalence, disposition, and label declarations of a logical unit. The EQUIP statement precedes the RUN card or the entry point name statement of the program in which the logical unit is referenced.

$^7_9$EQUIP,u=d_1,d_2,...,d_n

u logical unit number, 1-49, 61, 62, 69, 71-79

d_i declarations

Declarations for a single logical unit may be combined in one EQUIP statement.

Unrecognized declarations are ignored. When contradictory declarations are given, the last mentioned (rightmost) takes precedence. If the unit specified in the EQUIP statement is currently assigned, the original assignment will be released. All declarations refer to the master logical unit (Equivalence Declarations).

2.5.1 HARDWARE DECLARATIONS

Hardware type may be specified by d_i; the mnemonics depend upon the equipment used by the installation.

CR	card reader
CP	card punch
LP	line printer
DR	drum
CT	console typewriter
MT	magnetic tape

If no hardware mnemonic is specified, DR is assumed unless a density declaration (implying magnetic tape) is given.

Example:

$^{7}_{9}$JOB,3215079,JAYNE,15	
$^{7}_{9}$EQUIP,10=MT	an available magnetic tape will be assigned to 10
$^{7}_{9}$EQUIP,9=CP	an available card punch will be assigned to 9
object program	
$^{7}_{9}$RUN,7,300,7,1	
<end-of-file>	
$^{7}_{9}$EQUIP,11=MT	a second available magnetic tape will be assigned to 11
object program	
$^{7}_{9}$RUN,6,250	
<end-of-file>	

2.5.2 USAGE DECLARATIONS

Usage on logical unit u may be specified by d_i:

RW read/write; no restrictions exist other than those imposed by the hardware.

BY bypass; all references to this unit except MODE (usage), STATUS, and RELEASE are treated as no operation.

RO read only; no write operations (WRITE, MARKEF, ERASE, WRLABEL, WEOT) are permitted. Read-only units such as a card reader need not be declared as such.

WO write only; no read operations (READ, REOT, RDLABEL, SKIP) are permitted. Write-only units such as a card punch need not be declared as such.

If no usage declaration is given on an EQUIP card, RW is assumed unless usage is restricted by the hardware type (CR, LP, ...).

2.5.3 DENSITY DECLARATIONS

The operating density for logical unit u may be designated by d_i:

OP operator option; for input, defines the density to be the same as the density of the label or, if unlabeled, the density of the physical unit. For output, a standard density determined by the installation is selected.

LO low density (200 bpi)
HI high density (556 bpi)
HY hyper density (800 bpi)

A density declaration automatically implies a MT hardware declaration. If no density is given for a unit which has been declared MT, a standard density, determined by the installation, is assumed.

2.5.4 DISPOSITION DECLARATIONS

Disposition of logical unit u at the end of the job may be defined by d_i. If no disposition declaration is supplied, the unit is released at the end of the job. Additional disposition declarations may be defined by the installation. These declarations are meaningful only on drum units.

PR print PU punch

2.5.5 EQUIVALENCE DECLARATIONS

d_i may be another logical unit number in the range 1-64, 69, 70-79. In this manner, two logical units may be assigned to the same physical unit. Any number of units may be equated, but separate EQUIP statements must be used for each pair; only one d_i may be equated to one u per equip card. d_2, d_3, ... if given, are not recognized.

The equivalence is one-way and transitive; for example:

${}^{7}_{9}$EQUIP,30=61	unit 30 is equated to standard output unit (61)
${}^{7}_{9}$EQUIP,31=32	
${}^{7}_{9}$EQUIP,32=33	unit 31 is equated to units 32, 33, and 34
${}^{7}_{9}$EQUIP,33=34	

The last three EQUIP statements, above, are an example of an equivalence chain. The last unit in a chain, the unit which is equivalent to nothing, is the master logical unit (34 above).

2.5.6 LABEL DECLARATIONS

Label declarations may be used to define information contained in the label of a magnetic tape, or to declare an unlabeled tape. A label declaration implies magnetic tape, unless overridden by an explicit hardware declaration.

LABELED TAPE DECLARATIONS

d_i format: (name, edition, reel, creation date or retention code).

Parentheses are required.

name — 14 alphanumeric characters, including blanks; the first character must not be an asterisk. If less than 14 characters are specified, remaining positions on the right are filled with blanks. More than 14 characters are truncated from the right. The name is checked on input, written on output.

edition — edition number, 01 to 99. Optional, but should be used if two or more labels have the same name. Checked on in input; written on output. If 00 is specified, a blank is written.

reel — reel number, 01 to 99. Optional, but should be used if two or more reels have the same name. If no reel number is specified for input, the lowest numbered reel is taken; for output, the reels are numbered beginning with 01.

creation date — date written; mmddyy (mm = month, dd = day, yy = year). The date specified is checked against the date written for an input tape. It is ignored on an output tape.

retention code — 3 or 6 digits, 000 to 999. Indicates number of days an output tape is to be saved from the date written. 999 results in permanent retention. A retention code is ignored on an input tape.

Examples:

$^{7}_{9}$EQUIP, 25=(INVENTORY, 1, 1, 012965), LO, RO

Logical unit 25 is a tape labeled INVENTORY edition 1, reel 1, written on January 29, 1965. It is a low density, read-only unit.

$^{7}_{9}$EQUIP, 10=(LEXICON, , 1, 030), HI

Label a high density tape (logical unit 10) with LEXICON, reel 1, retention code of 30 days.

LOGICAL UNIT NUMBER DECLARATIONS

d_i format:

(*nn, edition, reel, retention)

The asterisk is required; nn is the 2-digit logical unit number (01-49) contained (input) or to be written (output) in character positions 4-5 of the tape label. If no tape with nn in its label is found when the first request for unit u is a read request, Drum SCOPE will ask the operator to supply the tape or to terminate the job.

Edition, reel, and retention are the same as for labeled tape declarations. Only u may be used in a request for that unit. If the programmer wishes to use a request with nn, he must first give EQUIP, nn=u.

Example:

$^{7}_{9}$EQUIP, 28=(*35, 1, 1)

When logical unit 28 is referenced in a read operation, the system will search for a label with logical unit 35.

For a write operation, this statement may be used to label a tape referenced as unit 28 with logical unit number 35.

NON-STANDARD LABEL DECLARATION

Using a d_i of ** declares non-standard labeling.

A write request is handled in the same manner as for a labeled device; that is, the system will assign any unit which:

1. is magnetic tape
2. is not currently assigned
3. has a status indication of ready
4. contains a label in which the retention has expired

A read assignment consults the operator with a message requesting assignment of a particular device. The assignment will be made if the first three conditions stated above are met.

The following examples demonstrate some typical forms of the EQUIP statement:

Example 1:

${}^{7}_{9}$JOB,20,BETA,15

${}^{7}_{9}$EQUIP,10=CR

${}^{7}_{9}$EQUIP,11=CP

${}^{7}_{9}$EQUIP,20=(FILE A,1),RO, LO

${}^{7}_{9}$EQUIP,21=(CHANGE FILE),RO, LO

${}^{7}_{9}$EQUIP,22=(FILE A,2), LO

${}^{7}_{9}$LOAD,49

${}^{7}_{9}$RUN,13,3000

FILE A, edition 1, is the name of unit 20. CHANGE FILE is the name of unit 21. Unit 22 has the same name as unit 20, but it is the second edition.

Example 2:

```
 7
 9JOB,30,QWERTY,30

 7
 9EQUIP,5=(COSY TAPE,1,1),HI,RO       Logical units 5 and 20 are high
                                       density (556 bpi).  COSY input
 7
 9EQUIP,20=(COSY TAPE,21,1),HI,WO     to the COMPASS assembler is
                                       on logical unit 5.  COSY output
 7                                     on logical unit 20.
 9COMPASS,Y=5,C=20,L,X

      IDENT
        .
        .
        .
      SCOPE

 7
 9EQUIP,25=(*40),MT                    Logical unit 25 with a label
                                       bearing logical unit number 40;
                                       input/output requests within the
                                       program which reference 25
                                       will reference this tape.

 7
 9EQUIP,26=**,HY                       Hyper density tape referenced
                                       as logical unit 26 with a non-
                                       standard label.  The physical
                                       unit must be specified by the
                                       operator if a read request is
                                       given for this unit.

 7
 9LOAD

 7
 9RUN,28,1000
```

2.5.7 DATA ORGANIZATION

Data organization declarations specify a logical unit to be random access rather than sequential access. This declaration may be made only on units for which it is logically permissible; at present, the drum. No other declarations are recognized on an EQUIP card when RA is used.

d_i format: RA(m_1, m_2)

RA declares random access. When used alone, the largest number of contiguous words remaining in the random access area on the drum is assigned to the unit--equivalent to RA(0, 0).

m_1 minimum word requirement. If m_1 contiguous words are not available in the random access area of the drum, the job will be abandoned. May be used alone: RA(m_1).

m_2 If at least m_1 contiguous words remain in the random access area of the drum, up to m_2 contiguous words will be assigned to the unit. If m_1 words are not available, the job is abandoned. If m_2 is less than m_1, it is assumed equal to m_1.

Associated with each random access unit, the system maintains two values: the actual base address of the random access area on the drum, and the current relative address. The current relative address dictates the starting address for the next read or write operation. The following requests are permitted on random access units. The number of words in a data transmission request is indicated by the symbol wdct.

Request	Relative Address Value (after request has been honored)
LOCATE	set to specified value
REWIND, UNLOAD	0
READ, REOT	wdct + old value
WRITE, WEOT	wdct + old value
RELEASE	releases unit

The following requests are ignored on random access units:

DISPOSE	BSPF
ENTER	MARKEF
BSPR	ERASE
MODE	SKIP

The LOCATE request (Sec. 3.1.2) may be used to determine the size of the area. Words are assigned in an integral number of contiguous drum blocks; therefore, the number of words allotted may exceed that requested.

Example:

${}^{7}_{9}$EQUIP,5=RA(600,700)

700 words are desired, but 600 will be acceptable.

2.5.8 DEFERRED ASSIGNMENT

When this declaration is used, the system will query the operator for assignment of an output unit rather than assign the unit automatically. This allows assignment to particular physical units, such as a special reel of tape. This declaration is ignored in the case of an input unit.

d_i format: DA; indicates deferred assignment

Example:

${}^{7}_{9}$EQUIP,13=DA,MT

The physical unit assignment will be made by the operator at Drum SCOPE's request.

2.6 FAMILY STATEMENTS

The FAMILY statement groups logical units into a family for the purpose of sharing buffer areas.

Option 1

${}^{7}_{9}$FAMILY,f=$u_1,u_2,\ldots,u_k$

f family number, 0 to F; F is a limit, less than 64, determined by the installation.

u_i logical unit 1-62, 69, 71-79

The specified units are attached to family f. Any u_i currently attached to another family will be re-attached to the specified family.

Each family that contains one or more buffered units will be assigned two buffer areas.

Option 2

$^{7}_{9}$FAMILY(n),f=$u_1,u_2,\ldots,u_k$

n 0-2 decimal. Indicates the number of buffer areas to be allocated.

Family 0 is the system family and should be avoided to maintain efficient I/O processing.

Unless other specifications are made, Drum SCOPE automatically assigns units 60, 62, and 69 to family 1, and unit 61 to family 2. Two buffer areas are assigned for each.

2.7 PHASE CONTROL STATEMENTS

Phase control statements control program loading and execution and provide a means to load and correct routines on the system library. A description of the loader is in Chapter 5. The LOADER request (Sec. 3.3) allows additional programmer control over loading operations.

2.7.1 LOADING STATEMENTS

Library Program

Library Programs[†] are called by control statements which name the entry point to the library program. When the program is loaded, it assumes control, performing assembly or compilation. Upon return of control to Drum

[†] Library programs should not be confused with Drum SCOPE library subroutines which are called by symbolic reference in a subprogram.

SCOPE, the A register indicates whether errors were (non-zero) or were not (zero) detected. If the errors were detected, the loading and execution of the binary object program is inhibited on a load-and-go run.

ENTRY POINT

The program will be loaded from the production file if it is there. If not, it will be loaded from the auxiliary library if one exists and the program is on it. Otherwise, it will be loaded from the regular library.

${}^{7}_{9}$entry point name,p_1,p_2,...p_m

entry point name may be one of the following: FTN, COMPASS, COBOL, ALGO, ALDAP, Installation Program

$p_1, \ldots, p_m$ assembly options differ with each program; see pertinent reference manuals.

CORRECT

The system is directed to load a routine from the auxiliary library if it exists and the program is on it. Otherwise it will be loaded from the regular library.

${}^{7}_{9}$CORRECT, epname

epname an entry point on the library

Only the routine specified by epname will be loaded. The routine may be corrected by Octal Correction Cards (Sec. 7.10) directly following the CORRECT card. The corrections terminate with the first non-loader card.

LOAD

Relocatable binary subprograms are loaded into memory from programmer units or the load-and-go unit.

${}^{7}_{9}$LOAD,u

u logical unit number 1-49, or 69. When u is 0 or omitted, standard load-and-go (69) is implied.

The first time unit u is referenced in a loading operation, it is rewound prior to loading; otherwise, loading starts at the current position. Loading terminates on unit u at an end-of-file mark or a non-loader control card.

Binary Program

A binary object program on the standard input unit (60) is loaded into memory until a non-loader control card or an end-of-file is encountered.

Example:

$^{7}_{9}$JOB,1234-A,JRH,7

binary object program
.
.
.

2.7.2 EXECUTION STATEMENTS

RUN

RUN initiates program execution by transferring control to the object program in memory.

$^{7}_{9}$RUN,t,p,r,m,d

t — time limit (m. s) in minutes and seconds; the period may be omitted, if no seconds are specified. If blank or zero, an amount determined by the installation is assumed. If the time requested in the RUN statement exceeds the remaining time requested in the JOB statement, execution continues only until job time is depleted. The entire job is terminated if either limit is exceeded.

p — maximum number of print or write operations which may be requested on the standard output unit (61) during execution. This includes debugging dumps and any other execution output. If blank or zero, a standard limit determined by the installation is used. This field may also be in the form

(dd=ii, dd=ii, ... dd=ii)

where dd is a disposition code and ii is a decimal limit. For instance; (PR = 1) 1 is the limit for standard print output. The entire job is terminated if the limit is exceeded.

r dump key; specifies information to be dumped in case of abnormal termination or forced dump. All dumps provide a console scoop at the point of program termination.

r	dumped area, written on standard output
0 or blank	none
1	programs including library routines
*1	programs not including library routines
2	labeled common
3	programs and labeled common, including library routines
*3	programs and labeled common, not including library routines
4	numbered common
5	programs and numbered common including library routines
*5	programs and numbered common not including library routines
6	numbered and labeled common
7	all memory assigned to the job
*7	all memory assigned to the job except library routines
8	all assigned memory including library routines

m map suppress key; if non-blank, maps are suppressed. If blank or zero, or if snap or trace facilities have been employed, a memory map is printed on the standard output unit showing absolute locations of programs, program extension areas, common blocks, and entry points.

d phase termination dump key. If non-blank, the dump specified by the r parameter will be written on standard output at the end of an error-free run.

LOADMAIN

This statement loads and initiates execution of a previously partitioned program (Overlay Processing, Chapter 6).

$^{7}_{9}$`LOADMAIN,u,t,p,r`

u logical unit number, 1-49.

t time limit (m. s) for the run in minutes and seconds; the period may be omitted if no seconds are specified. If blank or zero, a standard time limit determined by the installation is used. If the time requested in LOADMAIN exceeds the remaining time requested in the JOB statement, execution continues only until the job time is depleted. The entire job is terminated if the limit is exceeded.

p maximum number of print or write operations which may be requested on the standard output unit (61) during execution. This includes debugging dumps and any other execution output. If blank or zero, a standard limit determined by the installation is used. The entire job is terminated if the limit is exceeded.

r dump key; if non-blank or non-zero, an octal dump of all memory assigned to the job will be provided in the event of an abnormal termination.

The specified unit will be rewound and the main partition loaded and given control. The main partition must be the first partition encountered. LOADMAIN may not be preceded by any previous loading operations performed by other phase control statements, or debugging aids planted by a snap or trace.

2.8 LIBRARY STATEMENT

This statement directs Drum SCOPE to augment the current system library.

$^{7}_{9}$`LIBRARY,u,Lname`

u logical unit, 1-49, 70-79, containing the file which is to augment the system library.

Lname name of the file which is to augment the system library. This name must begin with an L.

If any of the routines in the new file have the same names as routines on the current system library, the new routines effectively replace the current routines. If Lname is not found, the job is abandoned.

A LIBRARY card remains in effect until the end of the job or until another LIBRARY card is used. A LIBRARY,70 statement nullifies the effect of the preceding LIBRARY statements; that is, the original library is reinstated.

Examples:

$^7_9$LIBRARY,30, LIBFILE

$^7_9$LIBRARY,70

2.9 COMMENT AND PAUSE

The COMMENT statement allows the user to write comments on the standard output comment unit (console typewriter) during a program phase.

$^7_9$COMMENT, comments

An alternate COMMENT statement allows the comments following the comma to be written on LUN as well as on the typewriter.

$^7_9$COMMENT (LUN), comments

The PAUSE statement stops the current job; but the system continues processing background tasks. To continue the job, the operator must signal the system with an appropriate comment.

$^7_9$PAUSE

2.10 MASSFILE

This card is required for all jobs containing MSIO calls which refer to mass storage devices. The primary function of the MASSFILE card is to allocate memory for the file definition tables.

$${}^{7}_{9}\text{MASSFILE},k_1=d_1,k_2=d_2,\ldots,k_n=d_n$$

The k and d parameters may assume the values described below:

FILES=dd — This declaration is required; dd is a one- or two-digit decimal number defining the total number of files to be processed by the job. This parameter should be specified first and appear only once in any set of MASSFILE cards for a job.

The remaining declarations are optional and apply to one particular file. Each MASSFILE card may contain information about only one file; continuation cards are formed by repeating the FN declaration on every card of the set. Declarations on the MASSFILE card override those declared in the FET in the running program.

FN=dd — dd is a one- or two-digit decimal number defining an internal file number. Within a COBOL program all files are numbered consecutively from 1 through n as they appear in SELECT specifications within the ENVIRONMENT division.

ID=name — name is an external file name or label identifier; it may contain from 1 to 14 characters, including blanks. A comma before the fourteenth character causes the field to be completed with blanks.

EDT=dd — dd is a one- or two-digit decimal number defining a file edition number.

DATE=mmddyy — date written; mm=month, dd=day, yy=year. mmddyy must be six characters; for example, January 6, 1966 would be specified 010666.

DSP=SAVE — This declaration causes a file to have a retention code of zero, but to be retained until the end of the job before being released. If a file is not assigned a retention code and SAVE is not specified, the file is released at the end of the run.

DSP=ddd dd is a one- to three-digit decimal number defining the retention cycle for a file. 999 specifies permanent retention. Otherwise ddd is the number of days the file is to be saved from the date-written. This declaration is given when creating the file.

BLOC=±num num is a one- to eight-digit decimal number defining the number of blocks to be allocated. Plus indicates an addition to an existing allocation for the file, minus indicates reduction. Neither plus nor minus indicates an initial allocation.

SEG=dd dd is a one- or two-digit decimal number specifying how many segments allowed for the allocation. The maximum number of segments is 63; omission of the parameter, SEG=00, or any number larger than 63 causes the maximum to be used.

2.11 SOURCE DECK STRUCTURE

The entry point name statement is followed by the source program to be assembled or compiled by the library program.

A FTN card may be followed by either FORTRAN or COMPASS source language subprograms to be compiled or assembled.

A COMPASS card is followed by COMPASS subprograms to be assembled.

A COBOL card is followed by a COBOL source program to be compiled.

An ALDAP card may be followed by an ALGOL source procedure to be compiled or compiled and executed or a COMPASS subprogram to be assembled. An ALGO card is followed by an ALGOL source procedure to be compiled and executed or a COMPASS subprogram to be assembled.

If more than one library program (compilers or assemblers) is used for processing the subprograms for a single program, the source decks are stacked consecutively.

For each source language subprogram, a binary object program will be produced and stored on the logical unit designated by one of the parameters on the entry point name card. After each binary program is written on a logical unit, the library program writes an end-of-file mark and backspaces over it. Therefore, after all binary subprograms are written on the unit, only one end-of-file mark will remain; and the unit will be positioned so that any subsequent compilations for that unit will write over it. Consequently, at the end of the last compilation, an end-of-file mark remains on the unit.

A SCOPE card is required to indicate the end of the source subprograms following a FORTRAN or COMPASS card; also to indicate the end of the source procedures following an ALDAP card. An EOP card is required to indicate the end of the program following an ALGO card. COBOL does not require a SCOPE card. The word SCOPE begins in column 10; since it is not a Drum SCOPE control card, there is no 7,9 punch in column 1.

```
1        10
7
9COMPASS,C,L,R
         IDENT ASIMOV
           .
           .                       COMPASS subprogram
           .
         END
         IDENT BRENDA
           .
           .                       COMPASS subprogram
           .
         END
         SCOPE
```

```
1     7  10
7
9FTN,A,L,P
      PROGRAM ONE
         .
         .                         FORTRAN program
         .
      END
         IDENT TWO
           .
           .                       COMPASS subprogram
           .
         END
      SUBROUTINE TWO
         .
         .                         FORTRAN subroutine
         .
      END
         SCOPE
```

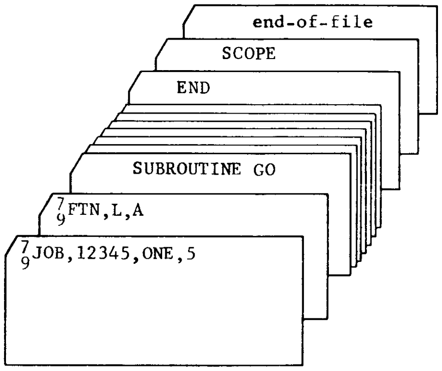
```
1        10
7
9ALDAP,L,P,A
PROGRAM ARCO
            .
            .                     ALGOL program
            .
         'EOP'
            .                     ALGOL procedure
            .
         'EOP'
         SCOPE
```

```
1      8
7
9COBOL,M,L,,P,T
       IDENTIFICATION DIVISION.
                 .
                 .                COBOL program
                 .
       END PROGRAM.
```

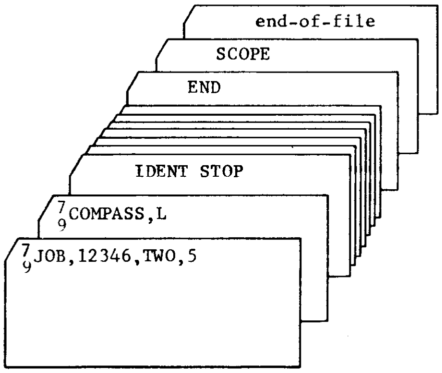
```
1        10
7
9ALDAP,L,P,A
         IDENT FOR
            .
            .                     COMPASS subprogram
            .
         END
            .
            .                     ALGOL procedure
            .
         'EOP'
            .
            .                     ALGOL procedure
            .
         'EOP'
         IDENT TAG
            .
            .                     COMPASS subprogram
            .
         END
         SCOPE
```

Examples:

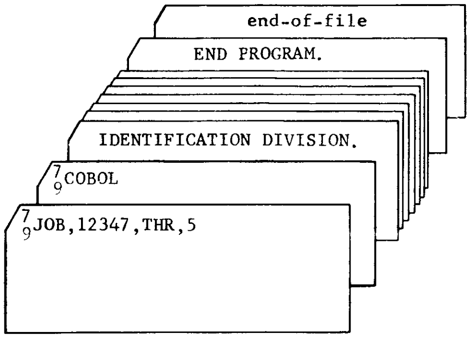
1) Compilation of a single FORTRAN subroutine.



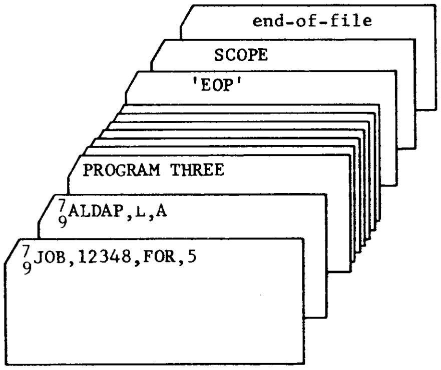
2) Compilation of a single COMPASS subroutine.



3) Compilation of a single COBOL program.

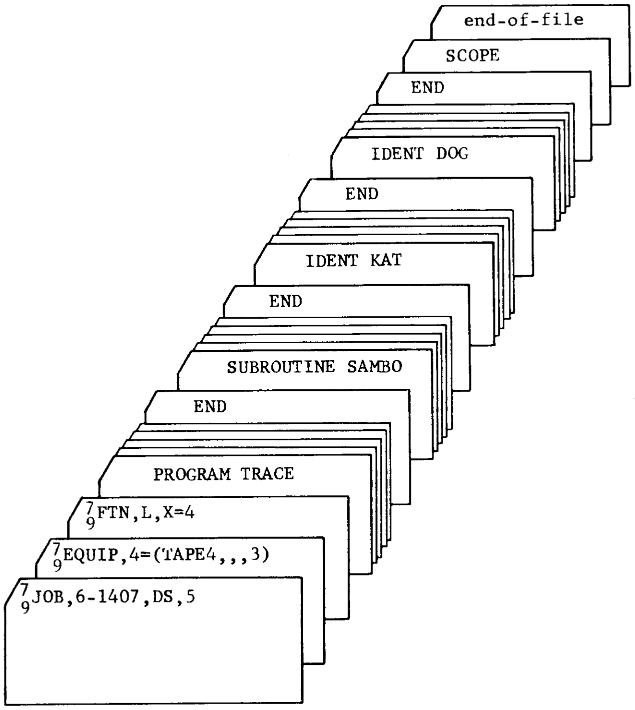


4) Compilation of a single ALGOL program.

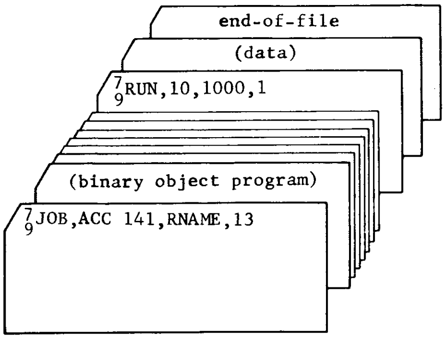


PROGRAM	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	REMARKS
ALGOL	P	R	O	G	R	A	M		n	a	m	e																					FIRST CARD OF SOURCE SUBPROGRAM(OPTIONAL)
										'	E	O	P	'																			LAST CARD OF EACH SUBPROGRAM
										S	C	O	P	E																			END OF COMPILE (ALDAP ONLY)
COBOL								I	D	E	N	T	I	F	I	C	A	T	I	O	N		D	I	V	I	S	I	O	N	.		FIRST CARD OF SOURCE PROGRAM
								E	N	D		P	R	O	G	R	A	M	.														END OF PROGRAM AND COMPILE
COMPASS										I	D	E	N	T		n	a	m	e														FIRST CARD OF EACH SOURCE SUBPROGRAM
										E	N	D																					LAST CARD OF EACH SOURCE SUBPROGRAM
										S	C	O	P	E																			END OF ASSEMBLY
FORTRAN							P	R	O	G	R	A	M		n	a	m	e															FIRST CARD OF EACH SOURCE SUBPROGRAM
							S	U	B	R	O	U	T	I	N	E		n	a	m	e												
							F	U	N	C	T	I	O	N		n	a	m	e														
							E	N	D																								LAST CARD OF EACH SOURCE SUBPROGRAM
										S	C	O	P	E																			END OF COMPILE

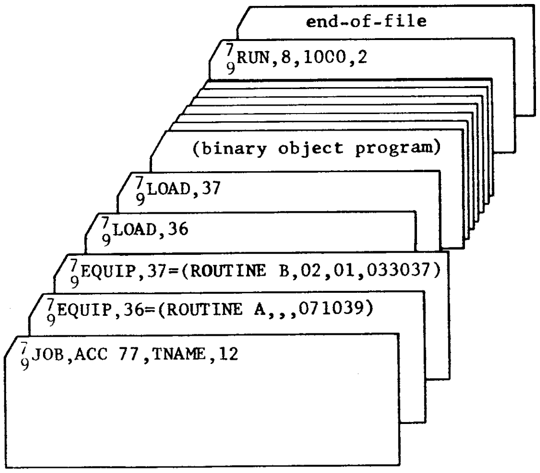
5) Compilation of a FORTRAN Program and several subprograms together with a COMPASS assembly. All binary object programs are placed on unit 4, which is magnetic tape.



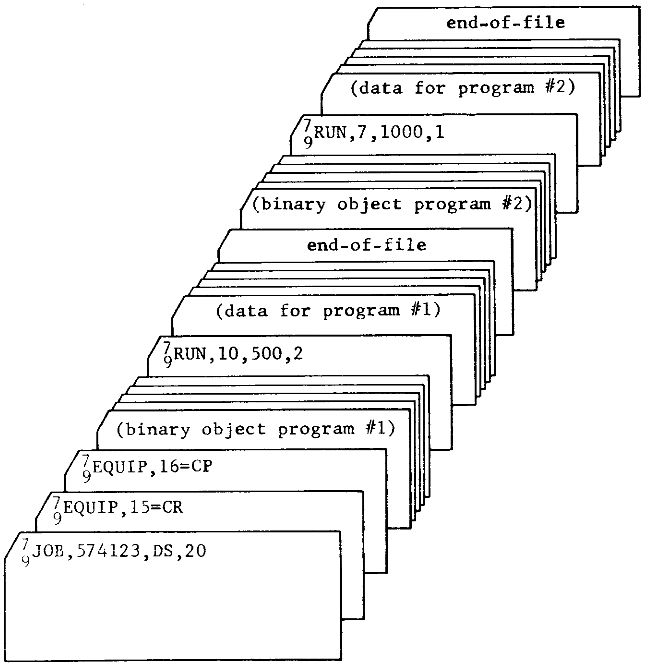
6) Execute directly from the standard input unit.



7) Load from two programmer units and INP.

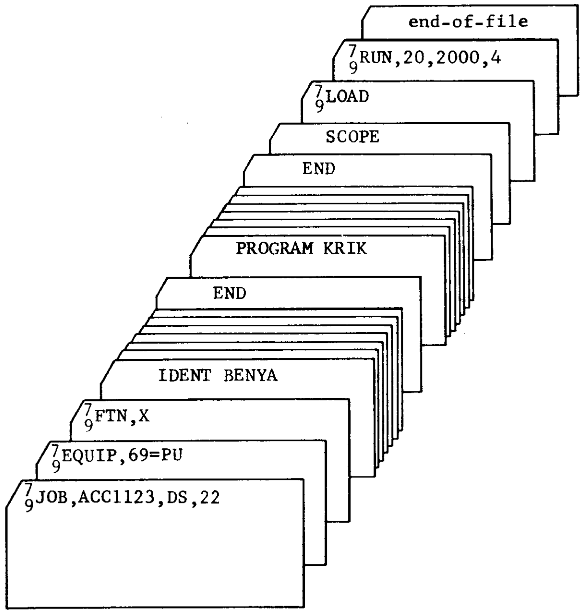

end-of-file
$^{7}_{9}$RUN,8,1000,2
(binary object program)
$^{7}_{9}$LOAD,37
$^{7}_{9}$LOAD,36
$^{7}_{9}$EQUIP,37=(ROUTINE B,02,01,033037)
$^{7}_{9}$EQUIP,36=(ROUTINE A,,,071039)
$^{7}_{9}$JOB,ACC 77,TNAME,12

8) Sequential execution using EQUIP cards.



Changes made because of EQUIP statements remain in effect for the duration of the job or until changed by another EQUIP statement.

9) Assemble a COMPASS subprogram and FORTRAN subprogram on the load-and-go unit. Execute the program. Punch the load-and-go unit.



$^7_9$JOB,ACC1123,DS,22

$^7_9$EQUIP,69=PU

$^7_9$FTN,X

IDENT BENYA

END

PROGRAM KRIK

END

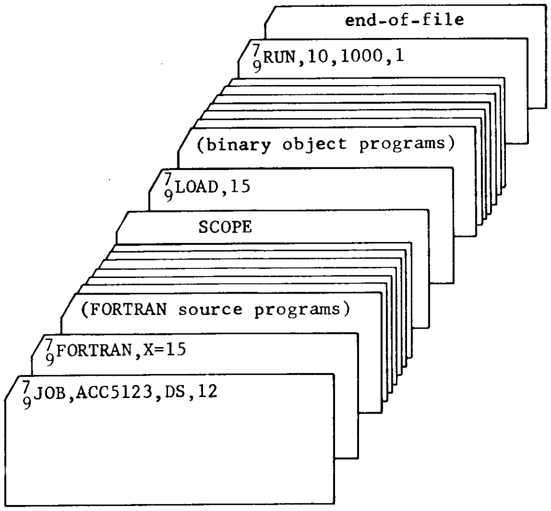
SCOPE

$^7_9$LOAD

$^7_9$RUN,20,2000,4

end-of-file

10) Compile and load a FORTRAN program. Load a binary subprogram from INP. Execute.



$^7_9$JOB,ACC5123,DS,12

$^7_9$FORTRAN,X=15

(FORTRAN source programs)

SCOPE

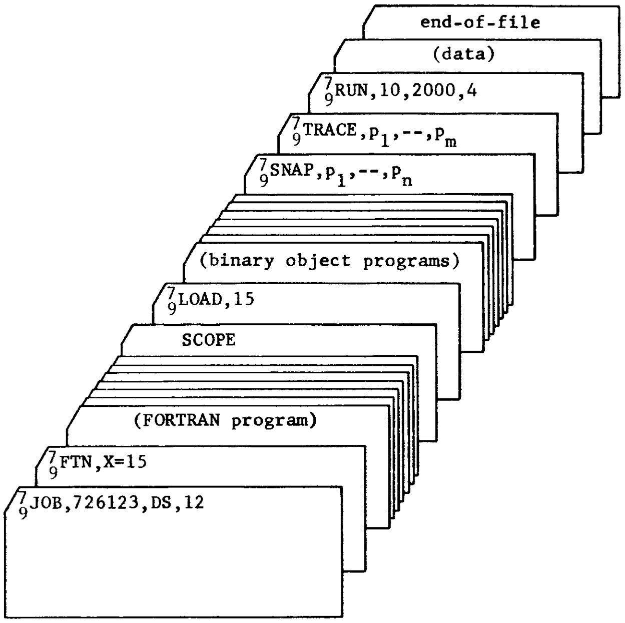
$^7_9$LOAD,15

(binary object programs)

$^7_9$RUN,10,1000,1

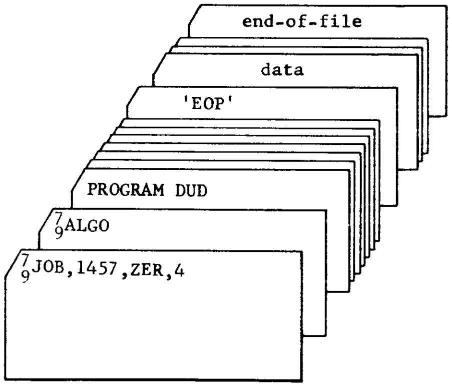
end-of-file

11) Include debugging aids.

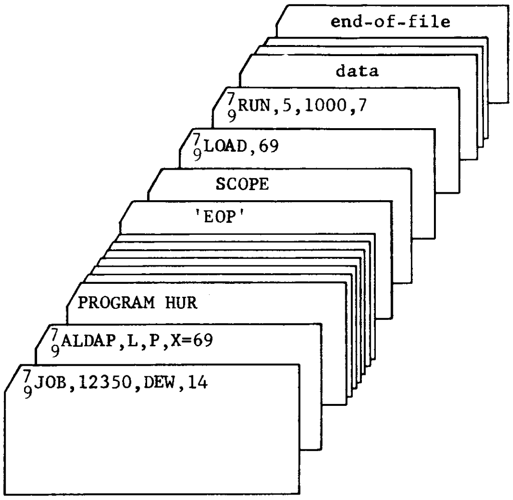


SNAP and TRACE cards precede the RUN card. (Refer to Chapter 6). The number of print requests includes SNAP or TRACE dumps.

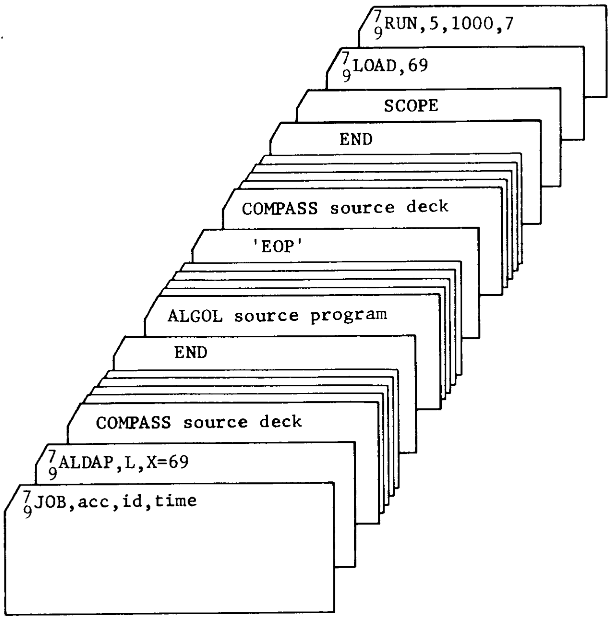
12) Compile and execute ALGOL program.



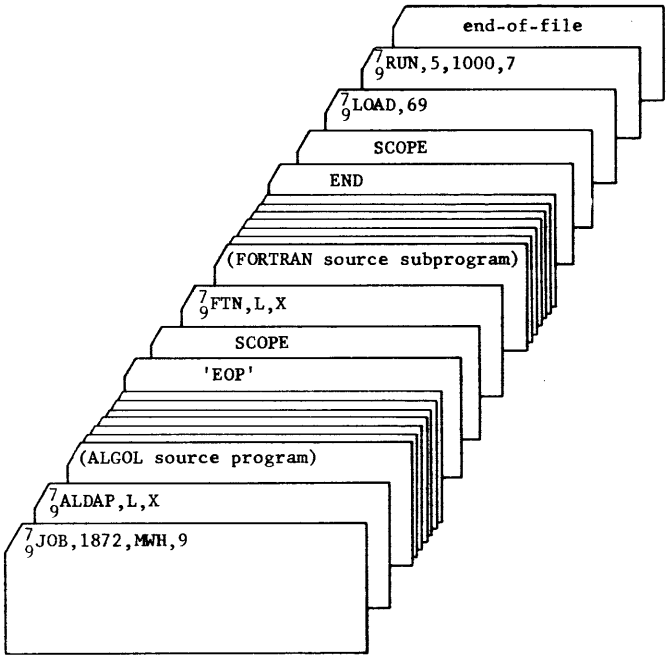
13. Compile and execute ALGOL program.



14) COMPASS subprograms in the form of subroutines may be assembled with an ALDAP compilation containing external declarations for the COMPASS subprograms. Neither a COMPASS nor FORTRAN subroutine may have a transfer card, since the ALGOL program always has a transfer card.



15) ALGOL programs may be compiled before or after FORTRAN programs for the same job. The basic operations for ALDAP, compile only, execute only and load-and-go are similar in format to those of other systems. A transfer address is generated by the ALDAP compiler and may not be provided by the programmer.



CENTRAL PROGRAMS 3

Central programs may consist of assemblies, compilations, and execution. Assembly or compilation result in a binary object program which may be executed as part of the central program, or transferred to a data storage device for later execution.

In the following text, a differentiation is made between the term "central program" and "user's binary object program." In actual practice, they may be the same, or the latter may be only a part of the whole central program. This depends upon how the central program is defined by the Drum SCOPE control cards discussed in Chapter 2.

3.1 NORMAL ENTRY AND EXIT

When the user's binary object program is given control by Drum SCOPE, an EXIT request (Section 4.3) is placed in the first program location, and control goes to the second program location. Therefore, the user may return control to Drum SCOPE by transferring control to the first location of his program or by specifying his own EXIT request.

3.2 INTERRUPTS

The logical flow of control in a central program may be interrupted by certain conditions.

With the SELECT request (Section 4.2.2) the user may choose conditions which interrupt the logical flow of control during the execution of his binary object program and cause a user-specified interrupt subroutine to be entered. Also, input/output requests (Section 4.1.2) contain an optional interrupt address parameter; the subroutine indicated by that parameter is entered when the I/O operation is complete.

Drum SCOPE enters an interrupt subroutine by placing a RETURN request in the first location of the subroutine and initiating control at the second. The programmer may return control to Drum SCOPE by transferring control to the first location of his interrupt subroutine or by specifying his own RETURN or RETURNM request.

3.2.1 REGISTER SAVING

When an interrupt subroutine is entered, the B1-B6 registers remain as they were when the interrupt occurred. The interrupt subroutine is responsible for maintaining these registers for their own use from enter to interrupt, however. When an I/O interrupt occurs, the A and Q registers contain the status; in all other cases, they remain unchanged.

When an interrupt subroutine returns control to the system by a RETURN or RETURNM request, the A, Q, and B1-B6 registers will not be modified; when control is returned to the central program, the system will restore their values. The values of the A and Q registers may be modified by the HERESAQ request. The location to which control is returned may be obtained by the WHERE request and modified using the RETURNM request.

When an interrupt subroutine is entered because of a bounds fault, illegal instruction or storage reference, a RETURNM request must be used. Otherwise, the error-producing instruction will be repeated. (See 3600 Computer System Reference Manual 60021300, table 4-1.)

3.3 ABNORMAL TERMINATION

A program will be terminated abnormally (abandoned) if illegal or invalid conditions arise. Any I/O operations which have been initiated will be processed, but no interrupt subroutines will be entered. Programs will be abandoned under the following conditions:

1. Violations of the rules given for the requests
2. An invalid or unrecognizable request
3. Exceeding the time limit imposed by the JOB, LOADMAIN, or RUN card in a central program
4. Exceeding the print limit imposed by a RUN or LOADMAIN card in a central program
5. An ABORT request
6. A bounds fault
7. The occurrence of an illegal instruction as defined by the monitor protect feature with the exception of DISABLE and ENABLE in a central program
8. A storage reference fault
9. Operator termination of a central program

Conditions numbered 6, 7, and 8 will not cause program termination within central program enabled code if an interrupt subroutine has been selected for these conditions with SELECT or BOUND requests.

When a program is abandoned, a diagnostic message is produced on OUT with a record of the console at the time the condition arose. If an execution phase is terminated, a core dump, as requested on the RUN or LOADMAIN card, is produced.

3.4 ENABLE/DISABLE MODE

A central program is originally activated (given control) in the enabled mode. In this mode, the logical flow of the program may be interrupted by a user-selected interrupt condition. When the interrupt subroutine is entered, the central program enters the disabled mode.

In disabled mode, the following conditions hold:

1. Bounds specified with the BOUNDS request are ignored; only the Drum SCOPE job bounds are in effect.
2. A bounds fault, illegal instruction, or storage reference fault will result in abnormal termination.
3. Except for the abnormal conditions listed in (2), the logical flow will not be interrupted by the occurrence of a selected interrupt condition. Execution will continue without entering an interrupt subroutine; the conditions will be held until the central program exits from the disabled mode either through a RETURN or RETURNM request or through an ENABLE request.

The central program may also enter the disabled mode with this request:

```
DISABLE
```

The same conditions as stated above hold. If the disabled mode is in effect, when this request is issued, no change occurs.

The central program will enter the enabled mode with this request:

```
ENABLE
```

This request should be used only to return to enabled mode after a DISABLE request. Any other use may cause endless loops.

Both ENABLE and DISABLE must be used with caution. Their purpose is only to permit enabled code to logically lock out for a period of time those interrupts which disrupt the logical flow of the program.

3.5 STORAGE ALLOCATION

The central program may use any portion of memory except certain areas of bank zero. Bank zero is divided into four areas:

Fixed System Area

The size of this area is fixed when the system is initialized.

Background-Driver Area

The size of this area is fixed at the start of each phase and is adjusted by the system between phases. The user may control the size of this area as follows:

Direct Control

> Operator request for a background program. The adjustment will not be made until a phase break.
>
> DEMAND card. The background program and drivers will be removed, if necessary, to meet the memory requirements.

Indirect Control

> Creating strings that require a background program for processing.
>
> Establishing a unit which requires a driver.
>
> Calling a production program that requires more space than is currently available.

No Control

> Background program or driver no longer needed.

Central Program Service Area

The size is fixed at the start of each phase, and is adjusted by the system between phases. A minimum size exists; otherwise, the programmer has direct control over the size of the area. There are three components of the central program service area:

Logical Unit block

The size in words is one plus five times the number of logical units allowed. This number is set when the system is initialized and can be modified to some extent by the JOB card. This size is fixed for the duration of a job.

Label Table

The size in words is ten times the number of labels allowed. This is set by EQUIP cards and may be increased between phases.

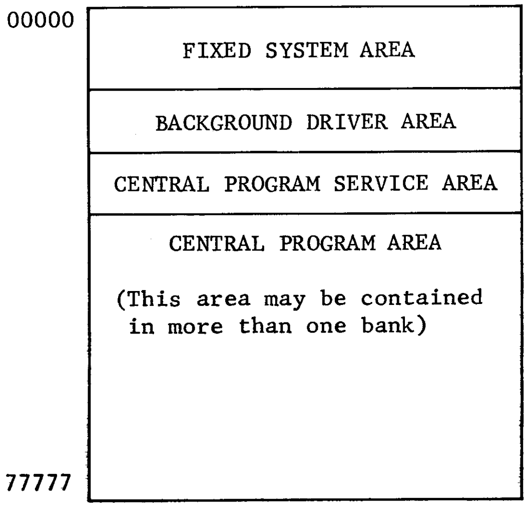
The Buffers

The size is the number of buffers assigned times the block size. The number of buffers assigned depends on the FAMILY card and BUFFER request history. This size is adjusted between phases.

Central Program Area

This area may contain central program code or numbered common. Any remainder is available through the MEMORY request.

The following diagram illustrates the areas in bank zero:


00000
FIXED SYSTEM AREA
BACKGROUND DRIVER AREA
CENTRAL PROGRAM SERVICE AREA
CENTRAL PROGRAM AREA
(This area may be contained in more than one bank)
77777

PROGRAMMER REQUESTS 4

Programmer requests are statements which can be included in assembly language programs. They specify operations for input/output control, internal interrupt, and system requests. They may be written as system macros or in any manner that generates the proper calling sequence to Drum SCOPE.

4.1 INPUT/OUTPUT REQUESTS

Drum SCOPE processes all input/output requests, including read/write, equipment status checks, and tape handling, and performs the following operations:

- Assigns logical unit numbers to physical units
- Selects an available channel
- Stacks a request if a channel is not available
- Responds to external interrupts
- Initiates input/output operations
- Locates a continuation tape when needed to complete an input/output operation or initiates one when end-of-tape is reached

All input/output requests must comply with the following rules:

1. Logical unit numbers used in requests originating from central programs must fall in the range 1-64, 69-79.
2. All parameters specifying destination of data must fall within the bounds allotted to the originating program.
3. In requests involving control word chains, control words must fall within the bounds allotted to the originating program. When control words specify input data transmission, both the first and last word addresses must fall within bounds.
4. The maximum number of control words (excluding IOJP) in control word chains on unblocked units is determined by the installation.

5. Parameters, unless otherwise specified, may be indexed or indirectly addressed. Indexing is performed in one's complement arithmetic using the values of the index registers at the time of the request and operand bank set to the bank from which the request originated.

4.1.1 ASSIGNMENT

The user may control the assignment of equipment. However, in many cases, these requests add information already supplied by EQUIP cards or implicitly assumed.

LABEL

The user may change information previously given in conjunction with the assignment of labeled devices.

LABEL u, addr, edition, reel, retention or creation date

u — logical unit number (decimal); this unit must have been assigned to a labeled unit by a previous EQUIP card. If a unit is currently assigned, it will be released.

addr — address of the first of two computer words containing the name. The name may be:

14 characters, alphanumeric or spaces; parentheses and commas cannot be used, nor can the first character be an asterisk. Characters beyond are ignored.

*nn in the first three character positions of the first word; nn is a number, 01-49. Results are the same as *nn in EQUIP label request.

** in the first two character positions of the first word, signify a non-standard labeled tape.

edition — 1-99. If not specified, blanks will be written in the output label; or any edition number will be accepted on an input label.

reel — 1-99. If not specified, reel 1 is written on an output label or the lowest numbered reel is read from an input label.

retention — retention days for an output tape (0-999) 999 specifies permanent retention.

creation date — date written for an input tape. (mmddyy; month, day, year)

DISPOSE The user may set or change the disposition of the specified unit. Disposition set by the system or by EQUIP cards may be overridden by the DISPOSE request.

DISPOSE u, dd, ea

u	logical unit number 1-49, 61-62, 69, 71-79. If the unit is not a drum, disposition is ignored.
dd	disposition mnemonic; cannot be indexed PR print PU punch Other mnemonics may be included by the installation.
ea	error address; if the specified disposition is not recognized, an exit to the error address is taken.

Example:

DISPOSE 39, PR, ERROR

Logical unit 39 is to be printed when released.

RELEASE To release an assignment, the user may specify:

RELEASE u, ra, s

u	logical unit number, 1-62, 69, 71-79.
ra	reject address to which control is transferred if the unit is busy. A reject address must always be specified.
s	Any character, meaningless unless u is magnetic tape. If s is non-zero, the unload is suppressed.

The RELEASE request will not remove any hardware declarations for central programs. Thus, if an EQUIP card declares

EQUIP, 5 = MT, (label declaration)

and unit 5 is assigned later, for example by a READ request, and then released by a RELEASE request, a new READ for logical unit 5 will again assign magnetic tape. The label declaration is released, however.

SAVE, UNSAVE These requests are ignored; they are provided for compatibility with Tape SCOPE.

$\left\{\begin{array}{l}\text{SAVE}\\ \text{UNSAVE}\end{array}\right\}$ u

4.1.2 DATA TRANSMISSION

In these requests, the following parameters are uniformly defined:

u logical unit number

cwa address of the I/O control word or the first I/O control word in a chain. (See the 3600 REFERENCE Manual for control word format.) Maximum length of a chain is determined by the installation.

ra location to which control is transferred in case the specified unit has not completed the former operation.

ia address of the programmer's interrupt subroutine to which control transfers when the requested operation is terminated.

READ/WRITE Reading and writing of data are initiated with these requests.

$\left\{\begin{array}{l}\text{READ}\\ \text{WRITE}\end{array}\right\}$ u, cwa, ra, ia

If a multi-reel operation on magnetic tape has been indicated, the READ/WRITE request initiates the search for the new reel and the release of the old reel. The direction of read may be designated by a parameter in the MODE request.

The operation will terminate if either the control word chain has been satisfied or an abnormal condition, according to the hardware, arises.

Examples:

READ (INP, CONTROLA, SAM, INTRPT)

Data is to be read from the standard input unit, the first control word is at CONTROLA. If the request is rejected, control is to be transferred to SAM. When the read operation is completed (or an abnormal condition occurs), control transfers to the interrupt subroutine at location INTRPT.

WRITE (OUT, CONTROLB, *, INTRPTB)

A write operation is to be performed on the standard output unit. The first control word is at CONTROLB, and the write request will be executed when OUT becomes available. When interrupt occurs, control transfers to INTRPTB.

REOT/WEOT

These requests are tape movement controls which allow reading and writing after the physical end-of-tape, before a continuation reel is assigned.

$\left\{\begin{matrix}\text{REOT}\\\text{WEOT}\end{matrix}\right\}$ u, cwa, ra, ia

If the request is issued before a physical end-of-tape, the reading or writing occurs and a logical end-of-tape condition is set. At the next READ or WRITE request, a continuation reel is assigned.

Example:

REOT 25, RDCWA, *

Logical unit 25 is read; a logical end-of-tape condition is set after reading is complete. Upon the next READ request for unit 25, a continuation reel will be assigned. Since ia is blank, an interrupt subroutine will not be entered upon completion of request.

RDLABEL

This request permits the user to examine contents of the label on a standard labeled unit.

RDLABEL u, ba, ra, ia

u	logical unit number of a device with a standard label.
ba	first word address of a 10-word buffer.

Drum SCOPE reads the label from u, which must be at the beginning of the volume[†] or unassigned, into the 10-word area beginning at ba. If the system cannot assign the unit, and the operator signals the system to continue by typing in NONE, the word at ba is set to machine zero.

[†] When the amount of space required for a file on a storage device exceeds the amount of contiguous space available, the file will be stored in two or more non-contiguous areas, or volumes. Volumes are then linked together logically. Drum SCOPE opens and closes the various volumes automatically. In the case of tape, volumes are reels of tape for multi-reel files.

Example:

RDLABEL 20, BUFFER, *

The standard label on logical unit 20 will be read into the location beginning at BUFFER. Since ia is blank, an interrupt subroutine will not be entered upon completion of request.

WRLABEL

With this request, the user may write information into the optional text area of the standard label. (The unit must be labeled, and it must be either at the beginning of volume or assigned.)

WRLABEL u, ba, ra, ia

Upon completion of the operation, the label written by the system will be in the area beginning at ba. The system will supply the first 32 characters and use the 48 characters beginning at ba + 4 (refer to Standard Label Format, Chapter 5).

RDBLOCK/WRBLOCK

RDBLOCK u, cwa, ea, ia

WRBLOCK u, cwa, ea, ia

These requests, valid on blocked units only, transmit full blocks of information rather than single records. They are primarily intended for use by background programs and system programs such as assemblers and compilers. They may be used by other central programs but require thorough knowledge of blocked I/O processing. Some restriction on the use of these requests follow:

1. Record control words (RCW) are checked for legality on WRBLOCK (and WRITE) requests. The RCW's must either string to the end of the block or to an RCW. E control word. The program will be terminated if these conditions are not met.

2. The CWA in the request must point to a word containing (in bits 17-00) the location of a block-size buffer. The upper half of the word will be ignored--BLSIZE words will be transmitted regardless of the word count, or type of IOxy. Chaining will also be ignored.

3. If the CWA points to a family buffer the following points must be carefully observed.

   Any other I/O on the same family will destroy information in the buffer (example: RDBLOCK on LUN 60 and WRITE on LUN 62 may cause trouble).

If the family contains two buffers, all RDBLOCK and WRBLOCK requests must specify the primary buffer. (A member request will return with the primary buffer location in QL and the secondary in QU. In addition bit FAT.PP, if set, signifies that the buffer is being read from or written onto the drum and should not be distrubed until the bit becomes zero.) The primary and secondary buffers are alternated; once the primary buffer is determined, it is necessary only to switch buffers on requests.

4. Mixing READ with RDBLOCK and WRITE with WRBLOCK requests require further precautions:

   The first word of the buffer (normally, the BCW) contains in the lower 9 bits, a pointer to the word following the last record READ if a RDBLOCK follows a READ. The entire buffer will be transmitted, but part of it has already been processed. (This will usually be true in the case of the first RDBLOCK on LUN 60 since at least some READS have been done by the system).

   A SETUPT request must be used to set the pointer if the user has not processed all records in a block after a RDBLOCK request if READ requests on the same unit are to follow. (For example, if COMPASS does a RDBLOCK on LUN 60 and encounters a SCOPE card part way through the block, a SETUPT request must be made pointing to the next record to keep the central program from being terminated on the next READ request.)

   Before issuing a WRBLOCK request on a family buffer, a GETUPT request should be made to find out if any information has been stored in the buffer by previous WRITE requests. GETUPT will return with the A Lower containing the pointer. (Zero if no writes have been made, otherwise the location in the buffer where the next RCS should go.) A SETUPT request must be issued to set the pointer back to zero before the first WRBLOCK if the pointer was non-zero to avoid losing information.

SYSIO With this request the user may write accounting data or a message to the operator.

SYSIO f, k, fwa, wdct

f function; unindexed digit, 0 or 1
0, write message to operator
1, accounting information

k relevant only if f = 0

k = 0	message appears always
1	if stop key 1 is set
2	if stop key 2 is set
3	if stop keys 1 or 2 are set
4	if stop key 3 is set
5	if stop keys 1 or 3 are set
6	if stop keys 2 or 3 are set
7	if stop keys 1, 2, or 3 are set
fwa	18-bit unindexed address
wdct	15-bit word count, unindexed; with fwa, this parameter defines a buffer area

The message beginning at location fwa, containing wdct words will be written on the unit specified by f.

f = 0-3	reserved for the system
4-7	available for installation use
0	write output comment
1	write accounting
2	unused
3	unused

Example:

```
SYSIO 0, 5, (($)MSG),3
```

The 3-word message beginning at location MSG will be output on the typewriter if stop keys 1 or 3 are set. The message is in internal BCD mode.

CONTROL REQUESTS

control name u, ra, ia

Control names are listed below:

BSPF — Backspace until an end-of-file mark or beginning of volume is reached, whichever occurs first. On tapes, a BSPF given at load point will cause a backward motion; remounting will probably be necessary.

BSPR — Backspace one record or to beginning of volume, whichever occurs first. On tapes, a BSPR given at load point will cause a backward motion; remounting will probably be necessary.

REWIND — Rewind to beginning of current volume.

SKIP — Skip to end-of-file or end of volume, whichever occurs first.

ERASE — Erase 6 inches of tape (unblocked tapes only).

MARKEF — Write end-of-file mark.

The formats of the following control requests differ slightly from those listed above:

UNLOAD u, ra, ia, p — Rewind to beginning of current volume and unload. If p is 0, the assignment is released; otherwise, the assignment is kept. (Sec. 3.1.4)

LOCATE u, lra, ra — Position random access unit at address contained in location lra. Unless u is a random access unit, the request is bypassed. The relative address contained in location lra must be between 0 and the total number of words in the file minus one. Upon return, either normal or reject, the A register contains the total number of words in the file.

STATUS

When a status request is issued, information regarding the unit involved is returned in the AQ register. A unit on which an operation has been requested may be in one or four phases; quiet, held, busy, or stacked. Requests are rejected on units which are not quiet.

In the quiet phase, no operation has been requested, or the operation is complete and the interrupt subroutine has been entered.

In the held phase, the previously initiated operation is complete, but the interrupt subroutine has not yet been entered.

In the busy phase, an I/O operation is currently in process.

In the stacked phase, an operation has been requested but has not yet been initiated by the system.

STATUS u

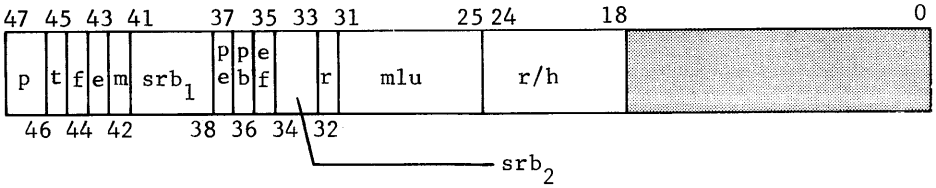
This request permits the user to detect the completion of an operation and obtain the status response. The status on a quiet or held unit reflects the status of the unit at the completion of the prior operation or at the last instance of a DYSTAT request.

DYSTAT u

This request is like STATUS except that if the unit is assigned and is quiet, a dynamic status, the current status of the physical device, is returned. The DYSTAT request should be used with care since it can take considerably longer than the STATUS request. If the unit is unassigned or if the unit is a blocked unit, the DYSTAT request behaves exactly like STATUS.

The reply to the STATUS and DYSTAT requests is entered in the A and Q registers as follows:

Q Register



p phase: 00, quiet
01, held
11, busy
10, stacked

t tape flag: 0, unit is a tape or a blocked unit, examination of the r/h field will determine which.

1, unit is not a tape or a blocked unit. The r/h field contains the hardware code.

f bypass indicator: 0, unit is not bypassed
1, unit is bypassed

mlu master logical unit; may differ from u in the request if units are equivalenced.

r/h if t = 0 and:

r/h = 0, unit is blocked.

r/h ≠ 0, unit is tape and r/h is current volume number.

if t = 1:

r/h contains the ordinal of the hardware type.

The remainder of the status is meaningful only on a quiet or held unit. If a unit is busy or stacked, the accuracy of the flags and fields cannot be guaranteed.

e end of volume indicator

0, end of volume condition does not exist.

1, end of volume condition does exist; this may arise from an REOT or WEOT request, or physical end of volume. The condition is cleared when a continuation volume is assigned, or when a backward motion request is issued and the physical end of volume condition is removed.

pe physical end of volume indicator

0, not physical end of volume.
1, physical end of volume.

pb physical beginning volume indicator

0, unit not positioned at beginning of volume.
1, unit is positioned at beginning of volume.

ef end-of-file indicator

0, end-of-file has not been read.
1, end-of-file has been read. This condition arises after:

Reading end-of-file with READ, REOT
Writing end-of-file with MARKEF
After BSPF, SKIP, or BSPR which read a file mark

r ready indicator

0, unit is not ready or not assigned
1, unit is assigned and ready

m transmission error indicator

1, parity or mode error occurred during prior transmission. A retry is recommended.

0, no transmission error.

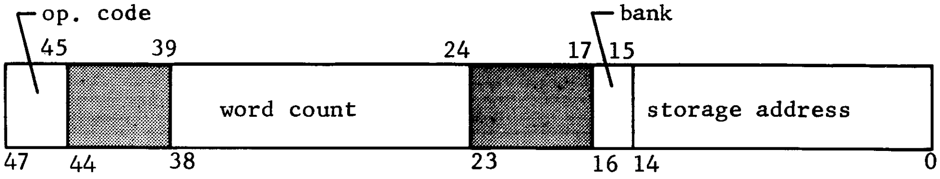
Interpretation of the status reply bits (srb_1 and srb_2) depends upon the hardware device. (See table 4.1)

Table 4-1. Hardware-Dependent Status Reply Bits[†]

	Card Reader	Card Punch	Line Printer	Unblocked Magnetic Tape	Random Access	Blocked Units
Q(34)	next card binary			write ring present		next card binary
Q(35)	card with 7-8 punch in column 1 has been read			EOF mark has been read		
Q(36)				load point is present	beginning of volume	beginning of volume
Q(37)	input tray empty			end of tape reflective spot has been read	end of volume	end of volume
Q(38)	end-of-file switch ON and last card has been read	feed failure or card jam	paper supply low	unit set to HI density		
Q(39)	stacker full or card jam	compare error on card prior to last one punched	paper is at last line (channel 7)	unit set to HY density		
Q(40)	read compare or pre-read error			lost data condition		
Q(41)				end of operation		

[†] Blanks indicate the bit is meaningless for that particular hardware.

A register:



If the previous operation was a read or write, the control word, on completion of operation, is returned in the A register. The unused fields in A and Q are of no significance to the user.

MODE

A MODE request defines the usage of a unit or specifies density or recording mode. A MODE request can be honored only if the unit is quiet; otherwise, control returns to the reject address.

MODE u, ra, d_1, d_2, d_3, d_4

A logical unit designator and a reject address must be present; d_i declarations are defined below. If declarations contradict the last mentioned, or rightmost, takes precedence. All declarations apply to the master logical unit. Certain declarations are implied by the hardware, for example, LP is WO.

Only one (the rightmost) parameter from each of the following groups will be recognized. The order is not important, except that there may be no null parameters within the declaration list. (The processor will not process any parameters beyond the first blank or null indication.)

Usage specifies an operating condition for the unit:

RW	(read and write) all legal requests will be performed.
BY	(bypass) STATUS, MODE, and RELEASE requests will be processed. Other operations will be accepted and interrupt subroutines entered.
RO	(read only) WRITE, WEOT, MARKEF, ERASE, or WRLABEL requests are illegal.
WO	(write only) READ, REOT, RDLABEL, or SKIP are illegal.
FO	(forward only) BSPF, REWIND, or UNLOAD requests are illegal. Drum SCOPE releases the drum area on which the information is recorded while reading or writing is taking place, allowing the user, for example, to start printing prior to the release of the entire file. One BSPR following a READ or WRITE is permitted provided that the record size is less than the system buffer size, but subsequent BSPR requests cannot be guaranteed. If a BSPR is requested and the data is not available, a diagnostic will result and the job will be abandoned. FO may be cleared only by a RELEASE.

The declarations RW, RO, WO, and BY are mutually exclusive and reset any former declaration except FO.

Recording mode selection remains until modified.

BIN	binary mode
BCD	BCD mode

Density may be declared for magnetic tape; it is ignored elsewhere. A density declaration in MODE request does not automatically imply magnetic tape, as it does in the EQUIP control card.

OP operation option; for input, defines the density to be the same as the density of the label; for an unlabeled tape, the operator selects the density when he mounts the tape. For output, a standard density determined by the installation is used.

HY	hyper density tape (800 bpi)
HI	high density tape (556 bpi)
LO	low density tape (200 bpi)

Direction may be declared for magnetic tape; it is ignored elsewhere.

ND normal direction

RV reverse direction; READ, BSPR, and REOT employ reverse read as defined in 3600 Reference Manual. Although FO and RV are seemingly contradictory, FO applies only to blocked units and RV applies only to tapes.

CHECK

This request terminates input/output operations. The response returned from the CHECK request gives the status at the time the operation terminated.

CHECK u

The response depends upon the phase of the unit at the time the operation terminates. If the unit is quiet, the CHECK request returns a quiet status, as in the STATUS request. If the unit is held, the request returns a held status, but the interrupt subroutine is cancelled. If the unit is busy, the system will wait until the unit is no longer busy. If the unit is stacked, the request returns a stacked status and the stacked request is cancelled.

After a CHECK request, the unit is quiet.

READY With this request the user may sense a ready condition on a unit; the interrupt subroutine is entered when the unit is ready.

READY u, ra, ia

This request has little meaning if no interrupt address is specified; the programmer would have to constantly check the status of the unit with one of the status requescs.

A ready condition is generally defined as ready and not busy, that is, ready for an operation.

GETUPT GETUPT lun

This request returns, in A lower, the pointer to the next record control word on the blocked unit specified by lun. It is to be used with RDBLOCK/WRBLOCK requests. A pointer of zero points to the first record control word in the buffer (the second word of the buffer).

SETUPT SETUPT lun, val

This request sets the pointer to the next record control word to val, on the blocked unit specified by lun. It is intended for use with RDBLOCK/WRBLOCK requests.

4.1.3 I/O BUFFERING

Buffers are used as intermediate storage areas for data transmitted between internal memory and external storage. Buffer size is determined by the installation, within limits of 129-511 words.

To provide efficient buffering on blocked units, Drum SCOPE requests allow the programmer to group logical units into families and assign a pool up to 2 buffers to a family. The system family, 0, contains the system units. For central programs, family 1 is initially defined to contain units 60, 62, and 69; family 2, unit 61. These associations may be changed with the FAMILY request.

Drum SCOPE is able to buffer one member of each family at a time; thus optimal performance is obtained for consecutive forward motion operations. Assigning two buffer areas allows maximum efficiency for input/output operations; Drum SCOPE will attempt to transmit one block of the data from one buffer while the other buffer is refilling.

Closely alternating input or output of two or more members of one family slows transmission; Drum SCOPE must unload the buffer areas for the old member and initiate buffering for the new one each time. If the need for fast transmission of units is not simultaneous, the time used in unloading the buffers for the old member and initiating the new one is not prohibitive. For example, consider an application where a program reads a large amount of data from one unit and then writes a large amount of data on some other unit. In other cases, slow buffering could possibly be tolerated, for example, with low frequency units.

A two-buffer family is more efficient. If only one buffer is available, no one-ahead buffering is possible; each time a new block of data is required, a delay will occur.

Another important consideration is the amount of storage used by buffers. Two family 0 buffers are always present; the central programmer can normally assume that there is room for four more buffers, nominally assigned to families 1 and 2.

FAMILY

With this request a programmer may attach a unit to a family.

FAMILY u, f

f family number, 0 to F, where F is a quantity determined by the installation.

u logical unit number 1-62, 69, 71-79 (or limited to valid references from a background program)

To maintain efficient I/O processing, avoid attaching a unit to family 0.

BUFFER

This request declares a buffer pool for a family.

BUFFER u, a, c

u unit number

a, c 18-bit addresses or zero

Non-zero a and c define the starting addresses of the buffer areas which the system may use for the family which contains unit u. If only one buffer is to be used, specify c = 0. No check is made to detect storage conflicts; however, an attempt to locate a buffer area outside of the job bounds will be detected.

If more than one BUFFER statement is made for units in the same family, the last one encountered by Drum SCOPE overrides the others. The MEMBER request as well as the . POOL common block (Section 7.1) may be used to determine the size of buffer areas.

MEMBER

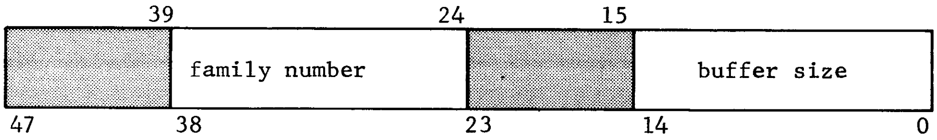
This request determines the family membership and buffer area allocations of a unit.

MEMBER u

u logical unit number

The reply is contained in A and Q as follows:

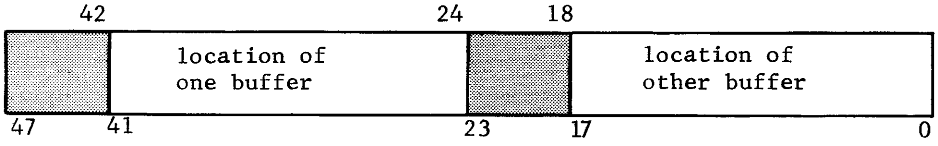
A register:



family number number of the family to which this unit is currently attached.

buffer size current system buffer size.

Q register:



4.2 INTERRUPTS

The user may define certain events as interrupt conditions. When such an event occurs, normal processing is discontinued, and control goes to a user-specified interrupt subroutine. Control must be returned to the interrupted program via the RETURN or RETURNM requests.

4.2.1 I/O INTERRUPTS

Interrupts may be specified by the ia parameter of certain input/output requests. When this type of interrupt subroutine is entered, the A and Q registers contain the status of the interrupted unit as defined in the STATUS request.

4.2.2 INTERNAL INTERRUPTS

Drum SCOPE contains eight additional requests with which the user may control internal interrupts.

SELECT

This request selects the condition to be detected by an interrupt subroutine. When the interrupt feature occurs, the subroutine indicated by the interrupt address will be entered.

The fault will be cleared when the SELECT request is given. That is, an error which has occurred prior to the issuance of the SELECT request will not cause the interrupt subroutine to be entered.

If the feature has been requested before in this job, the old interrupt address is saved within the request. Thus, a form of push-down selection can be obtained.

SELECT m, ia

m interrupt feature mnemonic; may not be indexed.

SHIFT	shift fault
DIVIDE	divide fault
EXOV	exponent overflow
EXUN	exponent underflow
OVER	arithmetic overflow
ADDR	storage reference fault
TRACE	trace mode alert
INST	illegal instruction
MANUAL	operator message
ABNORM	abnormal termination - only one ABNORM may be given in any phase. The specified routine is entered if any abnormal termination occurs.

The user is given an amount of time, determined by the installation, to perform desired termination procedures (such as writing out partial buffers, guaranteeing termination of all I/O and so forth). While operating in the disabled mode, the logical flow of the program will not be interrupted by the occurence of a selected interrupt condition.

I indirect - ia must be the address of a REMOVE or another SELECT request where an interrupt selection has been saved. Both the feature and interrupt address will be obtained from this request.

ia interrupt address, or if m = I, the address of a SELECT or REMOVE address; ia may not be zero.

REMOVE This request removes the selection of an interrupt.

REMOVE m

m any interrupt feature mnemonic except I, as explained in SELECT.

If any of the features ADDR, ABNORM, or INST is removed, an automatic selection will take effect to cause abnormal program termination if they occur. The fault is cleared.

BOUND Drum SCOPE is protected by upper and lower bounds. The upper bound is the highest location in the highest numbered bank available, excluding the system and background programs. The lower bound is the lowest numbered location in bank 0 excluding Drum SCOPE. A bounds fault causes program termination unless a BOUND request has been used previously to select a programmer interrupt subroutine.

This request may be used to impose bounds inside the Drum SCOPE bounds.

BOUND lb, ub, ra, ia

lb lower bound; 18-bit, unindexed address; zero indicates Drum SCOPE lower bound.

ub upper bound; 18-bit, unindexed address; zero indicates Drum SCOPE upper bound.

ra reject address; control transfers to this location if the requested bounds do not fall within Drum SCOPE bounds.

ia interrupt address; control transfers to this location if a bound fault occurs, if zero, UNBOUND is simulated.

The current bounds setting is returned in the A register.

If lb = ub = 0, IA is ignored and normal return taken.

Example:

BOUND (((*) LIMIT1), LIMIT2, RA, IA)

The lower bound is LIMIT1 in the bank containing the BOUND request. The upper bound is LIMIT2 in the bank containing LIMIT2. ($) LIMIT2 is assumed.[†]

UNBOUND

The Drum SCOPE bounds are restored with this statement. UNBOUND cannot remove the bounds set by the system.

UNBOUND

MEMORY

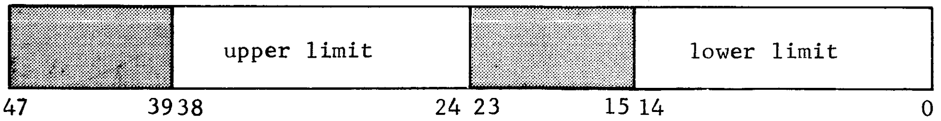
The limits of available storage may be obtained or changed by the MEMORY request.

MEMORY bank designator, lower limit, upper limit

bank designator bank number (0-7)

lower/ upper limits limits set by the programmer; may not be indexed. If zero, limits are set equal to Drum SCOPE job limit. Lower limit must be less than upper unless upper limit is zero.

When the limits have been changed by MEMORY, the new limits are returned in the A register. A non-existent bank is indicated when the A register is equal to zero. If no limits are supplied in the request or if both limits are zero, the current storage limits for the specified bank will be entered in the A register in the following binary format:



[†] See the 3600 COMPASS manual for meaning of bank designator ($).

LIMIT

A time limit is set after which control will be transferred to the interrupt address.

LIMIT du, ra, ia

du — duration in seconds of the time limit; milliseconds may be appended with the parenthesized expression (seconds, milliseconds)

ra — transfer location if the limit is not accepted

ia — location to which control transfers when the limit is reached

No more than a standard number (determined by the installation) of limits may be in effect at one time; each must fall within the time set by the last executed LIMIT request. If the new limit is greater than the time remaining for the previous selection, control will transfer to the reject address.

Example:

```
LIMIT)) 1000, 500), RA1, IA1)
```

FREE

This request releases the last time set by a LIMIT request (the smallest in the list of time limits) and re-establishes the next previous time set (the smallest in the list). Limits, such as JOB and RUN, set by the system cannot be freed.

```
FREE
```

RETURN

Control returns to Drum SCOPE from an interrupt subroutine. When it is issued, from a central program enabled routine, it is treated as an EXIT request.

```
RETURN
```

RETURNM

Modifications may be made to the address to which control will be returned.
RETURNM lower/upper, address, operand bank

lower/upper — L indicates control is to return to the lower half word. (A BJPL instruction is generated.)

U indicates upper half word. (A UBJP instruction is generated.)

address — 18-bit address to which control is to be returned.

operand bank — 3-bit operand bank value.

RETURNM may originate from a central program interrupt subroutine only; when it is issued from a central program enabled routine, it is treated as an EXIT request.

4.3 SYSTEM REQUESTS

The remaining requests are used to interrogate and make requests of Drum SCOPE.

TIME

Upon receiving a TIME request, Drum SCOPE enters the time of day into the Q register in BCD and the time remaining before the next time interrupt into the A register in binary.

TIME

The time of day is based upon a 24-hour clock (one minute before midnight is 235900) and is given in hours (hh), minutes (mm) and seconds (ss). This time is entered in BCD in the Q register in the format:

∧hhmmss∧ (∧indicates a blank)

The time remaining until the next user, RUN, or JOB time interrupt is entered in the A register in the format:

▨	milliseconds	seconds
47	38 24	23 0

DATE

When this request is received, Drum SCOPE enters into the A register the current date in BCD: mm/dd/yy (mm = month, dd = day, yy = year) and the Julian date in the Q register: ∧∧∧yyddd (yy = year, ddd = day)

DATE

WHERE

With this request, a central program interrupt subroutine obtains the last location executed in the normal central program. The bank jump command stored by the machine when the interrupt occurred is returned in A.

WHERE

EXIT

This request returns control from a running program to Drum SCOPE. Re-entry to the program will not occur; the program is complete.

EXIT

ABORT

The Drum SCOPE system contains a diagnostic routine with stored messages which are definable by the installation. When the program issuing an ABORT request is terminated, post-mortem procedures are taken and the diagnostic (specified by the BCD diagnostic key) is printed. If no diagnostic message exists which corresponds to the diagnostic key, the key will be printed.

ABORT   diagnostic key

LOVER

Drum SCOPE is directed to load the overlay partition[†] specified by the parameters.

LOVER u, f, n

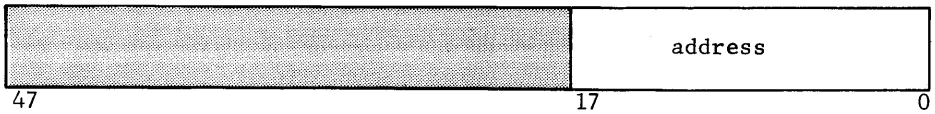
u   logical unit number containing the partition; must be 1-49, or 70 if the program is included in the production file.

f and n specify the type of partition.

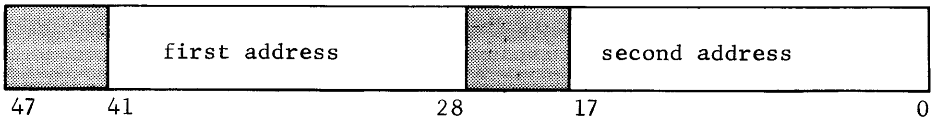
n	f	partition
0	O or S	main
overlay number	O	overlay
segment number	S	segment

When control returns to the system, the A register contains:

One address encountered, A =



Two addresses encountered, A =



[†] See OVERLAY PROCESSING, Chapter 6.

LIBRARY

This request positions the system library in front of the routine whose name is at address ln, left justified with trailing blanks.

LIBRARY ln, ea

ln address of library routine name

ea location to which control is transferred if the routine does not exist in the library.

When the last card of a routine has been read from the system library, end of volume will be set in a STATUS reply.

LOADER

A running program uses this request to call the loader.

LOADER

If the loader is not in storage, Drum SCOPE will read it from the library. It is loaded into the lower part of bank zero, and it uses the area immediately following for tables. The user should avoid saving any information in the lower part of bank 0. Just pricr to this request, the parameters specifying what is to be loaded must be placed in the A and Q registers (section 5.2). Drum SCOPE passes these parameters to the loader and retains control until the call is complete. When control is returned, the recording modes of the units referenced by the loader (which may include LIB, OUT) may be changed.

HERESAQ

This request permits the programmer to change the values of A and Q of the interrupted enabled code while in disabled central program code. The values are retained by Drum SCOPE until control is returned to the central program, and are then restored to the A and Q registers for the enabled code.

HERESAQ

CLBCD

This request converts a column binary card image to a BCD card image.

CLBCD bufa, bufb

bufa first word address of a 20-word area containing the column binary card image to be converted.

bufb first word address of a 10-word area in which the BCD conversion will be stored.

There is no checking for illegal characters. Column one of the resulting card image is set to 60_8.

Each column is converted as follows:

12 punch = 20_8

11 punch = 40_8

0 punch = 60_8 if no 12 or 11 punch
= 12_8 if either a 12 or 11 punch

1-9 punch = 01_8 to 11_8

The resultant character, c, is formed from the punch values, u_i, as follows:

Set c = 0

$c = c\ v^{\dagger}\ u_i \quad i = 12, 11, 0, 1, \ldots, 9$

if c = 0 then set c = 60_8 or

if c = 60_8 then set c = 0

BYNBY This request enables a background program to select a time interrupt. Control will be given to ia following a time delay of less than one minute. This request is value only from a background program.

JOB.ID JOB.ID returns an address in bank 0 at which is located a file word block of information.

Word 0 contains the priority of the current job in bits 17-15 and the sequence number (in binary form) in bits 14-0. Bits 47-18 are 0.

Words 1 and 2 contain the first 16 non-blank characters of the JOB card i field with trailing blanks.

Words 3 and 4 contain the first 16 non-blank characters of the JOB card c field with trailing blanks.

SWAP SWAP ln, ea, ia

ln location of name left adjusted with trailing blanks. The name must be within the bounds of the requestor.

ea address to which control transfers when SWAP is honored, a retry is possible. The following indicators will be in A.

[†] v is logical or operation

A	Q
0 Error	Diagnostic key in Q
1 A SWAP is already in process	
2 No room in PNL for the program	
3 There is no entry point to the program	

ia address to which control transfers if SWAP is honored. The following indicators will be in A.

A	Q
0 It is loaded	
1 The name is unknown	
2 It used bank 0	
-0 Exit	
-1 Aborted	Diagnostic key in Q

When this request is issued, the bank 1 portion of the central program (level 0) is moved from memory onto the drum after all program I/O has ceased. The production program named at ln will be loaded into memory in the area of the central program, and it will be run as a background program at the next available level. The program must use only bank 1. If the SWAP request is not rejected and the program is known and has an entry point, I/O will be stopped, and bank 1 will be saved. The program will be loaded and entered at its entry point.

This operation will have a lower priority than the requestor and therefore cannot operate unless the requestor is not in execution. Normally the instruction following a SWAP request will not be executed; therefore the built-in return in SWAP is not effective because control will always come to either ea or ia. The requestee will be given the A, Q, D registers and the index registers B1-B6. Also, the requestors two's complement status will be set.

LABELS, ASSIGNMENT AND RELEASE PROCEDURES 5

A label is a record which may be used to identify and protect a file. A data storage device may have standard labels in a format recognizable to Drum SCOPE or non-standard labels for which a user must code his own label-checking routine. Labels may also be omitted.

The manner in which a file is labeled affects the manner in which assignment is made.

Standard Label Format

Character Position	Content	Meaning
1	2, 5, or 8	density indicator; 2 = LO (200 bpi), 5 HI (556 bpi), and 8 HY (800 bpi)
2-3	( )	Unique label identifier
4-5	uu	Logical unit number (01-49) or blank
6-8	rtn	Retention code (000-999); number of days tape is to be saved from date written. 999 signifies permanent retention.
9-22	name	14-character alphanumeric file name
23-24	vv	volume (reel) number (01-99)
25-30	mmddyy	creation date; mm month, dd day, yy year.
31-32	ee	edition number (01-99 or blank)
33-80	user's information	may be referenced by the user with RDLABEL, WRLABEL.

5.1 ASSIGNMENT

Assignment may be defined as the association of a logical unit number with a particular file of information; this association takes place during execution. A forward motion request (read and write), the ASSIGN request from a background program, and certain operator messages cause assignment. EQUIP and LABEL statements provide information necessary to make the assignment, but do not cause assignment. These are three types of assignments: read, write, and unit assignments.

Read Assignment

A read assignment procedure is executed when a unit is unassigned and a READ, REOT, RDLABEL, or SKIP request is issued. It may also be executed when a unit is unassigned, an end-of-volume condition exists, and a READ or SKIP request is given. With READ or SKIP the old assignment is released, and if a standard labeled device is specified, the reel number is increased by one.

An EQUIP statement specifying hardware for a logical unit must appear before that unit may be assigned by a read request from a central program. Similarly a background program must issue an assign request before reading.

Units outside the range 1-49, 71-79 have non-standard labels even if the device is labeled.

Write Assignment

A write assignment procedure is executed when a unit is unassigned and a WRITE, WEOT, MARKEF, WRLABEL, or ERASE request is issued. It will also be executed when an ASSIGN request specifies a hardware mnemonic without ordinal, or when a unit is assigned and an end-of-volume condition exists when a WRITE request is issued. The latter condition can arise only on units 1-49, 71-79.

Unit Assignment

A unit assignment is executed either as a result of an operator query or when an ASSIGN request of the hardware-plus-ordinal form is issued.

5.1.1 UNLABELED DEVICE

A write assignment for a specified type of hardware will assign any device which:

1. is of the specified hardware type
2. is currently unassigned
3. is ready when the status is checked

A read assignment of a specified type of hardware consults the operator with a message; the operator assignment will be made only if the three conditions stated above are met.

5.1.2 LABELED DEVICE

Standard Labels

A write assignment on a standard labeled device is made on the first unit which:

1. is of the specified hardware type
2. is currently unassigned
3. is ready when status is checked
4. contains (a) a label with expired retention code; that is, the creation date plus retention code occurred prior to the day assignment is requested or (b) no recognizable label
5. contains a write enable ring if magnetic tape

A read assignment on a standard labeled device is made on the first unit which:

1. is of the specified hardware type
2. is currently unassigned
3. is ready when status is checked
4. has the required label

Label checking is performed as follows:

a. If an EQUIP statement or LABEL request specifies a logical unit number, the logical unit number field on the label must contain the proper unit number.

   Example:

   ```
   EQUIP, 5 = MT or
   EQUIP, 5 = (*02)
   ```

   In the first example, the proper unit number is 05, in the second, 02.

b. If the label text specifies a name, the 14 characters of the name must agree.

c. If the requested reel number is zero, the unit satisfying (a) or (b) with the lowest reel number is selected.

d. If the requested reel number is non-zero, the reel number field must agree.

e. If the requested edition number is zero, the first volume encountered which satisfies the above conditions is selected.

f. If the requested edition number is non-zero, the edition number field must agree.

g. If the creation date on the EQUIP or LABEL formats is zero, the first volume encountered which satisfies the above is selected.

h. If the creation date is non-zero, the date field must agree.

If a labeled device is declared in either an EQUIP statement or a LABEL request to contain a non-standard label, a read assignment is treated the same as for an unlabeled device. A write statement, however, is treated the same as for a labeled device.

5.2 RELEASE PROCEDURES

When an assignment is released, the file is no longer needed and has been returned to the system for disposition. A release procedure is executed as follows:

Explicit Requests

When RELEASE, UNLOAD, ENTER, and LABEL requests are issued, release procedures are executed on a previously assigned unit. RELEASE and UNLOAD may modify the procedure.

Program Termination

When a program terminates, all units assigned to it are released.

Loading Termination

At the conclusion of a central program loading operation which uses unit 69, unit 69 is released.

Phase Termination

At the conclusion of a central program phase, the units in the range 50-59 are released.

Continuation Volumes

When an end-of-volume condition exists for a central program unit, and a continuation volume is to be assigned, the old volume is released.

The action taken by Drum SCOPE depends upon the conditions stated below:

Blocked Unit--No disposition

When a blocked unit is released, and no disposition is specified (DISPOSE request) the information in the file is discarded. For the drum, the blocks which contained the information are made available. For tape, the tape is moved forward in search of the next file; if there is none, the tape is unloaded if a write ring is absent; otherwise, it will be blank labeled.

Blocked Unit--Disposition specified

When a blocked unit with a non-blank disposition is released, the information currently contained in the file is saved and becomes a drum unit of type equal to the current disposition. Thus the data may be found by a background program via an ASSIGN request. A tape is rewound to beginning of the file and becomes a "drum" unit of the disposition type. The file may not be disturbed until the data has been processed and released with a blank disposition.

Standard Labeled Device

When a standard labeled unit is released from a central program, the information will be saved if there is no ring (tapes only) or the retention cycle has not yet expired. Tapes will be unloaded unless overridden by the RELEASE request. If there is a ring and the retention cycle has expired, the unit becomes available and the data is lost.

Non-standard Labeled Devices

The unit is unloaded when released.

Labeled Devices--Background Assignments

When a labeled device assigned to a background program is released, the device is not unloaded; it is rewound and becomes unassigned.

Unlabeled Devices

When an unlabeled device is released, it is unloaded. The interpretation of unloading is hardware dependent and may in some cases, such as the printer, produce no external operation.

DEBUGGING AIDS 6

Selected areas of storage may be dumped each time a particular instruction is encountered (SNAP) or before execution of each jump instruction in a designated area (TRACE). Recovery dumps may be designated for abnormal termination; and a memory map, giving a listing of absolute addresses assigned to the program by the loader, may be obtained.

SNAP and TRACE dumps consist of a console scoop and a storage dump. The A and Q registers are printed in the mode requested. The index registers, bounds register and D register are printed in octal. The interrupt register and interrupt mask register are printed in binary.

Each printed line contains an absolute octal address, an octal address relative to the name in the first word address (fwa), and four to ten computer words, depending upon the mode requested. One or more lines of identical words are omitted.

6.1 SNAP DUMP

SNAP dumps are periodic dumps of specified areas. The programmer specifies the instruction address where the dump request is executed, the frequency, and the areas to be dumped. Drum SCOPE replaces the instructions at the dump addresses with jumps to the SNAP routine. The SNAP routine dumps the specified areas onto OUT, executes the instructions originally at the dump address, and returns control to the program.

More than one snap dump may be specified for an address and any number of addresses may produce snaps. After the last snap is produced, normal program operation resumes. The only restrictions on the number of snap dumps are the print request limit in the RUN statement and the amount of available storage remaining after the program is loaded.

$^{7}_{9}$SNAP, a, fwa, lwa, f, d_1, d_2, d_3, id

a the program address where the dump is initiated. The program address may be a program name or an entry point name plus or minus an octal displacement, p±n; p is an entry point or program name and n is an octal number. If a program name is used, the program address will be the first location in the program. If an entry point name is used, the program address will be the entry point location. If the program and entry point names are identical, the program name will be used.

fwa, lwa — The first word address and the last word address of the area to be dumped may be:

1) A 6-digit absolute octal location; the leftmost digit is the bank designator. If less than 6 digits are given, bank 0 is assumed.

2) $p_y \pm n_y$, a common block name, an entry point name, or a program name, p_y; plus an octal displacement, n_y, relative to the name. If the program and entry point names are identical, the program name will be used. If a common block name is used, it is enclosed in slashes: /name/. If the block name is blank, two slashes are given: //.

3) $\pm n_x$, an octal displacement relative to the last mentioned entry point, common block, or program name on this SNAP card.

4) blank (fwa and lwa), no area will be snapped; only the console will be snapped. Where fields are omitted, commas must be placed, unless no non-blank fields follow.

f — the format of the dump on the standard output unit is designated by:

O or blank	octal dump
M	octal dump with mnemonic operation codes
I	fixed decimal dump, integer
S	floating decimal dump, single precision
D	floating decimal dump, double precision
B	BCD dump
C	is suffixed to the designator, if a snap of the console is to be included

d_1, d_2, d_3 — control the start, stop, and frequency of the SNAP dump. A dump will be produced at the d_1 encounter of address a, and at every d_3 encounter thereafter until d_2 is reached. If these parameters are blank, a dump is produced at every encounter of the address. If d_3 is blank, a dump is produced at every encounter of the address between d_1 and d_2.

id — is an optional identification for each dump on the standard output unit. It may be up to five alphanumeric characters.

The SNAP cards are placed immediately before the RUN card or the named library card in the execution phase of the job.

Rules for Planting SNAPs

More than one SNAP may be placed at one location. However, since the location will be replaced by a control jump, certain restrictions apply. The user should be aware that the snap location will be executed elsewhere in storage; and, if it is not a jump, control will be returned to the following location. Therefore instructions which depend upon their relative position in the program should not be snapped, for example, RTJ, BRTJ, RXT P,A, SCAN. The location should not be modified or used as data. A TRACE should not have been previously planted in the location. SNAPs may be placed in either enabled code only or disabled code only, not both.

Examples:

$^{7}_{9}$SNAP, ANNA, +5, +30, MC, 1, 100, 5, JACK

The area of storage occupied by locations ANNA+5 through ANNA+30 will be dumped. A dump is produced the first time location ANNA is encountered, and every 5th time thereafter until the 100th time. JACK is printed with each dump as identification. An octal dump with mnemonics and a console scoop are produced.

$^{7}_{9}$SNAP, BETA, +0, +50

The snap is triggered by location BETA. The area of storage to be dumped is BETA through BETA+50. An octal dump will be produced every time BETA is encountered.

$^{7}_{9}$SNAP, BETA, /SAM/, +50, S, 1, 20, 2, JM

The common storage area SAM through SAM+50 is dumped in floating decimal, single precision when BETA is encountered. A dump is produced the first encounter and every alternate encounter until the 20th. JM is the identification.

6.2 TRACE DUMP

The TRACE statement produces a dump whenever jump instructions within a specified range of the program are executed. The dump will be written on the standard output unit in the same format as the SNAP dump.

$^7_9$TRACE, a_1, a_2, fwa, lwa, f, d_1, d_2, d_3, id

a_1 is the first address of the trace area; may be an entry point or program name plus or minus an octal displacement (p±n).

a_2 is the last address of the trace area; a_2 must be greater than a_1. a_2 may be one of the following:

$\pm n_x$ an octal displacement relative to the program entry point or program name, p.

$p_y \pm n_y$ an octal displacement relative to program entry point or program name, p_y.

Any number of ranges (a_1 to a_2) may be specified. When the last trace is produced, normal program operation resumes.

fwa is the first word address of the area to be dumped.

lwa is the last work address of the area to be dumped.

The parameters, fwa and lwa correspond to those of the SNAP statement.

f is the format of the dump on the standard output unit. The various designators are described following the SNAP statement.

d_1 specifies the number of times the area to be traced is passed through before a jump may produce the first dump. d_1 must be less than 4096.

d_2 specifies the last time through the trace area that a jump instruction will cause a dump. d_2 must be less than 4096.

d_3 specifies how often tracing occurs when passing through the trace area. d_3 must be less than 4096.

The area is traced at the d_1 encounter of a_1, and at every d_3 encounter thereafter until d_2 is reached. During tracing, the counter is not incremented until a_2 is encountered; jumps to a_1 in TRACE mode will not affect the count of the trace. If the parameters are blank, tracing is initiated at every encounter of a_1.

The specified area (fwa to lwa) is dumped before the jump instructions are executed. If the jump transfers control to a location outside of the tracing limits, trace output is halted. Upon returning within limits, trace output is resumed, beginning with the instruction which returned within the limits. If a jump instruction is located at a_1, it is traced; at a_2, it is not traced.

The TRACE cards are placed immediately before the RUN statement. If both SNAP and TRACE cards are used, their order is not significant, as long as they are the last cards before the RUN card.

Rules for Using Trace

If two or more tracing limits overlap, the last limit encountered takes effect and any previous limit is canceled. The same rules apply to a_1 and a_2 as apply to snapped locations. A TRACE may not be initiated at a location in which a previous TRACE has been planted. Only enabled code may be traced.

Example:

${}^{7}_{9}$TRACE, ALPHA, BETA, +5, +30, MC, 1, 100, 5, JACK

Jump instructions in the range ALPHA - BETA will be traced. The locations BETA+5 through BETA+30 will be dumped whenever a jump instruction is executed. Tracing will begin with the first encounter of ALPHA and every fifth encounter until the 100th: BETA must be executed in order to increment this count. An octal dump with mnemonics and a console scoop will be given. JACK is written as identification on the standard output unit.

6.3 POST-MORTEM DUMP

On the RUN control statement (sec. 2.3) a post-mortem dump may be specified for abnormal termination of a program.

Post-mortem dumps are given in octal with mnemonics and have the following form:

Console scoop, if requested

Print lines containing an absolute octal address, an octal address relative to the beginning of the subprogram or common block, and the contents of four words. When a new subprogram or common block is encountered, its name is printed and the relative address reset to zero.

One or more lines of identical words are omitted.

6.4 MEMORY MAP

If a memory map is to be suppressed, this must be indicated in the RUN control statement (sec. 2.7.2). When debugging aids are used, a map is always given. The locations are given as six octal digits, the leftmost designates the bank. The memory map lists the absolute location of the following items:

subprograms

program extension areas

labeled common

numbered common

entry points

LOADER 7

7.1 LOADER FUNCTIONS

The loader performs the following functions:

Loads and links subprograms

Detects errors and provides diagnostics

Patches subprograms and labeled common

Selects banks

Assigns program extension areas

A program may be divided into several subprograms, each separately compiled or assembled. As each subprogram is loaded into storage, the names and locations of all entry points are entered into a symbol table. External symbols are also stored in the symbol table and, as loading progresses, they are linked with corresponding entry points. When a RUN statement is encountered, the loader searches the library for subroutines corresponding to the names of all undefined external symbols. These routines are then loaded and linked by the loader. If external symbols remain which have not been linked to an entry point of a subprogram or library subroutine, a loader diagnostic is written on OUT and the job is terminated.

When loading is completed, control is returned to the calling program. Errors detected during loading are written as diagnostics on OUT. In most cases, the loader continues to process the subprograms to detect additional errors; however, it is possible that new errors will be caused by preceding ones.

The loader provides for patching of subprograms and labeled common through the use of octal correction cards. Instructions which do not fit in a patched area are loaded into a program extension area assigned by the loader.

A subprogram or common block may be assigned to a bank by the programmer through the BANK control card (sec. 7.12) or by the loader. When there is no bank declaration, the loader selects the bank, other than bank 0, into which the subprogram or common block most tightly fits. Bank 0 is assigned only when the other banks cannot provide enough available memory space. After each subprogram or block is loaded, the limits of available memory are reduced for that bank.

Programs and library subroutines are loaded at the high end of available memory in the banks assigned to them. They share both labeled and numbered common.

Labeled common is also assigned storage at the high end of available memory; it may be preset with data. The size of a labeled common block whose name POOLxxx, will be assigned by the loader to equal the current buffer size of the system. This definition will override the one specified by the programmer in his BLOCK request. The buffer size is selected by the installation, between 129 and 511 words.

Numbered common is assigned storage beginning at the lower end of available memory in the assigned bank.

The length of any common block may vary in size from one subprogram to the next as long as it does not exceed the first declared length. The last block of numbered common assigned in each bank may vary in length from one subprogram to the next; the maximum value will define the length of the block.

7.2 CARD FORMAT

The Drum SCOPE loader processes subprograms and subroutines to be executed as a running program under Drum SCOPE control. The loader assumes that the programs contain certain elements that enable it to relocate the coding and tie the subprograms and subroutines together. These elements are contained on binary cards, listed below. Generally, they are produced by COMPASS, FORTRAN, COBOL, and ALGOL and are of no concern to the programmer. It is possible, however, to prepare all cards directly without using an assembler or compiler.

1)	IDC	Subprogram Identification Card
2)	EPT	Entry Point Symbol Table
3)	BCT	Block Common Table
4)	RBD	Relocatable Binary Subprogram Deck
5)	EXT	External Symbol Table
6)	LAT	Linkage Address Table
7)	BRT	Bank Relocation Table
8)	OCC	Octal Correction Card
9)	TRA	Transfer Card
10)	LCC	Loader Control Card

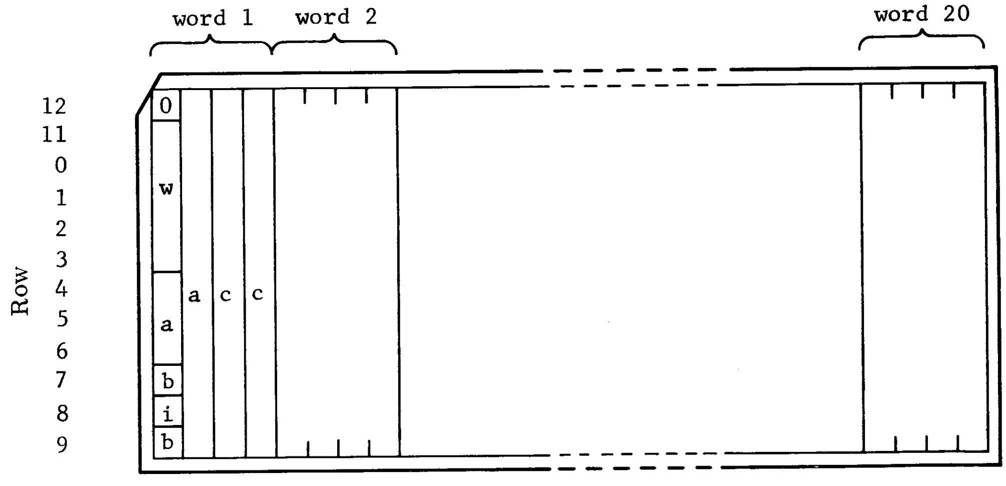
Card types 1-9 must occur in the order listed; however, only the IDC, RBD, and TRA cards are required in every subprogram. Type 10 occurs only between subprograms or ahead of the first subprogram in a series.

Unless identified as octal or BCD, information is assumed to be in binary on cards or in card format. Each column on a card represents 12 bits of a 48-bit computer word. The correspondence between card positions and computer word bit positions for each group of four card columns is shown below:

	Corresponding Bit Position			
Row	Column 1	Column 2	Column 3	Column 4
12	47	35	23	11
11	46	34	22	10
0	45	33	21	9
1	44	32	20	8
⋮	⋮	⋮	⋮	⋮
9	36	24	12	0

All Drum SCOPE loader cards have a 7,9 punch in column 1. On most loader cards the first four columns identify the type to provide a means of checking its contents. Cards with bit 47 (row 12 of column 1) punched are bypassed.

The format of the first four columns (word 1) is shown below:



Row 12 of column 1 must be zero (no punch)

w	5-bit word count identifies the card type

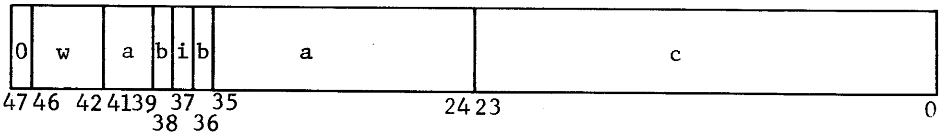
a	15-bit address or sequence number

i	checksum indicator; 1 (punch) ignore checksum,
	0 (no punch) check checksum

c	checksum

b	7, 9 punch in column 1 indicates binary card; required for all loader cards.

Bit structure of first computer word:



7.3 IDC CARD

The identification card names the subprogram which follows it and provides information about the subprogram to the loader.

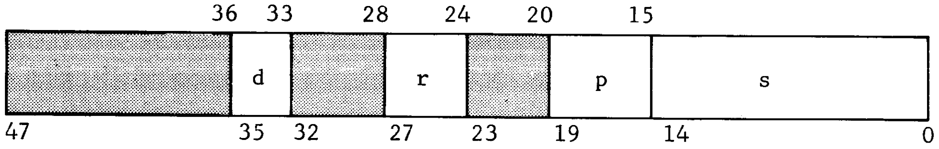
Word 1:

w	31_8

a	0 (no punches)

c	checksum

Word 2:



d	word number, 3-6, of the first word on the subsequent cards which contain data to be loaded. Data begins in the (d+1) word of all RBD cards.

r	length, 2 to 8, of the relocation byte on the RBD cards.

p	number of relocation bytes, 6 to 24, per word on the RBD cards.
s	length of subprogram, 0 to 77776_8

The unused portions of Word 2 are zero.

Words 3-6:

Name of subprogram in BCD code. The name may be specified in one of two formats:

> If the name is 8 characters or less, the leftmost character of word 3 must be non-numeric. Only word 3 will be used for the name.
>
> If the name is from 9 to 31 characters in length, the leftmost character of word 3 must be numeric, 2-4, specifying the number of words comprising the name.

Words 7-20:

Zero (no punches) unless the subprogram is to be a background program. In this case, word 7 upper address contains the highest logical unit number used and word 7 lower address contains the number of families used by the background program.

7.4 EPT CARD

The entry point symbol table gives the names and addresses of entry points.

Word 1:

w	32_8
a	sequence number, 00000, 00010, 00020, etc.
c	checksum

Words 2-20:

These words define entry points. Each entry point requires from 2-5 whole words; the name of the entry point occupies from 1-4 words (either form given for subprogram name on IDC card), and the address of the entry point in the subprogram occupies positions 14-0 of the last word. Relocatability is indicated by bit 47 of the last word: bit 47 = 0, address is relocatable; bit 47 = 1, address is absolute. The entry point definitions are specified contiguously, and definitions may continue from word 20 of one card to word 2 of the next card. As many EPT cards as are necessary may be specified; however, the cards must be in sequence ("a" field of word 1).

7.5 BCT CARD

The block common table defines the name and length of each common block declared in the subprogram.

Word 1:

w	33_8
a	sequence number, 00000, 00010, 00020, etc.
c	checksum

Word 2:

Zero (no punches)

Words 3, 5, 7, ... 19:	names, in BCD code, of the blocks of common assigned. A name may not contain more than 8 characters; it may be numeric (for numbered common) or alphanumeric (for labeled common).
Words 4, 6, 8, ... 20:	length of the common block named in the preceding word.

Common blocks are data storage areas which are shared by subprograms and library subroutines. Common may be labeled or numbered. For labeled common, the first character must be alphabetic; for numbered common, it must begin with a numeric character.

Labeled common is assigned from high order to low order storage, following the subprogram in which it was first referenced. Numbered common is always assigned to locations in lower storage following Drum SCOPE. Numbered common may not be preset; it may be assigned to the area used by the Drum SCOPE loader and overlays the loader when it is referenced.

As many BCT cards as are necessary may be used; each card but the last must define 9 common blocks; up to 126 blocks may be defined. All BCT cards must be in sequence ("a" field in word 1).

7.6 RBD CARD

The relocatable binary deck contains the data words to be positioned in memory. A relocation increment or decrement is applied to the address portions of each word.

Word 1:

w word count; number of words to be loaded ($1 \leq w \leq 21_8$)

a initial load address (a = first data word address, a + 1 = second data word address, and so forth)

c checksum

Words 2-6, as needed:

These words contain relocation bits arranged in constant length bytes which determine the kind of relocation of the load address (a) and the two address portions of each data word.

The number of words reserved for the relocation bytes depends on the length of each byte, which, in turn, depends on the number of common blocks defined by the subprogram.

No. of common blocks	Relocation byte length in bits	No. of words required for relocation bytes
0	2	2
1-2	3	3
3-6	4	3
7-14	5	4
15-30	6	4
31-62	7	5
63-126	8	5

As many complete bytes as possible are arranged in a word, left justified. The first byte in word 2 designates relocation for the load address. The second byte designates relocation for the left address portion of the first data word; the third, for the right address portion of the first data word, and so forth.

The value of a byte is used to specify the relocation quantity. The leftmost bit specifies the sign of relocation (0 = incrementing, 1 = decrementing). The remaining bits in the byte specify the relocation value, as follows:

0	no relocation
1	program relocation
2	relocation of first declared common block
3	relocation of second declared common block

.
.
.

7.7 EXT CARD

The external symbol table gives the names, in BCD code, of symbols external to the subprogram. These names are entry points to other subprograms and library subroutines.

Word 1:

w	34_8
a	sequence number, 00000, 00010, 00020, etc.
c	checksum

Words 2-20:

External symbol names may be specified in either of the forms described for subprogram names (see IDC card, sec. 7.1.1). The names are arranged contiguously on a card and may be split between successive cards. That is, a name may continue from word 20 of one card to word 2 of the next card. As many EXT cards as are necessary may be used; however, the cards must be in sequence ("a" field of word 1).

7.8 LAT CARD

The linkage address table points to locations within the subprogram at which references to external symbols occur.

Word 1:

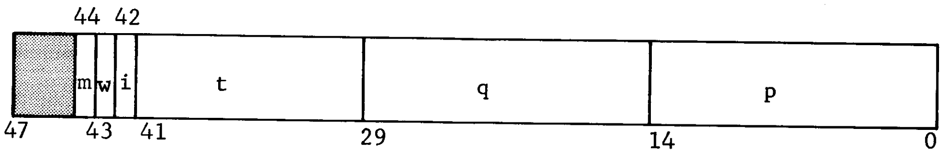
w 35_8

a sequence number, 00000, 00010, 00020, etc.

c checksum

Words 2-20:

Each LAT entry, one word in length, points to a single string of references to one EXT symbol. The LAT entries for each EXT symbol are threaded through the table.



m relocation mode indicator for the external symbol

1 normal (incremented) references
0 complementary (decremented) references

w upper/lower string indicator

0 string of addresses beginning at p refers only to upper addresses

1 string of addresses beginning at p refers only to lower addresses

i 1

t indicates next LAT entry in this thread. The first LAT entry (in word 2) would be referred to as 0001_8, the second as 0002_8, and so forth. If no more LAT entries are needed, t = 0000.

q quantity added to relocated address of the EXT symbol (q may be zero)

p relocatable address of first word in which the external symbol is used. This address contains the location of the next reference, and so on. The last address in the string contains 77777_8.

For each EXT symbol there is at least one LAT entry. If the identical symbol appears in both the upper and lower positions of a word, there are two LAT entries. There are other LAT entries for each modification value q, of the symbol. Finally, there are separate entries for each q-valued portion of the symbol for either value of the relocation mode indicator, m. There may be many LAT entries for a single external symbol.

For example, each of the following references will require two LAT entries if each is referenced from an upper and lower portion of a word; eight entries would result from:

EXT + Q

EXT - Q

-EXT + Q

-EXT - Q

The EXT and LAT tables are arranged in parallel so that the ith LAT begins the series of references to the ith symbol. When more than one LAT entry exists for an EXT symbol, successive entries continue after all initial LAT entries. The related LAT entries are threaded together by the t designation in the entry.

7.9 BRT CARD

The bank relocation table indicates the subprogram locations at which bank designators are dependent upon the banks to which this or other subprograms or common blocks are assigned.

Word 1:

w	36
a	sequence number, 00000, 00010, 00020, etc.
c	checksum

Words 2-20:

The rest of the card contains a threaded list of BRT entries.

The BRT table has three sections; the first parallels the EXT table and each word contains the first two entries for that EXT symbol. If no bank designators depend on that symbol, there are no entries. The second section consists of a pointer to the beginning of the thread of entries for this subprogram and each common block it defines. Section three holds the rest of the BRT entries, two per word, to which sections one and two point.

Format of the words in sections one and three:

	42	30	15	
y	i	t	a_1	a_2
47 43		41	29	14 0

y specifies one of the five bank designator positions to be relocated within the word at a_1 and a_2. The coding of the designator position is as follows:

Code	Bit Positions of Designator
0	none
1	10-8
2	17-15
3	26-24
4	34-32
5	41-39

The value of y may be one of the 30 possibilities, $1\text{-}36_8$:

y = (code for a_1 designator) + 5 times (code for a_2 designator). If a_2 is not present, y is 1 to 5.

i 0

t designates the next BRT word (two entries) in this thread.

a_1 and a_2 are addresses in this subprogram in which a relocatable bank designator appears. If more than one designator in a word is relocatable, each has a BRT entry.

Section two parallels the RFTABLE with four entries per word:

36	24	12	
t_i	t_{i+1}	t_{i+2}	t_{i+3}
47	35	23	11 0

t_i is the ordinal in section three, relative to the beginning of BRT, at which the first two BRT entries corresponding to the ith RFTABLE entry occur. t_1 points to the entries depending on the bank into which the subprogram is loaded.

t_2 through t_{127} (or the last t field) point to the entries depending on the banks to which the corresponding common blocks are assigned. t_0, and any fields not pointing to a section three entry are zero. As many t fields are required as the number of declared common blocks.

7.10 OCC CARD

OCC subprograms may be corrected with octal correction cards. Corrections may be loaded over portions of the subprogram; or, if additional memory is required, into the loader-assigned program extension area. This area is assigned to the highest locations in available memory; its size is limited only by the amount of available storage in the bank receiving the program and corrections.

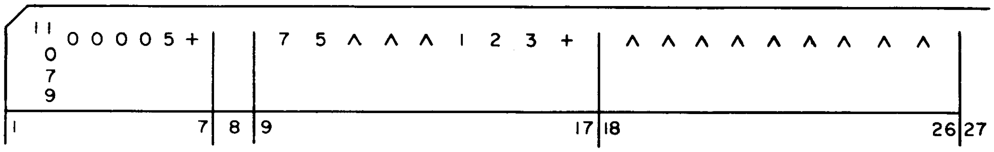
All octal correction cards contain a load address, the contents of one to four computer words, and relocation designators for the address portion of each instruction:

Column	Contents
1	punches in rows 11, 0, 7, and 9
2-6	relocatable load address, aaaaa, for the first correction field on the card
7	relocation factor for address aaaaa
8	blank
9-17	data field 1 - upper instruction or data word to be loaded at the address aaaaa
18-26	data field 2 - lower instruction or data word to be loaded at the address aaaaa
⋮	⋮
63-71	data field 7 - upper instruction or data word to be loaded at the address aaaaa + 3
72-80	data field 8 - lower instruction or data word to be loaded at the address aaaaa + 3

Octal correction cards are placed immediately before the TRA card of the binary subprogram to which they pertain.

Example:

To correct a single instruction in a subprogram:



In the fifth instruction of the subprogram, the upper instruction will be changed to an SLJ to relocatable 123 in the subprogram. The lower instruction will not be affected.

7.10.1 RELOCATION FACTOR

The relocation factor, which follows the load address in card column 7, may be any one of the following:

Factor	Relocation
E	Relative to the first location of the program extension area
+	Relative to the first location of the subprogram
1	Relative to the first location of the first declared common block
2	Relative to the first location of the second declared common block
⋮	⋮ ⋮
9	Relative to the first location of the ninth declared common block
0	Relative to the first location of the tenth declared common block

Only labeled common blocks may be corrected, data cannot be prestored in numbered common blocks. In selecting the correct factor for a common block, however, both numbered and labeled common blocks in the program must be counted in the order in which they are declared. Only the first ten blocks can be corrected.

Examples:

A FORTRAN program contains the following statement:

COMMON /1/A/B3/G, H/B4/F/6/Z/COG/P

To alter data in block B3, the relocation factor 2 would be used. To alter data in block COG, factor 5 would be used. Data cannot be prestored in the numbered common blocks.

A COMPASS program contains the following statements:

```
21        BLOCK 10
          COMMON AFLAGS(5), BFLAGS(5)

H30       BLOCK 200
          COMMON A(10, 10), B(10, 10)

26        BLOCK 3
          COMMON R1, R2, R3

RTABLE    BLOCK 100
          COMMON R(10, 10)
```

To alter data in block H30, the relocation factor 2 would be used. To alter data in block RTABLE, the relocation factor 4 would be used. Data cannot be prestored in blocks 21 and 26 because they are numbered common.

7.10.2 DATA FIELDS

The format of each data field, card columns 9-80, is:

nnnxxxxxi

Data fields 1-8 are loaded into sequential half-words in storage starting with address aaaaa (columns 2-6).

nnn the upper 9 binary digits of an instruction or data word.

xxxxx the address of an instruction, or lower 15 binary digits of a data word.

i the relocation factor for the address, xxxxx. Any of the factors in the relocation list may be used with two additions:

Factor	Relocation
blank	no relocation
-	Relative to the complement of the first address of the subprogram.

Since program instructions may refer to both numbered and labeled common, the value of the relocation designator, i, must be determined by counting each declared common block -- both labeled and numbered -- up to the one to which the address refers.

Blanks in the nnn and xxxxx fields are converted to zeros; if the entire field is blank, the releated portion of storage is not altered.

Example of two cards in an octal correction deck.

	11 0 700001+ 9	b	bbbbbbbbb	75400002E	75bbb123+	bbbbbbbbb	
card column	1 7	8	9 17	18 26	27 35	36 44	45

	11 0 7 9 00002E	b	75077777b	125001045	75000002E	bbbbbbbbb	
card column	1 7	8	9 17	18 26	27 35	36 44	45

In the subprogram:

at location 00001+ UI is unchanged because of the blanks

LI contains a return jump to the 3rd word (00002E) of the program extension area

at location 00002+ UI contains an SLJ to relocatable 123 in the subprogram (blanks fill to zeros).

LI is unchanged, because it is blank.

In the program extension area:

at location 00002E UI contains an SLJ **. The blank designator indicates no relocation.

LI contains a LDA with the contents of relocatable 104 of fifth declared common block.

at location 00003E UI contains an SLJ to 00002E, the third location of the program extension area, to exit back to the program.

LI is unchanged.

The program extension area is allocated 4 words, 0, 1, 2, and 3.

7.11 TRA CARD

The transfer card signals the end of each subprogram and may also specify the name of the entry point to which control is to be given when the entire program has been loaded.

Word 1:

w	37_8
a	0
c	checksum

Word 2: 48 bit sum of words 1 of the preceding deck, including the TRA card.

If an entry point name is used, it is specified in Hollerith, beginning in column 9. When there is no transfer name, columns 9 through 80 must be blank.

Usally only one TRA card in a program specifies an entry point transfer name. Control is given to that address when the program is run. However, a second transfer address may be specified in a later subprogram, or more typically, in a library subroutine. When the program is run, control is given to the second address; the first address is noted in the A register, bits 41-24.

7.12 LCC CARDS

Loader Control Cards permit the programmer to overlay subprograms, correct library routines, and declare at load time the memory banks to which subprograms and common blocks are to be assigned.

$\overset{-}{0}$
${}^{7}_{9}$BANK, (b_1), ..., name$_1$, ... (b_2) ..., name$_k$, ...

or

$\overset{-}{0}$
${}^{7}_{9}$BANK, (m), sym$_1$, sym$_2$ ...

$\overset{-}{0}$
${}^{7}_{9}$MAIN, u

$\overset{-}{0}$
${}^{7}_{9}$OVERLAY, u, 0

$\overset{-}{0}$
${}^{7}_{9}$SEGMENT, u, n

Described in Chapter 8.

7.12.1 BANK CARD

Option 1

The programmer may specify a particular bank for each subprogram and common block; he may also specify that particular subprograms and common blocks go into the same bank.

11
0
${}^{7}_{9}$BANK, (b_1), ..., name$_i$, ..., (b_2), ..., name$_k$, ...

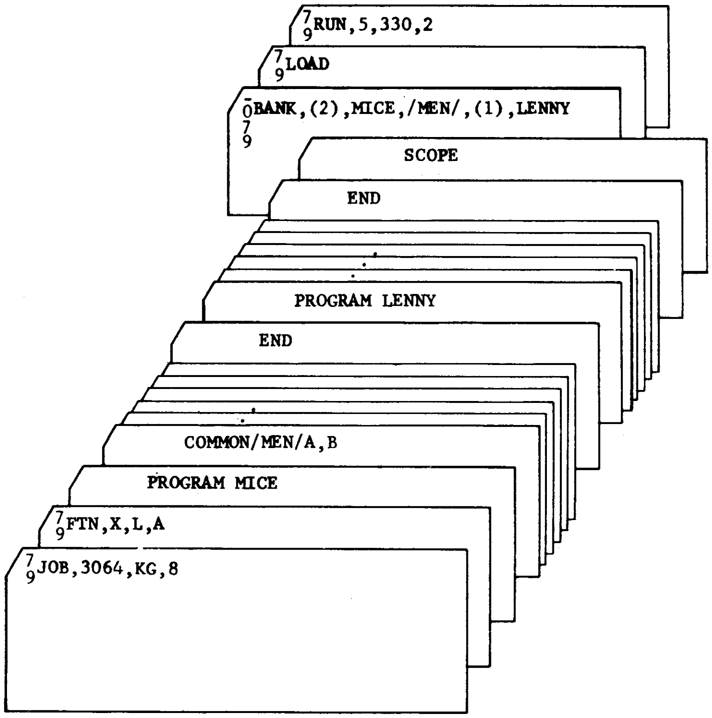
b a bank number (0-7), an entry point, or a common block name.

name an entry point, program, or common block name. A common block name is enclosed in slashes.

If b is an entry point or common block name, the names which follow it are allocated to the same bank as the entry point or common block name, and the loader places the subprograms in the bank having the largest amount of available storage, other than bank zero. If there are several entry points in a subprogram, only one of these need appear in the BANK statement.

Programs compiled or assembled by systems such as FORTRAN, COMPASS, ALGOL, must have provisions for bank relocation before they may appear in a BANK statement.

Example:



```
7
9RUN,5,330,2
7
9LOAD
0BANK,(2),MICE,/MEN/,(1),LENNY
7
9
SCOPE
END
.
.
PROGRAM LENNY
END
.
.
COMMON/MEN/A,B
PROGRAM MICE
7
9FTN,X,L,A
7
9JOB,3064,KG,8
```

Two FORTRAN subprograms are to be compiled and written on the load-and-go unit. The BANK statement precedes the LOAD statement. Subprogram MICE and the common block MEN are to be placed in bank 2 and subprogram LENNY is to be placed in bank 1.

Option 2

Various combinations of subprograms or common blocks may be forced into a particular bank.

11
0
7
$_9\text{BANK}, (\text{m}_1), \text{sym}_1, \text{sym}_2, \ldots, (\text{m}_2), \text{sym}_3, \text{sym}_4, \ldots$

m_i is a bank number, 0-7

sym_i may be the following designators:

SP. = subprograms
NC. = numbered common } on binary input unit
LC. = labeled common

LSP. = library subprograms
LNC. = library numbered common } from library subroutines
LLC. = library labeled common

APC. = SP. + NC. + LC.

ALC. = LST. + LNC. + LLC.

ALL. = APC. + ALC.

The designated subprograms and common blocks will be allocated to the specified bank. These declarations apply only to subprograms or common blocks for which no previous bank declaration defining a unique bank has been given.

Examples:

```
11
 0
 7
 9BANK, (0), APC., LSP.
```

The succeeding subprograms, labeled, and numbered common, read from the binary input unit and library subprograms will be stored in bank 0. Numbered and labeled common blocks from the library are dynamically assigned by the loader.

```
11
 0
 7
 9BANK, (A), B
```

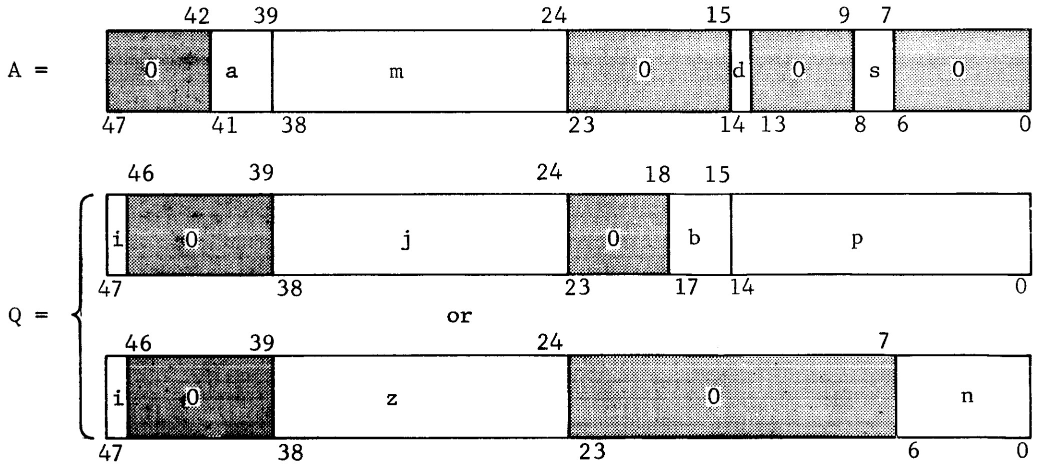
```
11
 0
 7
 9BANK, (0), A
```

```
11
 0
 7
 9BANK, (1), ALL.
```

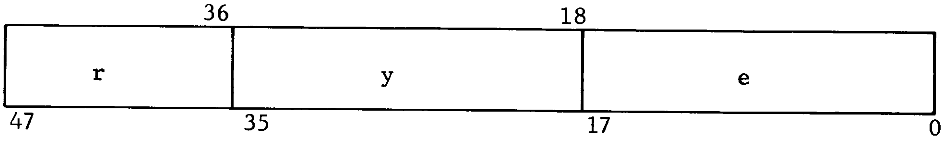
Subprograms containing entry points A and B will be forced into bank 0 by the first two bank statements. The remaining subprograms ·and common blocks will be loaded into bank 1.

7.13 LOADER CALLS

When the LOADER request is made (sec. 4.3), the A and Q registers must contain certain parameters:



am an 18-bit address, or zero, specifying a location for preset entry points which have been defined by the calling program. The contents of the location specified by am are in the following format:



r value specifying the number of entry points preset into the entry point symbol table ($r < 4096$).

y first word address (18 bits) of the list of entry point names. The names must be in contiguous storage locations and may extend to 31 characters per name (either form given for subprogram names on IDC card).

e first word address (18 bits) of the list of entry point addresses. This list must be in contiguous storage locations, with one address contained in the lower 18 bits of each word. The number of entry point addresses must equal the number of entry point names, both of which must equal r.

d is a map parameter:

0 no map after loading
1 map after loading

s specifies the kind of loading operation:

00 load library programs from library

01 load program from n (lower Q is used)

10 load library program from library, and octal corrections to it from INP (upper Q is used)

11 complete loading operation after interruption - (ignore remainder of A and Q)

j the number of names in the list starting at bp

bp an 18-bit address specifying the beginning of a list of library subroutine entry point names to be loaded from the library. A name may be in either form allowed for the subprogram name on the IDC card.

z the location, in bank zero, of the first binary card image of the program to be loaded. The rest of the cards are found on n. If $z = 0$, the first card is also found on n.

n designates the logical unit from which binary cards, or images of cards, one per record, are to be loaded.

The unused portions of A and Q must be zero.

i non-zero specifies that the loader is to initialize its tables to start a new loading operation. Set only if last loading operation was not completed and loader has not been reloaded.

The four loading operations, keyed by the s parameter in the A register, are as follows:

s=00

Load library subroutines called by the entry point names at bp in the order listed. No externally referenced programs will be loaded. When the loader returns to the calling program, bit 47 of the A register will be 1; the rest of A has no meaning.

Q upper will contain the first location in bank zero of the address list. This list gives the relocated addresses of the entry point names found at bp. The list is in contiguous storage locations, with one address contained in the lower 18 bits of each word. Loading may be continued with further s=00, s=01, or s=10 calls. Loading may be completed with an s=11 call.

s=01

Load subprograms from unit n until an end-of-file or Drum SCOPE control card is encountered. If z is non-zero, it gives the location of the first card image in bank 0. An entry point symbol table is preset if am is non-zero. If an end-of-file is encountered on n, control is returned to the calling program with bit 47 of the A register 1 and bits 14-00 zero. If a Drum SCOPE control card is encountered, bit 47 of the A register is 1 and bits 14-00 specify the location in bank 0 of the card image of the control card. In neither case are library subroutines loaded. Loading may be continued with an s=01, s=00, or s=10 call. Loading is completed with an s=11 call.

In the event that this s=01 call is the first call to the loader, and the first card encountered by the loader is a MAIN control card, the loader will completely process the overlay program and return with bit 47 of the A register equal to zero. Bits 17-00 will contain the last transfer address, bits 41-24 will contain the first transfer address if two were encountered, else zero. Q upper will contain zero if loading terminated with an end-of-file card, or it will contain the location of the card if loading terminated with a SCOPE control card. Q lower is non-zero if errors were encountered while loading. The transfer addresses above refer to the main partition only.

s=10

This is the same as the loading operation with s=00 except that j is assumed 1 and OCC cards are accepted. After loading the named program, the loader loads OCC cards from INP until it encounters a TRA card or a SCOPE control card. If a SCOPE control card is read, the loader returns with bit 47 of the A register 1 and bits 14-00 specifying the location in bank 0 of the card image. Loading may be continued with s=10 or s=00; it will be completed with an s=11 call.

s=11

Either load a library subroutine and all externally referenced programs, or complete loading after an s=00, s=01, or s=10 call. Return is made with the transfer address in A lower if one named transfer card was found; if two were found, the first transfer address is in A upper and the second in A lower.

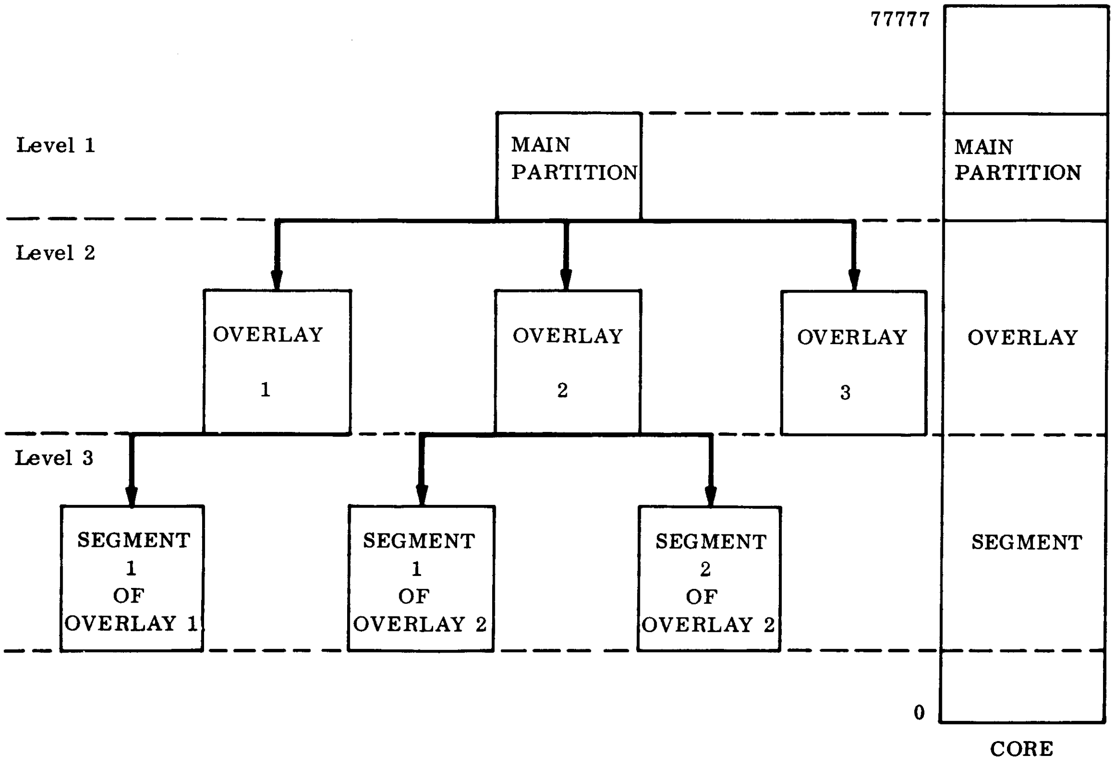
Q upper contains the first location of the address list in bank zero. This list gives the relocated addresses of the entry point names found at bp. The list is in contiguous storage locations, with one address in the lower 18 bits of each word. If more than one address list is specified on s=00 and/or s=10 calls, the last one encountered will be used. In all cases, bit 47 of the A register is 0, and Q lower contains the number of errors encountered.

OVERLAY PROCESSING 8

A program which is too large to fit completely into available storage at one time can be divided into independent partitions to be called and executed as needed; a partition is an absolute binary record previously linked by the loader. There can be only one partition of highest functional level; it is called the main partition, and resides permanently in core memory during execution of the program. The main partition may have any number of associated overlay partitions, and each overlay may contain any number of segments. However, only one overlay and one segment can reside in memory with the main partition at one time.

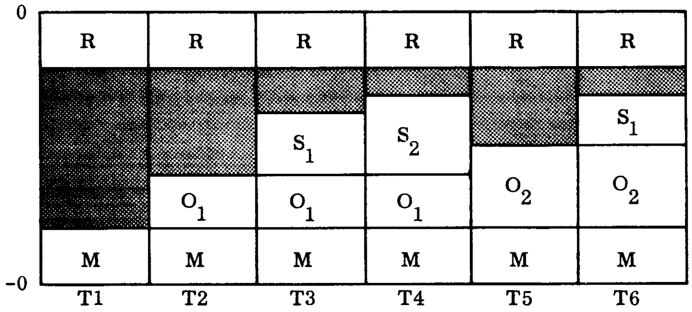
For example, in the diagram below, a partition on one level (indicated by broken lines) may not call (load into core) any other partition on that same level or any higher level; OVERLAY 1 may not call OVERLAY 2, OVERLAY 3, nor the MAIN PARTITION. A partition may call any other partition on a lower level if these two partitions are connected by a line of call (shown as heavy solid arrows).

Interrelationship of Overlay Partitions



Each partition consists of one or more subprograms. An entry is an entry point or common block within a partition. For example, any entry defined at level n may be referenced only by a partition at a subordinate level (n+1 or n+2) which is connected to the partition defining the entry by a line of call. Entries defined in the main partition may be referenced by all overlays and segments. Entry points defined in a particular overlay may be referenced by all segments associated with that overlay. Within a segment, entries may be referenced only by that segment.

STORAGE DURING OVERLAY
PROCESSING AND EXECUTION



R = Resident
M = Main
O_i = Overlay
S_i = Segment
Tn = Time n

8.1 OPERATING PRINCIPLE

The loader is called to create a partition by a partition control card (MAIN OVERLAY, or SEGMENT). This card specifies the type of partition and the unit on which it is to be placed. When the loader encounters such a card on a particular unit, it loads and links all relocatable binary subprograms from that unit and the library until the next control card is encountered. The loader then writes the absolute partition record on the specified paritition unit.

In a program containing overlays, the loader control statements, MAIN, OVERLAY, and SEGMENT, precede the relocatable binary subprograms which comprise the partitions. These statements are read by the loader as it loads from the load-and-go unit or from INP. Each subprogram or common block is assigned to the bank of core into which it can fit most tightly (bank 0 is used only if there is no room elsewhere) and the limits of memory are adjusted to reflect the assignment. Bank assignments made with the BANK card override the dynamic bank assignment. OCC cards are permitted unless the program is on the library. Each partition must have one transfer address.

Overlay processing loads the relocatable binary subprograms into storage and writes each partition as a separate record in absolute binary on a partition unit. This unit is then called in sections for execution. The absolute records do not require the relocatable binary loader to perform the usual relocating and linking functions. Debugging aids (SNAP and TRACE) cannot be used.

Initially, control is transferred to the main partition which resides in storage continuously; it in turn calls the overlays when they are needed during program execution. Segments may be called either by the main partition or by an overlay. FORTRAN and COMPASS subroutines are available to call overlays and segments during execution; calls to these subroutines must be included in the source subprograms. After a partition unit has been created, it is executed with the SCOPE control statement, LOADMAIN. All partition control cards are designated by 11, 0, 7, or 9 punched in column 1.

8.2 CONTROL STATEMENT CARDS

Each partition deck begins with a control statement which specifies the type of partition, the logical unit on which it should be written, and applicable overlay and segment numbers.

The following control statements are used:

MAIN   This statement defines the main partition; it may not be omitted.

```
11
0MAIN,u
7
9
```

u   logical unit number on which the partition is to be written, (1-49). The logical unit numbers for the main partition, overlays, and segments may be 70 if the program is to be included in the production file of the library tape as a production code (PROGRAM card, section 9.2.5).

The MAIN control statement must precede the object subprograms which comprise the main partition.

When overlay files are being prepared, the main partition remains in storage. After all partitions have been written, the main partition may be executed. The main subprograms must precede the first overlay. Portions of the main partition may be assigned to several banks with the BANK statement.

OVERLAY

This statement defines an overlay partition.

```
 11
  0OVERLAY,u,n
  7
  9
```

u   logical unit number on which the partition is to be written, (1-49). The logical unit numbers for the main partition, overlays, and segments may be 70 if the program is included in the production file of the library tape as a production code.

n   overlay number; must begin with 1 and increase consecutively for subsequent overlays.

The OVERLAY control statement precedes the object subprograms which comprise the overlay; when it is encountered, SCOPE creates an overlay partition and writes this section in absolute binary on the partition unit specified.

Calling sequences to call each overlay into storage must be included in the main partition. The overlay may be assigned to several banks with the BANK statement.

SEGMENT

This statement defines a segment partition.

```
 11
  0SEGMENT,u,n
  7
  9
```

u   logical unit number, 1-49, on which the partition is to be written; segments must be written on the same unit as the overlay with which they are associated.

n   segment number; within each overlay, segment numbers must begin with 1 and increase consecutively.

The SEGMENT control statement precedes the object subprograms which comprise the segment; when it is encountered, SCOPE creates a partition and writes it in absolute binary on the overlay unit specified.

Calling sequences for each segment must be included in the main or overlay partition. The segment may be assigned to several banks with the BANK statement.

The following rules will aid in partitioning a program into overlays and segments:

1. Numbered and labeled common blocks and all entry points declared in the main subprograms may be referenced by an overlay and any segment.
2. Numbered and labeled common blocks and all entry points declared in an overlay may be referenced by that overlay and its associated segments, but not by the main subprograms, another overlay, or segments contained in another overlay.
3. Numbered and labeled common blocks and all entry points declared in a segment may be referenced by that segment only.
4. The first overlay card must be preceded by the main program.
5. The overlay numbers must start with one and continue in consecutive order for all overlays written.
6. The segment numbers, within an overlay, must start with one and continue in consecutive order.
7. Only four overlay units may be written. If these units are tape, no overlay may occupy more than one reel.
8. Each overlay and segment must have a single named transfer point.
9. All segments within a particular overlay must immediately follow that overlay.

8.3 PARTITION UNIT PREPARATION

A partition unit is composed of absolute binary records followed by two end-of-file records. Each record constitutes a main subprogram, overlay, or segment. These records may be composed of many subprograms. In absolute form, a particular main partition, overlay, or segment may occupy any number of banks.

The overlay unit is prepared under direction of loader control statement cards. Each control statement specifies the logical unit on which the resulting absolute binary program will be stored. SCOPE control statements may also be included if compilation or assembly must occur before the overlay unit is created.

Each main partition, overlay, or segment must contain only one transfer address; the main partition may contain two transfer addresses.

8.3.1 DECK STRUCTURE

In the preparation of a partition unit, deck structure differs according to the type of input. Relocatable binary subprograms, preceded on INP by the loader control statements MAIN, OVERLAY, and SEGMENT are loaded and then written on the overlay unit in absolute binary. Source language subprograms must be compiled or assembled onto a load-and-go tape before they can undergo overlay processing; the loader control statements MAIN, OVERLAY, SEGMENT, preceding each portion coded in source language, must be transferred to the load-and-go unit. The load-and-go unit then becomes input for the overlay unit. Relocatable binary subprograms and source language subprograms may be combined; the source language may be compiled onto a load-and-go unit, and the binary deck may be transferred to this same unit. As each loader control statement is read, it is written on OUT. The section following the loader control statement is loaded into storage, assigned absolute locations and a map is written on OUT showing the absolute locations assigned to each main partition, overlay, and segment.

8.3.2 BANK ASSIGNMENT

Any subprogram within a main partition, overlay, or segment may be assigned to a specified bank. In source language subprograms, a bank may be selected by the appropriate source language pseudo instruction.

If relocatable binary subprograms comprise the input for an overlay tape, they may be assigned to specific banks by the loader control statement:

$\begin{matrix}0\\7\\9\end{matrix}$BANK, (b_1), name$_1$, . . . , (b_k), name$_1$, . . . , name$_n$

The BANK statements must precede the binary subprograms to which they pertain, and follow the OVERLAY or SEGMENT statement to which they pertain. If the relocatable binary subprograms are contained on a load-and-go unit as the result of a compilation/assembly, the BANK statements may not precede the LOAD statement which loads the load-and-go unit for overlay processing.

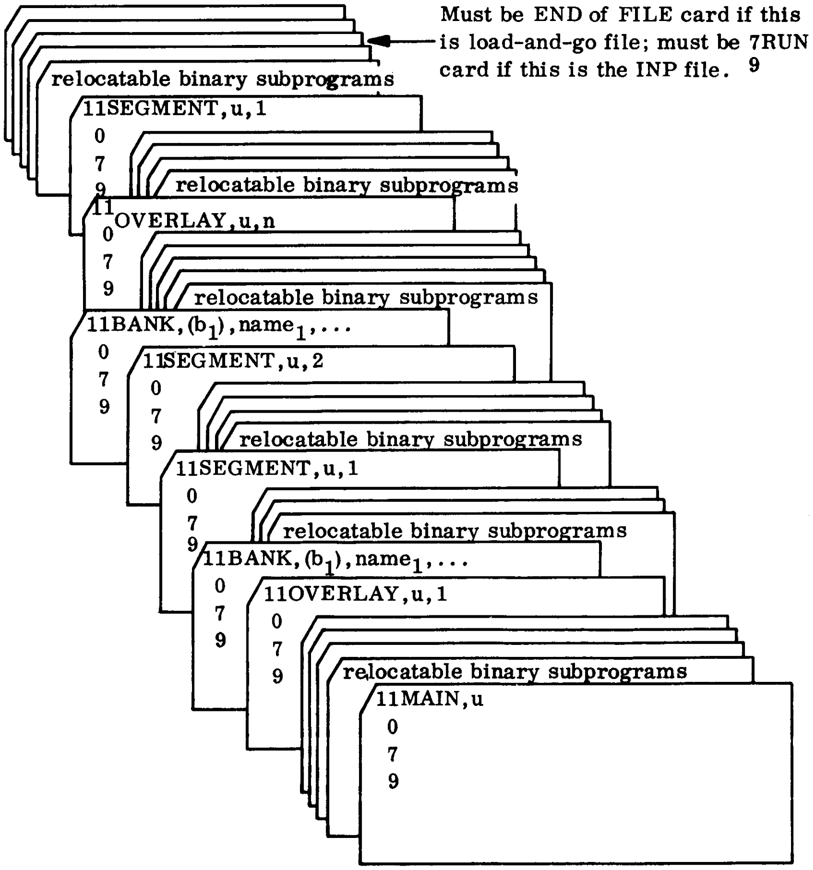
If there are no BANK statements, normal bank assignment occurs.

8.4 LOADING AND EXECUTING PARTITIONS

The preparation and processing of partitions depends on whether the input consists of binary or source language subprograms, or a combination of both.

BINARY SUBPROGRAMS ONLY

To create an overlay unit when the deck consists only of relocatable binary subprograms, each partition must be preceded by a control statement — MAIN, OVERLAY, or SEGMENT. If a subprogram is to be assigned to a specific bank, a BANK statement must be included before the binary subprogram and after the loader control card. The order of subprograms in this example must be followed in all overlay programs.


relocatable binary subprograms
11SEGMENT,u,1
0
7
9
relocatable binary subprograms
11OVERLAY,u,n
0
7
9
relocatable binary subprograms
11BANK,(b1),name1,...
0
7
9
11SEGMENT,u,2
0
7
9
relocatable binary subprograms
11SEGMENT,u,1
0
7
9
relocatable binary subprograms
11BANK,(b1),name1,...
0
7
9
11OVERLAY,u,1
0
7
9
relocatable binary subprograms
11MAIN,u
0
7
9

Must be END of FILE card if this is load-and-go file; must be 7RUN card if this is the INP file. 9

The overlay unit numbers may be the same or they may differ; however, segments must follow the overlay on the same overlay unit. If the information in the preceding example were on a unit other than INP, it could be loaded by a LOAD control statement and processed just as though it were on INP. With the deck on INP, processing proceeds as follows:

1. Relocatable binary subprograms following the MAIN statement and continuing up to the first OVERLAY statement are loaded into storage and linked by the loader. They are then written in absolute binary on partition unit u. The control statement, MAIN, followed by a map of the main section showing the assigned absolute locations, is written on OUT.

2. The binary subprograms following the first OVERLAY statement are loaded into an area beginning with the first location after the main area. They are written in absolute binary on partition unit u. The control statement, OVERLAY, followed by a map of the overlay section showing the assigned absolute locations, is written on OUT. The BANK statement following the OVERLAY statement controls the assignment for subprograms contained in the first overlay.

3. Step two is repeated for each overlay and segment. In storage, the segment begins with the first location after its related overlay. During this process, the main partition remains in storage.

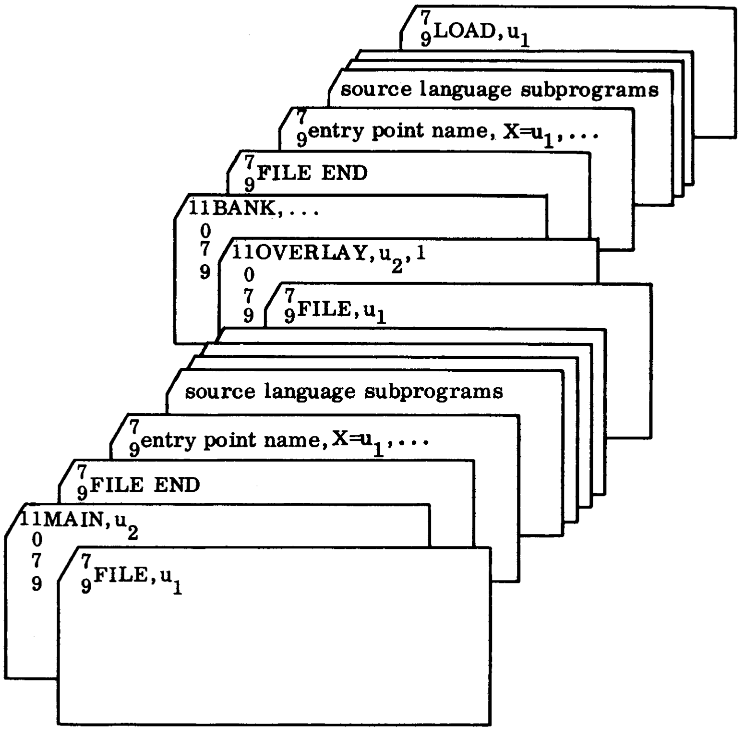
   All overlays are loaded into the same area following main before they are written on the overlay unit. All segments are loaded into storage beginning at the same first location following the associated overlay before they are written.

   The BANK statement following the second SEGMENT statement controls bank assignment for subprograms contained in segment 2 of overlay 1.

4. The end of overlay processing is signaled by an END-OF-FILE record. Upon encountering a RUN statement, execution of the partitioned program begins.

SOURCE LANGUAGE SUBPROGRAMS

If source subprograms comprise the overlay deck, they must be compiled or assembled onto a load-and-go unit. Loading of the load-and-go unit for overlay processing is initiated by a LOAD statement. The loader control statements MAIN, OVERLAY, and SEGMENT, must precede the subprograms on INP and the load-and-go unit just as they would on INP if only binary subprograms were used; they are transferred to the load-and-go unit by FILE, FILE END control sequences. Bank assignment may be included in the source language subprograms; BANK statements may be transferred to the load-and-go unit by FILE, FILE END control sequences. COMPASS and FTN are entry point name statements; they must immediately precede the subprograms to be assembled or compiled. Entry point name statements must follow each FILE, FILE END sequence to return control to the compiler/assembler program. The load-and-go unit, u_1, must not be the same as the partition unit, u_2.



With this deck on INP, processing proceeds as follows:

1. The loader control point statement (MAIN for the first time, OVERLAY or SEGMENT for following instances) is transferred to the load-and-go unit, u_1.

2. The subprograms following the first FILE END statement will be processed by the named library program. The entry point name statement should specify that the binary object subprograms be stored on the load-and-go unit, u_1.

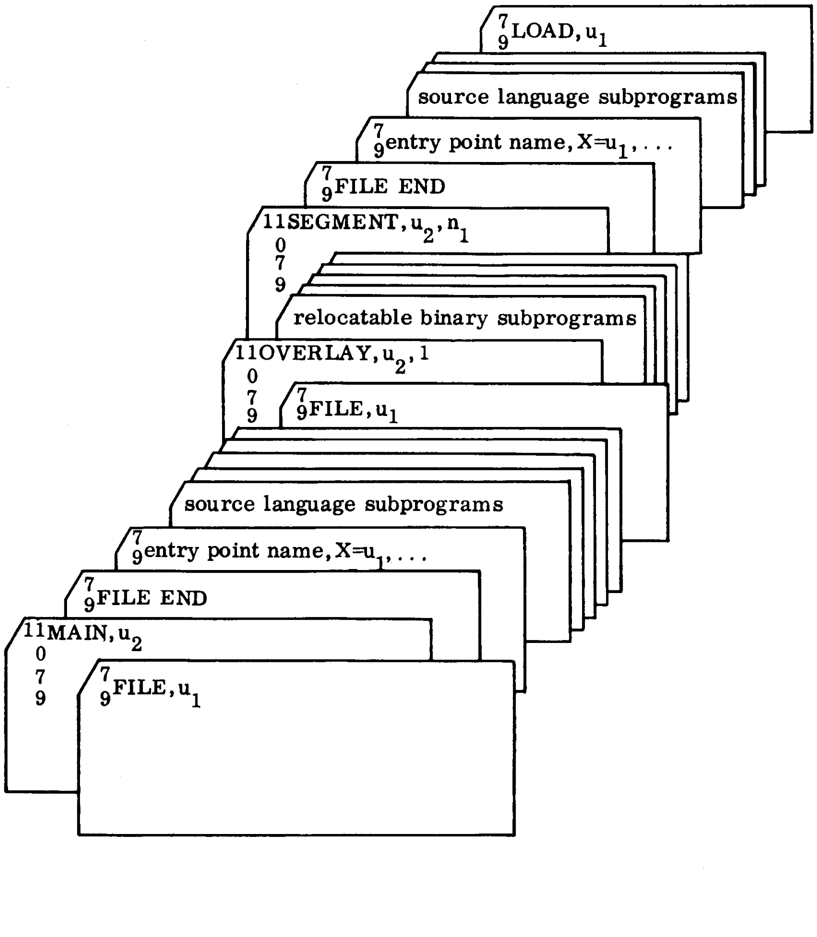
3. Steps 1 and 2 are repeated until the LOAD statement is encountered. At this time, the load-and-go unit, u_1, contains the same deck structure as INP would have contained if binary subprograms alone had been included in the deck, as follows:

```
11
0MAIN,u2
7
9
      relocatable binary subprograms

11
0
7OVERLAY,u2,o
9

11
0
7BANK,...
9
      relocatable binary subprograms
             .
             .
             .
```

4. When the LOAD statement is encountered, the load-and-go unit, u_1, is loaded, and processing is identical to the overlay processing for binary subprograms only. The partitioned program is on logical unit, u_2.

MIXED BINARY AND SOURCE LANGUAGE SUBPROGRAMS

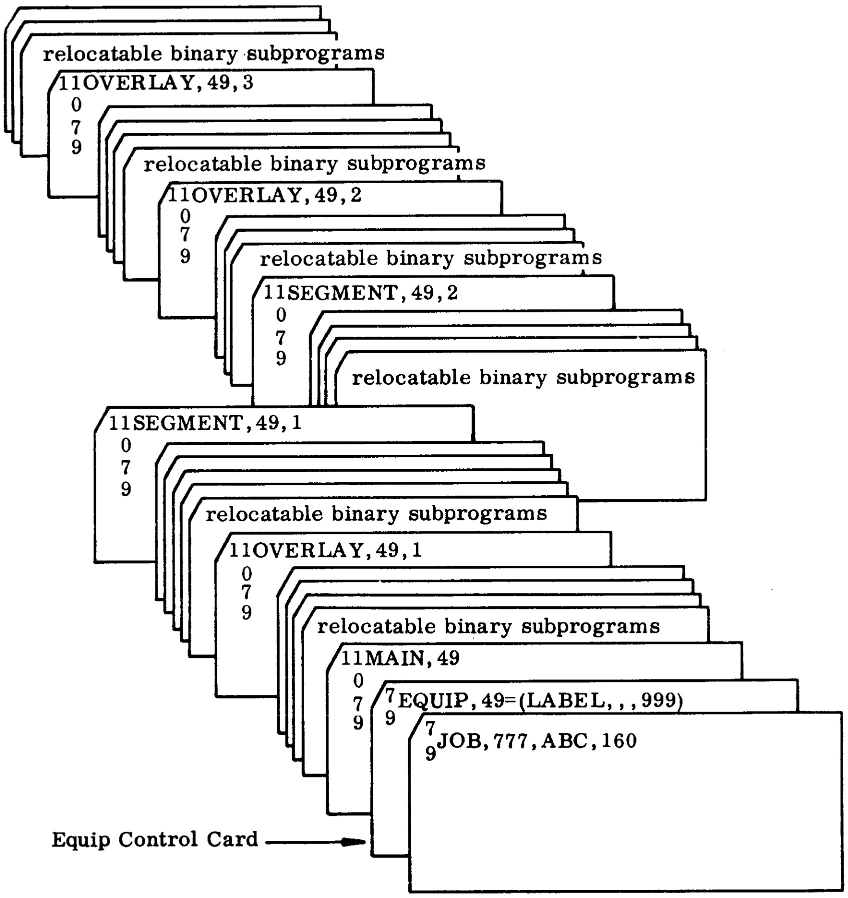
If relocatable binary subprograms are mixed with source language subprograms in an overlay deck, the binary subprograms must be transferred to the load-and-go unit by FILE, FILE END sequences. The order on the load-and-go unit must be the same as that on INP in the previous example. Processing is similar to that for source language subprograms. Source language and binary subprograms may be placed within the same partition.


7 9 LOAD,u_1
source language subprograms
7 9 entry point name,X=u_1,...
7 9 FILE END
11SEGMENT,u_2,n_1 0 7 9
relocatable binary subprograms
11OVERLAY,u_2,1 0 7 9
7 9 FILE,u_1
source language subprograms
7 9 entry point name,X=u_1,...
7 9 FILE END
11MAIN,u_2 0 7 9
7 9 FILE,u_1

8.4.1 LOADING AND EXECUTING THE MAIN PROGRAM

SAVING AN OVERLAY TAPE

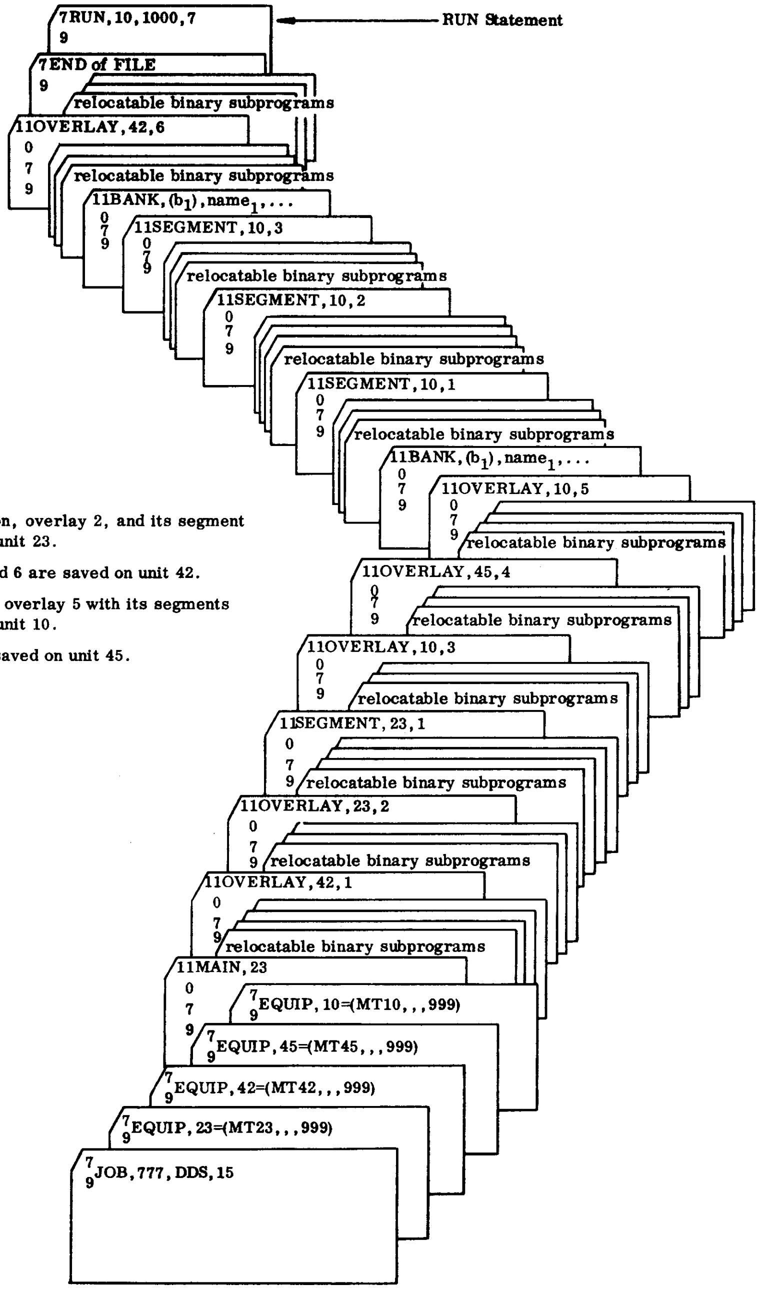
When an overlay tape is being prepared, it may be saved for subsequent executions with an EQUIP control statement. If the overlay tape is to be prepared for subsequent executions only, no RUN control statement is required.


relocatable binary subprograms
11OVERLAY,49,3
0
7
9
relocatable binary subprograms
11OVERLAY,49,2
0
7
9
relocatable binary subprograms
11SEGMENT,49,2
0
7
9
relocatable binary subprograms
11SEGMENT,49,1
0
7
9
relocatable binary subprograms
11OVERLAY,49,1
0
7
9
relocatable binary subprograms
11MAIN,49
0
7
9
7EQUIP,49=(LABEL,,,999)
9
7
9JOB,777,ABC,160
Equip Control Card

EXECUTION IMMEDIATELY AFTER PARTITION UNIT PREPARATION

While the partition units are being prepared, the main partition is retained in storage. To execute the program immediately after the units are ready, a RUN control statement must be placed at the end of the overlay deck or after the LOAD statement on INP. (LOADMAIN is not used.) When the RUN statement is encountered, the units are rewound and control goes to the transfer address in the main section. The main section then calls the overlays and segments from the partition units.

Example 1



```
7RUN,10,1000,7
9
7END of FILE
9
relocatable binary subprograms
11OVERLAY,42,6
0
7
9
relocatable binary subprograms
11BANK,(b1),name1,...
0
7
9
11SEGMENT,10,3
0
7
9
relocatable binary subprograms
11SEGMENT,10,2
0
7
9
relocatable binary subprograms
11SEGMENT,10,1
0
7
9
relocatable binary subprograms
11BANK,(b1),name1,...
0
7
9
11OVERLAY,10,5
0
7
9
relocatable binary subprograms
11OVERLAY,45,4
0
7
9
relocatable binary subprograms
11OVERLAY,10,3
0
7
9
relocatable binary subprograms
11SEGMENT,23,1
0
7
9
relocatable binary subprograms
11OVERLAY,23,2
0
7
9
relocatable binary subprograms
11OVERLAY,42,1
0
7
9
relocatable binary subprograms
11MAIN,23
0
7
9
7
9EQUIP,10=(MT10,,,999)
7
9EQUIP,45=(MT45,,,999)
7
9EQUIP,42=(MT42,,,999)
7
9EQUIP,23=(MT23,,,999)
7
9JOB,777,DDS,15
```

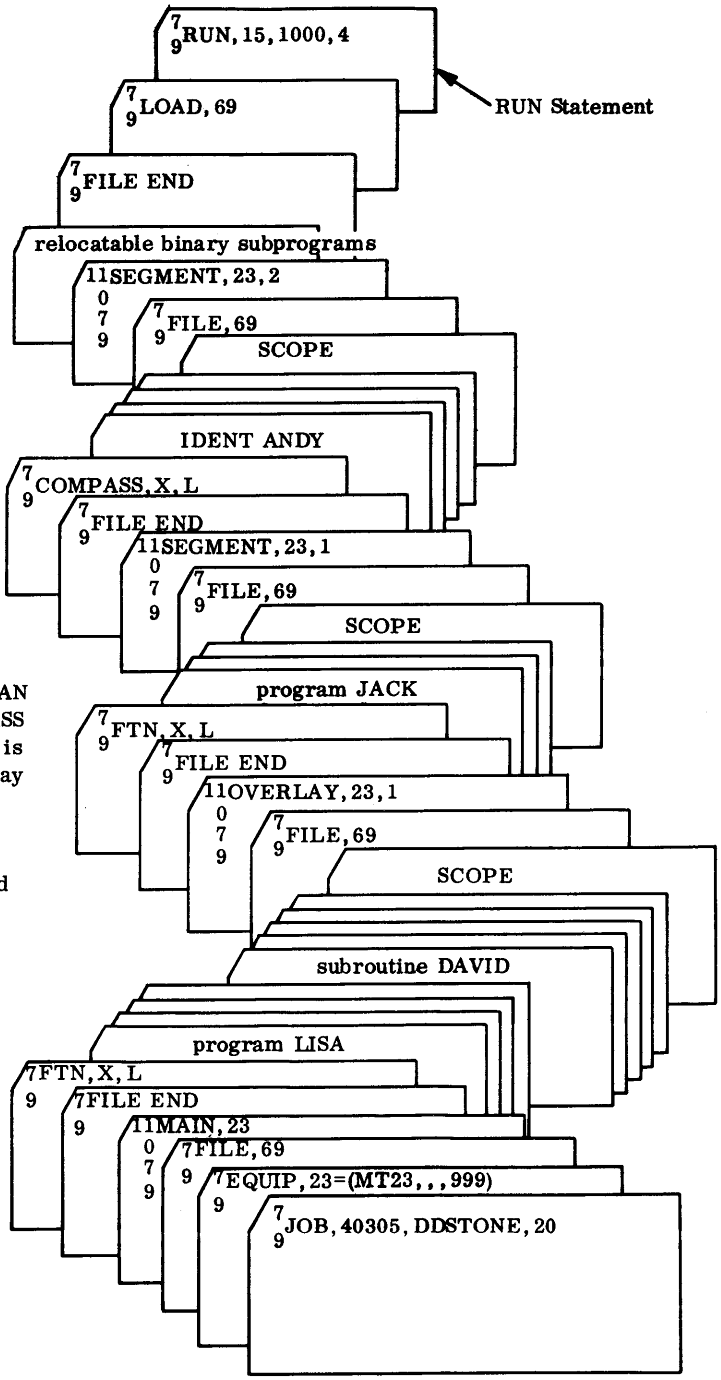
Main partition, overlay 2, and its segment are saved on unit 23.

Overlays 1 and 6 are saved on unit 42.

Overlay 3 and overlay 5 with its segments are saved on unit 10.

Overlay 4 is saved on unit 45.

Example 2



Overlay statements are transferred to load-and-go unit 69 with the relocatable binary output from FORTRAN compilations and a COMPASS assembly. Logical unit 69 is loaded to prepare the overlay tape; an overlay tape is created on logical unit 23. When RUN is encountered overlay tape is rewound and control goes to transfer address in main section.

EXECUTION OF PREVIOUSLY PREPARED OVERLAY TAPE

Once an overlay tape has been prepared, the overlay program in absolute binary can be loaded and executed with the SCOPE control statement, LOADMAIN. The LOADMAIN card is described further in section 2.7.2.

$^{7}_{9}$LOADMAIN,u,t,p,r,d

u unit that contains partitioned program in absolute binary.

t time limit in minutes (optional)

p print limit (optional)

r request for octal dump or console scoop on abnormal termination

non-blank, octal dump of all core

blank, console scoop

d request for octal dump on normal termination

LOADMAIN loads the main partition from the overlay tape on logical unit u and transfers control to the transfer address in the main section. During execution the main partition calls the overlays and segments from the overlay tape. A RUN control statement is not required to execute overlays loaded by LOADMAIN.

The system configuration should be the same as the one under which the overlay tape was prepared. For example, if bank 0 and a DEMAND card are used in preparing the partition tape, the same DEMAND card should be used when loading the overlay tape. If an error occurs while LOADMAIN reads the MAIN program from tape, SCOPE writes a diagnostic on OUT and terminates the job.

Example:

$^{7}_{9}$JOB,7777,ROG,50

$^{7}_{9}$LOADMAIN,25

data to be read by program

⋮

8.4.2 LOADING AND EXECUTING OVERLAYS AND SEGMENTS

Overlays and segments are loaded into memory with the LOVER request. This subroutine operates under the following rules:

1. A LOVER request does not initiate execution of the specified partition; it only locates and loads the partition into core.

2. An overlay may be loaded only by a request from the main partition.

3. A segment may be loaded from the associated overlay or from the main partition if the associated overlay is in core storage. A segment belonging to an overlay not currently in storage cannot be loaded.

4. Overlays may be loaded in any order.

5. Segments within a given overlay may be loaded in any order.

6. A job is abandoned under the following circumstances:

   Irrecoverable parity error is detected when loading a partition

   Logical unit number specified in the request is not 1 to 49, or 70 if the program is included in the production file of the library tape used to load the system

   Specified partition could not be located

   Rules 2-5 violated

7. When an overlay is loaded by the LOVER request, all previous overlays and segments are erased from core. The loading of a segment erases any previous segment from core.

Normally overlays and segments are designed to enter with the bank return jump instruction and to exit via their entry point. However, they may also exit directly to the calling overlay or main program.

Both FORTRAN source programs and COMPASS subprograms may issue calls to LOVER.

FORTRAN CALL

FORTRAN source language subprograms use the following call statement to load and execute overlays and segments. (The overlay or segment call uses LOVER to load overlays and segments.)

$$\text{CALL} \quad \begin{matrix}\text{SEGMENT}\\ \text{OVERLAY}\end{matrix} \quad (o,s,u,d,p_1,\ldots,p_n)$$

SEGMENT loads and executes a segment

OVERLAY loads and executes an overlay

o overlay number, specified for both segment and overlay

s segment number, blank or zero for an overlay.

u logical unit number

d dummy parameter which must be present if any actual parameters appear; otherwise, may be blank. d may also be the literal 8H.RECALL. or a variable containing that literal if the last loaded overlay or segment is tó be recalled. In RECALL mode, the overlay or segment is not reloaded; initialization of variables local to the overlay or segment is the programmer's responsibility. In RECALL mode, the overlay, segment, and unit parameters should appear.

p_i actual parameters to be passed to the overlay or segment routine; may not exceed 59

If o, s, u, or d is blank, a comma must appear; their order is fixed.

One subprogram in each overlay and segment must begin with the FORTRAN statement, PROGRAM name. This statement may contain a maximum of 59 parameters:

PROGRAM name $(p_1,\ldots,p_n)$

name transfer address for overlay or segment

$p_1,\ldots,p_n$ formal parameters; actual parameters in the CALL must correspond to these formal parameters

CALLING SEQUENCE

The following calling sequence is generated during compilation for the CALL statement:

```
         BRTJ    ($) {SEGMENT}
                     {OVERLAY} ,,* .
    +    SLJ     *+m
         0n      DICT.
    +    00  ($) o
         00  ($) s
    +    00  ($) u
         00  ($) d
    +    00  ($) p1
         .       .
         .       .
         .       .
         00  ($) pn
```

In this sequence,

$m = \frac{n+1}{2} + 1$, * + m is the return address.

n = number of parameters specified in FORTRAN CALL statement $(o \ldots p_n)$

The above calling sequence jumps to the OVERLAY or SEGMENT subroutine which passes the parameters o, s, and u to LOVER. LOVER loads the segment or overlay and returns either a loading error code or the transfer address for the overlay or segment loaded.

If no errors occur during loading, the following call to the transfer address is generated by the SEGMENT or OVERLAY subroutine:

```
         BRTJ    ($)name,,*
    +    SLJ     *+m
         nn      DICT.
    +    00      p1
         .       .
         .       .
         .       .
         00      pn
```

In this sequence,

name the transfer address for the loaded overlay or segment

$m = \frac{n+1}{2} + 1$

n = number of actual parameters in the FORTRAN CALL statement, $(p_1, \ldots, p_n)$

$p_1, \ldots, p_n$ actual parameters in FORTRAN CALL statement

Example:

CALL SEGMENT(3, 2, 25, , A, B, C)

The first FORTRAN card is: PROGRAM SUB2(X, Y, Z)

The transfer address in segment 2 of overlay 3 is SUB2.

The call to load the segment is:

```
BRTJ      ($)SEGMENT,,*
SLJ       *+5
07        DICT.
00    $   =D3
00    $   =D2
00    $   =D25
00        0
00    $   A
00    $   B
00    $   C
00        0
```

The call from SEGMENT to the transfer address SUB2 in segment 2 of overlay 3 is:

```
BRTJ      ($)SUB2,,*
SLJ       *+3
03        DICT.
00    $   A
00    $   B
00    $   C
00        0
```

ERRORS If errors occur during loading overlays and segments in response to the FORTRAN CALL, the job is terminated. The A register will contain the contents of the parameters for the last LOVER call specified in the CALL OVERLAY/SEGMENT statement.

(A) =	n	0	o	0	s
	47	41	38	23	14 0

n = Logical unit number
o = Overlay number
s = Segment number

Contents of the A register are written on OUT together with one of the following messages:

READ PARITY ERROR

LUN OUT OF RANGE

USE OF TOO MANY LUN

RECORD NOT ON THIS LUN

ILLEGAL SEQUENCE

OUT OF BOUNDS LOAD

COMPASS CALLING SEQUENCE

Two calling sequences for COMPASS subprograms can be used to load an overlay or a segment.

Sequence 1

a CALL LOVER

a+1

u	0	o	0	s
47	41	38 24	23	14 0

a+2 (return point)

In this sequence,

n = logical unit number

o = overlay number

s = segment number (0 if an overlay)

This is equivalent to the Tape SCOPE call.

Sequence 2

The MACRO instruction LOVER can be used:

a LOVER lun,f,n

a+2 (return point)

In this sequence,

lun = logical unit number

f = (O for overlay and S for segment)

n = overlay or segment number

In both cases, if the overlay or segment has been loaded correctly, the A register is set to 0, a BRTJ to the transfer address of the overlay or segment is placed in the Q register, and processing returns to the return point.

ERRORS

If errors are encountered while loading the overlay or segment, the A register contains one of the following error codes, justified to the right.

Error Code	
1	Non-recoverable parity error was encountered in loading the overlay or segment record.
2	Specified logical unit number was not 1-49.
3	More than four logical units were addressed in reading overlay and segment records.
4	Overlay or segment specified in the calling sequence was not on the specified logical unit.
5	Overlay or segment specified in the calling sequence was not consistent with the last overlay or segment loaded. (Rule 3 in section 8.4.3 has been violated.)
6	An attempt was made to load an overlay or segment out of bounds.

8.5 DATA FORMAT

Each partition is written in the form of absolute records in a file-structure format. The main partition always occupies the first file; each overlay with its segments comprises a separate file. The format for each main partition, overlay, or segment, is shown below.

The first record in each overlay file must be an overlay record which may be followed by segment records referenced by that overlay. The overlays on each overlay tape must be numbered in ascending order. Segments within an overlay file must be numbered and ordered consecutively. All segments of one overlay must be contained in one overlay file on one reel of one logical unit.

8.5.1 HEADER RECORD

The header record describes and defines the limits of the subsequent partition as follows:

	bits	content
Word 1:	47-42	00; defines record as header record.
	38-24	overlay number; zero denotes main partition
	14-00	segment number; zero denotes main or overlay partition
Word 2:	41-24	Second transfer address
	17-00	First transfer address
Words 3-10:		Control words for loading successive data records, one word for each record. If less than eight records follow, words at the end (beginning with word 10) are filled with zeros. Each control word is an input-output transmit record (IOTR) instruction.
Words 11-17:		Memory table at end of partition loading

8.5.2 DATA RECORDS

The data records occur in the same order as the control words in the header record.

	bits	content
Word 1:	47-42	record number; record 1 is loaded by the control word in the third word of header record, record 2 by the fourth, etc.
	38-24	overlay number; zero denotes main partition
	14-00	segment number; zero denotes main or overlay partition
Word 2:		unused
Words 3 to n+2		n data words
Word n+3		300 00000 000 00000

An EOF mark follows if:

Preceding partition was the main partition

Preceding partition was the last segment of an overlay

Preceding partition was an overlay and there are no segments for that overlay

An additional EOF mark follows if the preceding partition is the last partition on the unit.

LIBRARY MAINTENANCE 9

During execution, the Drum SCOPE library resides on the drum. This drum version of the library, called the internal library, is prepared by the PREDRUM program from a set of cards or card images, called the external library. With the LIBEDIT routine, an existing external library may be on tape or a new one may be created; the internal library may be modified by the LIBRARY control card (sec. 2. 8).

9.1 EXTERNAL LIBRARY FORMAT

The external Drum SCOPE library is composed of five files, each terminated by an end-of-file mark. An external library to be used as an auxiliary library unit need have only the library file present. An external library used to load the system must have the first four files present in the order specified below; the production file may be absent:

Autoload file

System file

Background file

Library file

Production file

Each file except the autoload file contains a *FILE card immediately following the end-of-file.

9.1.1 AUTOLOAD FILE

The autoload file is loaded into memory as a result of autoloading the external library. The information in this file is dependent upon the specific hardware device, (conventionally a magnetic tape unit). LIBEDIT assumes that any tape with a logical unit number of 72-79 has an autoload file and that a tape with a logical unit of 1-49 does not have an autoload file. The autoload files transfer control to PREDRUM, the first routine in the systems file.

9.1.2 SYSTEM FILES

The systems file contains the Drum SCOPE system and installation drivers. The order of routines within the systems file is rigid; any variance will inhibit the successful preparation of an operating Drum SCOPE System.

PREDRUM — Routine that prepares the internal library

PARAMS — System parameters, may be selected by the installation. A complete list is in appendix D.

HTL — Hardware Type List, BCD routine containing descriptions of the hardware known to the system

AET — Available Equipment Table, descriptions of each separately connectable device in the computer system

DCODES — List of up to 62 disposition mnemonics. The following codes must be included.

Mnemonic	Use
IN (Input)	EXEC will search for IN strings to operate as central programs.
HT (Held, Tried)	Each time EXEC scans all held programs, it saves those it cannot process as HT.
HU (Held, Untried)	When scanning all held programs, EXEC will temporarily set them to HU.
PR (Print)	Logical unit 61 is initially declared PR.
PU (Punch)	Logical unit 62 is initially declared PU.
AC (Accounting)	Accounting output is disposed of as AC.
TP (Tape)	Used by CRP to dispose of card to tape disposition.
BT (Background Trouble)	Used by EXEC to dispose of dump data for an aborted background program.
SW (SWAP Scratch)	Used by EXEC to save bank 1 of the central program while a SWAP request is being processed.
CF (Comment File)	Used by EXEC to save console messages for writing on standard output.

PNL — Program Name List, list of resident background programs

TITLE — Listable comment which appears at the beginning of each job's listing

SYSUNITS — System units, I/O units assigned to the system

RDL — Resident driver list

RESIDENT — Resident system, relocatable routine loaded into low memory

Drivers — Drivers are named for each hardware specified in HTL:

DRhhDxxx

hh is the hardware mnemonic

xx are arbitrary characters

Overlays — in the form: xxxxPyyy

xxxx is the name of the overlay

P is the letter L, X or H

yyy are arbitrary characters

Modules — in the form: xxxxMyyy

xxxx is the name of the module

yyy are arbitrary characters

BOOT — Routine that provides the final system initialization when PREDRUM has completed its task

9.1.3 BACKGROUND FILES

This file contains all the background programs available to Drum SCOPE; it is composed of relocatable routines each constituting a program. A background program named in PNL will be loaded immediately and not recorded on the drum. An installation may add programs to the background file. A background program must indicate to the system the number of logical units and families that will be employed as indicated in the IDC card word 7; the number of units is given in bits 38-24 and the number of families in 14-00. Each of these values is restricted to less than 64. In COMPASS, the values are given on the IDENT card as follows:

```
IDENT      name, nlun, nfam
```

Routines specified in PNL should be included in this file.

9.1.4 LIBRARY FILES

The library file contains routines available to the central program including the BCD routine, DIAGNO, which contains the diagnostics.

9.1.5 PRODUCTION FILES

Programs to be relocated at autoload time and stored on the drum as unblocked absolute records are contained in this file. This enables frequently used programs to be loaded several times faster than the blocked relocatable records in the library file.

9.2 LIBEDIT

LIBEDIT has two modes of operation:

Preparation of a new library requires the structure to be completely specified in the control deck.

Editing requires specification of changes to an existing library.

9.2.1 LIBEDIT INPUT

Input to LIBEDIT consists of a deck of control cards on INP (lun 60), and optionally, other libraries or data tapes contain routines to be included on the external library. LIBEDIT derives all information concerning the library being written from INP, unless directed to consult some other unit.

No use is made of the internal version of the library by LIBEDIT, except in loading of LIBEDIT by the monitor. To edit the current library, therefore, the external library must be used, not the internal version on unit 70. All data processed by LIBEDIT as input is checked, when relevant, for parity errors and checksum errors.

9.2.2 LIBEDIT OUTPUT

LIBEDIT produces an external library on logical unit 71 composed of the control cards, routines, programs and file marks derived from input as controlled by the statements on INP. Labels on external libraries are not produced by LIBEDIT. They may be specified by EQUIP cards prior to calling LIBEDIT.

LIBEDIT control on the external library is subject to the following limitations:

> Programs or routines which are not contained within programs may not be duplicated within one file. The second and subsequent copies will be bypassed if they occur.
>
> If a routine is duplicated within a program, the second and subsequent copies will be bypassed.
>
> If two or more files have the same name, the second and subsequent copies will be bypassed if they occur.

9.2.3 LIBEDIT CONTROL CARDS

LIBEDIT functions are called by the control cards shown below:

$^{7}_{9}$PREPARE

This card initiates external library preparation. It must be the first card, and may not be used elsewhere. When PREPARE is used, the following LIBEDIT control cards are illegal.

*MODIFY	*PATCH
*INSERT	*DELETE

The EDIT card initiates external library editing. It must be the first card, and may not be used elsewhere.

$^{7}_{9}$EDIT,u

u logical unit number (1-49, 71-79) of the library being edited.

9.2.4 LIBEDIT DIRECTIVE CARDS

These cards direct preparation or editing of the external library. They may occur on INP only; their occurrence elsewhere will be ignored. All cards contain a 7, 9 punch in column one, and an asterisk in column two. Following completion of the indicated activity, LIBEDIT returns to INP for controls.

$^{7}_{9}$*REWIND,u

LIBEDIT issues a REWIND request on logical unit u, 1-49, 72-79. If u is 72-79, the autoload file will be skipped after the rewind.

$^{7}_{9}$*BSPF,u

LIBEDIT issues a BSPF request on logical unit u, 1-49, 72-79.

$^{7}_{9}$*SKIP,u

LIBEDIT issues a SKIP request on logical unit u, 1-49, 72-79.

$^{7}_{9}$*EXTRACT,u,m_1,m_2,...,m_k

m_i names of routines, files, and programs to be extracted from logical unit u; embedded blanks are not permitted. The list terminates with the first blank not preceded by a comma. If the list is continued on subsequent cards, each card except the last terminates with a comma followed by a blank and continues on the following card in column 3; the first two columns of each continuation card must contain $^{7}_{9}$*.

The unit will be read and those routines and programs specified on the EXTRACT card will be placed on the new library as they are encountered, subject to the limitations stated in section 9.2.2. If a double end-of-file is encountered, the unit is rewound and searched again. This request terminates when all routines requested have been located, or the double end-of-file is encountered for the second time; in this case, a diagnostic will result.

$^{7}_{9}$*COPY,u,r,m_1,m_2,...,m_k

u   logical unit number. If the following conditions exist the autoload file will be copied. If not, it will be skipped if it exists.

1. u is in the range 72-79
2. an EDIT is being performed
3. u is the tape being edited

r   name of routine, file or program on unit u. All routines, programs and files are considered to be entities; therefore, if R specifies a file or a program, routines that are part of that file or program will not be excluded if they appear on the ignore list.

m_i name of routine, file, or program on unit u uses same format as EXTRACT.

The routines, programs, and files from the current position of unit u through the routine, file, or program named by r will be copied omitting all the m_i. A double end-of-file encountered before r will terminate the copying. The m_i follow the same format rules as stated in EXTRACT, above.

$^{7}_{9}$*MODIFY,m

m   name of routine, file, or program on unit u

The current library is positioned at the named routine. Following the MODIFY card are PATCH cards. This card is valid only for editing.

A one-word patch may be made on the current library by this statement. A binary card will be appended to the routine being modified so that when it is loaded, the patch will be loaded over the word which is to be modified. Any number of PATCH statements may be used for any routine, but each must be preceded by a MODIFY card naming the routine, or a PATCH card. This card is valid only for editing.

$^{7}_{9}$*PATCH,name+n=upper,lower

$^{7}_{9}$*PATCH,/cname/+n=upper,lower

name+n	location of word to be modified. name is the routine being modified; cname, enclosed in slashes, is the labeled common region. n is an octal number specifying displacement of word to be modified. n must be less than the program or common block length.
upper/lower	value and relocation specification of the patch.

The load address is specified in one of the above forms. The value of the patch is specified by half words, upper and lower, from one to eight octal digits; both must be specified. Both are suffixed by a relocation specification, as follows:

P	program relocation
blank or none	no relocation
A. . . I	relocation of the first nine declared common blocks. Only these may be specified.
-P, -A, . . . , -I	corresponding negative relocation

The program being patched must be relocatable.

$^{7}_{9}$*DELETE,m

m	name of routine, file, or program on current library

All routines, files, and programs up to the first occurrence of m are extracted; for input, m is bypassed. This card is equivalent to the following if u specifies the old library.

$^{7}_{9}$*COPY,u,m,m

This card is valid only if editing.

$^{7}_{9}$*INSERT,m

m   name of routine, file, or program on current library

All routines, files, and programs on the old library up to and including the first occurrence of m are extracted for input. This card is equivalent to the following if u specifies the old library.

$^{7}_{9}$*COPY,u,m

This card is valid only if editing.

9.2.5 LIBEDIT INPUT

Unlike the directive control cards which may be used in INP (60) only, the input cards described below may be either on INP or encountered on some other unit as a result of a directive card.

The input control cards may define routines and programs, in which case they will be placed on logical unit 71 subject to the rules on page 9-2.

$^{7}_{9}$*FILE,name

Signals the start of a new file identified by name. An end-of-file mark will be written on 71, followed by the *FILE card. The file terminates with the next *FILE or *FINISH card. The name must begin with one of the following letters, which identifies the type of file:

S	Systems file
B	Background file
L	Library file
P	Production file

If the *FILE card is encountered while extracting or copying and its name is in the extraction or ignore list, the entire file will be extracted or ignored.

```
7
9*PROGRAM,name
```

Signals the start of a production program identified by name. If this card is encountered while extracting or copying, and its name is in the extraction or ignore list, the entire program will be extracted or ignored. The program terminates with the next *PROGRAM, *FILE, or *FINISH card.

This card is not permitted in the systems, background, and library files.

IDC Card

The occurrence of an IDC card (w=31_8) signals and names a relocatable routine. The routine consists of a relocatable binary deck and terminates with a TRA (w=37_8) card. The name of the routine is extracted from this IDC card.

```
7
9*BIN,name
```

Identifies and names the subsequent binary cards as a binary routine. The routine terminates with a TRA card (w=37_8).

The data cards are similar to RBD cards except that the word count field contains the count of data words on the card from 1 to 23_8, and the data always begins in word 2 (column 5). The location field specifies the relative origin of the data within the routine.

This card is not permitted in the background and production files.

```
7
9*BCD,name,terminator
```

Identifies and names the subsequent cards as a BCD routine. Any number of BCD cards may be specified; they are terminated by the first occurrence of a card with the 8-character terminator in the first 8 columns (blanks included). This card is not permitted in the background and production files.

```
7
9*MACRO,name
```

Signifies the start of a set of BCD cards containing the definitions of the library macros for COMPASS. LIBEDIT will process the cards and produce a binary data routine with the same name.

This card is not permitted in the background and production files.

9.2.6 SPECIAL CONTROL CARDS

```
7
9*ABS,u
```

Indicates a standard relocatable binary deck consisting of one subprogram with no entry points, external symbols, or common blocks and not exceeding 5000_8 locations in length. The deck is on unit u, 1-49 or 60. If blank, 60 is assumed.

With this card the user may define the contents of the autoload file, the only one with this type of record. When preparing an external library for use, the autoload file must be specified in this way, if it is to be present at all. When editing, the absolute file of the old library will be copied unmodified, unless the *ABS card is the first data encountered by LIBEDIT. The only cards which may precede the *ABS card are *REWIND, *BSPF, *SKIP.

```
7
9*FINISH
```

This card must be the last control card on INP. An end-of-file mark will be written on logical unit 71, followed by the *FINISH card and two more file marks. If a *FINISH card is detected by LIBEDIT while reading a unit other than INP, it is assumed to signal the end of an external library and is equivalent to a double end-of-file.

However, if an edit is being performed, the occurrence of a *FINISH card on INP will cause LIBEDIT to read and process all data remaining on the old library up to its *FINISH cards, subject to the rules on page 9-2.

9.2.7 LISTING EXTERNAL LIBRARIES

When LIBEDIT produces an external library, a list of the control cards, files, programs, and routines on the external library will be written on OUT.

Also, the LISTLIB named entry card can be used to obtain a listing on OUT of the contents of an external library.

```
7
9LISTLIB,u
```

u	specifies the logical unit number (1-49, 71-79) of the external library.

In both cases, the listing will be formatted to indicate grouping and order on the external library. The names and lengths, where appropriate, of all files, programs, and routines on the library will be written on OUT.

APPENDIX SECTION

DRUM SCOPE BACKGROUND PROGRAMS

I GENERAL INFORMATION

The 861 drum is the principal I/O device in Drum SCOPE. Two drum features in particular are inherent in the operating system design:

High rate of information transfer between drum and core storage (4 usec per 48-bit word at 1-1 interlace)

Random access organization with minimum access time (average access time = 17 ms)

For example, loading a library program into core from drum storage requires less time than loading it from magnetic tape.

Drum SCOPE uses blocked records to facilitate the scheduling of drum storage. Drum storage is partitioned into blocks of the same size as the blocked records used by the system. A sequence of blocked records linked by pointers in each block defines a string which may be assigned to a logical unit. Block addressing is used internally for drum reference.

In Tape SCOPE, standard input and output units are magnetic tapes. The information on the tapes is divided into contiguous units called jobs, and the position of the job on the standard input unit determines the order of execution.

In Drum SCOPE, a card-assigned priority number associated with each job determines the sequence in which jobs are processed. If no assignment is made, priority is zero.* Jobs having the same priority number are executed in the sequence with which they were introduced to the system. The system also disposes of output on the priority basis.

Background programs (card-to-drum, drum-to-printer, drum-to-punch) prepare the central job input files on the drum and dispose of the output files.

* Or whatever parameter the installation assigns.

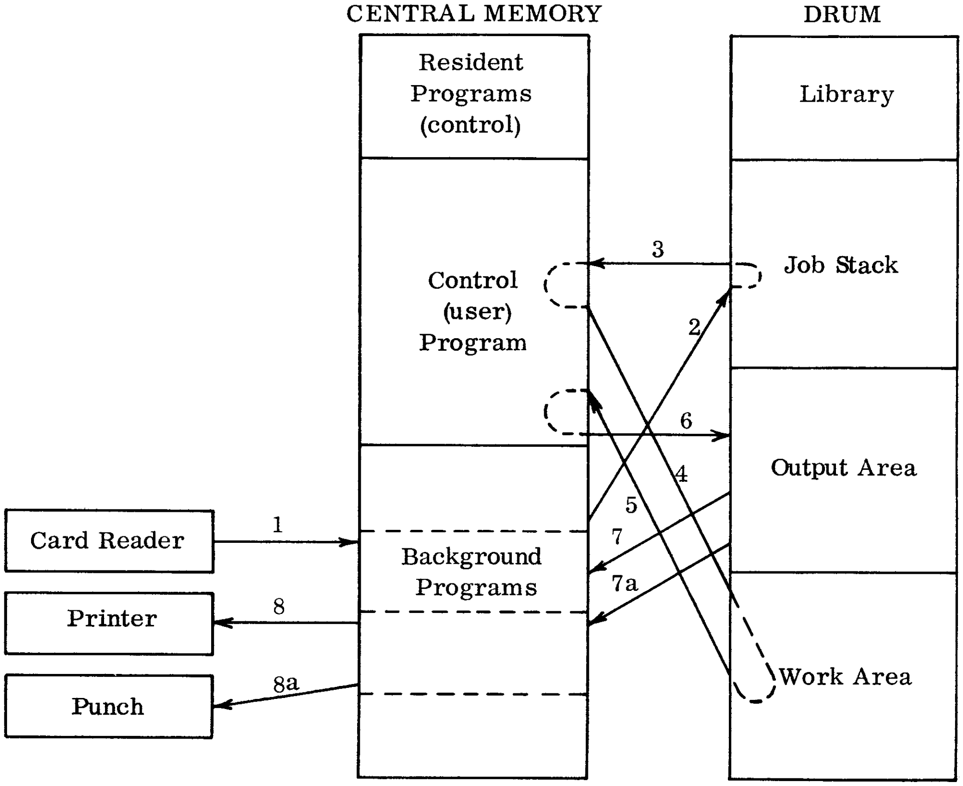


Figure 1. Information Flow in Drum SCOPE System

INTERRUPT SYSTEM

The 3600 interrupt system transfers control to location 1 of bank 0 when specified conditions occur. Interrupt conditions are either internal, such as a time interrupt, or external, such as an I/O interrupt.

External interrupts are used by the system for I/O control; interrupts are selected for each I/O operation. Internal interrupts are used to sense conditions, such as storage reference faults, to produce time signals, and to control storage usage. The bounds fault interrupt is used to signal a request from an executing program to the system. Arithmetic fault interrupts are not used functionally but are processed by the system for programs selecting them. All user programs, including interrupt routines, are executed with interrupt active.

CENTRAL PROCESSING UNIT

In Tape SCOPE, the CPU is used by the executing program or by the system. The system has higher program priority. This is program priority rather than job priority.

In Drum SCOPE, there are 64 levels of program priority for the CPU. System function programs carry a higher priority level than a job or executing program. In figure 2, the center circle represents the interrupt control routine INC. Although a priority level number is not assigned to INC, it has immediate use of the CPU each time an interrupt occurs. A new condition does not cause an interrupt while INC is using the CPU. INC performs such bookkeeping tasks as interpreting interrupts and recording the anticipated CPU usage for the program levels involved.

After these tasks are performed, INC seeks the highest level that can use the CPU and gives control to the program at that level. Not all levels are equivalent with respect to the usage a program may make of the CPU. Levels 63, 62, and 2 comprise a privileged group, the background programs another group, and levels 0, 1, and 3 the central program group. The privileged group does not run under monitor protection (it has no bounds limits); certain requests are allowed in background programs but not in central programs.

STORAGE

Portions of the system, such as interrupt control, I/O processing, and status tables, must be in core storage at all installations.

Background programs declared resident on the system's external library may vary from installation to installation.

The EXEC portion of the system allocates storage available for the central program and for non-resident background programs.

INPUT/OUTPUT

All I/O channels and devices are scheduled and monitored by the system program on level 63 as shown in figure 2. Lower-level programs request this program to assign them the use of a specific device. A device is reserved for the requesting program until it is released.

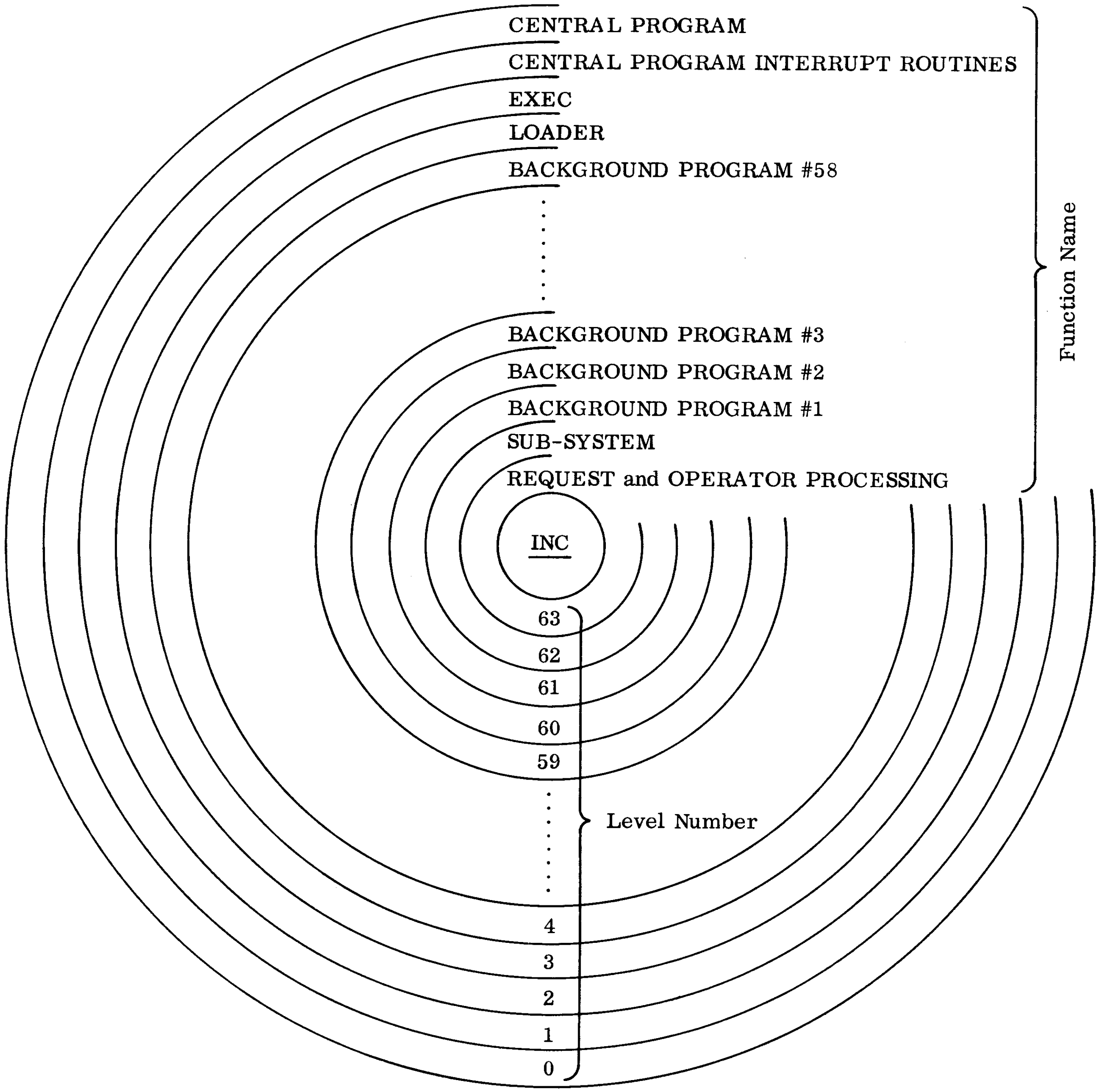


Figure 2. Priority Level Structure Diagram

Associated with the request for most I/O operations is an interrupt address to which control is transferred when the operation is completed. For any one program level, the transfer cannot occur until a RETURN request has been given by that level. However, in the central program, control transfers to level 1 at the time the operation is completed. This is the user's interrupt subroutine.

SYSTEM REQUESTS

Programs communicate with the system through system requests. Certain requests are not legal for all levels. Abnormal termination procedures are followed if a request is illegal for the level in which the program resides.

As an example, consider the following request from a background program.

```
BSPR     LUN, RA, IA
```

The request would appear in storage as the binary equivalent of:

```
LOC      63      5*8
         03       0
         00       0
         00      LUN
         00      RA
         00      IA
```

When CPU attempts to execute the instruction at LOC (the bank return jump to location 0, bank 0) a bounds interrupt is generated. This interrupt transfers CPU control to INC which saves the values of the CPU registers at the time of interrupt, and recognizes the interrupt as a bounds interrupt. Since the destination address of the transfer is (0) 00000, INC recognizes the interrupt as a request interrupt. The interrupt is then interpreted according to the value in the upper address position of the bank jump. INC records the interrupt as requiring the action of the request processor on level 63.

INC then transfers control to the request processor where the request and the status of the logical unit are checked; if they are acceptable, the backspace operation is initiated. The interrupt address, to which control is to be transferred after the operation is completed, is saved. If no other interrupts occur, control is directed through INC back to the program originally making the request. The registers are restored to their values at the time the request was executed, and the program continues from the point of the request. The requested operation is not complete, however, until the I/O operation is complete and interrupt routine has been entered. In a background program this may occur only after the program relinquishes control with a RETURN statement.

II BACKGROUND PROGRAMS

Drum SCOPE uses the background program facility to:

Place job input information on the drum

Transfer job output information to other media

Perform installation defined tasks

Dump onto magnetic tape

An installation may add its own background programs or replace the standard background programs provided by the system with its own version of programs to form and process job input/output strings. The only requirements are:

The System must be provided with strings of information on the drum that are declared as IN disposition

Output strings (PR and PU dispositions) should be removed from the drum

Additional disposition categories may be declared when a new system tape is prepared. Additional background programs are required to process newly defined dispositions. Background programs which process output strings should be named; for instance, BK.xx, where xx is the disposition mnemonic of the type of string processed by the program. Functions peculiar to an installation may be performed by background programs. Certain requests such as LIMIT, RDLABEL, etc., are not defined since background programs are not intended to accomplish objectives similar to central programs.

Background programs should occupy a small fraction of total CPU time. This is the situation within standard job I/O processing where the amount of time taken to initiate an asynchronous I/O is small compared to the time of the operation. Processing, which may take more time but which is not often required, might also be performed by background programs.

COMMUNICATION

Background programs communicate with Drum SCOPE, the computer operator, and other background programs. They communicate with the system through the requests listed in section III. Communication from system to background programs is implicit with control transfers and explicit through the A, Q registers. An example of implicit communication is the transfer of control to the reject address in response to a READ request. The value of the registers in response to a DATE request demonstrates explicit communication.

The console typewriter is used for communication between the system and the operator. By typing an asterisk followed by the name of the background program, the operator may request the system to transfer a message to a background program. This indicates to the system that the message following is intended for the named background program. However, the background program receives the message only if it has requested a READ on a unit assigned to the input comment medium.

With the SYSIO request, a background program can request the system to write a message on the output comment medium. The message is preceded by a tab and the name of the background program. The operator communicates with the system by pressing manual interrupt, waiting for the type-in light, and typing the message.

The communication path is not direct; the system relays the messages to the proper destination. The INFORM and XFER requests transfer information from one background program to another. The XFER request reassigns logical units of the requesting program to logical units of a designated program. XFER does not initiate action in the receiving program.

To be prepared to accept the message sent by INFORM, the receiving program must request a read on a unit assigned to the input comment medium. Since the message may also be from the operator, the message source may be determined by the content of the message. A maximum of 10 words of information can be transferred via the INFORM request.

Since the INFORM request is fulfilling a read operation, it initiates the interrupt routine associated with the read. If the receiving program is on a higher priority level than the sending program, the transfer to the interrupt routine takes place immediately after the information is transferred. However, if the receiving program is on a lower priority level the transfer to the interrupt address cannot take place until the sending program and any other programs on higher levels have no further action to be taken, and have issued RETURNS.

One background program can determine the status of another with the SIWOH request. SIWOH is used to determine whether a background program is present, if it is reading the ICM unit, and so forth. In a broad sense, the INVOKE request is also a background program communication request.

PHYSICAL PROGRAM STRUCTURE

The physical structure of background programs can be considered from two angles: as they appear in storage after loading, and as ready for a COMPASS assembly. In storage, the program is retained in a continuous portion of bank 0 memory. The area contains:

1. Program Instructions
2. Program data
3. Tables maintained by system
4. Buffer areas to contain records for blocked units

The third and fourth sections are used exclusively by the system. In preparing a background program for COMPASS assembly, the area for tables is indirectly specified by the statement:

```
IDENT name,n,f
```

where n is the number of logical units and f is the number of families used by the program. The buffer areas for blocked records should be specified in the program with either a BSS or .POOL xxx common block statement. The logic of the program should include BUFFER and FAMILY requests to properly utilize the areas. Alternately, buffer area 0, common to all programs, may be used; however, this is not recommended since inefficiencies would result.

The first location of the background program must be reserved for storing an EXIT request issued when program execution begins. Likewise, the location specified as an interrupt address must be reserved for a RETURN request.

LOADING AND EXECUTING

The loading action is dependent on whether or not the background program is to remain in storage. A background program can be introduced into the system only by being added to the external library with LIBEDIT. At the time the system is initiated, resident background programs, as declared in PNL, are loaded into memory and non-resident programs are stored on the drum. Since the loader for background programs does no linking, entry points and external symbols cannot be handled.

Background programs can be initiated in three ways if the loading operation is included and if the program is not present in storage:

1. EXEC looks for strings with a disposition code. When strings are available, EXEC loads the background program specified by BK. xx where xx is the disposition code. For example, if a PR string is available BK. PR will be loaded.
2. The operator may initiate a background program with a LOAD message on the console typewriter.
3. One background program may initiate another with the INVOKE request.

A loading operation is performed immediately only when initiation originates in EXEC. EXEC loads background programs only when storage is available for reassignment. Loading cannot occur during central program execution, and background program loading is dependent upon the central program status. Background programs can be loaded at phase or job breaks. When a background program is loaded, it is assigned the next lower priority level.

Effective background programming depends on an understanding of the external equipment, knowledge of the drivers used in the system, and familiarity with Drum SCOPE implementation techniques. Detailed information for coding drivers is contained in the Drum SCOPE reference manual and the maintenance documentation.

ENTRY

A background program is initiated by placing an EXIT request at relative location 0 and transferring control to 1. On entry, the A register indicates whether the background is resident or non-resident.

A negative, non-resident A positive, resident

The loser address portion of Q contains the location of the program control table for the background program.

Q(14 - 00): Location of PCT

III SYSTEM REQUESTS

Most system requests may be used by background programs. Requests such as READ and WRITE have their usual meaning with the restriction that the interrupt subroutine is not entered until the background program has relinquished control with a RETURN request.

SYSIO f,k,fwa,wdct

f	function; unindexed digit, 0 or 1 0, write message to operator 1, accounting information.
k	digit 0-7; relevant only if f = 0.
fwa	18-bit unindexed address.
wdct	15-bit word count, unindexed; with fwa, this parameter defines a buffer area.

With this request the user may write accounting data or a message to the operator on OCM.

The message beginning at location fwa, containing wdct words, will be written on the unit specified by f. If f = 0, the output can be conditioned with stop keys.

k = 0	message appears always
1	if stop key 1 is set
2	if stop key 2 is set
3	if stop keys 1 or 2 are set
4	if stop key 3 is set
5	if stop keys 1 or 3 are set
6	if stop keys 2 or 3 are set
7	if stop keys 1, 2, or 3 are set

Example:

```
SYSIO 0, 5, MSG, 3
```

The 3-word message beginning at location MSG will be output on the typewriter if stop keys 1 or 3 are set. The message is in internal BCD mode.

The write on OCM is controlled by the selective stop switch settings to permit the installation to govern the volume of output on the typewriter. The request is not buffered. When control is returned, the message has been transferred to systems buffer when the octal I/O occurs. Further SYSIO requests are rejected until the previous transmission is complete. To receive a message from the operator, the ASSIGN (ICM) request and READ request on the unit must be used. (See ASSIGN request.)

Accounting records are written on the drum; they are given a disposition of AC and released when the operator types AUDIT. A background program, BK.AC, processes the AC string. A simple version is supplied by Control Data Corporation; a more sophisticated version may be written by the user.

ASSIGN uu, y

Background programs may make logical unit assignments with the ASSIGN request. The number of logical units must be specified on the program IDENT card.

y = DR

Unit y is assigned to the drum. The sequence number assigned to the string is the one belonging to the current job being run. This may be changed by the PRISEQ request. Subsequent WRITEs will create a string in the INP area of the drum. The installation determines how much of the drum is reserved for this use by specifying LINP when defining PARAMS on the external library. If the installation expects considerable string generation and manipulation by background programs, this area should be expanded.

An ASSIGN may not be necessary when data is to be placed on the drum, since a WRITE given on an unassigned unit causes the unit to be assigned automatically to the drum. However, since only a limited number of drum strings can exist at any one time, the system will terminate the program issuing the WRITE in the event that there is no room for the new string. ASSIGN un DR permits the background program to detect this situation.

y = disposition mnemonic

This method locates and assigns particular strings on the drum. For example, BK. PR performs ASSIGN uu, PR when a print string is required. All strings on the drum have an associated disposition. When the request is honored the string with the highest priority and lowest sequence number is assigned. On return, the Q register contains information which may be used by the background program.

The sequence number Q(14-00) and priority Q(17-15) may be used for accounting purposes. The ID (Q23-18), may be used as a security classification or as a further indication of the disposition. For example, a print string, PR, which is to be transmitted to a particular remote site, "A", would have an ID of "A" (assigned through PRISEQ by a special background program when the job was introduced to the system). The printing background program would be able to distinguish this string and send it to remote site "A".

Disposition of the unit is set to 0 following a successful ASSIGN.

y = hardware mnemonic

This method locates an available piece of hardware. A write assignment for the hardware is attempted. For instance, the printing background program performs ASSIGN uu, LP when it requires a line printer. The hardware designate of the form HHnn (HH = hardware mnemonic, nn = ordinal) is returned in Q(23-00) upon return from a successful ASSIGN. Hardware may be reserved for background program assignment in the AET routine in the systems file of the external library.

y = hardware designate

This obtains a specified piece of hardware. The physical unit must be unassigned and not down. As an example, this method could be used to assign remote terminal "A" as a result of receiving the PR string with ID "A" as mentioned above. If RT were the hardware type and "A" were the first RT, the background program performs ASSIGN uu, RT01.

y = ICM

With this method, the background program can receive mcssages from the operator by performing a subsequent READ request on the unit. After performing ASSIGN uu, ICM, a READ request on uu is stacked until the operator directs a message to the background program. When the operator types in *name of background program, the message is stored as specified by the control words and the interrupt subroutine is entered. If no interrupt address is specified, the READ request is meaningless. To receive additional messages, additional READ uu requests must be issued. A RETURN request must be given before the interrupt subroutine can be entered. Ten words is the maximum message length.

When control returns to the program, the A register indicates the result of the request as follows:

A > 0 Assignment was honored

A = +0 Assignment was not made because no hardware or drum unit was available

A = -0 Assignment was not made because it requires a driver which cannot be loaded at present

A < 0 Assignment was not made because y was not recognized by the system

Following successful ASSIGN requests, units which are not needed should be released. For example, if a PR string is assigned but no printer is available, the disposition of the string should be reset to PR via DISPOSE and the unit released; similarly, if a printer is assigned and no PR string is available, the printer can be released for use by another program in the system. The WAIT or BYNBY request can be used for a delay before repeating the assignment request. Following an unsuccessful ASSIGN request for hardware requiring a driver which cannot be loaded, the system will load the driver at the next opportunity (job or phase break).

PRISEQ u, p, seq, id

u	logical unit number, must be a blocked unit assigned to the background program.

p	priority 0-7 with 7 being the most urgent, p cannot be indexed.

seq	sequence number which will be assigned to the unit. If zero, Drum SCOPE will assign a number. In either case, the A register contains the sequence number assigned in bits 14-00 and priority in bits 17-15.

id	a one-character identifier to be associated with a drum unit available later in the ASSIGN request. This could be used, for example, for security classifications.

The PRISEQ request gives priority sequence and ID information to the system concerning blocked units. A unit assigned to the drum, either through ASSIGN uu, DR or a WRITE request, is given the same priority, ID, and sequence number as the currently executing central program so that all units associated with a job have the same priority, ID, and sequence number. This allows a background program to determine which job is being run.

If other values are necessary, the background program must set them through the use of PRISEQ. If no sequence number is specified, the system assigns the next highest number. The first sequence number assigned is 1, the second 2, etc.

For example, CRP, the card reader processor, uses PRISEQ following assignment of a string for a job. Priority is taken from the priority card if there is one, otherwise zero is assigned by CRP; the sequence number is zero and the ID is zero.

The ID field is unused by the system; the installation may make any desired use of it.

The ID is passed along, unaltered, with all information associated with the job. A sequence number may be assigned if SEQUENCE cards are used by an installation and CRP is modified to accept and process the cards. (The standard version of CRP has an assembly option to ignore sequence cards.)

DISPOSE u, dd, ea

u logical unit number belonging to the background program. If the unit is not a drum, disposition is ignored.

dd disposition mnemonic; cannot be indexed.

The dispositions IN, PR, PU, AC, HT, TP have well defined meanings - an IN string is assumed to be available for processing as a central program job. PR and PU are print and punch strings. AC is the accounting string. HT and HU are used by EXEC for DEMAND card processing. TP is a string to be written on tape.

ea error address; if the specified disposition is not recognized, an exit to the error address is taken.

Example:

DISPOSE 2, PR, ERROR

Logical unit 2 is to be printed when released.

The DISPOSE request declares a disposition for a drum string when it is released. Judicious use of DISPOSE and the disposition codes can increase the capability of the system. Files (strings) may be written on the drum, given dispositions, and processed independently of other system operations. The unit is not released when DISPOSE is issued. Since DISPOSE is illegal on a unit declared as FO, DISPOSE should be issued before declaring a unit FO. Restrictions are:

The strings are not labeled and are identifiable only through disposition, ID, sequence number, and priority. A request cannot be made for a particular string within a disposition class.

Installations may define as many as 62 dispositions.

MODE

The MODE request is discussed here because drum units declared FO (forward only) have special significance. Background programs should declare as FO all units assigned to the drum if no backward operations will be performed on the units. REWIND, BSPF, UNLOAD are illegal on FO units. A limited number of BSPR requests may be given on a FO unit. BLSIZE words are always available to be backspaced over.

During reading, the blocks of the string are released after they have been read. During writing, the blocks of the string which have previously been written are made available for release. For example, the OUT string (with disposition PR) being generated in the central program, is declared FO. When a background program requests a PR string through ASSIGN, the system will release the previously written blocks comprising OUT when the next write is given on OUT, before OUT is released at end of job. The ASSIGN request will not be honored at this initial request, but by the time another ASSIGN is given some blocks may have been released. The ASSIGN request can, therefore, be honored before the string has been completed.

WAIT

The WAIT request relinquishes control to the system so that lower level programs can be executed. Interrupt subroutines for the program are not entered. Control is returned to the program after a time delay of less than one minute specified by an assembly parameter in RESIDENT.

WAIT is used whenever the background program has no task to perform, and no I/O operation is currently going on. If a RETURN request is used, the program is not re-entered.

WAIT can be used when attempts to assign units with ASSIGN are unsuccessful. WAIT can be used in conjunction with DYSTAT to simulate a READY request if other tasks may be performed.

BYNBY interrupt address

The BYNBY request is similar to WAIT. The program will be entered at the interrupt address following the same delay as for WAIT. BYNBY permits other interrupt subroutines to be processed, whereas WAIT delays processing until the WAIT is satisfied. BYNBY may be used by a program which handles more than one operation. For example, BK. PR (printing background program) would use BYNBY when waiting for another printer to become available. Otherwise, several flags would need to be set and checked before returning control to SCOPE in the reading/printing cycle.

RETURN

This request returns control to the system. The RETURN request should be issued whenever further execution awaits completion of an I/O operation. It should not be issued unless the program expects an interrupt (either I/O or BYNBY) to occur; otherwise the program will not be re-entered.

RETURN enables a program for an interrupt entry and gives control to the next lower level.

Two paths of control can be taken:

> If an interrupt has occurred for an I/O operation on a unit belonging to this background program, the interrupt subroutine will be entered.
>
> Control will be given to a program at a lower level.

Unless a background program releases control (RETURN, WAIT) no lower level program is executed. Since background programs can be introduced only through the external library, the installation must verify that every background will eventually release control.

EXIT

The EXIT request is issued only when the program has completed its functions and is to be terminated. A program that is not a resident background program is removed from core after all lower level background programs have been removed. Removal takes place only at job or phase breaks. A resident background program remains in core but cannot be re-entered until the LOAD statement is issued by the operator.

ABORT (diagnostic key)

The ABORT request is used only if the program is to terminate itself abnormally. A resident background program should not issue an ABORT as it will not be re-entered and there is no copy of the program on the drum. If trouble occurs SYSIO (OCM) and/or WRITE (on a unit with PR disposition) may be used. If a background program is aborted the memory occupied by the background program will be written on the drum and a dump produced later.

SIWOH ln

ln	location containing the name of a background program, left justified with blank fill.

This request enables a background program to determine the status of another background program. Upon return, the A register specifies the status as follows:

A = 0	No background program of this name is known to the system, either in memory or on the drum.
1	The program is known to exist, but it is neither in memory nor awaiting loading.
2	The program exists, but is awaiting loading.
3	This program is present in memory, but is not currently reading ICM; that is, an INFORM will be rejected.
4	This program is present in memory and is currently reading ICM.
5	Same as 3 except the program is of equal or higher level.
6	Same as 4 except the program is of equal or higher level.

SIWOH may be used to detect the presence of a background program and its capability of receiving messages via INFORM.

INVOKE ln, ea, ia

ln	address of the name of the background program to be loaded. The name must be left justified with blank fill.
ea	address to which control is transferred if the loading tables are full.
ia	address to which control is transferred when the program has been loaded.

With INVOKE, a background program requests loading and initiation of another background program. If the program is in core, an interrupt will be posted for the calling program and on return, the A register will contain a 3 or 4 depending on whether or not the invoked program was reading ICM (3 if not, 4 if so). If the invoked program has an equal or higher priority a 5 or 6 will be returned in A. A SIWOH should be given at this point to obtain a more current status. The calling program will be terminated if the background program is not in the directory. If the program is in the directory but not in core, it will be put on the invoke list to be loaded and initiated as soon as possible. In this case, the A register will contain a 2 on return.

INVOKE may be used by a small controller background program to load larger programs to perform specific tasks, or (via some particular signal) to load special background programs.

Only INFORM and XFER requests may be used to communicate with the loaded background program. If the named program is not known to the system, the background program will be terminated.

INFORM ln, fwa, wdct, ra

ln	location containing the name of a background program, left justified with blank fill.
fwa	first word address of a buffer area.
wdct	size of the buffer area, less than or equal to 10 words.
ra	reject address.

When this request is encountered the message defined by fwa, wdct is transmitted to the background program whose name is ln. If wdct is greater than 10 words, the first 10 words will be transmitted. If the background program is present but not reading a unit assigned to ICM (Input Comment Medium), control will be transferred to ra.

The INFORM request transmits a message from one background program to another if the receiving program has performed a read on ICM (see ASSIGN). INFORM can be used to signal other background programs to begin a task. Statements from the operator are transmitted in the same manner. Since the background program receiving the message must interpret the source, the message must be intelligible to the receiving background program.

XFER u, ln, lun, ra

u	logical unit number
ln	location containing the left justified name of the receiver program
lun	logical unit number
ra	reject address

This request permits the transfer of a logical unit assignment from one program to another. The current assignment of the unit specified by u of the sending program will be assigned as logical unit lun of the receiver program. If u is busy, or if the old assignment of unit lun in the receiver program is busy, the request will be rejected. If lun is out of range to the receiver program, or if ln specifies a program not currently present, the program is abandoned.

XFER may be used when a card reading program encounters a control card which requires special handling, such as, transmittal to a remote station. The remote station background program is loaded and notified about the sending XFER. The card reader assignment is given to the new background program which can then read cards directly. When the task is completed a new XFER must be issued so that the unit assigned to the card reader may resume normal operation.

Due to the lengthy procedures required - INVOKE, SIWOH, INFORM, XFER, INFORM, etc., - other methods of handling this type of operation may be superior. For example, the data could be written on the drum with a specific disposition to be processed by another background program at a later time, or the entire capability could be written into the original program.

READY u, ra, ia

With this request, the background program may sense a ready condition on a unit; the interrupt subroutine is entered when the unit is ready (ready for an operation). Actual definition of READY is determined by the I/O equipment driver. Normally, drivers which deal with special equipment are written in conjunction with the background programs.

The READY request is used in the normal manner. For instance:

> After establishing a link with a remote terminal or station, the READY request is followed by a RETURN. When the terminal is ready (as defined by the I/O driver) the background program is re-entered and data transmission is initiated.

This request loads background program overlays.

LOADBO

ln location of the name of the background overlay to be loaded. The name must be left justified, with trailing blanks, from 5 to 8 characters.

fwa 15-bit address of the beginning of the area to be used to load the overlay.

ra reject address. Control transfers to ra with A = -0 if the name is not found in the system directories. Control transfers to ra with A = +0 if the area specified is too small to contain the requested overlay.

ia interrupt address entered when loading is completed.

Background program overlays must be loaded within the bounds of the calling background program. No entry points or external symbols may be defined in the overlays.

EXPANSION OF REQUESTS B

All systems requests are expanded into a BRTJ command with a possible parameter list. The BRTJ is coded:

```
630 cccc0 030 00000
```

cccc is the call code for the request. Call codes 0-3777 are reserved for the system; call codes 4000-7777 are for installation use. The entry point, SENTRY, is defined by the loader to be 0 00000. This permits Tape SCOPE compatibility. Each request is specified by the request name, the call code value and the parameter list. If the expansions of two requests differ only in the call code value, a reference to another expansion will be given.

In the following expansions, parameters are used as follows:

pi denotes a parameter whose effective value is calculated with indexing. The index registers are those which were set at the time the request is issued. If indirect addressing is used, all references are within the bank of request.

(b)p denotes that the index field contains a bank term.

p denotes a value expressed in one field.

```
                                                          Call Code
EXIT                                                        C = 0
                       +   63    0*8
                           03      0

READ u,cwa,ra,ia
                       +   63    1*8                        C = 1
                           03      0
                           00    cwa,i
                           00    u,i
                           00    ra,i
                           00    ia,i

WRITE u,cwa,ra,ia          like READ                        C = 2

REOT u,cwa,ra,ia           like READ                        C = 3

WEOT u,cwa,ra,ia           like READ                        C = 4
```

```
                                                                        Call Code
BSPR u,ra,ia                                                            C = 5
                              +  63    5*8
                                 03      0
                                 00      0
                                 00    u,i
                                 00    ra,i
                                 00    ia,i

BSPF u,ra,ia                     like BSPR                              C = 6

REWIND u,ra,ia                   like BSPR                              C = 7

UNLOAD u,ra,ia,p                                                        C = 8
                              +  63    8*8
                                 03      0
                                 00    p,i
                                 00    u,i
                                 00    ra,i
                                 00    ia,i

SKIP u,ra,ia                     like BSPR                              C = 9

ERASE u,ra,ia                    like BSPR                              C = 10

MARKEF u,ra,ia                   like BSPR                              C = 11

MODE u,ra,p1,p2,p3,p4,p5                                                C = 12
                              +  63    12*8
                                 03      0
                                 00    ra,i
                                 00    u,i
                                 VFD   A23/,A5/P1,A5/P2,A5/P3,A5/P4.A5/P5
```

(A zero p_i is ignored)

where p_i have the following values:

BIN	6	OP	10B	RF	20B	ND	30B
BCD	7	LO	11B	RO	21B	RV	31B
		HI	12B	BY	22B		
		HY	13B	WO	23B		
				RW	24B		

Call Code

STATUS u C = 13

```
+   63    13*8
    03       0
    00       0
    00    u,i
```

LABEL u, addr, ed, rl, rtn or crd C = 14

```
+   63    14*8
    03       0
    00    addr,i
    00    u,i
    VFD   A24/crd
    VFD   A12/ed,A12/rtn
```

SAVE u C = 15

```
+   63    15*8
    03       0
    00       0
    00       0
```

UNSAVE u like SAVE C = 15

SELECT m, ia Direct Select C = 16

```
+   63    16*8
    03       0
    00    ** Saved old selection
    m     ia,i
```

m occupies positions 23-18. The codes are:

SHIFT	1	ADDR	12B
DIVIDE	2	TRACE	14B
EXOV	3	INST	16B
EXUN	4	MANUAL	20B
OVER	5	ABNORM	21B

SELECT I, ia Indirect Select C = 16

```
+   63    16*8
    03       0
    10    ** Saved old selection
    00    ia,i
```

Call Code

REMOVE m C = 17

```
+   63    17*8
    03       0
    00    ** Saved old selection
    m        0
```

m codes as in SELECT

LIMIT du, ra, ia C = 18

```
+   63    18*8
    03       0
    IOSW du
    00    ra, i
    00    ia, i
```

If d_u is compound, the parameter is:

```
+   00    milliseconds
    00    seconds
```

FREE like EXIT C = 19

TIME like EXIT C = 20

BOUND lb, ub, ra, ia C = 21

```
+   63    21*8
    03       0
    00    (b)lb
    00    (b)ub
    00    ra, i
    00    ia, i
```

UNBOUND like EXIT C = 22

DATE like EXIT C = 23

LOADER like EXIT C = 24

LIBRARY ln, ea C = 25

```
+   63    25*8
    03       0
    00    ea, i
    00    ln, i
```

```
                                                              Call Code
MEMORY b, l, u                                                C = 26
                            +  63    26*8
                               03       0
                               00    (b)u
                               00       1

HERESAQ                        like EXIT                      C = 27

RELEASE u, ra, s                                              C = 28

                            +  63    28*8
                               03       0
                               P     ra,i
                               00    u,i

     P occupies bit 47

RDLABEL u, ba, ra, ia                                         C = 29

                            +  63    29*8
                               03       0
                               00    ba,i
                               00    u,i
                               00    ra,i
                               00    ia,i

WRLABEL                        like RDLABEL                   C = 30

RDBLOCK u, cwa, ra, ia         like READ                      C = 32

WRBLOCK u, cwa, ra, ia         like READ                      C = 33

SWAP      ln, ea, ia                                          C = 34

                            +  63    34*8
                               03       0
                               00    ln,i
                               00       0
                               00    ra,i
                               00    ia,.i

JOB.ID                         like EXIT                      C = 35
```

Call Code

SETUPT lun, f — C = 36

```
+   63    36*8
    03       0
    00    f, i
    00    lun, i
```

GETUPT lun — like STATUS — C = 37

SYSIO f, k, fwa, wdct — C = 39

```
+   63    39*8
    03       0
    00    wdct
    fk    (b)fwa
```

f occupies bit 21. K occupies 20-18

f = 0: OCM

f = 1: ACC

RETURN — like EXIT — C = 40

RETURNM p, a, ob — C = 41

if p = U (upper)

```
+   63    41*8
    03       0
  UBJP    (b)a, , ob
```

if p = L (lower)

```
+   63    41*8
    03       0
  BJPL    (b)a, , ob
```

CHECK u — C = 42

```
+   63    42*8
    03       0
    00       0
    00    u, i
```

ASSIGN u, q — C = 43

```
+   63    43*8
    03       0
    VFD   H24/q
    00    u, i
```

```
                                                            Call Code
PRISEQ u,p,seq,id                                           C = 44

                          +  63    44*8
                             03       0
                             VFD   A6/id,A3/P,A15/seq
                             00    u,i

LOCATE u,lra,ra                                             C = 45

                          +  63    45*8
                             03       0
                             00    lra,i
                             00    u,i
                             00    ra,i
                             00       0

DISPOSE u,dd,ea                                             C = 46

                          +  63    46*8
                             03       0
                             00    ea,i
                             00    u,i
                             BCD   1,dd

INFORM ln,fwa,wdct,ra                                       C = 47

                          +  63    47*8
                             03       0
                             00    ln,i
                             00    wdct
                             00    ra,i
                             00    fwa

ABORT k                                                     C = 48

                          +  63    48*8
                             03       0
                             VFD   H48/k

WAIT                         like EXIT                      C = 49

LOVER u,f,n                                                 C = 50

                          +  63    50*8
                             03       0
                             x0    n,i
                             00    u,i

   x occupies bit 47:   x = 0   Overlay,  x = 1   Segment
```

```
                                                                          Call Code

MEMBER u                          like STATUS                              C = 51

BUFFER u,a,c                                                               C = 52

                               +  63    52*8
                                  03       0
                                  00       0
                                  00    u,i
                                  00    (b)a
                                  00    (b)c

ENTER u,ra                        like CHECK                               C = 53

FAMILY u,f                                                                 C = 54

                               +  63    54*8
                                  03       0
                                  00    f,i
                                  00    u,i

INVOKE ln,ea,ia                                                            C = 55

                               +  63    55*8
                                  03       0
                                  00       0
                                  00    ln,i
                                  00    ea,i
                                  00    ia,i

WHERE                             like EXIT                                C = 56
DYSTAT u                          like STATUS                              C = 57

READY u,ra,ia                     like BSPR                                C = 58

XFER u,ln,lun,ra                                                           C = 61

                               +  63    61*8
                                  03       0
                                  00    ln,i
                                  00    u,i
                                  00    ra,i
                                  00    u,i

SIWOH ln                                                                   C = 62

                               +  63    62*8
                                  03       0
                                  00     ln,i
                                  00      00
```

```
                                                                  Call Code
CLBCD bufa,bufb                                                   C = 63

                          +  63    63*8
                             03       0
                             00    bufa,i
                             00    bufb,i

BYNBY ia                                                          C = 64

                          +  63    64*8
                             03       0
                             00      00
                             00    ia,i

BINBCD                    like EXIT                               C = 65

LOADBO ln, fwa, ra, ia                                            C = 67

                          +  63    67*8
                             03      00
                             00    fwa,i
                             00    ln,i
                             00    ra,i
                             00    ia,i
```

TAPE SCOPE COMPATIBILITY C

Most programs written for the Tape SCOPE operating system will operate under Drum SCOPE. Points of incompatibility are discussed below; a knowledge of Tape SCOPE is assumed.

BACKSPACING INP

Backspacing INP (to correct parity errors, for example) is permitted in Tape SCOPE. Drum SCOPE guarantees only one backspace on INP.

BOUND and UNBOUND REQUESTS

In Tape SCOPE, BOUND requests are stacked by the system such that an UNBOUND request restores a previously selected bounds limit. In Drum SCOPE no such stacking is done. UNBOUND removes the programmer selected bounds.

CONTROL WORD ADDRESS

A control word address of 0 or -0 is legal in Drum SCOPE if it is within the program bounds.

CONTROL WORD CHAINS

Control word chains in Tape SCOPE are unrestricted; the programmer is required to guarantee their correctness. Under Drum SCOPE, all chains are checked to afford system protection and are of restricted length. Control words currently directing data transmission should not be modified until the transmission is complete.

DATE REQUEST

The Q register is used as well as the A in Drum SCOPE.

DEFAULT HARDWARE

In Tape SCOPE, any logical unit not declared otherwise is assumed to be magnetic tape. In Drum SCOPE, any unit not declared otherwise is automatically assigned to the drum. All programmer input units and nondrum output units must be declared in EQUIP statements.

EOF on INP

Although reading past an end-of-file on standard input is illegal in Tape SCOPE, it is permitted in Drum SCOPE.

EQUIP, U=SV

The SV declaration on and EQUIP card is ignored in Drum SCOPE.

FILE OF SCOPE CONTROL CARDS

SCOPE control cards may be included in the range of a FILE card in Drum SCOPE.

ICM, OCM

In Tape SCOPE when ICM = OCM, each RHT entry refers to the AET entry. In Drum SCOPE, one RHT entry refers to the other RHT entry which in turn refers to the AET entry.

JOB AND RUN TIME LIMITS

In Drum SCOPE, the time limit is specified as M.S. (minutes and seconds). If the specification is blank or zero, installation standard time limits are used.

LIBRARY REQUEST

The LIBRARY request has a different meaning in Drum SCOPE because of the revised library format. All such requests must be modified.

LIBRARY STATEMENT

The form and usage of the LIBRARY statement are changed in Drum SCOPE.

LOAD CARD

In Drum SCOPE, the unit is rewound the first time it is mentioned in a phase; thereafter loading starts from the current position. In Tape SCOPE, a backspace file occurs before each load.

LOADER CALLS

The parameters passed to/from the loader are changed in Drum SCOPE.

LOCATION OF EQUIP CARDS

In Drum SCOPE, EQUIP cards may not be intermingled with phase control cards.

LOCATION ZERO

In Tape SCOPE, when an interrupt subroutine is entered, location zero contains the location of the interrupt. In Drum SCOPE, this information is obtained with the WHERE request.

MACHINE INTERRUPT MODE

When an interrupt subroutine in Tape SCOPE is entered, the computer is in interrupt mode. In Drum SCOPE, the machine interrupt system is active at all times.

MAGNETIC TAPE ASSIGNMENT

Magnetic tapes declared on EQUIP cards are not assigned until they are referenced under Drum SCOPE. In Tape SCOPE, EQUIP 11=MT causes a write assignment when the card is encountered.

MEMORY REQUEST

A request in Drum SCOPE for memory below the program limits is illegal. In Tape SCOPE, a default limit is set.

MODE REQUEST

Recording mode selections are made on the master logical unit in Drum SCOPE. In Tape SCOPE, the selection is made on the called logical unit.

NAMED ENTRY CARDS

The SCOPE parameters have been removed from the entry point card.

NORMAL TERMINATION

As part of normal program termination, Tape SCOPE permits the active interrupt subroutine after the EXIT request. In Drum SCOPE, the EXIT request will lock out such interrupt entries.

OPERATOR COMMUNICATION

Procedures and messages involving operator intervention and communication differ from Tape SCOPE.

OVERLAYS

The overlay tape format has been altered. The MAIN card is required in Drum SCOPE.

PAUSE STATEMENT

In Tape SCOPE, PAUSE is executed at the beginning of the specified job. In Drum SCOPE, PAUSE is executed whenever the statement is encountered.

POOLxxxCOMMON BLOCK

Drum SCOPE requires a reserved word of the form .POOLxxx, where xxx are arbitrary characters. The user-specified common block name with this form must be changed. This feature is used to allocate system size buffers.

REQUESTS FROM INTERRUPT MODE

Drum SCOPE does not provide the facility of holding requests; if the unit is busy, the request is rejected.

RUN-PHASE The definition of a RUN is new in Drum SCOPE.

SAVE REQUEST The SAVE, UNSAVE requests are recognized but ignored in Drum SCOPE.

SELECT Drum SCOPE does not permit the SELECT features OPER and M1604.

SEQUENCE STATEMENT The Tape SCOPE Sequence statement has been eliminated. A PRIORITY statement provides an equivalent function.

STATUS REQUEST The format of the reply to the STATUS Request differs in Drum SCOPE.

SYSTEM UNITS The Tape SCOPE systems units 60-80 have been redefined in Drum SCOPE. Units 60-62 are job units; units 65-68, 80 are reserved by the system without specifying actual usage.

INSTALLATION PARAMETERS

The second routine of the Systems File is a BCD routine called PARAMS which contains the systems parameters. Each parameter is represented on a card as follows starting in column 1:

parameter name = value

The parameter name is one of those listed below. An equals sign is used as a separator; a comma may also be used. The value is expressed as a decimal number (0-32767) or an octal number with a B suffix. Embedded blanks are not permitted. The routine ends with a blank card.

The range of permissible values is given with each parameter; suggested values are also provided. It is the responsibility of the installation to use only permissible values for the parameters. The size of RESIDENT is dependent upon the parameter value shown as follows:

R = f (parameter)

This expression is correct to within a constant independent of the parameter value. The total size of RESIDENT cannot exceed 32767.

BLSIZE

Block size for all blocked units.

Range:	129 - 511
Suggested:	256
Size:	R = 2*BLSIZE + 2^{18}*D/(3*BLSIZE) where D is the number of drums in AET

.CWORDS

The number of control words permitted in control word chains for unblocked units.

Range:	4 - 16383
Suggested:	4
Size:	R = .CWORDS*CHANS where CHANS is one greater than the maximum channel mentioned in AET

DRFUDGE

In selecting an optimum of several drum requests, DRFUDGE allows for the system time before the drum can be accessed.

Range:	0 - 32767
Suggested:	5000B
Size:	R = 0

FUDGE

Provides a space between the high point of the minimum system and the origin of numbered common for production programs. This allows space for system expansion at the same time that a production program is called. This space must be large enough to accommodate the following items:

non-resident drivers for resident background programs

non-resident drivers required by central program via equip cards

expansion of central program logical unit block if specified by job card

label area as required by the central program via equip cards

Additional space is desirable for the following items, but if space is short these last two items are removed. Background programs are sacrificed in deference to buffers.

important non-resident background programs and their non-resident

buffers for families 1 and 2

Range:	9 - 32767
Suggested:	4*BLSIZE + 1024
Size:	R = FUDGE

JTIME

Job time, in seconds, supplied in the absence of the t field on the job card.

Range:	0 - 32766
Suggested:	300
Size:	R = 0

LABTERM

Number of seconds allotted to an ABNORMAL subroutine.

Range:	0 - 2236
Suggested:	120
Size:	R = 0

LINP

The quantity LINP/(LSCR + LINP + LPOOL) defines the proportion of the drum area to be reserved for background program output.

Range:	0 - 32767
Suggested:	2
Size:	R = 0

LPOOL

The quantity LPOOL/(LSCR + LINP + LPOOL) defines the proportion of the drum area to be reserved for the library and central program output.

Range:	0 - 32767
Suggested:	13
Size:	R = 0

LSCR

The quantity LSCR/(LSCR + LINP + LPOOL) defines the proportion of the drum area to be allocated for scratch and random access area, subject to the restriction that no more than one entire drum can be allocated.

Range:	0 - 32767
Suggested:	1
Size:	R = 0

NCPFAMS

The number of families permitted to a central program in addition to family 0, that is, the highest permissible family designate.

Range:	0 - 63
Suggested:	7
Size:	R = 6*NCPFAMS

NCPLUNS

The number of logical unit designations permitted to a central program unless overridden by a JOB card. The units INP, OUT, ICM, OCM, and LIB are included.

Range:	2 - 75
Suggested:	10
Size:	R = 5*NCPLUNS

NSEG

Defines the average number of segments per file where 3 is one segment.

Range:	3 - 511
Suggested:	6
Size:	R = (NSEG - 3) * NF

NF = number of files declared on a MASSFILE card.

RLINES

Print limit, in request count, supplied in the absence of the p field on the RUN or LOADMAIN cards.

Range:	0 - 32767
Suggested:	1000
Size:	R = 0

RTIME

Run time in seconds supplied in the absence of the t field on the RUN or LOADMAIN cards.

Range:	0 - 32766
Suggested:	300
Size:	R = 0

STDENS

The standard tape density; 3 = HI, 4 = LO, 6 = HY.

Range:	3, 4, or 6
Suggested:	3
Size:	R = 0

.TIMES

The number of concurrent TIME selections.

Range:	2 - 16383
Suggested:	5
Size:	R = 2*.TIMES

BLOCKED TAPE FORMAT E

Drum SCOPE blocks and buffers user information on the drum. This facility is also available on magnetic tapes; however, the user may not write blocked tape under system control. Preparation and handling of blocked tape outside the Drum SCOPE system is described here.

Information on tape is organized into groups, separated by file marks. The BOT and EOS groups mark the beginning and end of the reel of tape and are not part of the data. The SID group identifies the data contained in the DATA group.

Data on blocked tape is formatted in strings; when a tape is introduced into the system, each string is processed according to its type. The tape may contain several data strings; but each must be complete; that is, a data string may not begin on one reel and continue on another.

BOT GROUP

The first physical record on each reel of the file is the BOT group. It contains one record, the label, followed by an end-of-file mark. The label must be in the standard label format, given in Chapter 5, and the file name must begin with an asterisk in column 9. The reel number corresponds to the physical reel sequence number.

SID GROUP

This group follows the BOT group and may also follow a DATA group. It identifies the data string which follows, and consists of a 10-word BCD record followed by an end-of-file.

Positions 1-3	SID
4-3	(blank)
9-10	mnemonic which classifies the disposition of the following data string
11-80	irrelevant

EOS GROUP

The EOS group follows the last data group on the tape. It consists of a 10-word BCD record followed by a file mark.

Positions 1-3	EOS
4-8	(blank)
9-80	irrelevant

DATA GROUP

This group immediately follows the SID group which identifies it. It consists of one or more binary records in constant length blocks. The length of a block is determined by the installation, between 129 and 511 words, inclusive.

The first word of each block is a block control word:

Bit position	Contents
47-45	zero
44-30	sequence number
29-15	predecessor block sequence number
14-00	successor block sequence number

The first block of data contains a sequence number of 1, predecessor block sequence number of 0, and successor block sequence number of 2; these numbers are incremented by 1 for each successive block. The last block contains a successor block number of 0. The data group is terminated with a file mark.

Spanning blocks are records. Record control words appear at the beginning of each record and also at the beginning of each block immediately following the block control word. This record control word contains:

Bit position	Contents
47	file flag
46	continuation flag
44	BCD flag
38-24	number of truncated trailing blanks in BCD, number of zeros in binary.
14-00	segment length

The continuation flag is 0 for the first record control word of each record. If a record cannot be stored entirely within the block which contains the initial record control word, the second and sebsequent segments of the record are preceded by a record control word with continuation flag 1. The actual data record is composed of all the segments; thus, any length record may be represented.

The file flag is 1 if the record consists of an end-of-file. It is relevant only in the first record control word. If the record length is non-zero, the data contained within the record will be transmitted. Drum SCOPE will write a one-word record consisting of 1717 0000 0000 0000, and will set the binary flag to 0.

The BCD flag contains a 1 if the record is BCD. It is relevant only on the first record control word. Drum SCOPE writes an end-of-file mark in BCD when the BCD flag is 0.

The binary flag contains a 1 if the record is binary. It is relevant only on the first record control word. Drum SCOPE writes an end-of-file mark in BCD when the binary flag is 0.

The segment length defines the number of words of the record immediately following the record control word. If a record control word occupies the last word in a block, the segment length is zero.

DIAGNOSTICS

LOADER DIAGNOSTICS

All loader diagnostics are written on OUT in the following format:

LOADER ERROR p_1 p_2 p_3

p_1, p_2, p_3 are eight-character designators specifying information concerning the detected error.

When the loader detects an error, the diagnostic is written on OUT, and the loading continues as though the error had not been encountered. For example, after writing, the checksum error is processed as though the checksum were correct. The only exception is the memory overflow error which causes the loading operation to be terminated. When errors are encountered during the loading operation, no library subroutines are loaded.

Following is a list of errors and values for p_2 and p_3 for each error; p_1 will be the first subprogram name encountered by the loader, or p_1 will be blank.

p_2	p_3	Condition
BAD LAT	suppressed	LAT string looping (more than 77776B entries in string.)
BANK	BANK card contains format error	BANK card format error.
CARD SEQ	column 1 and 2 of current card	Card out of sequence, i.e., LAT card followed by EXT card.
CHKSUM	column 1 and 2 of current card	Checksum error; punched checksum on a binary card does not agree with computed checksum.
COM LNG	name of common block	Common block length error; attempt made to define labeled common blocks with different lengths or multiple numbered common blocks in one bank having various lengths.

P_2	P_3	Condition
CD TERM	column 1 and 2 of current card	Previous card not terminated. When a name continues to the next EPT or EXT card, the second card was not found.
EOT	suppressed	End-of-tape encountered in writing an overlay tape.
FEW BRT	suppressed	Either the T portion of a BRT has made a reference past the end of the BRT table or there are fewer BRT's than EXT's.
FEW LAT	suppressed	Either the T portion of an LAT has made a reference past the end of the LAT table or there are fewer LAT's than EXT's.
ILL BYTE	column 1 and 2 of current card	Illegal byte value in R field of the RBD card. Byte 10...0, which is not used, was encountered.
ILLEQUIV	column 1 thru 8 of OCC card	EQUIP,xx = yy EQUIP,yy = xx is illegal.
ILL CHAR	column 1 thru 8 of OCC card	A character which is not octal or blank appears in a correction or an illegal relocation designator is used.
ILL EOF	suppressed	End-of-file read from an INPUT UNIT does not immediately follow a TRA card.
ILL PNCH	column 1 thru 8 of OCC card	Illegal punch configuration appears on a OCC.
LAT OV	LAT(T)	Computed value of T, when entering an LAT into the permanent loader tables is greater than 4095.
LAT RNG	address of attempted reference	LAT range error. An attempt was made to reference SCOPE in a LAT string.
LOAD ADD	column 1 and 2 of RBD card or column 1 thru 8 or OCC card	Illegal load address on RBD card or OCC card. The byte for the load address specified either a fixed or decremented address on RBD card. An OCC load error may be one of the following:

P_2	P_3	Condition
		Blank load address. Relocation designator specifies: decremented address fixed address numbered common
MAIN C	suppressed	No main card in overlay processing.
MD BANK	EPT name	Multiply defined bank for EPT. Bank specified for EPT on bank statement does not agree with the bank assigned to the entry point.
MD C BK	name of common block	Multiply defined common bank. Common block assigned to two or more banks of memory by BANK statements or automatic assignment of block followed by a BANK statement.
MD EPT	entry point name	Multiply defined entry point. Same entry point name defined at two addresses.
MD P.N.	suppressed	Multiply defined program name. Same subprogram name encountered more than once.
MD TRA	illegal transfer name	Multiply defined transfer name. More than two TRA cards containing a name encountered.
OVERLAY	overlay number	Current overlay card violated overlay rule.
OV MEM	column 1 and 2 of current card	Memory overflow. Not enough memory available to assign block of common, subprogram, or program extension.
OVT MEM	column 1 and 2 of current card or 1-8 of OCC	Memory overflow. Not enough memory available for construction of loader tables.
PARAM	A register parameter for current loader call	Parameter error. An illegal call has been made to the loader: S = 11 call after loader has completed previous load request. S = 10 or 00 call before loader has completed previous load request. S = 10 or 00 and QU = 0.

p_2	p_3	Condition
PARITY	suppressed	Non-recoverable parity error encountered on loading unit or overlay tape.
ROOMLOAD		No room in core for loader after loader request.
SEC LIM	column 1 thru 8 of OCC card	Current value of load address lies outside program section being corrected. Program section is defined as a portion of the program referenced by a relocation designator of + or 1-9, 0 on OCC.
SEGMENT	overlay number followed by segment number	Current segment card violated one rule.
SEQ NO	column 1 and 2 of current card	The sequence number on an EPT, BCT, EXT, LAT or BRT was out of sequence.
TAPE NO	requested tape number	Logical unit on LCC is out of range.
UN COM	column 1 and 2 RBD card or column 1-8 of OCC	Undefined common block reference. No common block was declared for a byte value on the RBD card or a relocation field on an OCC card. Or the associated common block resulted in a COM LNG error.
UN EXT	external symbol	Undefined external symbol. A reference has been encountered to an external symbol that is not defined as an entry point to any of the subprograms loaded or to any library subroutines.
UN TRA	transfer name	Undefined transfer name. The symbolic transfer name contained on one of the TRA cards was never defined as an entry point to a subprogram.
127 BLK	column 1 and 2 of current card	Too many common blocks. More than 126 common block name have been encountered in a subprogram.

SNAP/TRACE CARD DIAGNOSTICS

If an error occurs while processing SNAP or TRACE cards, the following message is written on OUT, followed by the card image:

CARD yyy	(SNAP) (TRACE) ERROR at Column XXX mmmmm
yyy	Number of card
XXX	Column being processed when error was detected
mmmmm	Diagnostic listed below
BIG NUM	Octal value more than 6 digits
BIG PAR	Parameter was greater than 4096
EXC CARD	Required field missing
FWA	FWA exceeds LWA
ILL MODE	Illegal character in mode field
ILL OCT	Illegal character in octal field
ILL PAR	Illegal character in parameter field (non-numeric)
ILL SLSH	Slash appears illegally
LOC1 OCT	Initial location is absolute octal
LOC1 REL	Initial location is relative
LOC2 OCT	Location is absolute octal
NO LOC2	No LOC2 on TRACE card
NO LWA	No last word address
NO ROOM	Not enough available memory in program bank for tables
NOT LOAD	Name not loaded
REL ERR	Relative address with no preceding name
LOC1, BIG	LOC1 greater than/equal to LOC2 on TRACE card

JOB ABANDONMENT

When a job is terminated abnormally, a diagnostic and an optional dump will be written on OUT in the form:

ABORT, mmmmmmmm, MESS

mmmmmmmm is a diagnostic key and MESS is the message in the routine DIAGNO corresponding to the diagnostic (search) key.

The messages may be modified by changing DIAGNO. If more than one card in DIAGNO has the same key, the second and all subsequent cards will be listed without the diagnostic key.

If the job is terminated because of an illegal I/O request or operation an additional diagnostic line will follow, in the form:

LUN = nn

nn is the logical unit number.

If an abnormal subroutine is used which aborts, a diagnostic of the same form will appear preceded by ABNORMAL INTERRUPT ROUTINE ABORTED. The diagnostics that follow appear on the listings in full capitals. They are shown here in upper and lower case for readability.

ASSEMERS	Library phase errors so loading not attempted
BAD FAM	Attempt to declare buffers for FAMILY 0
BAD FEAT	Select feature invalid
BAD UNIT	I/O equipment failure
BADDECLR	Illegal declaration on EQUIP card
BADDEM	Error on demand card
BADEQUIV	Illegal equivalence on EQUIP card
BADFAMCD	Error on FAMILY card; the FAMILY card has two
BADFAMCD	forms, (1) FAMILY (N), F=U, U,U--- (2) FAMILY,
BADFAMCD	F=U, U,U---in form (1) there may be no) otherwise
BADFAMCD	The error is one of the following:
BADFAMCD	No comma preceding F or no = after F
BADFAMCD	F greater than maximum number of families (NCPFAMS)
BADFAMCD	U=63-68, 70, 80 = infinity
BADLABEL	Illegal label on EQUIP card

BADLIBCD	Library card not proper
BADMODHD	The first card of a module on the library file is
BADMODHD	illegal either it was a BCD ward or a control card
BADMODHD	Other than *BIN or *BCD
BIG FAM	FAMILY number too large
BIGXDIR	Not enough space to make auxillary library
BIGXDIR	Either directory too large or *BIN module too large
BOU CWD	Input control word out of bounds
BOUNDS	Bounds fault
BSR FO	Too long a backspace on a FO unit
CHAIN L	Control word chain too long
COREUSED	UC card preceded by loading operations
CORNONAME	No 7/9NAME card to execute a correct sequence
COSYCARD	Improper control card
ENDLESS	Endless control word chain
EXT EPT	EPT card came after UC cards in program module
FORCDUMP	Dump asked for on UC card
ILBINBIN	An illegal binary card in *BIN module (24-36) card
ILL FO	Illegal request on FO unit
ILL INP	Illegal request on INP (60)
ILL INST	Illegal instruction
ILL IOCM	Illegal request on ICM/OCM
ILL LIB	Illegal request on LIB
ILL LNAME	Invalid format for library name
ILL LUN	Illegal logical unit number on UC card
ILL REQ	Illegal systems request
ILL SCR	Illegal request on scratch unit (50-59)
ILL UNIT	Invalid unit
INSUF EQ	Insufficient equipment
JOBCDBAD	Improper JOB card. Probably one of the following:
JOBCDBAD	JOB) JOB (and no) JOB (N)X where X ≠ ,
JOBCDBAD	JOB, C, I, MX or JOB, C, I, M, SX where X ≠,

JTIMESUP	Job time limit exceeded
LAB FUL	Label ordinals used up
LAB NBOT	Label request issued on unit not at BOV
LAB TEXT	Incorrect label text
LCWBAD	Control word on overlay tape was improper
LCWBAD	LWA exceeded 77777 in specified bank
LINCT	Line count exceeded
LMEML	Bank zero memory limit exceeded on overlay tape
LNOOV	Specified overlay could not be located
LNOPART	Specified partition could not be located
LNOSEG	Specified segment does not exist in current file
LOADERER	Errors in loading
LOADILL	Attempt to load following a SNAP or TRACE card
LONGEPT	Name on EPT card longer than 8 characters
LONGJT	The time requested on the job card is
LONGJT	Greater than or equal to 99900 seconds
LOVLIB	Library overlay not coded yet
LPARITY	Irrecoverable parity error detected on overlay tape
LUB BIG	Not enough space for logical unit block
MODE BAD	Illegal feature in mode
NO 7-9	Control card with no 7-9 punch
NO DR	No DR in hardware type list
NO ENT	No entry point specified
NO .U=	EQUIP,· U= not proper
NOCOMMA	Field not terminated by a comma
NOCORE	Not enough core available for demands
NOEQUIP	Not enough equipment in system for demands
NOHTL MT	No MT in hardware type list
NOIDC	IDC (31) card out of place. Either there
NOIDC	Were two IDC (31) cards or some card
NOIDC	Other than LCC (30) cards preceded it
NOJOBCRD	No job card on in string

NOLOAD	Nothing loaded before RUN card
NONAME	No name on CORRECT card
NONAMLD	This name was not loaded
NONBCD	Non-BCD card in *BCD module
NONBIN	Nin-binary card in *BIN module
NONBINPR	Non-binary card in program module
NONE	Operator typed in none. Not enough equipment
NORASPAC	No random access area left
NOSPACE	Not enough space for program
NOXLIB	The auxiliary library file asked for was not on the unit
OPSAYNO	Operator said not enough equipment for demands
OPTERM	Operator termination
PAR BNDS	Parameter address out of bounds
REOV	Read attempt beyond end of information
RTIMESUP	Run time limit exceeded
SEL IA=0	Select interrupt address = 0
SMALCARD	Control card had less than 20 words
SMALRA	Not enough random access area for RA declaration
SREF CWD	Storage reference fault in control word
STO REF	Storage reference fault
TIMESUP	Time allowed for abnormal termination
TIMESUP	Clean-up has elapsed (LABTERM)
TOO MANY	Too many I/O units referenced
UN UNIT	Read attempt on unassigned unit must have EQUIP declaration
UNEXEOI	Premature end of input information
UNL LAB	Label request on unlabeled unit
UNLOAD	Illegal logical unit number on LOAD card
UNLOADM	Illegal logical unit number on LOADMAIN card
USAGE	Request incompatible with define usage
W-R SEQ	Write followed by read
WRONGEN	This name does not correspond to a loaded entry point

CONSOLE SCOOP

A console scoop will be given in the following form:

```
               REGISTERS                  LOGICAL UNITS
          ENABLE      DISABLED      LU HD F   A STATUS     Q STATUS    MODES

   (ZERO) XXXXXXXXXX   XXXXXXXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX
†CONTENTS OF X XXXXX   XXXXXXXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX,XX
        A XXXXXXXXXX   XXXXXXXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX,XX
        Q XXXXXXXXXX   XXXXXXXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX
        D XXXXXXXXXX   XXXXXXXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX
       BR XXXXXXXXXX   XXXXXXXXXX   XX XX  MASTER  (if equivalenced)
       B1      XXXXX        XXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX
       B2      XXXXX        XXXXX   XX XX XX  XXXXXXXXXX   XXXXXXXXXX  XXX,XX
       B3      XXXXX        XXXXX
       B4      XXXXX        XXXXX
       B5      XXXXX        XXXXX
       B6      XXXXX        XXXXX
       SC      XXXXX        XXXXX


                FAULTS
          FAULT   MASK   SELECT

  SHIFT     XXX    XXX     XX
 DIVIDE     XXX    XXX     XX
   EXOV     XXX    XXX     XX
   EXUN     XXX    XXX     XX
   OVER     XXX    XXX     XX
   ADDR     XXX    XXX     XX
  TRACE     XXX    XXX     XX
   INST     XXX    XXX     XX
 ABNORM     XXX    XXX     XX


                CONSOLE

   TWOS     XXX
    SS1     XXX
    SS2     XXX
    SS3     XXX
    SS4     XXX
    SS5     XXX
    SS6     XXX
    JK1     XXX
    JK2     XXX
    JK3     XXX
```

†Contents of location pointed to by (zero).

REGISTERS Contents of registers at last interrupt in the program ENABLED and DISABLED, registers corresponding to the mode. An * appears next to the mode in execution when the job is aborted. (ZERO) contents of zero at last interrupt in the program. A, Q, D, BR, B1-B6, SC contents of A, Q, bounds register, B-boxes and shift count register.

Logical Units Status of each logical unit used by the program:

LU	logical unit number
HD	hardware type
F	family number
A, Q Status	status
MODE	mode of unit; BIN, BCD, FO, RO, WO, RV
FAULTS	ON or OFF
MASK	ON or OFF
SELECT	NO or YES (fault selected) or HELD
CONSOLE	status of console, OFF or ON

SUPPORT PROGRAMS G

Support programs are library routines available to the central program user. Included in this catagory are LIBEDIT (chapter 9) FILE, and file manipulation.

FILE

The file program transfers data for INP (Unit 60) to some other unit. Output for FILE is copies of the cards, no listings will result.

This program is called by a named entry card in the format:

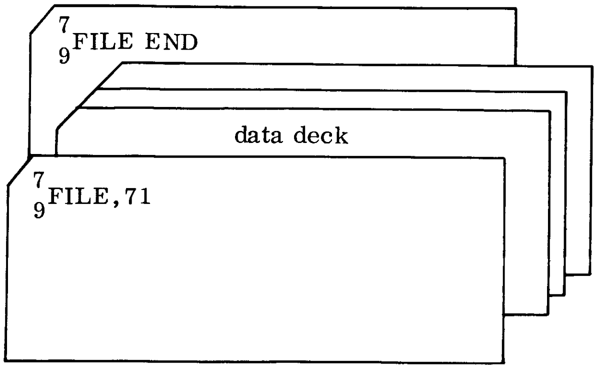
${}^{7}_{9}$FILE,u

u is a unit number 1-49,69,71-79

The FILE control card immediately precedes the data cards to be filed on the unit specified. The data cards must be binary cards. The data deck is terminated by an end-of-file mark or by a binary card containing:

${}^{7}_{9}$FILE END

Control is then returned to the system for job continuation.



File Manipulation

The file manipulation program performs indicated functions on the specified unit. If an invalid request is issued, the system terminates the job. The following functions may be performed:

REWIND, u	Rewind tape
BSPF, u	Backspace to end-of-file
SKIP, u	Skip to end-of-file
UNLOAD, u	Unload tape
BSPR, u	Backspace one record
MARKEF, u	Write end-of-file
RELEASE, u	Release assignment

INDEX

CONTROL DATA

CORPORATION

COMMENT AND EVALUATION SHEET

3600/3800 COMPUTER SYSTEM

Drum SCOPE Reference Manual

Pub. No. 60059200B Revised - July, 1967

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.

CUT ALONG LINE

PRINTED IN USA

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
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

CUT ALONG LINE

MD248

FOLD FOLD

STAPLE STAPLE

CONTROL DATA

1/2 3/4 1 1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

3600/3800 DRUM SCOPE REFERENCE MANUAL

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60059200

Litho in U.S.A.