CONTROL DATA

1604/1604-A COMPUTER

1604/1604-A

CO-OP MONITOR/PROGRAMMER'S GUIDE

CO-OP MONITOR/PROGRAMMER'S GUIDE

CONTROL DATA CORPORATION
8100 34th Avenue South
Minneapolis 20, Minnesota

ACKNOWLEDGMENT

Specifications and implementation of the CO-OP Monitor was a joint project between the CO-OP Programming Committee and Control Data Corporation Applications.

CONTENTS

INTRODUCTION

The CO-OP Monitor allows more efficient use of the Control Data® 1604/1604-A computer by providing such features as job processing, input/output control, interrupt control and debugging aids. In addition, certain routines within the monitor may be used to accomplish specific programming tasks.

JOB PROCESSING

Any programming job to be run under the monitor may consist of a combination of assemblies, compilations, and executions. At the beginning of each job the monitor assigns the input/output equipment required by the job, and records accounting information such as the accounting number and programmer's name or initials. The monitor controls the compilation, assembly and execution of all programs, subprograms and subroutines for each job. The loading of object programs is handled by the monitor loader, which controls storage address assignments and the loading and linking of required subprograms and subroutines. As each job is completed, the monitor computes and outputs the elapsed time. Provision is made for operator intervention if necessary. If the program, including required data storage, is too large for memory, the programmer may subdivide it into overlays and segments.

DEBUGGING AIDS

Debugging aids provided by the monitor include:

Error diagnostics which are printed out on the standard output unit when a program is compiled or executed.

SNAP control cards that designate the contents of control and index registers and specified portions of storage which are to be printed at specified times during execution of a program.

A memory map which can be requested when a program is loaded. This indicates the storage assignments made for the main program and associated subprograms, subroutines and data storage (common) areas.

If an error condition is sensed by the monitor (such as a request for a non-existent input/output device) the monitor terminates the entire job, and provides a limited or extensive dump of control registers and memory as requested by the programmer.

INPUT/OUTPUT CONTROL

Input/output is accomplished by the monitor subroutine READ*/WRITE*. This routine performs all necessary data conversion, and calls the equipment driver routines necessary to activate and transfer data to or from an external device (such as a tape drive) . The programmer may assign any arbitrary number from 1 to 49 to the input and output units used in the program. The unit is always referenced by this logical unit number, which has no relation to the physical unit assigned by the monitor. The assignment of physical units to the programmer-assigned logical unit numbers is done by the monitor at the beginning of each job. Magnetic tapes are normally assigned unless the programmer specifies differently in his program. READ*/WRITE* is used by both the monitor and the programming systems controlled by the monitor such as FORTRAN and COBOL.

INTERRUPT CONTROL

Interrupt processing on the 1604 is simplified by the monitor subroutines SELECT* and REMOVE*. The machine language programmer uses calling sequences to these subroutines to select or remove internal and external interrupts. Compilers such as FORTRAN and COBOL also use these routines for interrupt processing.

CO-OP MONITOR DESCRIPTION 1

The standard CO-OP Monitor consists of a main control system, called a job sequencer, several subordinate control systems, and a library of various assemblers and compilers such as CODAP-1, FORTRAN and COBOL. Also included in the library is a comprehensive set of function subroutines. All of these control systems, compilers and subroutines are normally contained on one master tape. More than one system tape, however, can be used. Additional system tapes called DEFINE tapes could contain additional compilers and subsystems. Included on the master system tape are the input/output equipment driver routines† used for all input/output operations, whether under monitor or programmer control.

The job sequencer initiates and terminates each job, providing automatic sequencing from one job to the next. Once a job is initiated, control is turned over to a subordinate control system. When the job is terminated control returns to the job sequencer. The programmer specifies the subordinate control system on a control card; the choice of system is dependent on the task to be performed. The CO-OP subordinate control system is used for compiling and executing FORTRAN, COBOL and CODAP-1 source programs. A sort-merge operation would use SORT as a subordinate control system, and execution of overlay programs would require the LOADMAIN system. Another subsystem, LIBEDIT, is used to generate and edit CO-OP Monitor system tapes. Additional subordinate control systems may be added to the CO-OP Monitor system. The reference manual for a given programming language or system will specify the required subordinate control system and associated control cards. A hierarchy of two of the control systems is shown in figure 1.

INPUT/OUTPUT EQUIPMENT ASSIGNMENT

Input/output equipments are referenced by logical unit numbers 0 through 63. Units 1 through 49 are reserved for programmer use; the remainder are reserved for monitor use. Equipments reserved for the monitor are called standard units and provide the following functions:

Name	Logical Unit Number	Functions
Systems Unit	0	Contains the monitor library (master) tape.

†A driver routine (or driver) is a monitor subroutine that controls a specific type of peripheral equipment. A separate driver is included for each type of equipment at an installation.

Name	Logical Unit Number	Functions
Standard Input	50	All control cards and all source and object programs are read from this unit unless otherwise indicated on a control card. Data may also be read from this unit.
Standard Output	51	All output under monitor control, such as printouts, listings, dumps and diagnostics, is to the standard output unit.
Standard Punch	52	All punched card output from the monitor is to this unit.
Comment from Operator	53	Used by the operator for communicating with the monitor.
Comment to Operator	54	Used for messages from monitor to operator.
Accounting Log	55	Accounting information for each job is output on this unit.
Standard Scratch Units (2)	56,57	Used by the compilers and assemblers for intermediate results, and for load-and-go operations. 57 is used for additional temporary storage during assembly or compilation; 56 is normally used for load-and-go tapes. These tapes may be used by the programmer when not required by a compiler or assembler.

Programmer assigned units are assumed to be low density magnetic tape units unless otherwise specified on the MCS control card (Chapter 2). The MCS control card is used for assigning all programmer logical units. The standard unit assignments made by each installation are entered in an Available Equipment Table (AET) (Appendix A).

If an installation has at least six magnetic tape units and facilities for off-line card-to-tape and tape-to-printer operations, the standard input, output and punch units could be assigned to magnetic tape units for more efficient use of the system. Otherwise, these three units would be assigned as a card reader, a line printer and a card punch. The accounting log is normally output to the paper tape punch in the 1604 console, and the comments to and from the operator units are usually assigned to the console typewriter. The standard scratch units are always magnetic tape units, but are not assigned unless specified on the MCS control card. All other standard units are assigned automatically, and do not require entries on the MCS card.

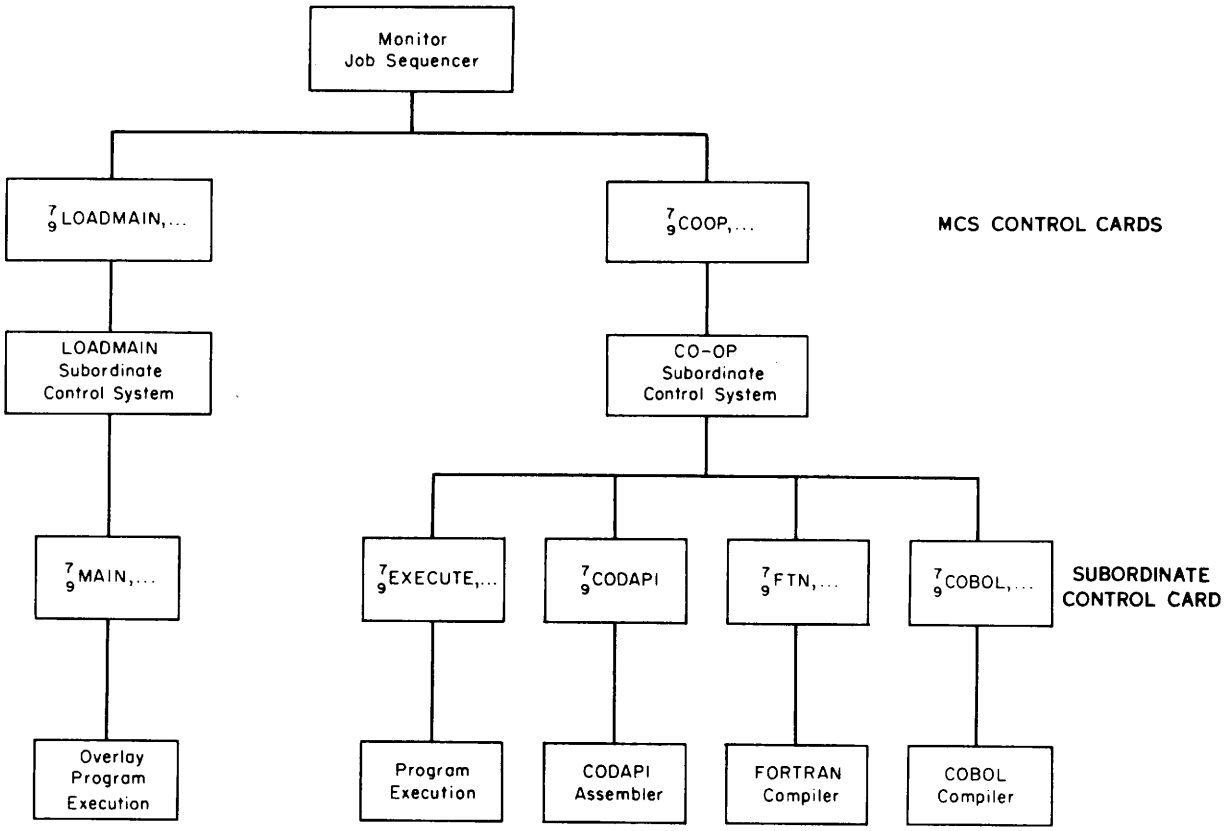
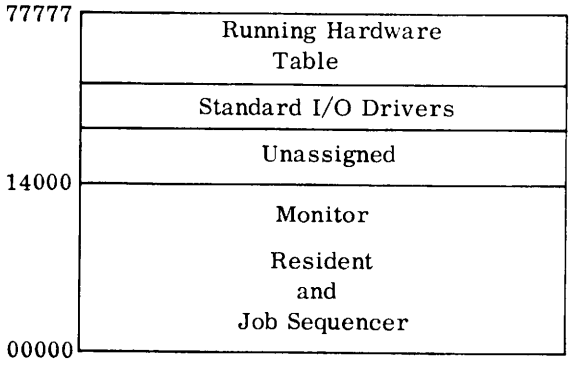


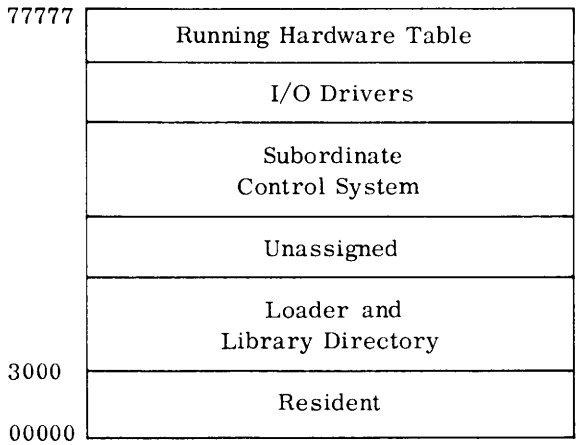
Figure 1. Levels of Control with CO-OP Monitor

PROGRAM STORAGE ASSIGNMENT

When object programs are loaded, the monitor loader routine insures that they are assigned free areas of storage. The memory map for an initial (bootstrap) monitor load is shown below. Addresses when shown are approximate and only indicate the relative magnitudes of the various assigned areas.



When the control cards of a job deck are read and interpreted, internal storage assignment looks like this:



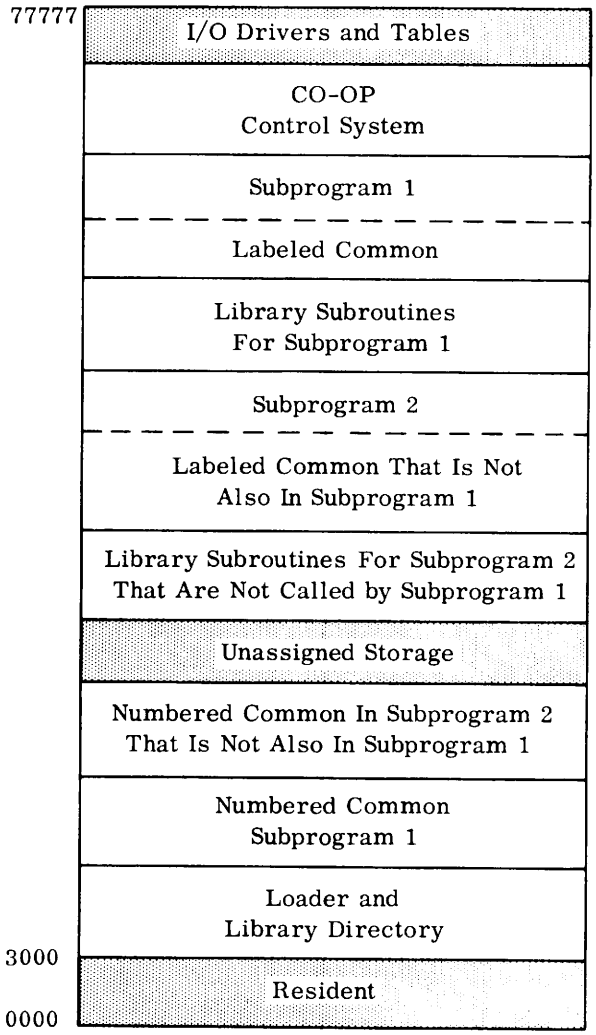
The amount of space assigned to the I/O drivers depends on the number and types of drivers required for each job. The subordinate control system is loaded in the upper part of storage immediately below the I/O drivers. The resident portion of monitor is that part of the job sequence that is required for job processing. The remainder of the area originally assigned the job sequencer is released for program storage.

When an object program is loaded by the monitor loader routine, it is assigned the storage area immediately below the subordinate control system. An object program usually consists of either one single program, or a set of subprograms and subroutines, depending on the requirements of the programmer.

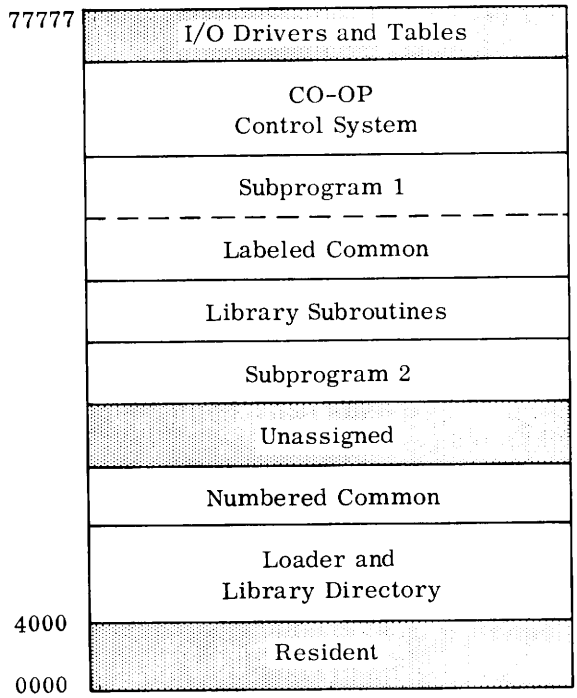
Large programs may be divided into subprograms. These subprograms and required library subroutines are loaded into consecutive storage blocks from top to bottom. Any required data storage is also reserved at this time.

The monitor provides two kinds of data storage, numbered common and labeled common. Labeled common may be pre-set with data when a program is loaded; numbered common cannot. Numbered common storage area is assigned to the lower portion of available memory (next to resident) working up. Labeled common is included in the storage area assigned to the subprogram defining the labeled common area. Labeled and numbered common storage areas are defined by statements in FORTRAN, COBOL, CODAP-1 or other source program languages.

The following memory layout is the result of loading a program that consists of two subprograms controlled by and calling the CO-OP subordinate control system.



If, however, all required number and labeled common areas were defined in subprogram 1, and the library subroutines were called from subprogram 1, the memory layout would appear as follows:



A brief description of the loader routine is given in Appendix C.

The amount of memory available for program storage may be increased by using the subsystem control card EXECUTER or the loader control card RELOCOM which are described in Chapter 2.

JOB PROCESSING 2

To compile and execute programs under the monitor, a source program is punched on cards; and with the addition of the proper control cards, it is submitted as a job for compilation and execution under monitor control. This chapter describes the sequence of events during job processing, as well as the control cards and job deck structure for performing various tasks under monitor control.

MONITOR OPERATION

After mounting the master library tape on the systems unit (0), the operator presses the auto-load button on the 1604 console (or simulates this action); this causes a bootstrap routine from the library tape to be loaded and executed. The bootstrap routine loads the job sequencer and resident portion of the monitor and the monitor enters an idling loop; a series of tones indicates it is idling.

The operator then types a job sequencing message which initiates job processing. These messages are listed in appendix B.

The monitor job sequencer reads the first two cards of the job deck from the standard input unit. The first two cards in a job deck are control cards which provide accounting information, identification of input/output units, time limits, recovery information, and name the subordinate control system to be used. After the control cards have been interpreted, the subordinate control system and required input/output drivers are loaded, and control is given to the subordinate control system.

The subordinate control system reads one or more control cards from the job deck on the standard input unit, and, if required, calls in an assembler, compiler, or special routine (specified on the subordinate system control card) and passes control to this routine. The subordinate control system maintains over-all control of the job until it is terminated.

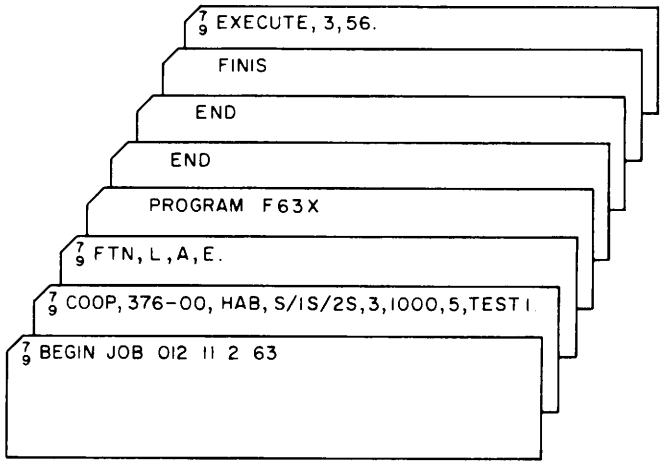
The job runs to completion or terminates if certain major errors are encountered. A description of the control cards for the COOP and LOADMAIN subordinate control systems are contained in this chapter and in Chapter 7. A manual supplement† describes the LIBEDIT system for generating and editing systems tapes.

†Control Data pub. #PSB AE04

MONITOR CONTROL CARDS

A programmer sets up a deck for compilation or execution with a Master Control card, a subsystem control card, and various combinations of END, FINIS, EXECUTE and BINARY control cards. The control card arrangement for compilation and execution of a FORTRAN-63 program is shown below. The Master Control (MCS) card is followed by a subsystem control card, (in this case FORTRAN-63). Next is the source program deck with two FORTRAN END cards. The FINIS and EXECUTE card follow. A data deck may follow the EXECUTE card.

In addition, a BEGIN JOB card precedes the MCS card in a job deck and an End-of-File (EOF) card terminates the job. These cards are normally supplied by the computer operator.



Typical Deck Structure, Compile and Execute

Whenever digits are stacked one above the other in a control card description or example, such as $\frac{7}{9}$, they are to be overpunched in the specified card column (usually column 1).

BEGIN JOB CARD

$\frac{7}{9}$BEGIN JOB xx mm dd yy

The BEGIN JOB card contains the number that is used to identify jobs in a series of job decks, and the date the job was assigned (optional).

Field 1 — 7,9 in column 1, BEGIN JOB starts in column 2, and the field ends in column 10.

Field 2 — xxx, is a three-character alphanumeric number provided by the operator for job identification, starting in column 12.

Field 3 — mm dd yy, which is optional, consists of a six-digit code indicating the date the job number was assigned. A two-digit code for the month, mm, is punched in columns 17 and 18; a two-digit code for the date, dd, in columns 20 and 21, and a two-digit code for the year in columns 23 and 24.

MCS CARD

$^{7}_{9}$COOP,A,I,IO,TL,LL,R,C.

The MCS card provides accounting information, establishes time and output line limits for the job, provides I/O equipment assignment information and specifies recovery procedures in case of abnormal (error) termination.

Field 1 (required) — 7,9 punch in column 1, followed by name of the required subordinate control system starting in column 2. (COOP indicates COOP subordinate control system.) The name is a maximum of eight characters.

Field 2 (required) — Accounting number used by monitor accounting log.

Field 3 (required) — Programmer's name or initials.

Field 4 (optional) — Input/output (I/O) assignment field. (Format is described below.)

Field 5 (optional) — Time limit (TL) estimate in minutes. If time allotted is not sufficient, the job is terminated at the end of the specified time, and a recovery dump indicated in field 7 is printed on the standard output unit. If this field is omitted, a standard time limit determined at each installation is set by the monitor.

Field 6 (optional) — Line limit (LL) estimate indicates the number of lines of output to be written on the standard output unit during the job. This number includes the amount required for assembly or compilation, as well as programmed output. If this field is omitted, a standard line limit determined at each installation is set by the monitor.

Field 7 (optional) — Recovery key (R) indicates one of the six recovery dump procedures listed under Recovery Dump Keys. A zero key is implied when this field is omitted.

Field 8 (optional) — Comments (C) or identification. Printed on standard output unit but not interpreted by the monitor.

A comma terminates each field except the last, which is terminated by a period. The card is free field after column 2. Up to seven continuation cards may be used if necessary; each card must have a 7,9 punch in column 1, and a Hollerith character in column 2. Fields may be broken at any point and continued on the next card. The period indicates the end of the last field used. Imbedded blanks are allowed after column 2 on any card.

In the MCS card example

$^{7}_{9}$COOP,54321-00,ABC,I/1/O/HD02/S/1S/2S,10,1000,1,TEST.

COOP is the subordinate control system, 54321-00 is the accounting number and ABC are the programmer's initials.

The I/O assignment field indicates that logical unit 1 is a low density input tape unit, logical unit 2 is a high density output tape unit, and the job is to be assigned two scratch tape units. The time limit for the job is 10 minutes, the line limit is 1000 lines, and the recovery option is 1.

If optional fields are omitted, the terminating comma for each omitted field must be included. The last field used is terminated by a period.

An example of an MCS card with omitted fields is

$^{7}_{9}$COOP,54321-00,ABC, , , , ,TEST.

In this example the I/O field, time limit, line limit and recovery key fields are omitted. If the comments field had also been omitted, the card would be punched

$^{7}_{9}$COOP,54321-00,ABC.

INPUT/OUTPUT ASSIGNMENT

This field assigns programmer logical units 1-49 and the standard scratch units 56 and 57 to each job. This field must indicate all programmer units required for the job, including scratch units required for compilations and load-and-go operations.

Standard units, other than 56 and 57, need not be defined, as they are always assigned to every job.

Four subfields within the I/O assignment field are as follows:

Subfield

Input (I) — Unit numbers immediately following I in the I/O list are assigned as input-only magnetic tape units. Any attempt to use them for output (unless they also appear in the O

Subfield	
Input (I) (cont)	subfield) will result in immediate job termination, using the specified recovery dump. After the job is terminated, all assigned input-only tape units are rewound with interlock.
Output (O)	Unit numbers immediately following O in the I/O list are assigned as output-only magnetic tape units. Any attempt to use them for input (unless they also appear in the I subfield) will result in immediate job termination. After the job is terminated, all assigned output-only units are rewound with interlock.
Scratch (S)	Unit numbers immediately following S in the I/O list are assigned as scratch units. A scratch tape unit is defined as one that can be used for either input or output. After job termination, all assigned scratch units are rewound to the load point.
Equivalence (E)	The equivalence subfield allows units to be equated to other units in the I/O list, or to standard units. The unit numbers are separated by an = sign; the number on the right must be a standard unit or have been defined in another subfield; the number on the left cannot appear elsewhere in the I/O list.

All programmer units are assigned as low density magnetic tape units unless the unit number is preceded by one of the prefixes listed below. These prefixes can be used in the I, O or S subfield unless otherwise restricted.

Prefix	Definition
T	Typewriter (O field only)
PT	Paper Tape Reader or Punch (I or O field)
CD	Card Reader (I field only)
PR	Printer (O field only)
MT	Low density magnetic tape unit
HD	High density magnetic tape unit
BY	Bypass (input/output statements using a unit prefixed by BY on the MCS card will be bypassed).

The programmer can assign a programmer unit (1-49) to a particular channel by prefixing a unit number with the channel number in the I, O and S subfields. Either the even or odd channel of a channel pair may be designated for any subfield. Unit numbers less than 10 must be written as 01, 02, etc, if a channel prefix is used. The leading zero is not necessary, however, if an equipment prefix immediately precedes the unit number. A unit number can be prefixed by both channel and equipment designators. For example, the I/O list entry 3HD3 indicates that programmer unit 3 is to be assigned as a high density magnetic tape unit on channel pair 3 and 4.

I/O FIELD FORMAT

The format of the I/O field list requires that each element in the list, except the last, be followed by a slash (/). The last element is followed by the terminating comma (or period) for the I/O field. The I subfield is first, followed by the O, S and E subfields. For example in the I/O field

. . . ,I/1/2/O/3/4/S/5/56/57. . . .

units 1 and 2 are defined as input/output units, 3 and 4 are output-only units, 5, 56 and 57 are scratch units. In the I/O list

. . . ,I/CD2/3HD3/O/PR4/S/10/56/57/E/47=50/48=51, . . .

unit 2 is assigned as a card reader, unit 3 as a high density magnetic tape unit on channel 3, unit 4 as the printer, units 10, 56 and 57 as scratch units,† unit 47 is equated to the standard input unit, and unit 48 is equated to the standard output unit. A tape can be used for both input and output, and be rewound with interlock (saved) at the end of the job, by using the same number in both the I and O subfields.

Precedence of I/O Equipment Assignment

The order in which input/output equipment is assigned to each job by the monitor is as follows:

1. All scratch units
2. All unit numbers in both I and O subfields
3. All unit numbers in I subfield only
4. All unit numbers in O subfield only
5. All equivalences

Within each subfield, precedence is as follows:

1. All units having both channel and equipment prefixes
2. All units having only a channel prefix
3. All units having only an equipment prefix
4. All remaining units

If an illegal condition is detected in the I/O list, or if not enough equipment is available for assignment, the entire job is terminated immediately. Possible errors include defining a unit as both an input-only, or output-only, unit and a scratch unit, and using the unit number on the left of an equivalence pair more than once, or in another subfield.

† Within the S subfield only, units 56 and 57 can be indicated as 1S and 2S, respectively. These alternate codes cannot be used in other subfields or in the program.

The following I/O fields are illegal, because of the first error described.

. . . ,I/1/O/2/S/1, . . .

. . . ,I/3/O/4/S/4, . . .

. . . ,I/2/O/2/S/2, . . .

The second error described occurs in this subfield.

. . . ,I/1/E/1=50, . . .

RECOVERY DUMP KEYS

All dumps are on the standard output unit (51) with the console conditions printed first. Dumps occur when the job is terminated because of an error detected by the monitor, by a program transfer to the monitor subroutine ERROR* (Appendix E), or by operator termination of the job.

Key	
0 (or omitted)	Octal dump of console conditions
1	Octal dump of console conditions and numbered common
2	Octal dump of program, labeled common, and console conditions
3	Octal dump of numbered and labeled common, console conditions, and program area
4	Octal dump of console conditions and all of memory except the monitor
5	Octal dump of entire memory and console conditions

The octal dump for console condition consists of:

A Register

Q Register

Program Address Register

Upper address of interrupt control word (or words)

Index registers 1 through 6

Buffer words 1 through 6

The format for data dumps in keys 1, 2 and 3 is four octal data words per line, preceded by a heading as follows:

-ABS-	DATA	CONTENTS
X	R	Y Y Y Y

X represents a 5-digit absolute octal address of the leftmost data word Y; R represents the relative decimal location of the leftmost data word with respect to the beginning of the data block being dumped; and each Y is a 16-digit octal data word. The program format for keys 3, 4, and 5 is eight instructions (four words) per line, preceded by a heading, as follows:

```
-ABS-     PROG.
  X       R       SYMppb  A SYMppb  A. . .SYMppb  A SYMppb  A
```

X is a 5-digit absolute octal address of the leftmost instruction pair; R is a decimal location relative to the start of the program region being dumped; SYM represents a 3-character mnemonic operation code; pp represents the octal M or Y term of the instruction.

For option code 3, the program is listed before numbered common. The data heading for options 4 and 5 has the word MEM in place of the word DATA.

SUBSYSTEM CONTROL CARDS

The format and number of subsystem control cards depends on the subordinate control system being used. This section describes the control (CSS) cards for the CO-OP subordinate control system. The control cards for LOADMAIN (overlays) are described in Chapter 7; the control cards for SORT are described in the SORT reference manual. Representative CO-OP job deck structures for FORTRAN-63, COBOL and CODAP-1 are illustrated later in this chapter. As additional programming languages are added to the monitor, the required control cards and deck structures will be described in the reference manual for each language.

The monitor allows compilation and execution (load-and-go) within a job. Load-and-go tape units are specified on the subsystem control cards. All load-and-go units must be defined either as scratch units or as both I and O in the I/O list on the MCS card. Unit 57 must also appear in the S subfield for compile and execute operations.

FORTRAN-63 CONTROL CARD

The FORTRAN control card causes the FORTRAN-63 compiler to be loaded, and the source deck in the specified input unit to be compiled into CODAP-1 language. This, in turn, is automatically assembled into a binary object program. The programmer may select from a variety of output formats, as well as indicate to the compiler the number of CODAP-1 symbols to be generated during compilation. (This is important only if the CODAP-1 assembler indicates that symbol table overflow has occurred while producing the object program.)

The FORTRAN-63 control card has the following format:

$^{7}_{9}$FTN, option keys.

Field 1 $^{7}_{9}$FTN is punched in columns 1 through 4, followed by a comma.

Field 2 Option keys may appear in any order separated by commas; they are free field and start in column 6. Illegal option keys and extraneous characters are ignored. The option field is terminated by a period at the end of the control card. If no option keys are present, only error messages and the basic assembler headings are printed. Any key can be abbreviated to its first character,

$^{7}_{9}$FTN,L,P,E.

Any option may be followed by = n, where n is a decimal integer. If n is 0, the option is interpreted as if it were not present.

$^{7}_{9}$FTN,LIST=1, E=10.

The description of each option key assumes that =n does not follow the option key. The n code for some options is not recognized unless it is zero.

Option Key		n ($\neq$ 0)
LIST	List source language program.	N/A
PUNCH	Punch binary object program deck on logical unit 52 (standard punch) or n.	Punch binary deck on unit n.
EXECUTE	Write load-and-go tape on unit 56 or unit n.	Write load-and-go tape on unit n.
ASSEMBLY	List assembled program in CODAP-1 language.	N/A
INPUT	Input source program from standard input or unit n.	Input source program from unit n.
TAPE	Assign unit n as assembler scratch tape. No scratch unit assigned if n=0 or option is omitted.	Assembler scratch tape n.
BCD	Punch generated symbolic CODAP-1 cards on standard punch unit or unit n.	Punch generated cards on n.
SYMBOLS	Allot 2048 or n words to assembler symbol table; if option is omitted, or n $\leq$ 1024 words, 1024 words are assigned.	Allot n words to assembler Symbol table (n > 1024).

REFERENCES*	Suppress assembler symbol reference table; if option is omitted, print table.	N/A†
NULLS*	Suppress null listing; if option is omitted, print null listing.	N/A

* Applies only if ASSEMBLY option is present.

COBOL CONTROL CARD

This card causes the COBOL compiler to be loaded, and control to be turned over to COBOL. The source program is read and compiled from the standard input unit. The programmer specifies the outputs required. The format of this card is as follows:

$^{7}_{9}$COBOL, option keys.

Field 1 The first column contains a 7,9 punch; columns 2 through 6 contain the word COBOL.

Field 2 contains the option keys listed below:

Option Key	
Z	Suppress source program listing.
X or 56	Load-and-go tape is written on scratch unit 56.
M	Print data map on the standard output unit.
P	Punch object program in relocatable binary form on standard punch unit.
S	Compile DATA division only.
T	Do not suppress diagnostics for trivial errors.
L	Print symbolic object program listing on standard output unit.

Each key is an alphabetic character (or 56), and is separated from the next by a comma. The last key is followed by a period or a blank. The keys may be specified in any order. If any of these functions is not required, the corresponding key and comma may be omitted.

$^{7}_{9}$COBOL,L,X,P.

† N/A = not applicable

CODAP-1 CONTROL CARD

This card causes the CODAP-1 assembler to be loaded. Control is given to the assembler; and the source program, on the specified input unit, is assembled into a binary object program. The format of a CODAP-1 control card is:

$^{7}_{9}$CODAP1, option keys.

Field 1 $^{7}_{9}$CODAP1 is punched in columns 1 through 4, followed by a comma.

Field 2 The option keys are free field, start in column 9, and may appear in any order separated by commas. Illegal option keys and extraneous characters are ignored. The option field is terminated by a period at the end of the control card. If no option keys are present, only error messages and the basic assembler headings are printed. Any key can be abbreviated to its first character.

$^{7}_{9}$CODAP1, L,P,E.

Any option may be followed by =n, where n is a decimal number. If n=0, the option is interpreted as being omitted.

$^{7}_{9}$CODAP1,L,P,E=10.

Option Key		n ($\neq$ 0)
LIST	List assembly output.	N/A
PUNCH	Punch binary object program deck on standard punch unit or unit n.	Punch binary deck on unit n.
EXECUTE	Write load-and-go tape on unit 56 or unit n.	Load-and-go unit is unit n.
INPUT	Input source program from standard input or unit n. (If option is omitted, unit 50 is implied.)	Source input on unit n.
SYMBOL	Allot 2048 or n words to assembler symbol table, if option is omitted, or n $\leq$ 1024, 1024 words are assigned.	Allot n words to assembler symbol table (n > 1024).
REF	Suppress symbol cross reference table on assembly listing.	N/A
NULL	Suppress listing of nulls in assembled listing.	N/A

EXECUTE CARD

This card is used to load and execute an object program from the specified input unit. The object program may be one that was previously compiled, or it may be the output of a load-and-go operation. For load-and-go units, the unit number on the EXECUTE card must be the same as the unit specified on the compiler or assembler control card, and must be defined as a scratch unit on the MCS card. The EXECUTE card has the format:

$^{7}_{9}$EXECUTE,t,n,m.

Field 1 — In column 1, a 7,9 punch, followed by the word EXECUTE and a comma or period in column 9.

Field 2 (t) — If an execution time limit is wanted, the number of minutes is entered starting in column 10. The monitor considers this limit to be a subset of the time limit specified (or implied) on the MCS card. If load-and-go or other operations have preceded the execution phase, the monitor deducts the time taken for these operations from the time limit obtained from the MCS card. The remaining time is compared to the EXECUTE time limit, and the execution time limit is set to the lesser of the two values. If this field is omitted, the time remaining for the job becomes the time limit.

Field 3 (n) — This field specifies the input or load-and-go unit containing the object program to be loaded and executed. If it is omitted, input will be from standard scratch unit 56 if the previous operation prepared a load-and-go tape. If not, standard input unit 50 is assumed to contain the object program.

Field 4 (m) — The m is a memory map key, 1 or 0. A memory map of the loaded program is normally output on the standard output unit. (See Chapter 6 for format.) This map may be suppressed by setting m to 1. If m is 0 or omitted, a memory map is produced.

Each field is followed by a comma, except the last which is followed by a period. A comma must be included for imbedded omitted fields. For example to omit the time limit field, the card would appear:

$^{7}_{9}$EXECUTE, ,n,m.

To omit the time limit and memory map fields, the card would appear:

$^{7}_{9}$EXECUTE, ,n.

All three fields may be omitted as follows:

$^{7}_{9}$EXECUTE.

In this case, the time limit is the time remaining for the job, the input unit is 56 or 50, and a memory map is produced.

When this card follows a source program deck in a compile and execute operation, the object subprograms on the load-and-go tape are loaded. The loading operation terminates whenever two consecutive transfer (TRA)† cards or an end-of-file is encountered, whichever occurs first. If an end-of-file mark is encountered first, the loader loads additional subprograms from the standard input unit. When two consecutive transfer cards are encountered, either on the standard input unit or the load-and-go unit, the loading process terminates and control is given to the object program.

The placement of the EXECUTE card in a job deck is shown in the deck structure examples later in this chapter. If one or more errors have occurred during compilation or assembly process, the EXECUTE card is skipped over. Errors which occur during the loading of the object program result in loader diagnostic messages on the standard output unit. These messages are described in Chapter 6.

EXECUTER CONTROL CARD

The EXECUTER control card has the same format as the EXECUTE card; it is used when extra memory is required for program storage. The EXECUTER card releases approximately ninety per cent of the CO-OP control system storage area for program use. Only the upper ten per cent of CO-OP remains in storage. No other CO-OP control cards may follow the EXECUTER card in a job deck.

LOAD CARD

This card loads object programs into memory when execution is not wanted. The format is:

$^{7}_{9}$LOAD,n,m.

Field 1 $^{7}_{9}$LOAD is entered in columns 1 through 5, followed by a comma or a period.

Field 2 The number (n) of the input or load-and-go unit appears here. If this field is omitted, the input unit is standard scratch unit 56 or standard input 50 as described for the EXECUTE card.

Field 3 When the memory map (m) is to be suppressed, the m key is 1; if the key is 0 or omitted, the map is produced.

In the example $^{7}_{9}$LOAD, ,1. the input unit is 56 or 50, and the map is suppressed.

In the example $^{7}_{9}$LOAD. the input is 56 or 50 and the map is produced.

† TRA cards are described in Appendix C.

If the LOAD card follows a compilation, it will be skipped over if compilation errors have occurred. Loader error diagnostics are listed in Chapter 6.

An EXECUTE card following a LOAD card will execute the object program loaded by the LOAD card.

DEFINE CARD

The DEFINE card changes the unit assigned to the monitor library tape. This allows programming languages to be on separate (DEFINE) tapes; each tape must be self-contained in that it includes all required library subroutines and a complete directory. The library tape remains defined throughout the job until another DEFINE card is encountered; therefore, at least two DEFINE cards are required if the master library tape (unit 0) is to be used again during the job. The definition is cancelled when the job is terminated. In the format of the DEFINE card shown below, n is the number of the new library tape unit.

$^{7}_{9}$DEFINE,n.

$^{7}_{9}$DEFINE is punched in columns 1 through 7, followed by a comma or a period. If n is not zero, it must appear as an input-only or scratch unit in the I/O list on the MCS card.

BINARY CARD

The BINARY card is used to perform card-to-tape or tape-to-tape operations from the standard input unit to another tape unit, such as a load-and-go tape. The cards in the job deck following the BINARY card are written on the specified logical unit in column binary format (two tape frames per column). The transfer operation continues until another subsystem control card (7,9 in column 1, Hollerith character in column 2) is encountered. This card stops the operation, and an end-of-file mark is written on the tape. The tape is then backspaced over the end-of-file mark. Loader control cards (11,0 and 7,9 in column 1) will be written in column binary format as they are encountered.

The format of the BINARY card is:

$^{7}_{9}$BINARY,n.

$^{7}_{9}$BINARY is entered in columns 1 through 7, followed by a comma or period. The n field which indicates the logical unit to be used is terminated by a period. If n is 0 or omitted, standard scratch unit 56 is used.

If the BINARY card follows a source language compilation, the card-to-tape operation is performed only if there were no compilation or assembly errors. If errors occurred, the BINARY card is skipped over.

This card allows the programmer to generate a tape consisting of a mixture of previously compiled object program decks, and the object program output of load-and-go operations in the current job. This card is also used in creating overlay tapes as described in Chapter 7.

REWIND CARD

This card rewinds to load point the tape units specified in the second field. Care must be taken not to rewind a unit that cannot become ready (such as rewound with interlock). If this is attempted, the job will hang up, but it will not be terminated until the time specified for the job has elapsed. The standard units (except 56 and 57) cannot be rewound by this card. This card cannot be used to rewind the currently defined library tape, even if it has been defined on other than 0.

The three formats on the REWIND card are:

$^{7}_{9}$REWIND, n_1, n_2, n_3, . . . ,n_m.
(n is 1 to 49, 56 or 57)

$^{7}_{9}$REWIND, ALL.

$^{7}_{9}$REWIND.

In each format, the $^{7}_{9}$REWIND begins in columns 1 through 7. The first format rewinds all listed units. The second format rewinds all programmer assigned tape units, including the scratch units 56 and 57, if assigned. The third format rewinds only unit 56. Unassigned logical unit numbers appearing in the second field are ignored.

This card can be used to initialize programmer assigned tape units. It may also be used to rewind a load-and-go unit, so that a program may be repeatedly executed and then rewound using different sets of data.

LOADER CONTROL CARDS

The four loader control cards, RELOCOM, LIBRARY, SNAP, and PATCH can be used in a job deck to increase available memory, load library subroutines, provide periodic (snapshot) dumps during program execution, and patch object programs. RELOCOM and LIBRARY are described below, and the placement of these cards in a job deck is shown later in this chapter. SNAP is described in Chapter 6, and PATCH is described in Appendix D.

RELOCOM CARD

This card releases additional memory for program storage. In effect, numbered common replaces a part of the upper portion of resident (Chapter 1). The following information appears in columns 1 through 9:

11
0
7RELOCOM.
9

LIBRARY CARD

This card loads all required subroutines before the object program is loaded. The LIBRARY card is used when the loader error FM (full memory) occurs. This error indicates the loader is attempting to load program instructions into the area reserved for the loader tables used for address linkage. When the LIBRARY card is used, the loader does not require the library directory. This reduces the storage space occupied by the loader and associated tables, thereby decreasing the storage space occupied by resident. This card also acts as a RELOCOM card in increasing the length of numbered common.

For this card, 11,0 and 7,9 appear in column 1, LIBRARY in columns 2 through 8. followed by a comma and a list of the names of the library subroutines to be loaded. Each name, except the last, is followed by a comma. The last field is terminated by a period. No spaces are allowed in the list.

11
0
7LIBRARY, a_1, a_2, a_3, . . . ,a_n.
9

As many LIBRARY cards as necessary may be used. The list must contain the names of all library subroutines needed by the object program, and the card must be loaded before the object program is loaded.

TERMINATION CARDS

JOB DECK

The last card in a job deck is an end-of-file card which has 7,8 punches in columns 1 and 2 and 12,1,4,7 punches in columns 3 and 4.

MONITOR RUN

To terminate a monitor run, an END MONITOR INPUT card follows the end-of-file card of the last job deck. This card is punched as follows in columns 1 through 18; it terminates automatic job sequencing, and returns the monitor to the idling tone loop.

7
9END MONITOR INPUT

JOB DECK STRUCTURES

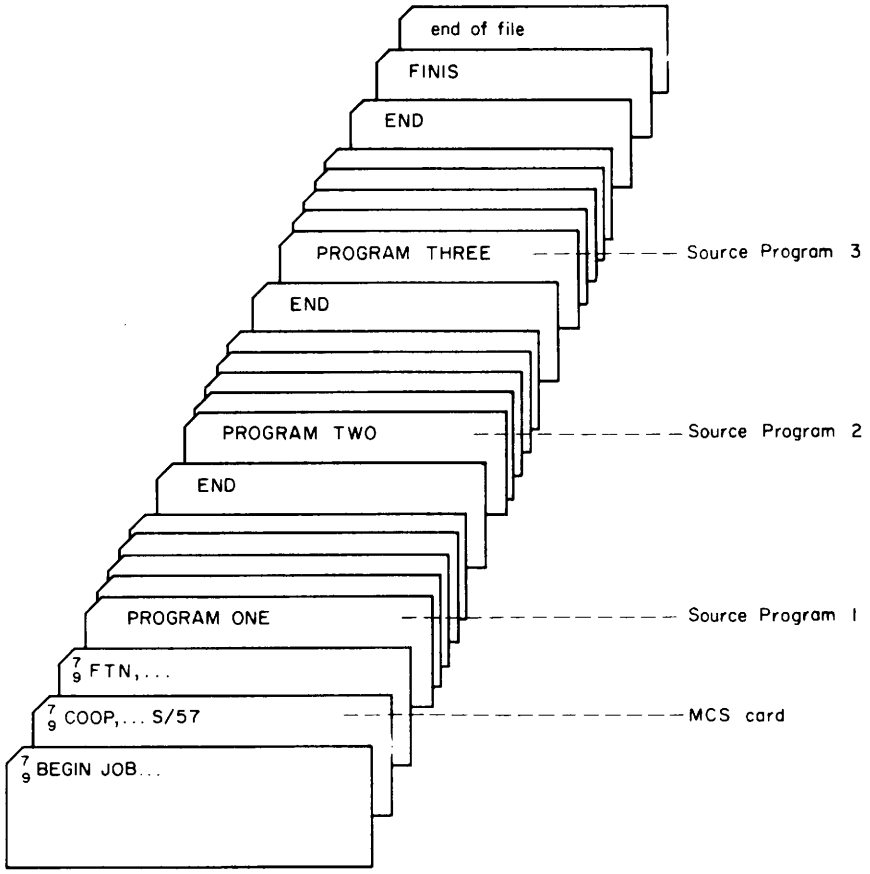
The deck structures in this section illustrate representative job decks for FORTRAN, COBOL and CODAP-1. BEGIN JOB cards and end-of-file cards shown are normally supplied by the operator. The various types of binary cards identified in object program decks are described in Part 2 of Appendix C. Deck structures for other programming systems are described in the programming manuals for these systems.

The data decks shown in the examples are entered into storage by READ statements within the program. The decks shown in this section require that the data be on the standard input unit.

FORTRAN-63

COMPILE ONLY

Following is the deck structure for batch compilations in FORTRAN-63. Each of the object program decks output by the compiler will have one transfer (TRA) card at the end. Since the last subprogram in a program deck must be terminated by two consecutive TRA cards, the second TRA card must be added to the object program deck before execution time.

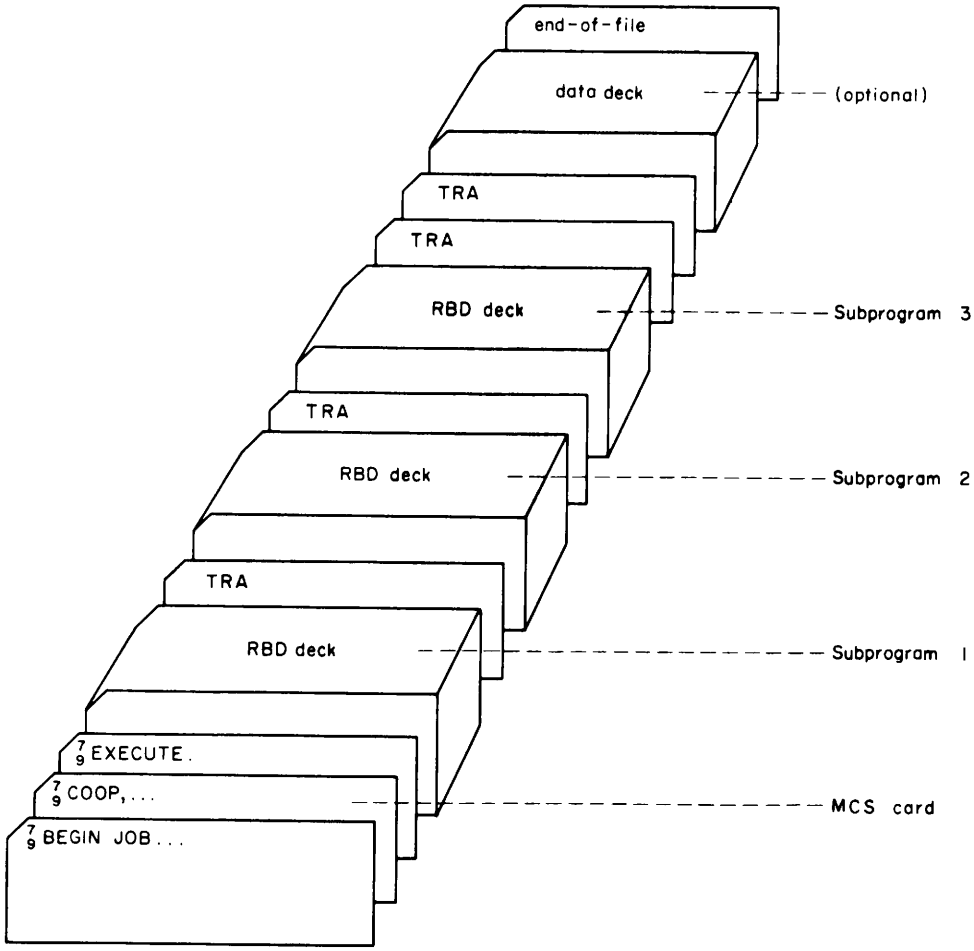


Batch Compilation

EXECUTION ONLY The following decks illustrate single, batch, and repeated execution from the standard input unit. In single execution, a program is executed once during the job in repeated execution, one program is executed more than once during the job. In the second (batch) example two previously compiled programs are executed within one job. Each is executed independently of the other.

Each relocatable binary (RBD) deck in the single execution example contains one transfer (TRA) card; the last deck contains a second TRA card. One of the TRA cards must contain the symbolic transfer address to the entry point at the start of the program. This named transfer card may be any of the TRA cards. See Appendix C for RBD deck description.

Any error that causes an error return to the monitor (such as a loader error or an error in logical unit reference) terminates the entire job. Cards remaining in the deck for the terminated job are skipped over until the end-of-file is encountered. If automatic job sequencing is in progress, the next job will be exectued.

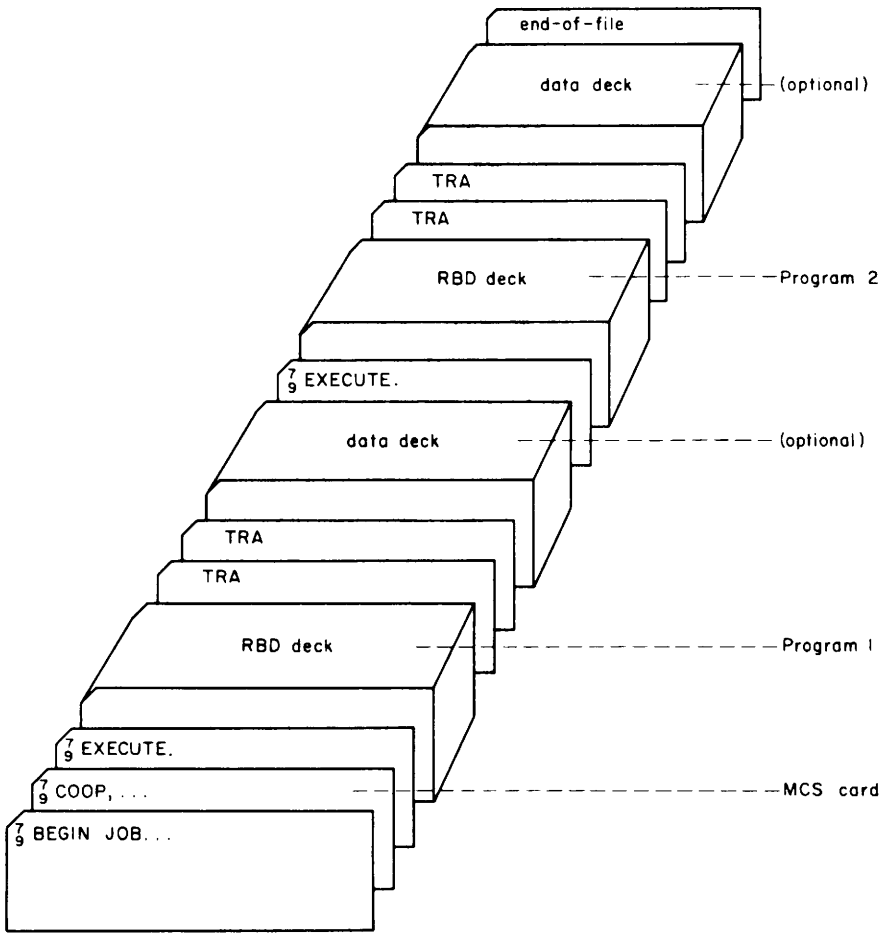


Single Execution

In the preceding example, no load-and-go operations occurred earlier in the job, therefore the EXECUTE card shown implies the standard input unit 50 rather than standard scratch unit 56.

Each RBD deck indicated in the batch execution deck may consist of any number of subprograms, each terminated by a TRA card. The last contains an additional TRA card. One of these TRA cards must contain the transfer address, punched in Hollerith code, starting in column 9.

If data is being read in a buffered read operation, a blank card must follow each data deck that precedes an EXECUTE card.

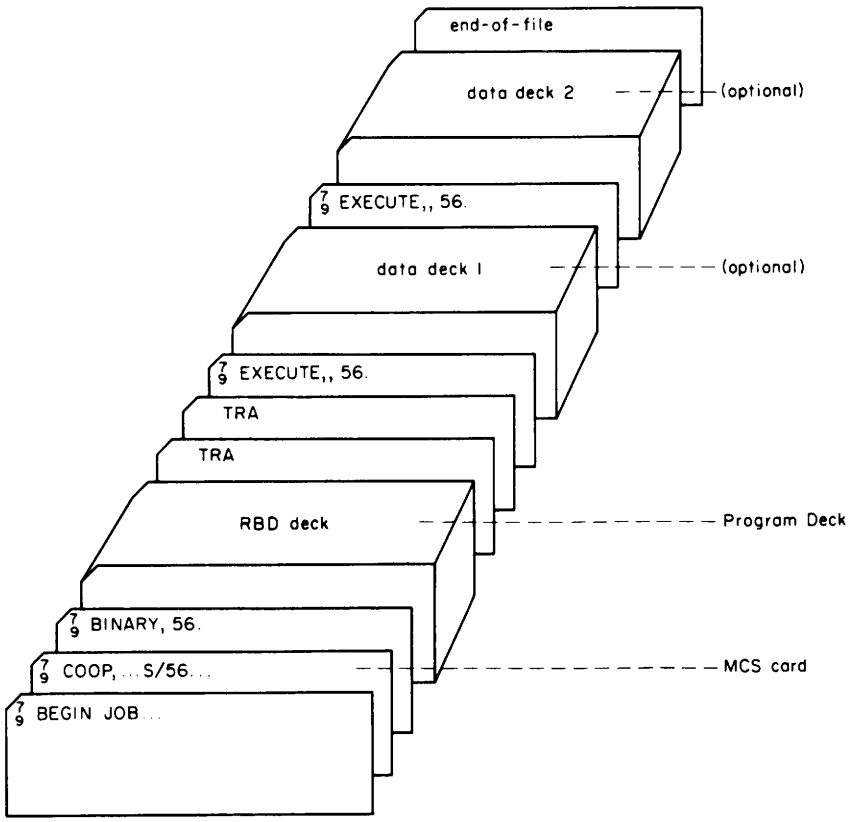


Batch Execution

The repeated execution job deck shown executes one program from scratch unit 56 with two sets of data decks on the standard input unit. The RBD deck terminated by two TRA cards is transferred to scratch unit 56 by a BINARY card. The scratch unit is rewound before each execution by the EXECUTE card inserted before each data deck.

The major difference in deck structure between this operation and the previous executions is the addition of the BINARY card. Data decks are separated by a blank card and an EXECUTE card.

If data is being read in a buffered read operation, a blank card must follow each data deck preceding an EXECUTE card.

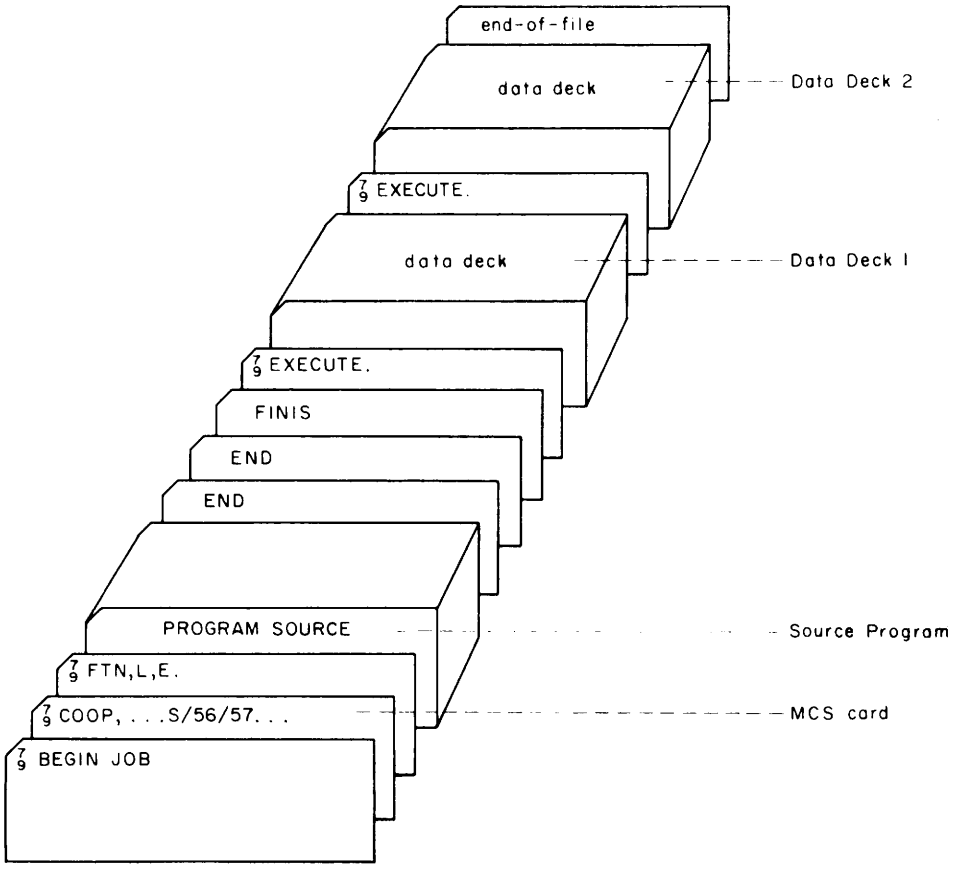


Repeated Execution

COMPILATION AND EXECUTION (LOAD-AND-GO)

To compile and execute a FORTRAN-63 program with two data decks, the following deck structure is used.

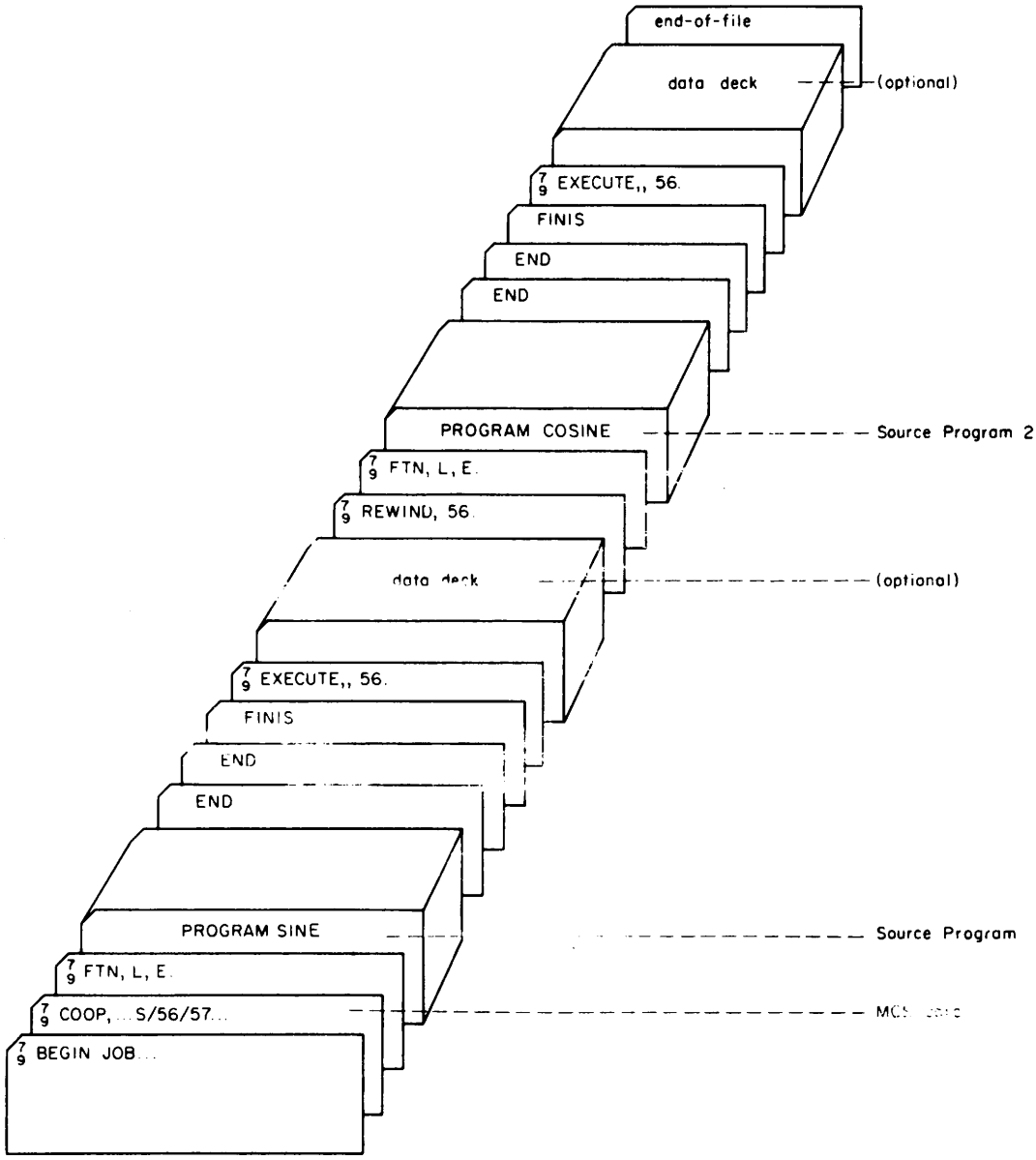
The program is compiled on unit 56, using 57 as an intermediate scratch unit. The data decks are on the standard input unit. The load-and-go unit is rewound each time the EXECUTE card is read, and the program re-executed with the new data deck. If loader or other errors occur causing the monitor to take control, the entire job is terminated and any remaining cards in the job deck are skipped over. If automatic job sequencing is in progress, the next job is executed. The data decks may be omitted. The end-of-file card would then follow the first EXECUTE card.



Compile and Execute One Program with two Data Decks

The following job deck is used to process multiple load-and-go operations within one job. The REWIND card is needed to rewind scratch unit 56 before the second compilation.

If the REWIND card were not used, the program COSINE would be compiled as the second program on scratch tape 56. Then, when the second EXECUTE card was read, unit 56 would be rewound before execution, and the first program on the SINE tape would be loaded and executed.



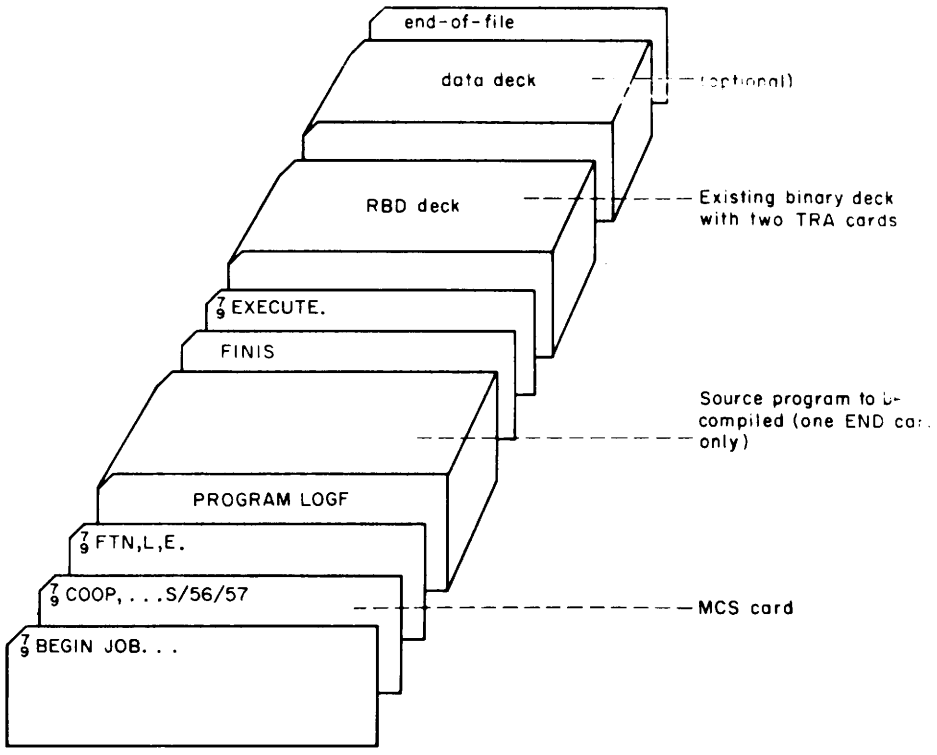
Compile and Execute (Two Programs)

PARTIAL COMPILATION AND EXECUTION

These procedures could be used to recompile or add a subroutine to an existing RBD deck and execute immediately, with or without data.

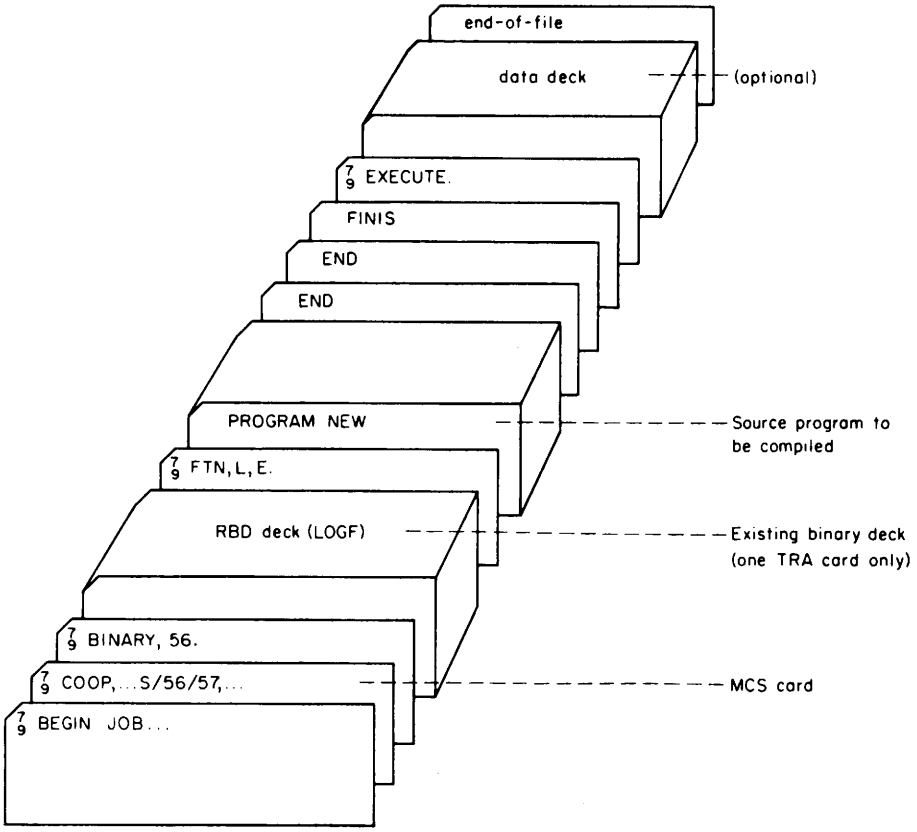
Procedure I loads the subprogram to be compiled before the existing binary deck; execution then takes place. Procedure II loads the existing binary deck first, followed by the newly compiled subprogram or subroutine.

If a special subroutine is to be used instead of an existing FORTRAN-63 library function with the same name, that subroutine must be the first subprogram in the job deck. For example, the program LOGF might be the programmer's own function subroutine. To make certain his routine, and not the library routine LOGF is used, the programmer's LOGF subroutine is the first subprogram in the job deck. If this subroutine is to be within this job, procedure I is followed. If this subroutine exists as a binary deck previously compiled, procedure II is followed. The BINARY card in II performs a card-to-tape or tape-to-tape operation to the load-and-go unit, so that the load routine will find the subprograms in proper order on the load-and-go unit.



Partial Compilation and Execution

Procedure I



Partial Compilation and Execution

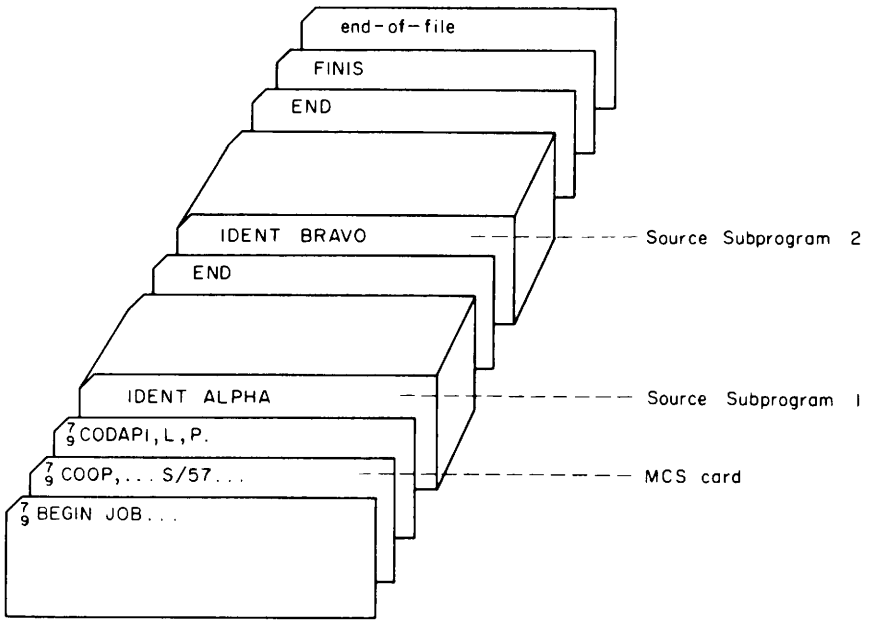
Procedure II

CODAP-1

The deck structures for CODAP-1 are the same as those for FORTRAN-63, with the CODAP1 card replacing the FTN card. END cards, however, are punched starting in column 10. Execution only decks are identical; only basic operations are shown.

COMPILE ONLY

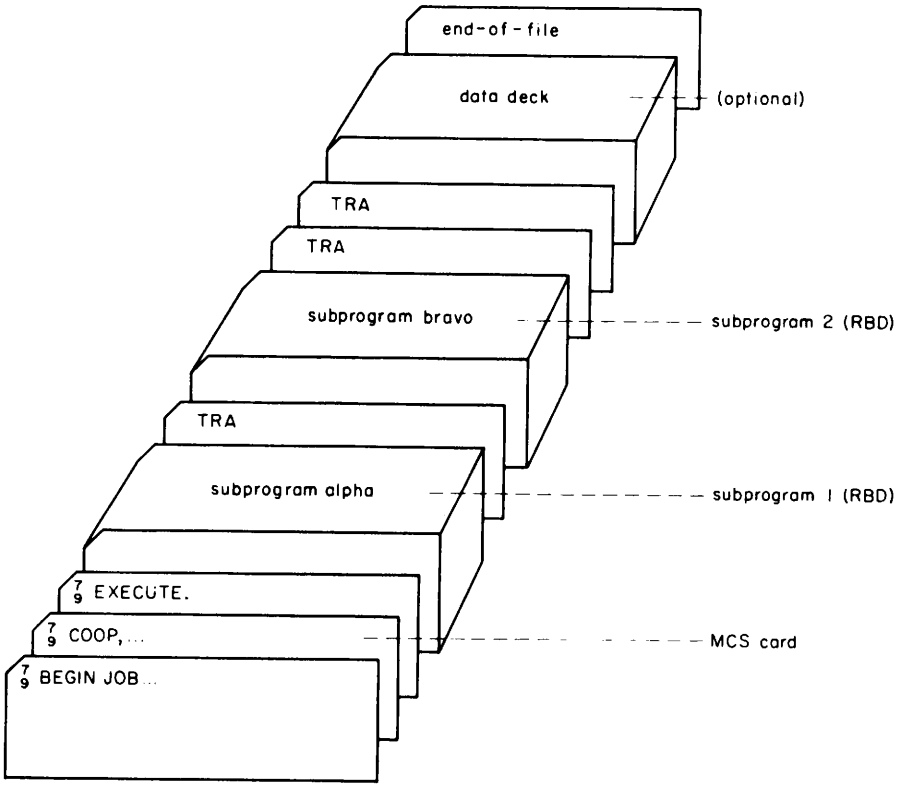
This deck results in two object subprograms being punched on the standard punch unit, each having one TRA card.



Compile Only

EXECUTE ONLY The decks used for execute only operations must be standard RBD decks as described in Part 2 of Appendix C.

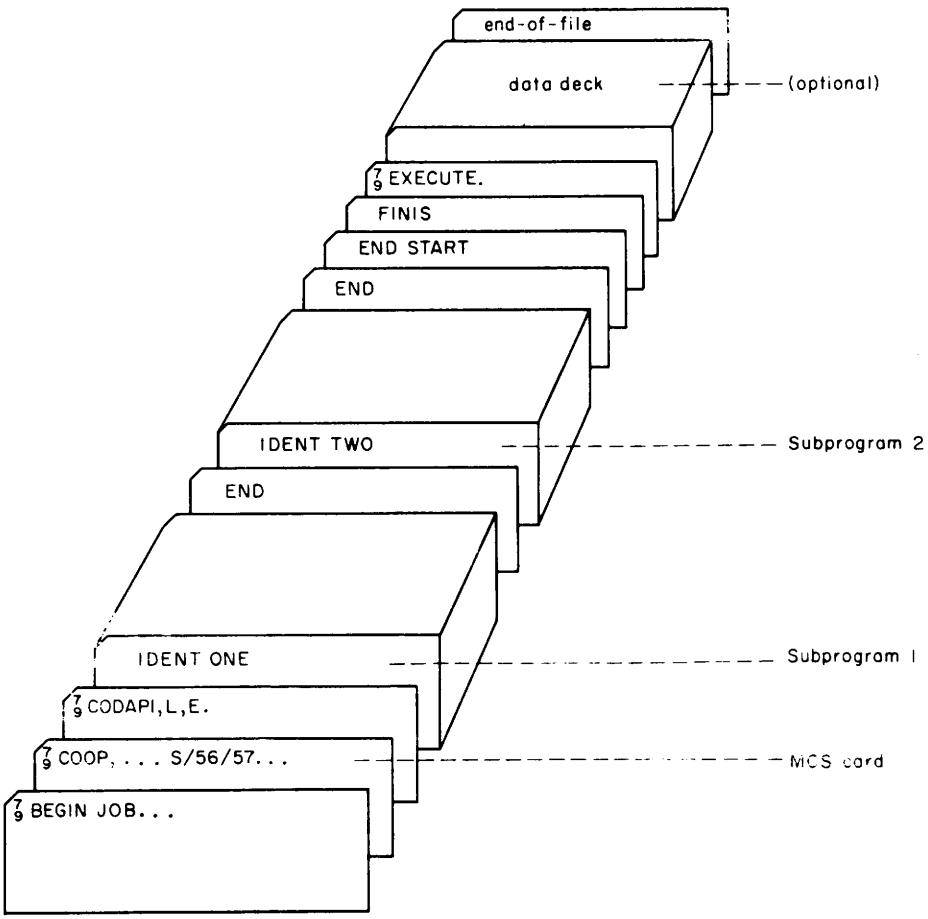
The two binary subprograms, Alpha and Bravo, will be loaded from the standard input unit and executed. A memory map will be printed. The TRA cards shown are required for each RBD deck. One TRA card must contain the symbolic transfer address in Hollerith code, starting in column 9.



Execute Only

COMPILE AND EXECUTE (LOAD-AND-GO)

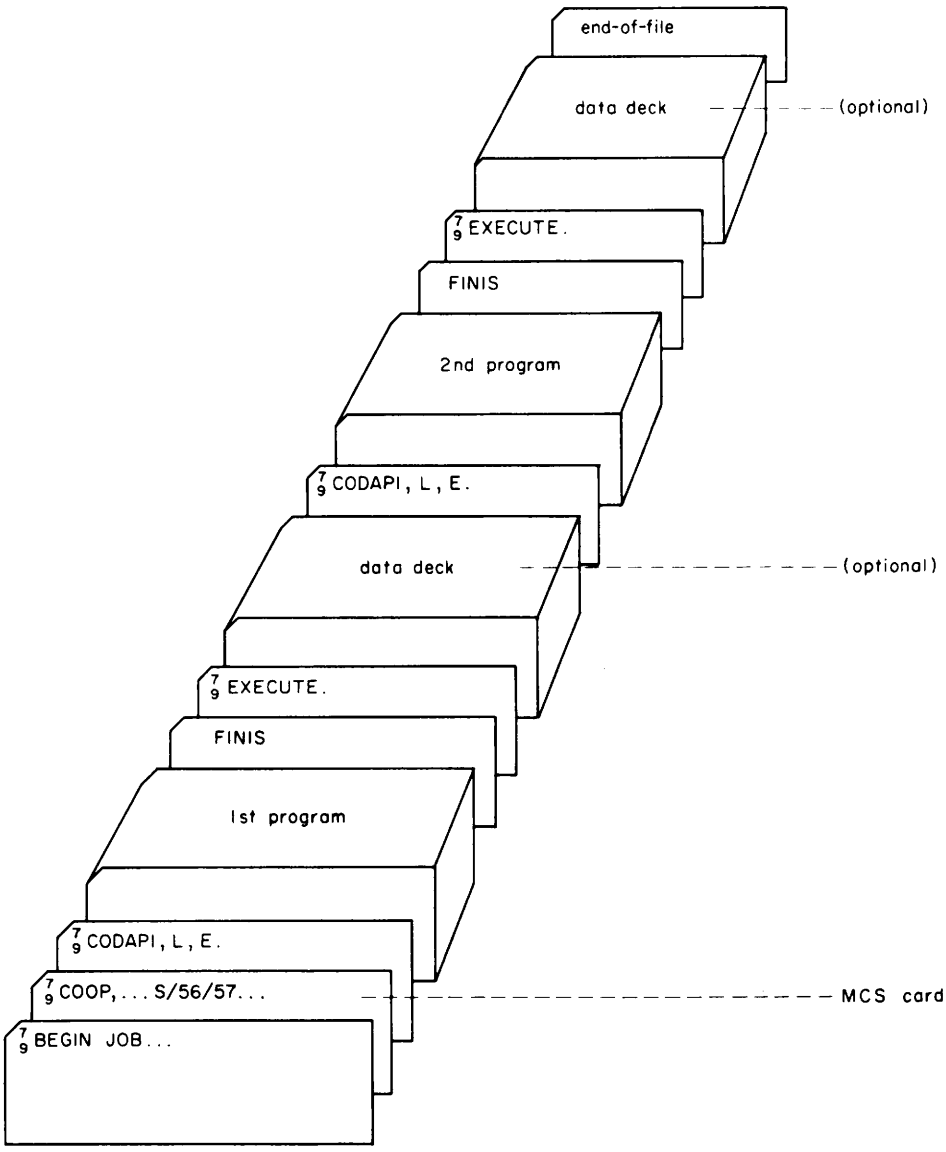
The two subprograms are compiled and executed as one program from standard scratch unit 56. Each source program deck is followed by an END card. The second deck includes an additional END card with the transfer address. The job deck is read from the standard input unit, and an assembled listing and a memory map are produced.



Compile and Execute (Single Program)

MULTIPLE COMPILATIONS

Multiple compilations and executions can be done within one job deck by placing the second and succeeding programs, preceded by a CODAP1 card, after the previous EXECUTE card or data deck, if one is used. Any loader or other error that returns control to the monitor terminates the entire job.

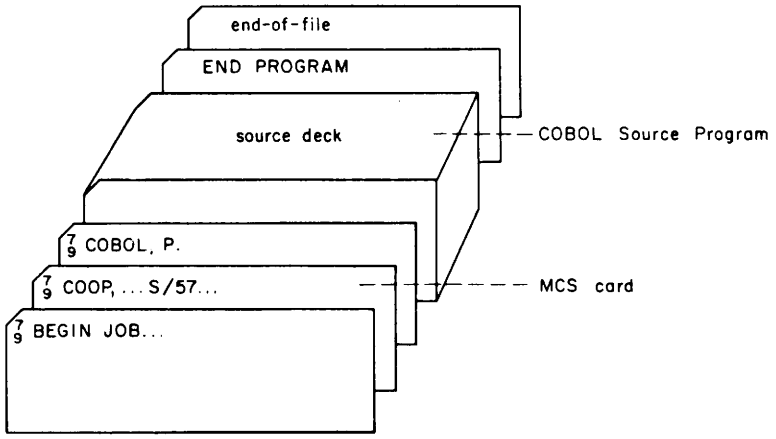


Multiple Compile and Execute

COBOL

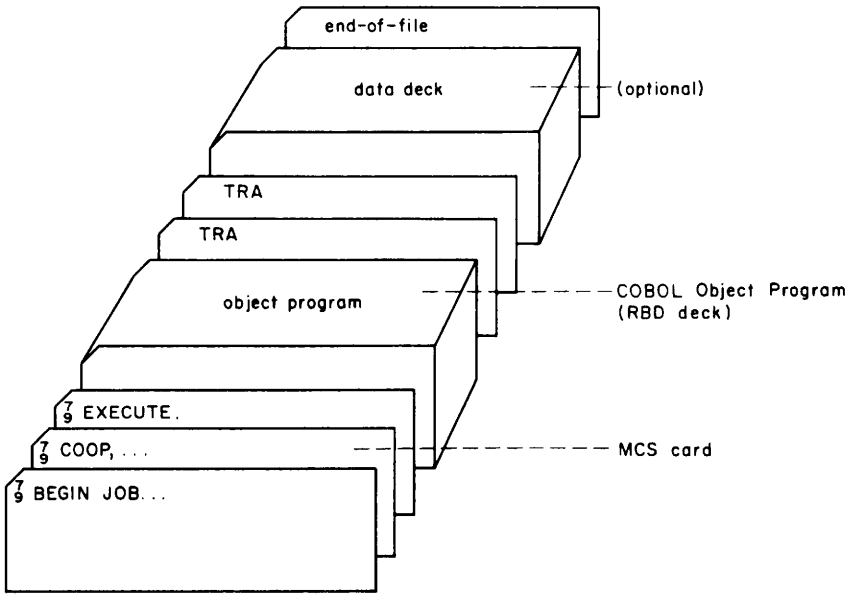
The basic operations for COBOL, compile only, execute only and load-and-go are similar in format to those of FORTRAN-63 and CODAP-1. For execution, however, the second TRA card must be supplied by the programmer, as the compiler can generate only one TRA card.

Using COBOL,P. a binary object deck will be punched on the standard punch unit, and a source language listing will be written on the standard output unit.



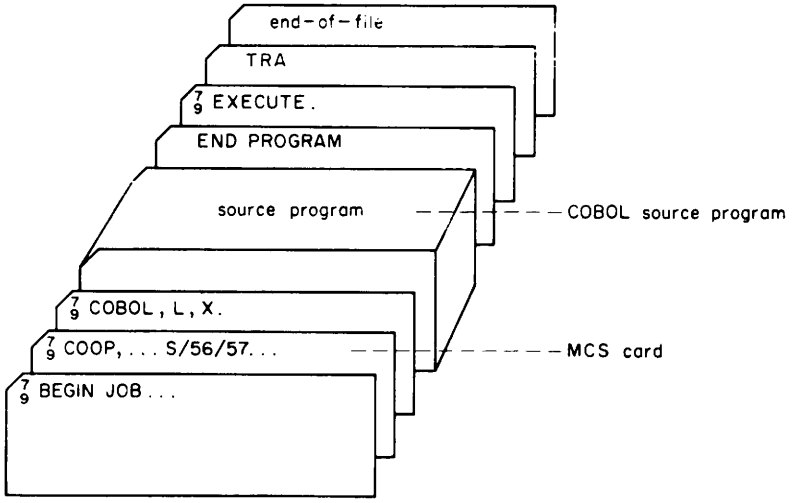
Compile Only

The following object program is loaded from the standard unit and executed, and a memory map is produced.



Execute Only

The load-and-go operations below compile, load, and execute the program from standard unit 56, producing a source listing, an assembled listing, and a memory map.

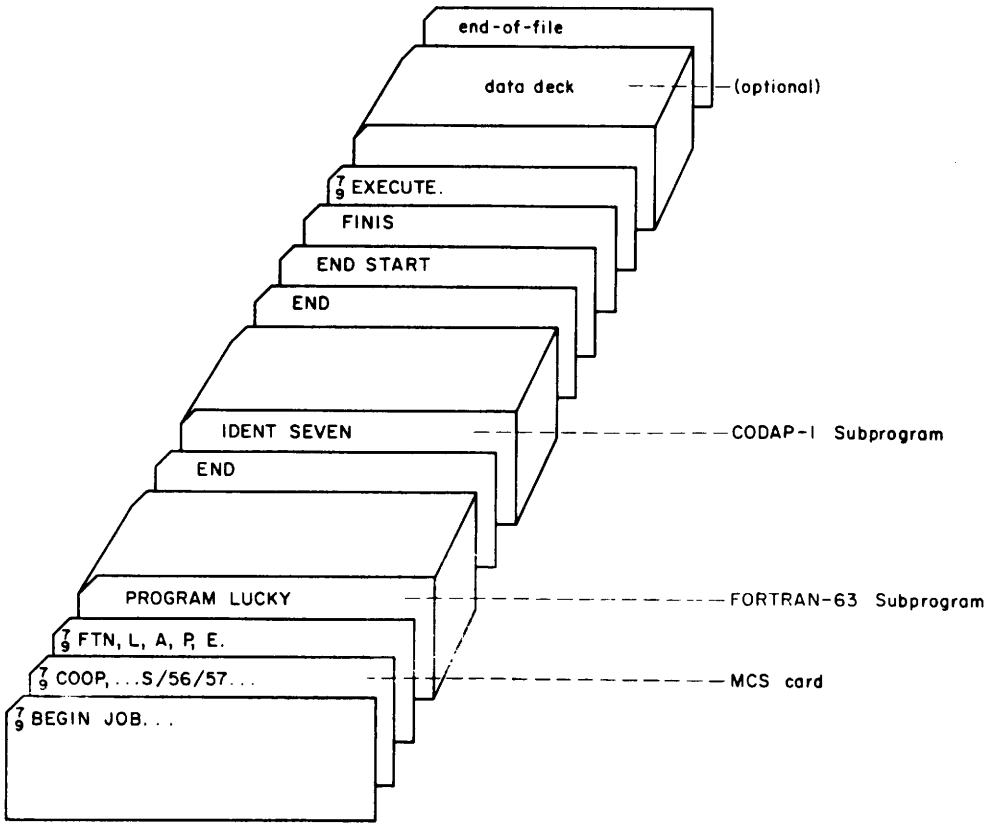


Compile and Execute

COMBINING FORTRAN-63 AND CODAP-1

Compilation and execution can be combinations of FORTRAN-63 and CODAP-1 subprograms. Job construction is similar to that for multiple compilations in each language. The double END or TRA card sequence may occur only once at the end of the program consisting of both FORTRAN-63 and CODAP-1 subprograms. The order of the subprograms depends only on the requirements of the program. A subprogram deck that has the same entry point names as a library function must be placed first in the program deck as this subprogram replaces the library subroutine.

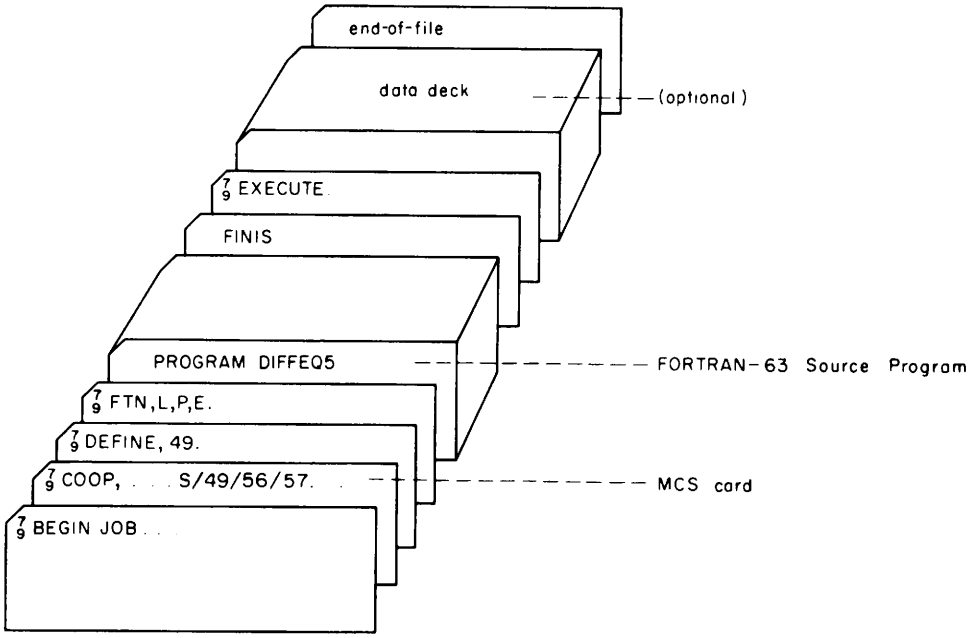
This job deck results in source and assembled listings for the subprograms, together with RBD decks punched on the standard punch unit. The resulting program is loaded and executed from unit 56, producing a memory map.



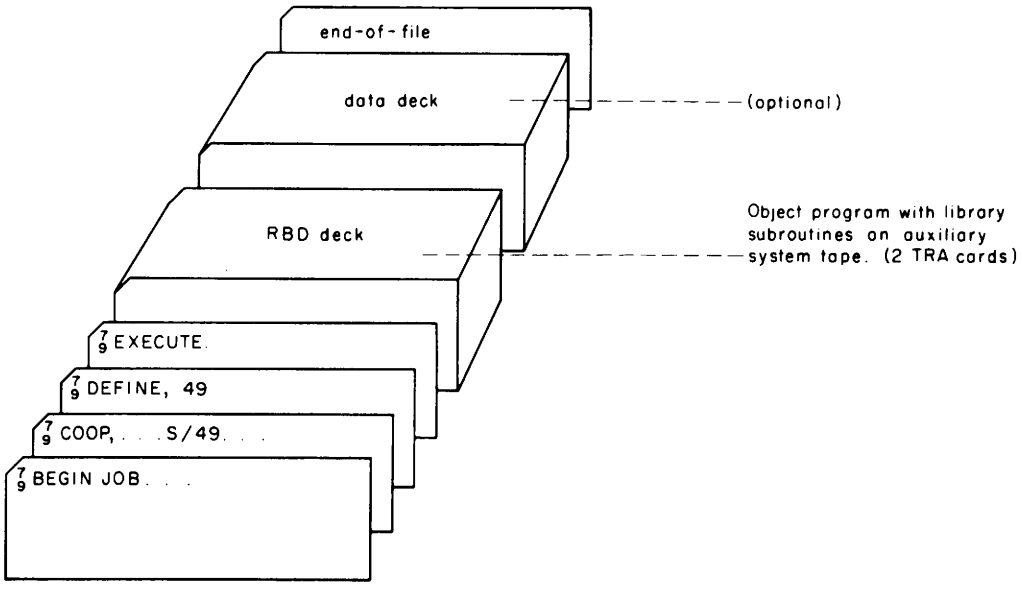
Compile and Execute (One Program)

USING DEFINE CARD

The following deck structure assumes that the auxiliary system tape is on logical unit 49. (49 must be defined either as an input-only or scratch unit on the MCS card.)



Compile and Execute using DEFINE

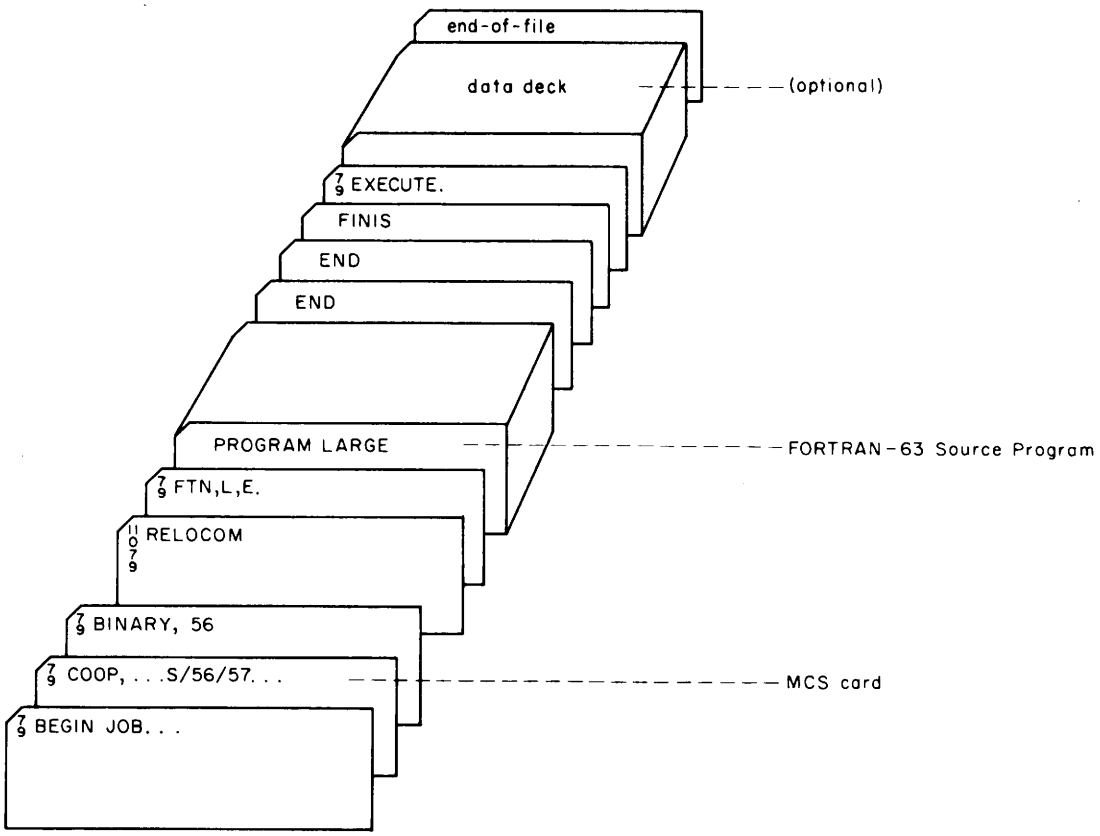


Execute Only using DEFINE

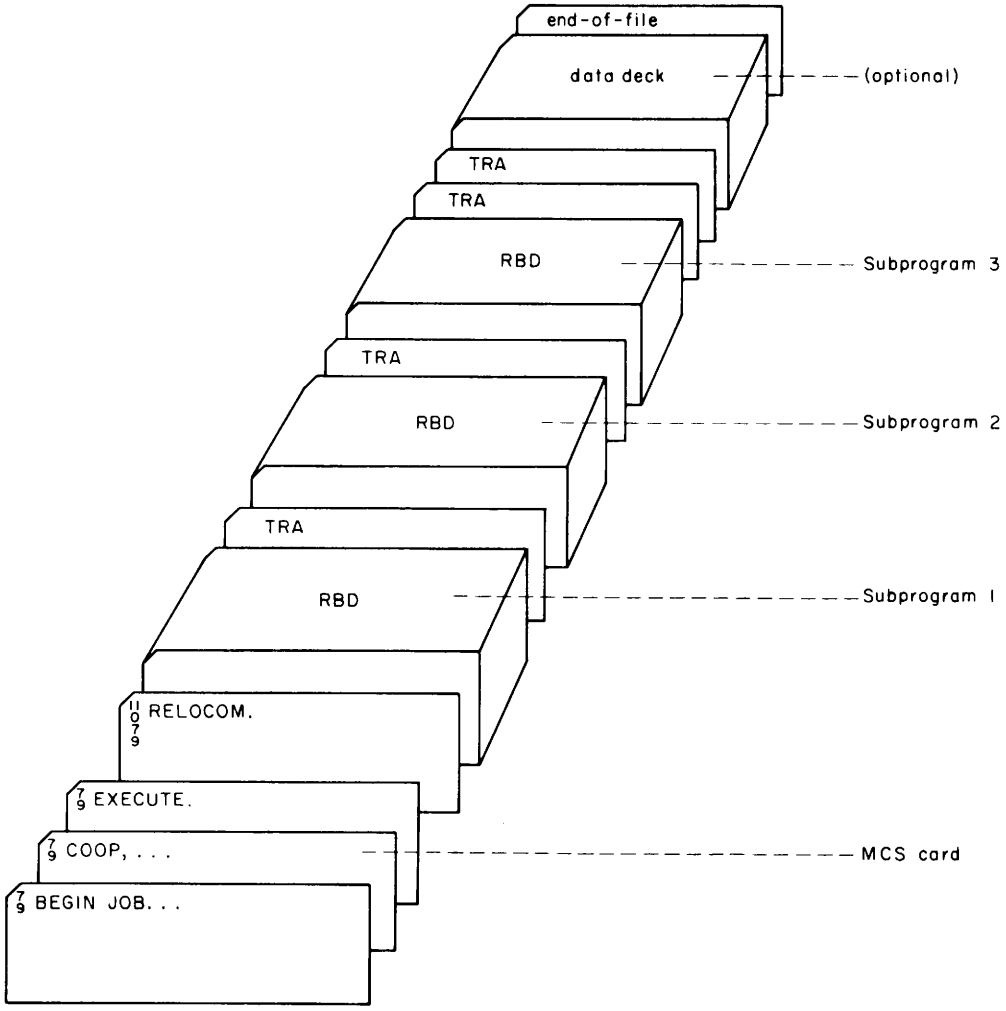
USING RELOCOM CARD

The RELOCOM card is used to increase available program and data storage area by replacing part of the resident portion of monitor with numbered common.

In an execution only operation, RELOCOM is the first card of the first sub-program to be loaded. In a compile and execute operation, RELOCOM is preceded by a BINARY card, and the two cards are placed immediately before the FTN, COBOL or CODAP1 card. If both RELOCOM and EXECUTER are used, the maximum amount of program and data storage is made available to the programmer. Since RELOCOM destroys part of the loader, the program using RELOCOM should either be the only program or the last program executed in a job deck.



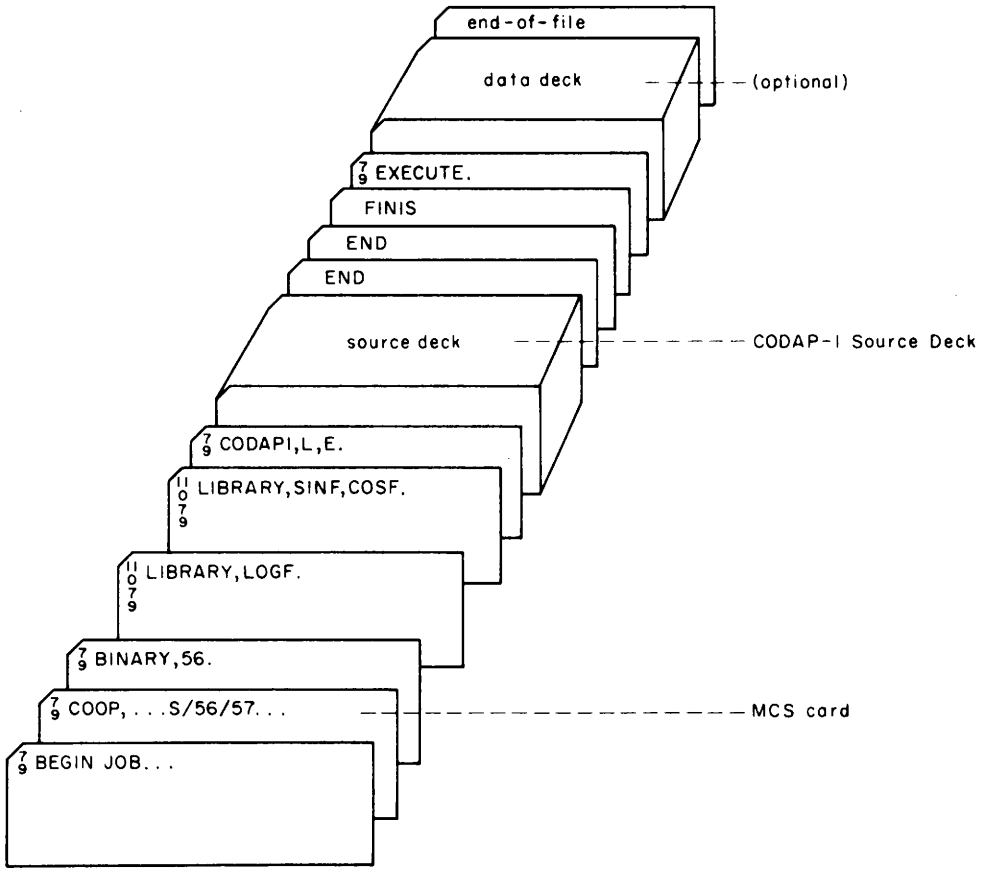
Compile and Execute with RELOCOM



Execution Only with RELOCOM

USING LIBRARY CARD

The LIBRARY card can be used to load all required library routines for a program. It also acts as a RELOCOM card by releasing a portion of resident for program and data storage. LIBRARY cards contain the names of all required library subroutines. Any number of LIBRARY cards may be used, but they must be grouped together at the beginning of the program deck. A BINARY card is required for load-and-go operations. A program that uses LIBRARY cards must be either the only or last program in a job deck.



Compile and Execute, Two LIBRARY Cards

INPUT/OUTPUT SUBROUTINES 3

The subroutine READ*/WRITE* is used for data input and output under monitor control. The subroutine GETCH* tells the programmer the odd number of the channel pair to which a logical unit has been assigned. The subroutine CHKSTD* determines if a particular logical unit has been assigned as the standard input unit.†

READ*/WRITE*

READ*/WRITE* is used for all programmed input/output. The calling sequence contains the logical unit number, the format of the information to be transmitted, and the type of operation to be performed. In addition, the calling sequence contains the location of the data storage area in the computer, and an interrupt key that indicates whether or not to sense for an interrupt condition on the input or output device.

Within READ*/WRITE* there are six channel busy flags (CBF) which indicate the status of a channel. All zeros (+0) in a CBF cell indicate a busy channel, all sevens (-0) indicate a free channel.

The calling sequence written in CODAP-1 assembly language is shown below:

Table 3-1. READ*/WRITE* Calling Sequence

	EXT		READ* or WRITE*	
	ENQ		Y	(address of interrupt subroutine)
L	RTJ	0	READ* or WRITE*	
	rm	0	n	(logical unit number)
L+1	fc	0	A	(FWA)
	i	0	B	(LWA+1)
L+2	alternate return			
			E	(error flag)
L+3	normal return			

†Programs using these subroutines need not save and restore index registers as this is done within these subroutines.

The parameter codes are defined below:

rm — An octal digit designating the recording mode (binary, coded, BCD, Hollerith) described in table 3-2.

fc — An octal number designating the function to be performed by the peripheral device as specified in table 3-3.

i — A single digit, 1 or 0 indicating whether or not to select an interrupt on equipment. If it is 0, the interrupt condition is not selected. If it is 1, the interrupt is selected. When the interrupt is recognized, an exit is made to the starting address of the interrupt subroutine supplied by the programmer. This address (Y) is stored in the Q register.

Y — Symbolic address of the interrupt subroutine supplied by the programmer (must be placed in the Q register).

n — Logical unit number of peripheral device to be used by READ* or WRITE*.

A — Symbolic first word address of input or output area.

B — Symbolic last word address + 1 of input or output area.

E — Error flag set in the lower half of L+2 by the READ* or WRITE* subroutine to indicate an error or abnormal condition (such as parity error or end-of-file) occurred during the operation. These flags are listed in table 3-4. The upper half of L+2 should be a jump instruction to avoid execution of the lower half of L+2.

If any of the parameter values in a calling sequence are illegal, the alternate return is taken and error code 0 is set in E.

In the calling sequence, the EXT pseudo instruction identifies the symbol READ* or WRITE* as being external to the program containing the calling sequence. EXT may be placed anywhere within the subprogram. The ENQ instruction enters in the Q register the entry point address of the programmer interrupt subroutine. If no interrupt is requested (i=0), ENQ may be omitted. The RTJ instruction which gives control to READ* or WRITE* must be an upper instruction. Parameters rm, fc, and i are placed in the OP-code portion of the lower half of L and the upper and lower halves of L+1. The n, A and B go in the corresponding M-terms of L and L+1.

An SLJ 0 instruction is usually placed in the upper portion of L+2 to provide an alternate exit (C) should an error occur during the execution of either READ* or WRITE*. The lower half of L+2 (E) is normally zero in the source code, to allow for insertion of an error code by READ* or WRITE*.

RECORDING MODES

The recording mode (rm) key (1, 2, or 3) indicates how the data is to be interpreted and converted.

For example, a card can be read in Hollerith, column binary, or row binary format and the information can be stored in either BCD or binary format. The recording mode key indicates both input and storage formats. Similar conversion options are provided for all the peripheral equipment normally connected to a 1604/1604-A computer.

LOGICAL UNIT

The logical unit number (n) must be assigned as an input, output or scratch unit on the MCS control card, unless a standard unit is used. Programmer units 1 through 49 are assigned to input or output on the MCS control card. Standard input and output units 50, 51, and 52, may be used by both monitor and programmer. These units, however, should not be rewound, or backspaced past an end-of-file or skipped to end-of-file. Because all standard units (except 56 and 57) are always assigned by the monitor, they do not require entries on the MCS control card. Other logical units, may be equated to the standard units. See Chapter 2 for explanation of logical units and assignments.

FUNCTION CODES

Since more than one type of operation can be performed on a given input or output device, the function code indicates the operation to be performed: read or write (punch, print) only, check only, read or write with checking, and sense equipment ready. Additional functions perform specific tasks such as rewind, page eject, punch 18-inch trailer, and so forth. The function codes listed in table 3-3 are described below.

A READ/WRITE calling sequence using a particular logical unit may be immediately followed by a READ/WRITE calling sequence using another logical unit number.

READ OR WRITE ONLY (fc=1)

is used when a buffered read or write operation is to be performed. During program execution the buffered operation is initiated, the CBF is set to +0 (busy) and control is returned to the main program.

To sense for the end of the buffer operation, interrupts may be selected in the calling sequence. The main program will be interrupted at the end of the operation and control transferred to interrupt location Y in the calling sequence. If interrupt has not been selected, the end of operation is sensed by the Check Only or Sense Equipment Ready function code.

Table 3-2. RECORDING MODE KEYS

Mode		Key (rm)
Magnetic Tape	Read/Write tape coded (even parity)	1
	Read/Write tape binary (odd parity)	2 or 3
Card Reader	Convert Hollerith card to BCD	1
	Convert row binary card to binary (40 low order bits of word)	2
	Convert column binary card to binary (48-bit word)	3
Card Punch	Convert internal BCD to Hollerith card (1 word - 8 columns)	1
	Convert internal binary to row binary card	2
	Convert internal binary to column binary card	3
Line Printer	Internal format is BCD	1
	Internal format is binary	2 or 3
Typewriter	Convert typewriter character code to internal BCD, 8 characters per word. Blanks are inserted for illegal BCD characters	1
	Convert typewriter character code to octal equivalent, 8 characters per word	2
	Convert typewriter character code to octal equivalent, 1 character per word	3
Paper Tape Read	Convert Flexowriter character code to internal BCD, 8 characters per word. Blanks are inserted for illegal BCD characters	1
	Read Flexowriter code and store, 8 characters per word (assembly mode)	2
	Read Flexowriter code and store, 1 character per word (character mode)	3

Table 3-2. RECORDING MODE KEYS (cont.)

Mode		Key (rm)
Paper Tape Punch	Convert internal BCD to Flexowriter codes, 8 frames per word	1
	Punch Flexowriter tape in assembly mode without conversion, 8 frames per word. (automatic 7th level punches)	2
	Punch Flexowriter tape in character mode without conversion, 1 frame per word	3

A Read Only or Write Only calling sequence should be followed by a Check Only calling sequence either in the programmer's interrupt subroutine, if interrupt is selected, or in the main program before the next READ or WRITE operation on that unit.

If the CBF is already set to +0 when the calling sequence is executed, no operation is performed. The alternate return is taken and error code 64 is set in E.

CHECK ONLY (fc=2)

checks for the end of a Read Only or Write Only operation. This function also performs any conversions required by the recording mode key for the Read Only or Write Only operation being checked.

If the Check Only function finds the logical unit busy, the program temporarily halts until the operation is completed. As soon as the unit is free, the CBF is set to -0 (not busy) ; and if errors occur, the appropriate E code is set in the lower half of L+2 in the calling sequence. The exit taken, normal or alternate, depends on this code. Upon exit, the number of words transferred to or from the logical unit is placed in the A register.

The Check Only function interrogates the last previous function performed on the channel of the specified logical unit. If the previous function was other than 1, 4, 5, (for input) or 11, no checking is done and a normal exit is made immediately. If the previous function was 1, 5, or 11, the Check Only function performs the tasks described below. If the previous function was 4 (Sense Equipment Ready) , the function performed prior to 4 is interrogated and the appropriate action is taken.

The READ* calling sequence checks read operations; the WRITE* calling sequence checks write operations. One calling sequence may immediately follow the other, as the two operate independently of each other. In the Check Only

calling sequence, the A, B and rm parameters are not required as they are obtained from the Read Only or Write Only operation being checked. The interrupt parameter i must be zero, as no interrupt is recognized in the Check Only operation.

READ OR WRITE WITH CHECKING (fc=3)

combines the Read or Write Only and Check Only functions for non-buffered input/output. Since the program halts until the operation is completed, no interrupt is recognized. Therefore, the interrupt parameter i should be zero. All other parameters except Y are required. The data storage area is defined by A and B. The subroutine exits through the normal or alternate return with the appropriate error code in E: the CBF is set to -0 before exiting. If the CBF was set previously, no operation is performed; the alternate exit is taken, and E is set to 64.

SENSE EQUIPMENT READY (fc=4)

interrogates the status of the equipment, channel, and CBF. The equipment is checked by requesting the status of the last unit selected. If busy, the alternate exit is taken; if not busy, the normal exit is taken. For example, assume that the logical unit in the Sense Equipment Ready calling sequence has been assigned by the monitor tape 2 of a 1607 or 606 equipment cabinet. If the last unit selected in that equipment was tape 3, the status of tape 3 would be sensed, and the exit would depend on the status of tape 3, not tape 2. The status of tape 2 is not checked. This restriction is due to the limitation of the 1604 sense codes which can check only the status of the last unit selected within a particular equipment. The Sense Equipment Ready function performs a sense operation only, it does not select any equipment or units. The value of the CBF for the specified logical unit is not changed by this function.

The E portion of the calling sequence for this function is set to a specific value according to the status of the channel, CBF, and last unit selected within the equipment.

Bit 0 of E is set to one if the last unit selected is busy.

Bit 1 of E is set to one if the channel is active.

Bit 2 of E is set to one if the CBF is set to +0 (busy) .

Otherwise these bits are set to a zero. The alternate exit is taken only when bit 0 is set to one.

This function is not intended for general programming use. When either a normal or alternate exit is made, the Running Hardware Table entry corresponding to the logical unit in the calling sequence is entered in the A register. The low-order 15 bits contain the assigned channel number.

The only required parameters in the read or write calling sequence are the function code (fc) , the logical unit number (n) , and the normal and alternate exits. The i code must be zero; values of all other required parameters are arbitrary.

SPECIAL PURPOSE FUNCTION CODES (fc = 5,6,7,10 OR 11)

provide facility for such operations as rewinding tape, restoring the page, skipping records or cards, ejecting blank cards, and executing a carriage return on the typewriter. These functions are listed in table 3-3. The decimal literal B in the descriptions indicates the number of times that function is to be performed.

For magnetic tape read functions 5 and 11, an interrupt can be selected to signal the end of the skip forward or backspace operation. For all other functions the interrupt key is zero. Parameters fc, n, and E are required for all special purpose codes. For functions requesting interrupt, Y is also required; all other parameters may have arbitrary values.

A special E code is entered in the functions shown with an asterisk in table 3-4 when the operation is terminated. These functions are primarily used for format control and the E code indicates that one of these functions has been executed on the particular unit.

INTERRUPT

Interrupt (i) may be selected in conjunction with certain function codes. The interrupt option is equivalent to the hardware select code interrupt on equipment ready, and is used primarily for buffered input and output (fc = 1) . It is also used for sensing the termination of skip forward or backspace operations on magnetic tape.

The Y parameter in the calling sequence specifies the symbolic entry point of the programmer interrupt subroutine. This entry point must be provided if interrupt is selected. If interrupt is not selected, the ENQ Y entry may be omitted from the calling sequence.

ERROR CODE

An error code (E) is entered in the lower half of L+2 in the read or write calling sequence by one or more functions depending on the error. For example, all functions check for parameter errors; however Check Only and Read or Write with Checking sense for parity and buffer length errors. There are special E codes for the no-operation and control function.

The E code is a composite of all errors or conditions sensed during the execution of the calling sequence. If the composite error code implies both normal

fc (octal)	Definition								
	Mag. Tape Read	Mag. Tape Write	Card Read	Card Punch	Line Print	Paper Tape Read	Paper Tape Punch	Typewriter Read	Typewriter Write
1	Read Only	Write Only	Read Only	Punch Only	Print Only	Read Only	Punch Only	Read Only	Write Only
2	Check Only	Check Only	Check Only	Check Only	Check Only	Check Only	Check Only	Check Only	Check Only
3	Read with Checking	Write with Checking	Read with Checking	Punch with Checking	Print with Checking	Read with Checking	Punch with Checking	Read with Checking	Write with Checking
4	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready	Sense Equipment Ready
5	Skip Forward B Records	No Operation	Skip Forward B Cards	Eject B* Blank Cards	Space* Forward B Lines	Skip* Forward B Frames	Eject B* Blank Frames	No Operation	Execute B* Carriage Returns
6	Skip Forward Past End-of-File	Write End-of-File	Skip* Forward Past End Card	Punch End* Card	Restore* the Page	No Operation	No Operation	No Operation	Execute* Carriage Return and Type End
7	Rewind	Rewind	No Operation	No Operation	Restore* the Page	No Operation	Eject 18* Inch Trailer	No Operation	Execute Six* Carriage Returns
10	Rewind with Interlock	Rewind with Interlock	No Operation	No Operation	Restore* the Page	No Operation	Eject 18* Inch Trailer	No Operation	Execute Six* Carriage Returns
11	Backspace B Records	Backspace B Records	No Operation	No Operation	No Operation	No Operation	No Operation	No Operation	No Operation

*These operations always give the special operation E code 16.

Table 3-3. READ*/WRITE* Function Codes

and alternate exits, the alternate exit is always taken. The bypassed operation code (128) is a result of a BY entry in the input/output list on the MCS card.

Not all conditions having an error code are considered true errors, therefore normal exits are used in certain cases. For example, a write buffer length error is a true error; however, a read buffer length error may or may not be an error, depending on the intention of the programmer.

Table 3-4. READ-WRITE ERROR CODES

E-Code (octal)	Condition	Exit
0	No error	normal
0	parameter error	alternate
1	parity error	alternate
2	write buffer length error	alternate
2	read buffer length error	normal
4	end-of-file sensed	normal
4	end-of-tape sensed	normal
10	no operation function code	normal
20	special operation function code	normal
40	illegal character, card output	normal
240	illegal character, card input	alternate
100	channel busy	alternate
200	bypassed logical unit	normal
400	typewriter case conflict (upper, lower)	alternate

OPERATOR MESSAGES

Two kinds of messages are typed or printed out as comments to the operator. One is the result of errors sensed by READ* or WRITE*; the other indicates the status of tape units, standard and programmer-assigned.

The following error conditions sensed by READ* or WRITE* result in job termination. The message, ILLEGAL I/O ON UNIT XX. is written on the standard output unit and control is returned to the monitor through ERROR*.

1. Logical unit is not defined on MCS card (no RHT entry) .

2. Logical unit number is out of range 1 to 63.

3. Equipment assignment conflicts with function requested (such as an attempt to write on tape assigned to input on the MCS card). Any tape, however, may be rewound or backspaced by either a READ* or WRITE* calling sequence. WRITE* cannot be used for rewind or backspace if the file protect ring is removed from the tape reel.

4. Attempt to read past end-of-file on a standard input unit.

5. Attempt to rewind a standard input or output unit.

6. Attempt to backspace a standard input or output unit more records than were read or written by the current job.

The message, LINE COUNT EXCEEDED, is written on the standard output unit if the line counter for the standard output unit exceeds the line limit specified on the MCS card, or the standard line limit if none is specified. (Standard limit is set by each installation.)

The message, JK2, printed out as a comment to the operator during program execution, indicates the first time a programmer-assigned logical unit is encountered. The READ*/WRITE* subroutine interrogates jump key 2 so that the operator can indicate whether or not the proper tapes are mounted on the assigned tape units. Once JK2 is set during a job, it is not sensed again.

The message, EOT XXXX, is printed out as a comment to the operator when end-of-tape is sensed on the standard output unit. If this occurs, three additional records are written on the standard output unit: and the tape ia automatically rewound with interlock. The monitor waits until a new tape is mounted on the standard output unit and the unit is restored to load-point position.

The three records written on the output unit are END OF TAPE END, end-of-file mark, and END MONITOR OUTPUT.

GETCH*

The subroutine GETCH* in the resident portion of the monitor system may be used to determine the channel assigned to a logical unit. It also is used to obtain the contents of the RHT entry corresponding to the logical unit number. The calling sequence is:

```
LDA     n (logical unit number)
RTJ     GETCH*
```

The subroutine enters into the low-order 6 bits of the A register the channel number assigned to n; the RHT entry is entered in the Q register.

CHKSTD*

The subroutine CHKSTD* is used to determine if a logical unit has been equated to the standard input unit (n = 50) on the MCS control card. The calling sequence is:

```
LDA   n (logical unit number)
RTJ   CHKSTD*
```

If n has been equated to the standard input unit, the subroutine exits with 50 in the A register, otherwise the A register will contain the number n in the low-order 6 bits.

INTERRUPT PROCESSING 4

Interrupts are processed under the CO-OP Monitor by the subroutines SELECT* and REMOVE*. Without these routines, it would be necessary to select an interrupt by an EXF instruction, and when an interrupt occurs, perform a test to determine the type of interrupt. Storage location 7† would then be set to the starting address of the programmer's interrupt subroutine, and when it terminated the interrupt selection would have to be cleared.

By using SELECT*, however, the programmer simply chooses one of the available interrupts and provides the starting address (entry point) of his interrupt subroutine. The SELECT* subroutine performs all the tasks associated with interrupt selection and recognition.

REMOVE*

REMOVE* cancels previously selected interrupts so that they do not carry over into the next program within the same job. REMOVE* may be used at any point in a program when the interrupt is no longer needed. All interrupt selections are automatically cleared when a job is terminated.

SELECT*

SELECT* selects the type of interrupt requested in the calling sequence. When the interrupt occurs, control transfers the programmer's interrupt subroutine, interrupt lockout is imposed, and the interrupt selection is cancelled. After the interrupt subroutine is executed, control returns to the main program, and the interrupt lockout is removed. A programmer must reselect an interrupt if that selection is to produce additional interrupts. Reselection would normally be made within the interrupt subroutine, but it may be done at any time.

Calling sequence:

```
L       RTJ             SELECT*
        t               NAME
L+1     error return
L+2     normal return
```

†The 1604-A uses location 7 through 17 for interrupt.

The parameters t and NAME will be transferred to the monitor interrupt subroutine for future use.

The parameter t is an octal number indicating the type of interrupt as listed in table 4-1. NAME is the symbolic entry point of the programmer's interrupt subroutine. SELECT* normally exits to location L+2; however, if an illegal parameter is encountered, the exit is to location L+1. L+1 may contain a jump to an error subroutine and L+2 may contain the next instruction pair in the main program.

A separate calling sequence is used for each interrupt to be selected.

The programmer's interrupt subroutine may be coded as follows:

```
L       NAME        SLJ     **
                    (first executable instruction)
                    .
                    .
                    .  subroutine coding
                    .
                    .
                    .
                    SLJ NAME
```

NAME is that of the interrupt subroutine. If the interrupt is to remain selected, the subroutine should contain another SELECT* calling sequence using the original parameters. Since interrupt lockout is imposed when the interrupt occurs, no interrupts will be recognized during execution of an interrupt subroutine.

Table 4-1. SELECT* Interrupt Keys

t (octal)	Type
1	interrupt on channel 1 inactive
2	interrupt on channel 2 inactive
3	interrupt on channel 3 inactive
4	interrupt on channel 4 inactive
5	interrupt on channel 5 inactive
6	interrupt on channel 6 inactive
7	interrupt on arithmetic fault
10	interrupt on overflow
11	interrupt on exponent overflow
12	interrupt on exponent underflow
13	interrupt on shift fault
14	interrupt on divide fault

MODIRET*

After the programmer's interrupt subroutine is executed, control normally returns to the location in the main program where the interrupt occurred; however, another return address can be specified by the subroutine MODIRET*. This call to MODIRET* appears in the programmer's interrupt subroutine and modifies the program address placed in the upper half of storage location 7 by the 1604 hardware when the interrupt occurs. The original program address (the next instruction to be executed in normal program sequence) is transferred to the A register by MODIRET* and a new address is inserted in location 7.

Calling sequence:

```
L       ENA             R
        RTJ             MODIRET*
L+1     normal return
```

The address parameter R is the location to which control is to be returned after completion of the interrupt subroutine. The 1604 sets a flip-flop indicating whether the upper or lower instruction in address location 7 is to be executed next; therefore, the programmer must provide both returns in the return address R.

The contents of R and R+1 would normally be as follows unless the upper return is to be handled in a different manner than the lower return.

```
R       SLJ     0       *+1 (if upper flip-flop set)
        SLJ     0       *+1 (if lower flip-flop set)
R+1     next instruction
```

REMOVE*

The following calling sequence removes interrupt selections that have not occurred:

```
L       RTJ     REMOVE*
        t
        6       0
L+1     error return
L+2     normal return
```

The octal parameter t indicates the selected interrupt to be removed as listed in table 4-2. The error return is taken if this parameter is illegal, otherwise the return is made to the upper instruction in L+2.

REMOVE* should be used at the end of a program to remove all selected interrupts that have not occurred, so that unwanted selections will not carry over to another part of the program or to the next program.

Table 4-2. REMOVE* Interrupt Keys

t (octal)	Type
1	remove interrupt on channel 1 inactive
2	remove interrupt on channel 2 inactive
3	remove interrupt on channel 3 inactive
4	remove interrupt on channel 4 inactive
5	remove interrupt on channel 5 inactive
6	remove interrupt on channel 6 inactive
7	remove interrupt on arithmetic fault (except clock)
10	remove all selected interrupts (except clock)

OPERATOR-PROGRAM COMMUNICATION

5

Within the resident portion of the monitor are subroutines that allow the operator to communicate with the program being executed. Using the typewriter, the operator can set up to 48 flags that are interrogated by the subroutine FLAGTST*, or he can type a message (maximum of 31 characters) that is interpreted by the subroutine OPCOM* and a subroutine provided by the programmer.

FLAG SETTING (SENSE SWITCHES)

The 48 flags (actually 48 bits in a flag word) are set when the operator control statement $f,a_1/n_1, \ldots a_1/n_i$ is entered on the typewriter. The parameter, a, is the number of the flag, 1-48, and n the flag value key. If n is 1, the flag is set. If n is 0, the flag is cleared. For example, if flags 1,30,40, and 48 are to be set, and flags 5 and 10 are to be cleared, the operator would type the message:

f,1/1,30/1/40/1,48/1,5/0,10/0.

Flag numbers may be in any sequence. The message cannot contain more than 31 characters, including the f, commas, slashes, and terminating period. The first six of these flags are called sense switches in FORTRAN-63. Bit 0 = flag 1 = sense switch 1, bit 1 = flag 2 = sense switch 2, and so forth.

To interrupt the current program, the operator strikes the carriage return key on the typewriter. The monitor then types a period and waits for the message to be typed. After typing the message, the operator returns control to the monitor by striking a period and a carriage return. If a program has been interrupted, processing resumes at the point where the interrupt occurred. Flags may be set at any time, whether or not a job is being processed.

FLAG TESTING

To test the flags, the programmer uses the subroutine FLAGTST*. The calling sequence is:

```
L       ENA     a
        RTJ     FLAGTST*
L+1             normal return
```

The parameter a is the number of the flag to be tested (1-48). The routine returns to L+1, and the A register contents indicate the value of the flag. The A register is negative if the flag is set; positive, if the flag is not set.

OPERATOR MESSAGES

The programmer can request a message from the operator of up to 27 characters including spaces, which is stored 8 characters per word, left justified in BCD format. Blanks are used for fill if the message is less than a multiple of 8 characters.

MESSAGE FORMAT

The operator initiates a message by striking the carriage return key on a typewriter. The monitor will interrupt the current program and allow the operator to type a message in the following format:

```
com, . . . (message). . .
```

The operator again strikes the carriage return key and the message is processed by the interrupt subroutine. As soon as the interrupt subroutine executes either a normal or alternate return, the monitor types an *, indicating that another message can be accepted from the typewriter. (Interrupt lockout is removed at this time).

To interpret this message within the program OPCOM* and an interrupt subroutine provided by the programming are used. OPCOM* tells the monitor to expect a message from the operator and supplies the entry point of the interrupt subroutine. When the operator types the message, control is turned over to the interrupt subroutine and an address is sent to the subroutine indicating where the first word address of the message can be found. The end of a message is always indicated by a period, therefore the programmer must search for this period to determine the length of the message. The calling sequence for OPCOM* is:

```
L       RTJ             OPCOM*
        ZRO             Y
L+1     normal return
```

The parameter Y is the location of the programmer's interrupt subroutine. Return is made to L+1 after Y has been transferred to OPCOM* for future use. This calling sequence to OPCOM* is only for initialization, and must be executed before a COM message is typed.

INTERRUPT SUBROUTINE FORMAT

The interrupt subroutine is written as a closed subroutine in the following format:

```
A       SLJ             **
          .
          . (subroutine coding)
          .
          .
        SLJ             A
```

A is the entry point of the interrupt subroutine. When the message is typed, the ** in the upper address of the entry point is replaced by an address from OPCOM*, and control is turned over to the interrupt subroutine. (The return to the main program must be made through OPCOM* after the message has been interpreted.) The first word of the message is located by indirect address procedures; the upper address of the entry point word is extracted and decreased by 1. This modified address from the UA of the entry point word is now the indirect address of the first word of the message location.

The return address to OPCOM* is also computed from the upper address of the entry point. In this case, the return address may be one greater than the upper address of the entry point.

NORMAL RETURN

The following coding illustrates a method of obtaining the first word address of the message block and the return address to OPCOM*.

A separate OPCOM* calling sequence must be initiated for each message, as the monitor cancels a message request after the message has been processed. An operator message of the type "com" is ignored if the current program has not executed a return jump to OPCOM* when the "com" message is typed. The main program continues normal processing.

```
                    ENTRY                       MESSAGE
MESSAGE             SLJ                         **
                    SIU     1                   SAVE
+                   LIU     1                   MESSAGE
                    INI     1                   -1
+                   SIL     1                   *+1
                    LDA     7                   **
                     .
                     .
                     . subroutine coding
                     .
                     .
                     .
                     .
                     .
                    LIU     1                   MESSAGE
                    INI     1                   1
                    SIL     1                   *+1
SAVE                ENI     1                   **
                    SLJ     0                   **
```

In this example, the first eight characters of the message are placed in the A register by the LDA 7 ** instruction. The return to OPCOM* is accomplished by the coding at the end of the subroutine. Any index registers used in the

subroutine should be saved and restored as shown. Interrupt lockout, imposed when the message was typed, is removed by the return to OPCOM*, which in turn returns control to the next instruction in sequence in the main program.

ALTERNATE RETURN

Control can be taken away from the main program by the programmer interrupt subroutine, and the interrupt subroutine can act as a new main program. The alternate return is accomplished by planting a return jump instruction in the subroutine, using as the M-term the upper address of the entry point location (unmodified). This will cause a return jump to OPCOM*, giving main program control to the interrupt subroutine at the location following the return jump instruction. The following coding illustrates a method of finding the message and giving main program control to the interpretive subroutine.

```
                ENTRY                       MESSAGE
MESSAGE         SLJ     0                   **
                SIU     1                   SAVE
                LIU     1                   MESSAGE
                INI     1                   -1
                SIU     1                   *+1
+               LDA     7                   **
                .
                .
                .
                .
                .
                .
                .
                .
                LIU     1                   MESSAGE
                SIU     1                   RETURN
RETURN          RTJ                         **
SAVE            ENI     1                   **
                .
                .                           New Sequence
                .
                .
```

The RTJ instruction transfers control to OPCOM*, which then returns control to SAVE rather than to the normal sequence in the main program. This removes the interrupt lockout.

DIAGNOSTICS 6

The diagnostics and programming aids discussed in this chapter consist of loader errors, memory maps and snapshot (SNAP) dumps. Snapshot dumps are requested by the programmer through SNAP cards interpreted at execution time. Memory maps can be requested whenever a program is loaded by using the proper key on a LOAD, EXECUTE or EXECUTER card.

LOADER ERRORS

When a program is loaded, the loader routine (Appendix C) checks for various error conditions. Errors detected by the loader are indicated on the standard output unit in the following forms:

LOADER ERROR XX YYYYYYYY ZZZZZZZZ

XX — a 2-letter mnemonic code denoting type of error

YYYYYYYY, ZZZZZZZZ — an 8-digit octal number or a symbol

LOADER ERROR CODES

XX	YYYYYYYY	ZZZZZZZZ	Type of Error
BC	last program name in loader tables	col. 1 and 2 current card	error in a loader control card (RELOCOM, etc.)
BR	last program name in loader tables	col. 1 and 2 current card	more than 62 block names in one subprogram
BT	name of incorrect library subroutine	not significant	library tape is in error
CK	last program name in loader tables	col. 1 and 2 current card	card checksum does not check
CS	last program name in loader tables	col. 1 and 2 current or previous card	cards of binary deck not in correct sequence
EF	last program name in loader tables	not significant	end-of-file encountered on standard input unit
ET	word 1 current card	word 2 current card	end-of-tape on overlay tape

XX	YYYYYYYY	ZZZZZZZZ	Type of Error
EX	not significant	erroneous symbol	undefined EXT symbol
FM	blank	blank	memory is full, loader cannot load
IO	program name on current card	col. 1 and 2 current card	illegal I/O assignment IDC card
LA	last program name loader tables	not significant	linkage address outside EXT table
LC	word 1 current card	word 2 current card	illegal data on LCC
LE	name of current library subroutine	col. 1 and 2 current card	illegal card on library tape
LL	last program name in loader tables	not significant	more than 100 library subroutines called
LT	word 1 current card	word 2 current card	too many overlay tapes specified
MD	last program name in loader tables	multiple defined symbols	symbol encountered by loader has more than one definition
OC	last program name in loader tables	not significant	loaded program has overflowed memory
OS	word 1 current card	word 2 current card	overlay number out of sequence
PM	blanks	1st instruction of LOADER*	error in LOADER* calling sequence parameters
RT	word 1 current card	named transfer	transfer address not within main program, overlay, or segment
SS	word 1 current card	word 2 current card	segment number out of sequence
TR	first named transfer	second named transfer	more than one TRA card with a symbolic transfer address
UN	last program name in loader tables	undefined symbol	symbol encountered by loader is not defined
WE	BCD overlay LCC card information		writing error on overlay tape

MEMORY MAP

When requested on the EXECUTE, EXECUTER or LOAD card, a memory map is printed on the standard output unit. This map lists the program names, block names for labeled common, block names for numbered common, entry point names, and the absolute addresses assigned to these names when the program was loaded. The program, block, and entry point names are those of both the program and the library subroutines loaded with the program. Also listed are the program names and entry points of the monitor resident routines used by the program and the library subroutines. (Resident entry points are followed by asterisks.) For overlay programs, a map is produced for the main program and each overlay and segment that is loaded.

```
PROGRAM NAMES

00043 RESIDENT          00030 ERRDUMP*          06000 FLMPY
10000 HORRORS           15000 PLEASE            27000 COMPUTE
50000 HOOTOWL           62000 MIMIC

LABELED COMMON

05650 A                 07200 AB                07700 ABC
14323 ABCD              25000 ABCDE             47400 ABCDEF
61200 ABCDEFG

NUMBERED COMMON

04327 1                 04377 20                04405 300
04505 4000              05506 50000             05600 6000

PROGRAM ENTRY POINTS

01000 READ*             01005 WRITE*            01326 SELECT*
01450 REMOVE*           02000 LOADER*           03700 CLBNBCD*
00300 EXIT*             00400 ERROR*            01372 MODERIT*
00700 BCDBN*            00100 OPCOM*            00050 RECRET*
00500 RECLIM*           04000 FLAGTST*          04120 MEMREC*
06000 START             06200 NORMALIZ          06317 ANSWER
14322 MASK              12000 INVERT            15000 PACK
16233 SEARCH            22300 STORE             27700 DIVIDE
27502 ERROR             30003 ROUND             50000 BEGIN
50013 PITCH             51120 TEMPO             62000 RDCARD
62100 CONVERT           62405 EXECUTE
```

SNAPSHOT DUMPS

Snapshot dumps are made of storage areas and console conditions at times selected by the programmer during the execution of a program. The dumps are triggered by inserting SNAP instructions at selected points in a program. A SNAP compiler routine interprets SNAP control cards prepared by the programmer.

These SNAP instructions replace the original instructions at the specified locations in the object program. The original instructions are retained in a SNAP execution routine which is called in from the library tape and acts as a library subroutine that is part of the program. When a snapshot dump has been produced on the standard output unit, the original instruction in that location is executed, and control is returned to the next instruction in the program sequence.

The programmer can specify on the SNAP control card the number of times a snapshot dump is to be produced at a particular location. This would occur only when a snapshot dump is requested within an iteration.

The programmer specifies the number of iterations to be performed before the first dump occurs, the number of iterations between each dump, and the total number of dumps for that location. After the last dump has been performed by the SNAP instruction in that location, the original instruction is re-entered in that location. The original instruction is executed during every iteration. If a SNAP dump occurs during an iteration, the original instruction is executed immediately following the snapshot dump. For example, in an iteration to be executed 100 times, the programmer can request that the first dump occur during the tenth iteration, and thereafter every fifth iteration, with a total of ten dumps to take place.

RESTRICTION ON SNAPSHOT DUMP LOCATIONS

An instruction to be replaced by a SNAP dump instruction must meet the following requirements:

1. The contents of the location containing the instruction to be replaced must not be referred to by that instruction or others in the program.
2. The replaced instruction must not be modified in any way, since modification would be to the SNAP instruction.
3. The replaced instruction must not be part of a subroutine transfer instruction calling sequence unless it is in the alternate or normal return location.
4. The address field cannot be an EXT symbol.

SNAPSHOT DUMP FORMATS

The dump produced may have any of five basic formats, with or without the console conditions. The format is specified in the mode key on the SNAP control card. Lines which are identical to the previous line, except for the address, will not be printed.

The SNAP dump contains facilities for a maximum of ten snap locations in any one program. The SNAP dump routine is in memory at execution time and will be included as a subprogram of the program being executed.

The console conditions (figure 6-1) will contain the contents of:

1. A register in the specified format and in octal
2. Q register in the specified format and in octal
3. Program address register in octal
4. Six index registers in octal
5. Input/output channel control words (storage locations 00001 through 00006) in octal
6. Fault indicators and interrupt lockout condition will be included with identifying titles and described as ON or OFF.

Octal Dump and Octal Dump with mnemonic

The octal dump and octal with mnemonic dump formats (figures 6-2 and 6-3) produce an absolute octal address and six words in each line.

Fixed Decimal Dump

The fixed decimal dump format (figure 6-4) produces an absolute address and six words in each line.

Floating Decimal Dump

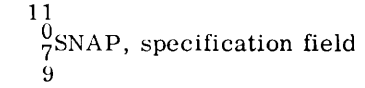
The floating decimal dump format (figure 6-5) produces an absolute address and six words in each line of the form, ±xxxxxxxxxxxx±xxx.

BCD Dump

The BCD dump format (figure 6-6) produces an absolute address and ten words of eight characters each in each line.

SNAP CONTROL CARD

When a SNAP card appears, the Snapshot Dump Routine inserts a snapshot dump at a specified location in a subprogram. The card format is:



```
   P=06001   A=0000000000000000                0   Q=0000000000000000                0
INDEX 1  00012      INDEX 2  00001     INDEX 3  00000      INDEX 4  00000     INDEX 5  00000      INDEX 6  00000
 IC1 77777 77777     OC2 77777 77777    IC3 77777 77777     OC4 07722 07722    IC5 04100 04100    OC6 77777 77777
```

Figure 6-1. Console Conditions Dump Format


```
00464  50000000 75402004  10000001 20000037  12000046 74000101  16000047 50000000  50100000 50200000  50300007 50400000
00472  50500000 50600011  75000007 50000000  74732505 75400502  50000000 75400526  50000000 75401147  50000000 75400750
```

Figure 6-2. Octal Dump Format

```
06670  STA 20575536 FAU 30407510  STA 20544430 QJP 23471240  STA 20507215 AJP 22450400  STA 20455644 LDQ 16672000
06674  STA 20424520 LAC 13710000  STA 20367346 ISK 54500000  STA 20335753 SAU 60400000  STA 20304611 FMU 32000000
```

Figure 6-3. Octal with Mnemonics Dump Format

```
06006          141        6749843246              1        -142149        62784        -9432784321
06014           -0     1234567890123           -186            14
```

Figure 6-4. Fixed Decimal Dump Format


```
06670   .1000000000+015   .1000000000+014   .1000000000+013   .1000000000+012   .1000000000+011   .1000000000+010
06676   .1000000000+009   .1000000000+008   .1000000000+007   .1000000000+006   .1000000000+005   .1000000000+004
```

Figure 6-5. Floating Decimal Dump Format

```
06556   ZROARSQR   SLRSENQA   LSQLSLLS   ENAINALD   ALACADDS   UBLDQLQC   STASTQAJ   PQJPMUID   VIMUFDVF   FADFSBFM
06570   UFDVSCAS   CQSSKSSH   SSTSCLSC   MSSULDLA   DLSBLSTL   ENIINILI   ULILISKI   JPSIUSIL   SAUSALIN   TOUIEQST
```

Figure 6-6. BCD Dump Format

The word SNAP appears in columns 2 through 5; a comma in column 6 is followed by the dump specifications which are free-field; fields are separated by commas. No blanks are allowed within the field, and the last field is terminated by a period. An omitted field is indicated by two successive commas. The specification is of the form:

$$P_1 \pm N_1, \text{ FWA, LWA, MODE, } K_1,K_2,K_3, \text{ I.}$$

$P_1 \pm N_1$ denotes the instruction location relative ($\pm N_1$) to the program P_1. P_1 is the program name, and N_1 is an octal number indicating the relative position of the instruction. This is determined from a CODAP-1 listing of a program or subprogram.

FWA, the first word address of the dumped region, may be in one of the following forms.

A_1	is an unsigned octal number denoting an absolute octal location.
$\pm N_2$	is a signed octal number denoting an octal location relative to P_1.
$P_2 \pm N_2$	denotes an octal location relative to a subprogram or common block P_2. P_2 is the subprogram or block name, and N_2 is a signed octal number.

LWA, the last word address of the dumped region, may be in one of the following forms:

A_2	is an unsigned octal number denoting an absolute location.
$\pm N_3$	is a single octal number denoting an octal location relative to P_2 if P_2 has been defined. If not, this location is relative to P_1.
$P_3 \pm N_3$	denotes an octal location relative to a subprogram or common block P_3. P_3 is the subprogram or block name, and N_3 is a signed octal number.

The MODE key indicates the format of the dump by one of the following codes:

O or Blank	octal dump
OC	octal dump with console conditions
M	octal with mnemonics dump
MC	octal with mnemonics dump and console conditions
D	fixed decimal dump
DC	fixed decimal dump with console conditions
F	floating decimal dump
FC	floating decimal dump with console conditions
B	BCD dump
BC	BCD dump with console conditions

K_1, K_2, K_3 are control parameters to specify the frequency of the dump. When the replaced instruction has been executed at the K_1 time, the specified dump is output every K_3 time the replaced instruction is executed until K_2 dumps have been output. If these parameters are omitted, the dump will be performed every time the replaced instruction is encountered.

I is an arbitrary alphanumeric used for identification with every dump, as in this example:

```
11
 0
 7SNAP, ALPHA+10,B,200,DC,1,20,1,A.
 9
```

This will dump the storage area B+200. The replaced instruction is the tenth instruction in program ALPHA, and the dump will occur the second and following times that this instruction is executed until a total of 20 dumps have been output. Each dump will be in fixed decimal format, it will include the console conditions, and will be identified by the character A.

COMMON SNAPSHOT DUMPS

When snapping a labeled or numbered common block, the parameters P_2 and P_3 take on the form . . , COMMON/block name/ ± n_2 (or n_3), . . . For blank numbered common, this takes the form . . . , COMMON/ / ± n_2 (or n_3), . . . The example below indicates that the second group of 64 words in the numbered common block 123 are to be output when the snapshot dump is executed in program BETA.

```
11
 0
 7SNAP,BETA+500,COMMON/123/+100,+100,DC,1,1,1,B.
 9
```

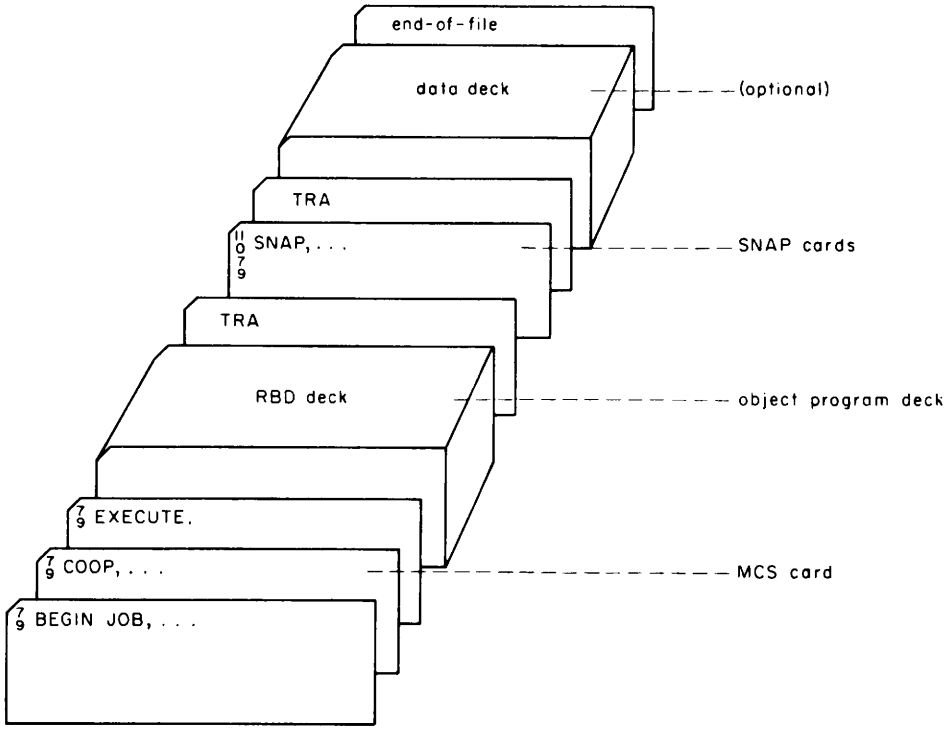
SNAP CARDS IN JOB DECKS

Any number of SNAP control cards can be included with a program in a job deck, but only the first ten are executed.

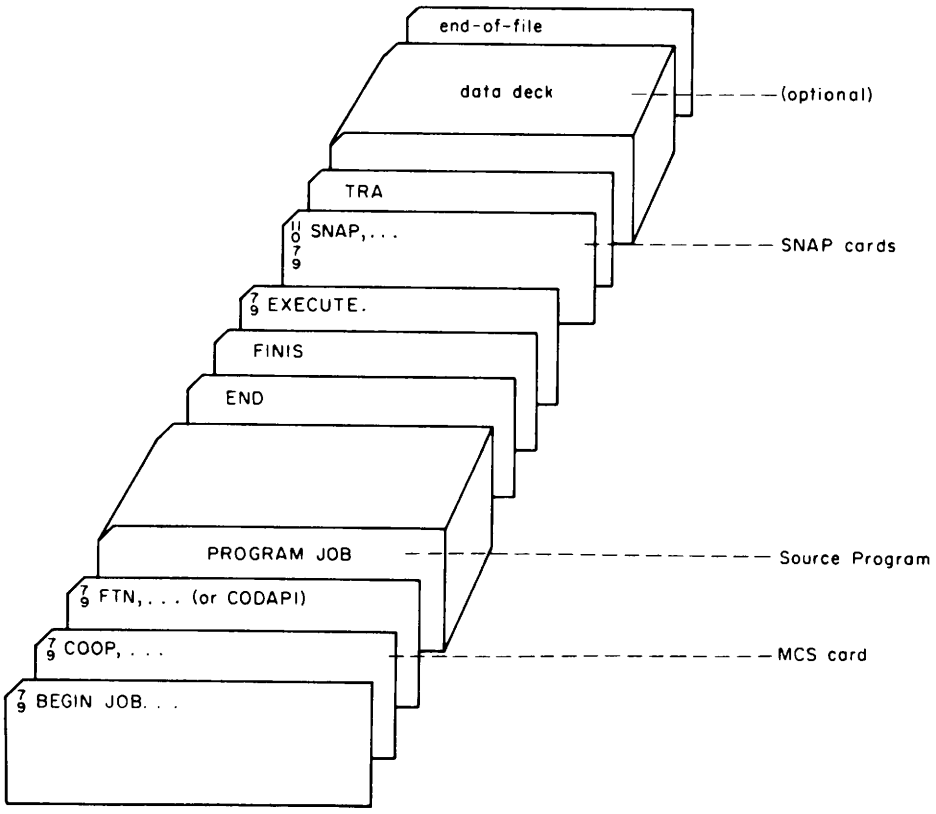
These cards are placed in any order between the two TRA cards that terminate an object program. SNAP cards can be included with a load-and-go deck for FORTRAN-63 or CODAP-1 by using a modified job deck. The modification for FORTRAN-63 and CODAP-1 consists of removing one of the two END cards at the end of the source deck and placing the SNAP cards after the EXECUTE (or EXECUTER) card, followed by a TRA card. The END card removed should not contain the transfer address for the program. For COBOL load-and-go job decks the SNAP cards are placed between the EXECUTE card and the TRA card.

SNAPS IN OVERLAYS

Snapshot dumps may be inserted in the main program or in overlay or segment. The SNAP control cards for overlays have the standard format. The control cards are inserted after the first transfer card following the main program, overlay or segment in which the dump is to occur.



Execute Only with SNAP Cards



Compile and Execute with SNAP Cards

OVERLAYS AND SEGMENTS 7

If a program is too large to be contained in core storage as a complete unit, it may be subdivided into overlays which are treated as subroutines and brought into storage by the monitor as needed. If necessary, overlays may be subdivided into segments. The LOADMAIN subordinate control system provides for execution of overlay programs.

A flow-chart showing the hierarchy of operations is helpful in determining which sections will be overlays and which will be segments. When the program is properly subdivided, each overlay and segment is coded as a separate subprogram or group of subprograms. A main or control program is required to call in the overlays during execution. Any of the library functions may be used by an overlay or segment, but library subroutines should be called by the main program, as some library routines must be in storage during the entire overlay program. An overlay or segment may use any portion of the main program as a subroutine, and a segment may use subroutines in any overlay of which the segment is a part. Only the main program, one overlay and one segment may be in storage at one time. Only the main program can call overlays, and segments may be called only by their associated overlay. The library subroutine, ROVER, is implicitly called whenever a call is made to an overlay or segment.

Information in the labeled and numbered common storage areas defined in the main program may be used by any overlay or segment. Labeled and numbered common storage areas and entry points assigned to overlays and segments can be referenced only when the associated overlay or segment is in storage.

A FORTRAN or CODAP source program consisting of a main program and one or more overlays and segments may be compiled and executed (load-and-go) at one time, or it may be compiled for later execution. In either case, the source program is compiled in the usual manner. If the operation is to be load-and-go, the relocatable binary deck is written on the load-and-go unit in the usual way with additional overlay control cards for generating an overlay tape. Once the load-and-go tape is prepared, the program is executed with the EXECUTE card. During the loading phase, each subprogram is loaded together with library subroutines and written as an absolute program on an overlay tape. The address assignments are made exactly as initially loaded by the EXECUTE card; therefore, the overlay program cannot be relocated once it is generated.

Compilation of overlays for later execution requires standard relocatable binary output; this binary output is used to generate the overlay tapes.

A maximum of four overlay tapes may be generated for one program. Overlays are numbered sequentially on each overlay tape, starting with 1. Segments within an overlay are also numbered sequentially, starting with 1.

CALLING OVERLAYS AND SEGMENTS

The following FORTRAN program illustrates how overlays are called from a main program in FORTRAN language.

FORTRAN CODING FORM

PROGRAM | CONTROL DATA NAME

ROUTINE | DATE | PAGE | OF

TYPE	STATEMENT NO.	CONT	FORTRAN STATEMENT (0 = ZERO, Ø = ALPHA O; 1 = ONE, I = ALPHA I; 2 = TWO, Z = ALPHA Z)	SERIAL NUMBER
			PROGRAM MAIN	
C			MAIN PROGRAM	
			COMMON A,B,C,EPS	
	1		READ 6,A,B,C,EPS	
			PRINT 7,A,B,C,EPS	
			IF (ABSF((A*A+B*B-C*C)-EPS)) 20,5,5	
	5		IF (ABSF((B*B+C*C-A*A)-EPS)) 20,10,10	
	10		IF (ABSF((C*C+A*A-B*B)-EPS)) 20,15,15	
	15		CALL OVER1(1,0,1)	
			GO TO 1	
	20		CALL OVER1(1,0,2)	
			GO TO 1	
	6		FORMAT (3F10.5,1F5.5)	
	7		FORMAT (1H0,3F10.5,1F5.5)	
			END	
			PROGRAM OVER1	
C			OVER LAY 1	
			PRINT 8	
	8		FORMAT (29H THIS IS NOT A RIGHT TRIANGLE)	
			END	
			PROGRAM OVER2	
C			OVER LAY 2	
			PRINT 9	
	9		FORMAT (25H THIS IS A RIGHT TRIANGLE)	
			END	
			END	

The calling sequence to an overlay or segment in FORTRAN is:

```
CALL OVERLAY(N,P,O)
```

The calling sequence for a CODAP-1 program is:

```
L       RTJ     OVERLAY
L+1     ZRO     L(N)
        ZRO     L(P)
L+2     ZRO     L(O)
        ZRO     0
L+3     Normal Return †
```

In CODAP-1, the location of N,P, and O are given in the calling sequence. These parameters are defined as follows:

- N is the logical unit number of overlay tape.
- P is the parameter to be passed to the overlay routine; L(P) is zero if no parameter is to be transmitted.
- O is the overlay number to be called.

Any logical unit number from 1 to 49 can be used as an overlay tape. Consecutive overlay tapes need not be numbered in consecutive order.

The calling sequence for segments is similar to that of overlays; to call a segment in FORTRAN, use the following:

```
CALL SEGMENT (N,P,O,S)
```

For CODAP-1, use the following:

```
L       RTJ     SEGMENT
L+1     ZRO     L(N)
        ZRO     L(P)
L+2     ZRO     L(O)
        ZRO     L(S)
L+3     Normal Return †
```

† For FORTRAN-62 master tapes the normal return is to L+4; L+3 is a dummy location.

N is the logical unit number of the overlay tape containing the segment.

P is the parameter to be passed to the segment. L(P) is zero if no parameter is to be transmitted.

O is the number of the overlay containing the segment to be called.

S is the number of segment to be called.

PARAMETER TRANSMISSION (FORTRAN)

In FORTRAN-63, P may be any standard variable type except logical (integer, real, double or complex). The parameter is picked up in the same manner as a function subroutine. Thus, if the parameter Q is to be transmitted to overlay 1 in the example above, the call to overlay 1 in the main program is:

```
CALL OVERLAY (1,Q,1)
```

The PROGRAM statement for overlay 1 is:

```
PROGRAM OV1(Q)
```

ROVER

ROVER is a library subroutine which loads and executes overlays and segments. It also controls the transmission of parameters, and the return of control back to the calling routine after the overlay or segment has been executed. ROVER is implicitly called by the statements CALL OVERLAY (RTJ OVERLAY) and CALL SEGMENT (RTJ SEGMENT) as OVERLAY and SEGMENT are entry points to ROVER. ROVER is automatically loaded with the main program at execution time if the main program contains a call to an overlay or segment.

The parameter identifiers need not be the same, as each is a local identifier.

PARAMETER TRANSMISSION (CODAP-1)

In machine language, the address of the parameter P is transmitted to ROVER, therefore the overlay or segment must obtain this address from ROVER. The calling sequence generated by ROVER to call the appropriate overlay or subroutine is as follows:

```
L       RTJ     Entry point of overlay or segment
        ENI  0
L+1     ENI  0  Parameter address
        ENI  0
```

When the RTJ instruction is executed, the upper address of the entry point word in the overlay or segment contains the address L+1 of the ROVER calling sequence. The upper address of L+1 contains the address of the parameter to be transmitted. The following CODAP-1 overlay example illustrates one method of obtaining the parameter through ROVER.

```
        IDENT       OV1
OV1     SLJ     0   **
        SIU     1   SAVE
+       LIU     1   *-1
        LDA     1   0          (L+1 in ROVER call)
+       ARS     0   24
        SAL     0   PARAM
PARAM   LDA     0   **         (Parameter now in A-register)
                .
                .
                .
SAVE    LIU     1   **
        ENI     0
        SLJ     0   OV1
        END         OV1
```

RULES FOR OVERLAYS AND SEGMENTS

1. Overlays and segments must be written as closed subroutines entered by return jump instructions.
2. Only the main program, one overlay, and an associated segment may be in storage at any time.
3. A segment cannot reference external symbols in another segment or an overlay of which it is not a part.
4. An overlay cannot reference internal symbols in another overlay.
5. The main program cannot reference external symbols in any overlay or segment.
6. The main program and the current overlay and segement can communicate via labeled common. Numbered common may be used to communicate with overlays and segments not in core.
7. Parameters may be transmitted from a main program to any overlay.
8. Parameters may be transmitted from an overlay to any of its segments.
9. A segment can be called only from its associated overlay or from the main program.

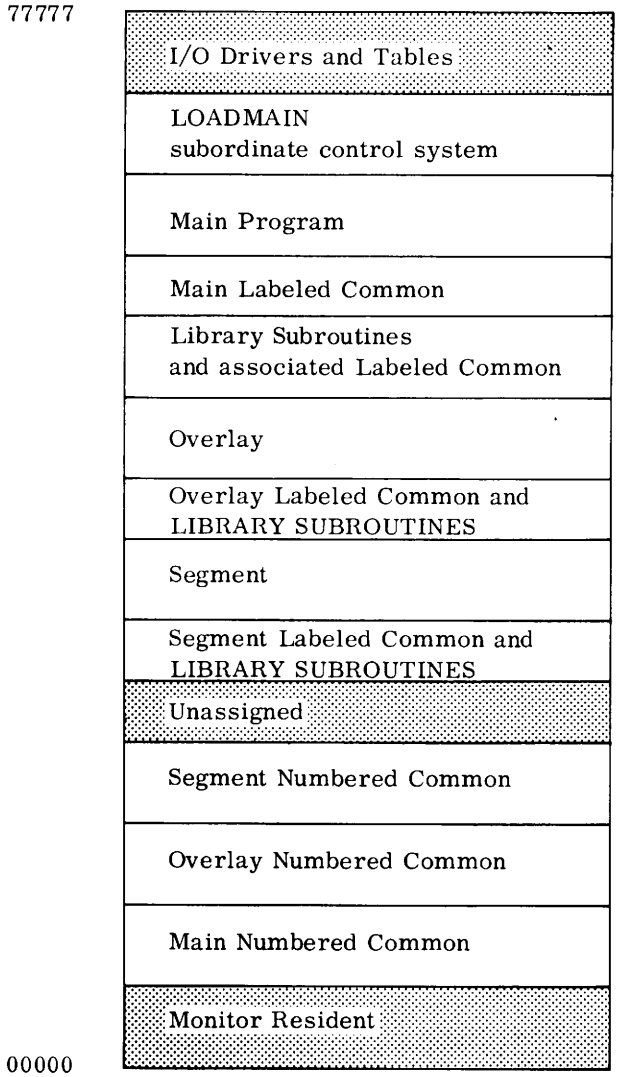
10. An overlay can be called only by the main program.

11. Overlays are numbered sequentially starting at 1, on each overlay tape.

12. Segments are numbered sequentially starting at 1, for each overlay.

OVERLAY STORAGE ASSIGNMENT

Only one overlay and segment can be in storage at one time. The storage assignment made to the main program, library subroutines and the currently loaded overlay and segment is shown below.



The block reserved for the library subroutines assumes they were called by the main program; otherwise, the library subroutine falls at the end of the overlay or segment.

Separate numbered common assignments are made if main, overlay and segment contain different numbered common block references.

If another segment of the same overlay is called in, the new segment is loaded on top of the old segment, leaving the original overlay intact, and destroying the previous segment. Labeled and numbered common for the new segment replace the common areas of the old segment. If a new overlay is called in, only the main program, associated library subroutines and numbered common remain intact, as the new overlay is loaded on top of the previous overlay.

When an overlay program is executed, the fixed portion of storage (resident CO-OP and I/O Drivers) must not occupy more storage space than when the overlay tape was originally made. Otherwise, the linkages between the overlay program and the monitor would not be correct.

CONTROL CARDS

Three control cards, MAIN, OVERLAY, and SEGMENT, are used in generating overlay tapes. These cards have the following formats:

```
11                  11                  11
 0                   0                   0
 7MAIN, n.           7OVERLAY,n,o.       7SEGMENT,n,o,s.
 9                   9                   9
```

Each field is separated by a comma; the last field is terminated by a period. The parameters are defined as follows:

- n is the logical unit numbers on which the associated main program, overlay or segment is to be written.
- o is the identification number of overlay. For SEGMENT cards, this is the number of the overlay containing the segment.
- s is the identification number of segment.

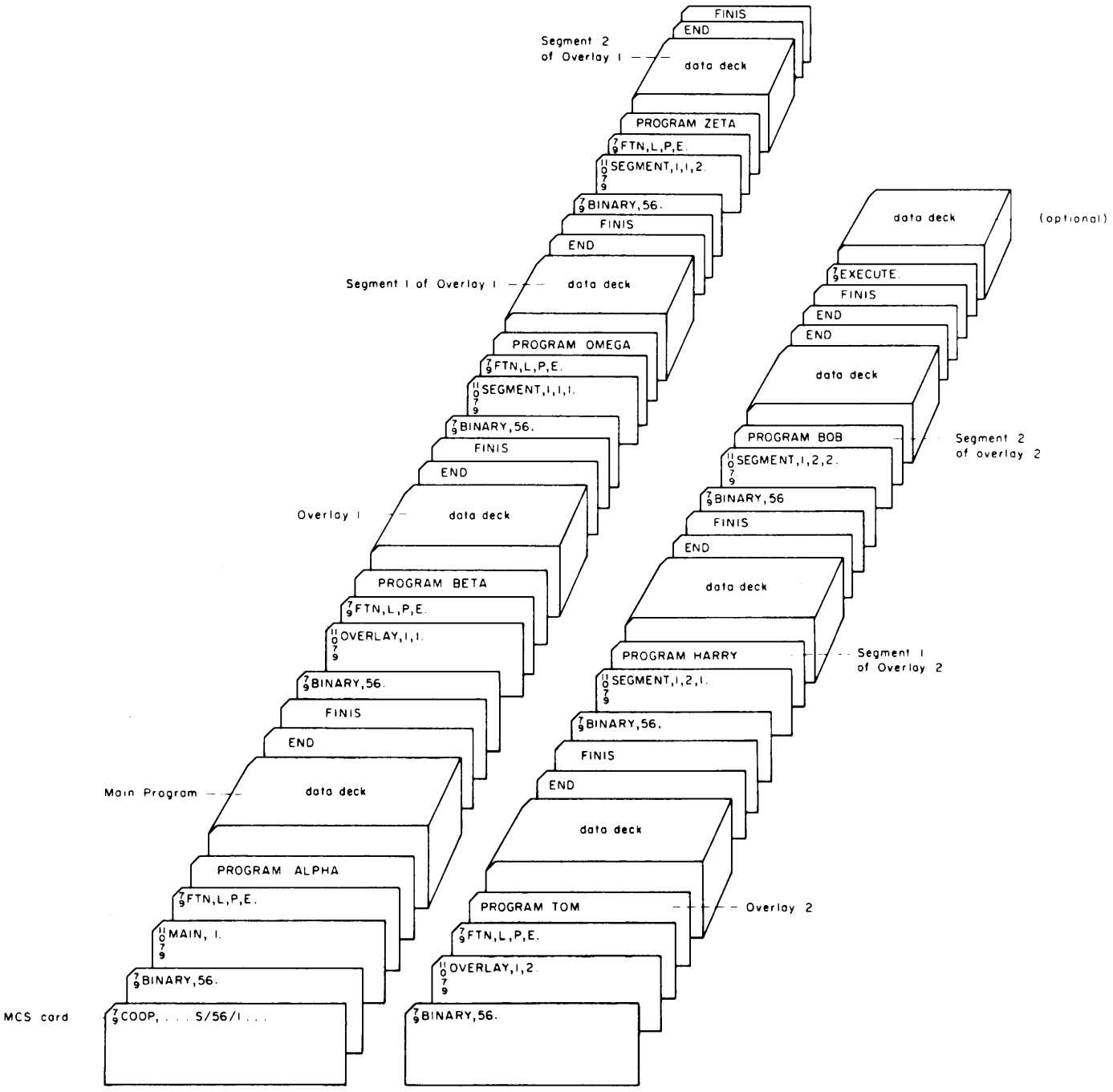
The parameters n, o, and s on the control card must correspond to the parameters n, o and s in the CALL statement or CODAP-1 calling sequence using the overlay or segment. The logical unit number n must also appear in the I/O list on the MCS card, defined either as a scratch unit or as both an input and output unit.

OVERLAY JOB DECK STRUCTURES

Overlay tapes may be generated and executed as a load-and-go operation or may be generated in one operation and executed at a later time.

LOAD-AND-GO JOB DECKS

The following diagrams illustrate the position of the required control cards for a FORTRAN overlay program, a CODAP-1 overlay program, and a mixture of the two, including previously compiled binary decks.



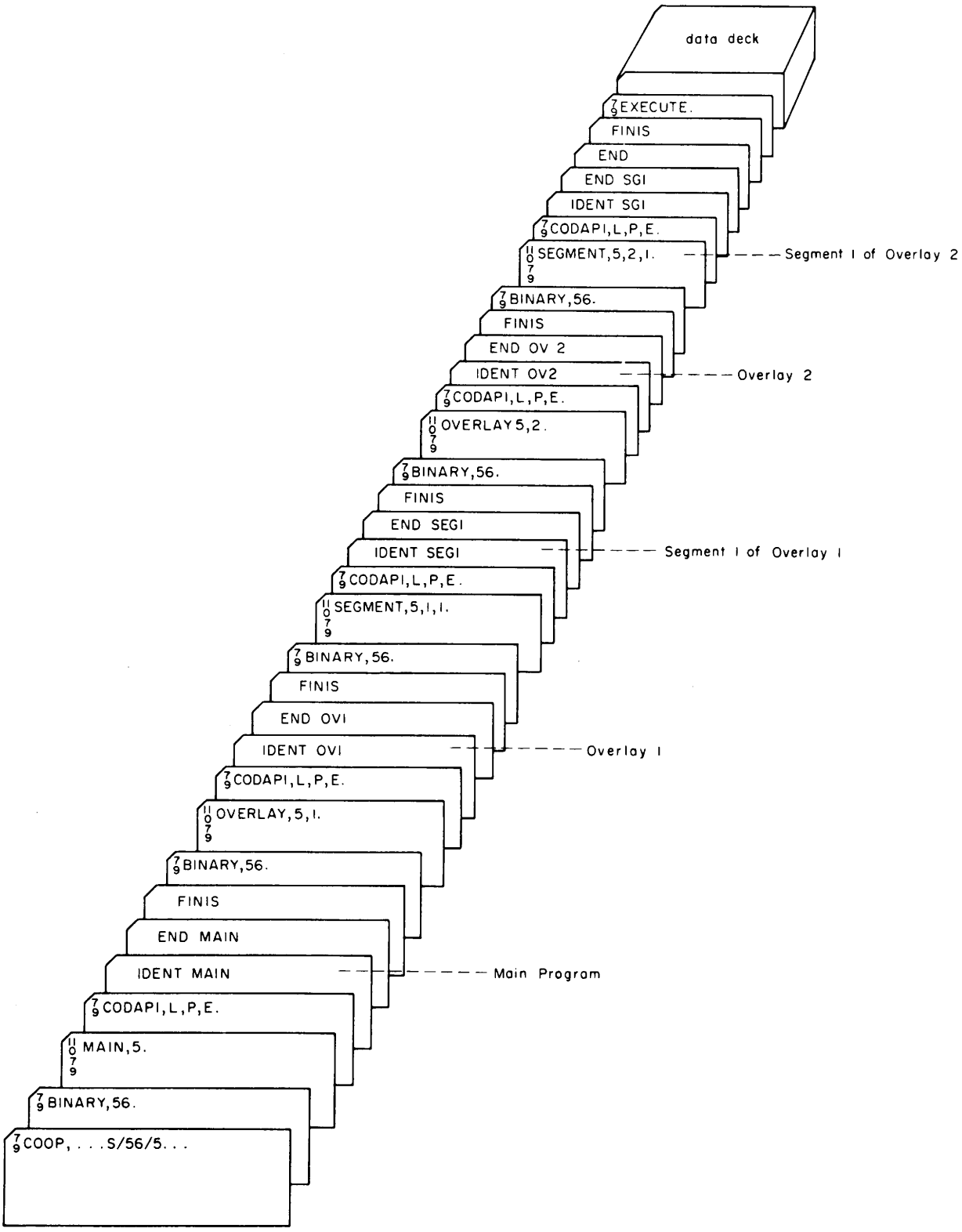
The job deck pictured above compiles a main program, two overlays and four segments. The intermediate (relocatable) binary output is written on unit 56, and the final (absolute) overlay tape on logical unit 1. These units are defined on the MCS card. The BINARY cards are used to transfer the overlay control cards to the load-and-go unit 56. These control cards are required for identification of the main program and each overlay and segment.

The overlay process starts with the normal compilation of the program, with the relocatable binary output written on the load-and-go unit specified on the FTN cards. On the load-and-go tape, each program is preceded by a MAIN, OVERLAY, or SEGMENT card and terminated by one END card. The last segment or overlay in an overlay program deck is terminated by two END cards. If the program size requires that more than one overlay tape be generated, only the last overlay or segment to be executed is terminated by two END cards. All other overlays and segments are each terminated by the one END card. The END card for each overlay or segment must contain the transfer address for that overlay or segment. The last overlay or segment to be executed must be physically the last program on the overlay tape. Only one of the END cards can contain a transfer address.

After the load-and-go tape has been written, the EXECUTE card is read and the overlay tapes are prepared. The main program and each overlay and segment, including all library routines, are written consecutively on the logical tape unit specified on the control card (1 in the example). All of these programs, subprograms and subroutines are written on tape in absolute binary form and are not relocatable. As soon as the tape generating process encounters two consecutive END cards, the main (control) program is loaded and execution begins.

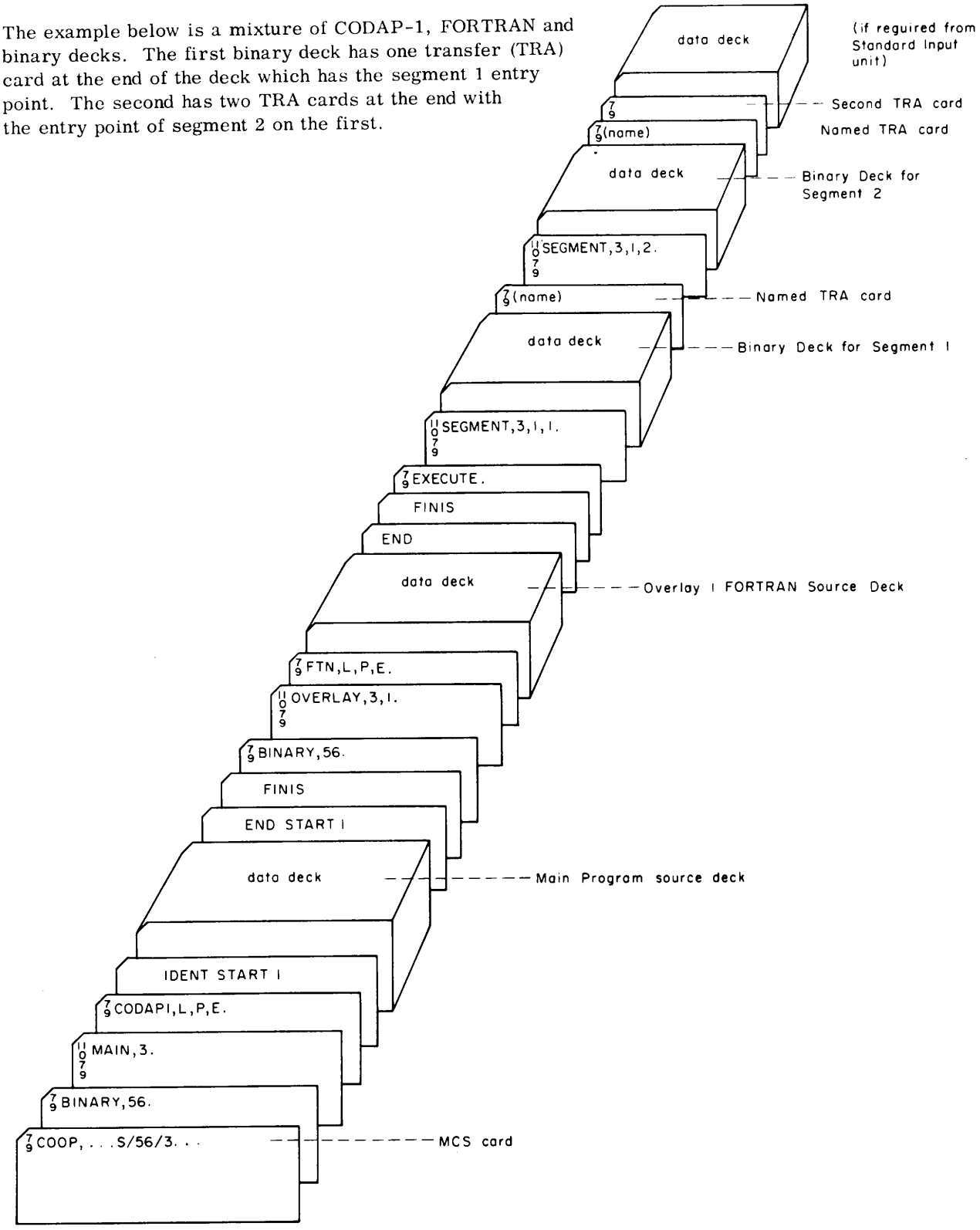
The following diagram illustrates a CODAP-1 program with two overlays and two segments. The load-and-go tape is on unit 56, and the overlay tape is written on unit 5. Each CODAP-1 source deck begins with an IDENT card, and is terminated by an END card containing a symbolic transfer address, and a FINIS card.

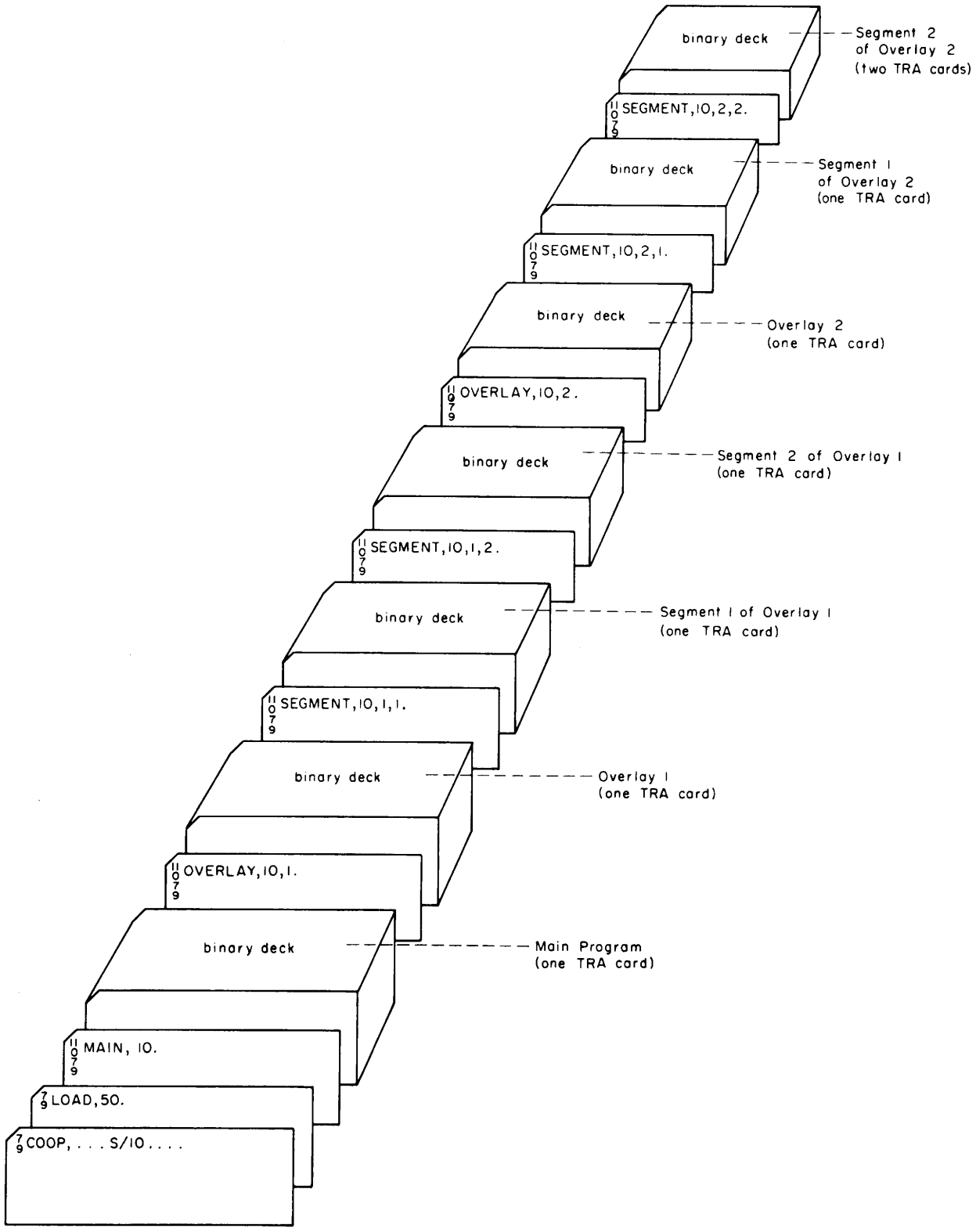
(The symbols in the example are arbitrary.) The last segment in the deck is terminated by a second END card.



CODAP-1 LOAD-and-Go Job Deck

The example below is a mixture of CODAP-1, FORTRAN and binary decks. The first binary deck has one transfer (TRA) card at the end of the deck which has the segment 1 entry point. The second has two TRA cards at the end with the entry point of segment 2 on the first.





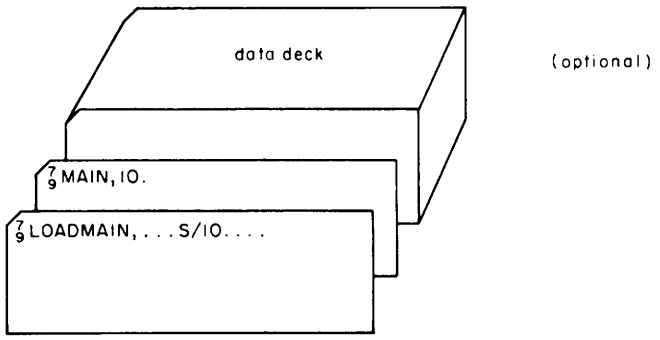
Job Deck for Preparing Overlay Tape from Binary Decks

GENERATING OVERLAY TAPE FOR LATER EXECUTION

The job deck in the preceding illustration is used to create an overlay tape from relocatable binary decks. The LOAD control card loads each program into storage from the standard input unit (50) using the standard binary loader. The loader interprets the MAIN, OVERLAY and SEGMENT control cards, and the relocatable binary decks are written on the overlay tape.

The binary decks are the output of either FORTRAN or CODAP-1 compilations. Each has only one TRA card at the end which contains the transfer address for that overlay or segment. The last binary deck in the job has two TRA cards indicating the end of the program deck. One of these must contain the transfer address for that overlay or segment.

To execute the overlay tape created by the job deck shown above, the following control cards are placed on the standard input unit as one job deck.



LOADMAIN acts as a subordinate control system in place of COOP. It reads in the MAIN control card and loads the main program from the logical tape unit indicated on the MAIN control card. This MAIN control card has only a 7,9 punch in column 1, instead of the 11,0,7,9 punch for the MAIN control card used in creating the overlay tape.

Control is turned over to the main program after loading and the overlay program is executed.

OVERLAY ERROR DIAGNOSTICS

If an error of the type listed below occurs in an overlay program, the monitor subroutine ERROR outputs an error message on the standard output unit in the following format:

SUBR. ERROR. IN OVERLAY / SEGMENT or AT TTTTT TYPE ZZ,A=X00000SSSSSNNNNN

TTTTT — indicates the absolute address of the location where the transfer to the error subroutine occurred.

TYPE ZZ — indicates the kind of error according to the following list:

Code	Error Type	Possible Cause
OV	Overlay	Requested overlay is not on tape. O = 0 in FORTRAN overlay call. L(O) = 0 in CODAP-1 overlay call. L(S) $\neq$ 0 in CODAP-1 overlay call.
SC	Segment	Associated overlay is not currently in core. Requested segment is not in current file. S = 0 in FORTRAN segment call. L(S) = 0 in CODAP-1 segment call.
LT	Logical unit number	Logical unit number does not fall into range: $1 \leq LUN \leq 49$
TO	Tape overflow	More than four logical tapes requested.
TP	Tape position	Requested overlay, segment, or tape numbers do not agree with those on tape record. First word of requested record was not read within 300 milliseconds after read command was activated.
PE	Parity Error	Parity check error encountered on three successive readings of requested record.

The code in A register is interpreted as follows:

X = 7	indicates that the word in error is an end-of-file word.
X = 0	indicates that the record associated with error is not an end-of-file word.
OOOOO	the number of the overlay containing the error.
SSSSS	the number of the segment containing the error SSSSS is zero if the error did not occur in a segment.
NNNNN	the logical tape number of the overlay.

After the error message is written, the overlay program exits to the monitor through ERROR*, and the job is terminated. ERROR* will produce a dump as indicated by the recovery key on the MCS card.

APPENDIX SECTION

EQUIPMENT ASSIGNMENT (AET - RHT)

A

The assignment and control of input/output equipment by the system is accomplished by the Available Equipment Table, the Available Equipment Driver Name Table, and the Running Hardware Table.

AVAILABLE EQUIPMENT TABLE

The AET table contains an entry for each input/output unit connected to the 1604 that can be used by the monitor. This table includes as a subtable the assignments for the standard input/output units; unit entries may be in any order. Each installation constructs its own AET table.

AET is in the first record on the library tape and is read into memory with the bootstrap loader.

AET ENTRY FORMAT

The entries of the AET consist of a 48-bit description of each unit and its status in the following format:

Bit Position	Description
47-45	Allowed Use of Unit 000 not available 001 input only 010 output only 011 input and output 100, 110 not used 111 not operable (down)
44-43	Assignment Key 11 assigned as I or O for this job 10 assigned as scratch for this job 01 assigned as I or O for last job 00 available
42	Assigned as a standard unit

Bit Position	Description
41-36	Used in AET entries 1 through 15 as a standard unit subtable entry giving the entry number of the standard units in the AET. These bits are not used in the remaining entries in the AET table.

The standard unit subtable has the following fixed format; the logical unit number is not contained in the subtable but is included for reference.

AET Entry Number	Description	Logical Unit Number
1	library unit	0
2	standard input	50
3	standard output	51
4	standard punch	52
5	comment from operator	53
6	comment to operator	54
7	accounting unit	55
8	standard scratch unit 1	56
9	standard scratch unit 2	57
10-15	not used	

For example, if entry 3 in an AET table contains an octal 20 in bits 41-36, it indicates that entry 20 in the AET table contains the status and description of the standard output unit.

35-24	Equipment Code
35-33	Channel Number
32-30	Cabinet Number
29-27	Sub-Cabinet Number
26-24	Unit Number
23-18	Hardware Code

00	typewriter
01	card unit
02	printer
03	paper tape
04	magnetic tape
05	magnetic tape (high density only)
06	cathode ray display
07	disc file
10	drum storage
11, 12, 13	not used
14	satellite magnetic tape unit
15	satellite magnetic tape unit (high density only)
16-77	not used

Bit Position	Description
17-15	Not Used
14-0	Location in AEDNT that contains name of equipment driver routine required for the unit described in this entry.

AET LISTING

The AET table can be listed and modified as described in Appendix B. The listing of an AET table having 20 entries appears below:

```
                Subtable
                   ↓
01 = 0005120104014576
02 = 0015120204014576
03 = 0006120304014576
04 = 0017120404014576
05 = 3121320104010301
06 = 3121320204010301
07 = 3024320304014576
10 = 3007320404014576
11 = 3010520104014576
12 = 3000520204014576
13 = 3000520304014576
14 = 3000520404014576
15 = 1100141001011243
16 = 1000142001014600
17 = 2100140201011640
20 = 2000760002012477
21 = 3100111000012201
22 = 0000112203014610
23 = 1000112103010723
24 = 2100112403011164
```

The third and fourth digits from the left in each entry comprise the standard unit subtable. This subtable indicates that entry 5 describes the library unit, entry 15 describes the standard input unit, entry 6 describes the standard output unit, and so on.

AVAILABLE EQUIPMENT DRIVER NAME TABLE

The AEDNT table contains the names of all equipment driver routines available either in resident or on the library tape. Each entry in the AEDNT consists of two words. The first word contains the driver name in BCD, left justified with blank fill. The second word indicates whether or not the driver is already in storage. If the driver is already in storage, the upper half of the second word consists of 77 0 XXXXX where XXXXX is the address of the driver in storage. If the driver is not in storage when the job sequencer

references the AEDNT, the upper half of the second word is zero. This indicates that the driver must be loaded from the library tape. If the driver is not on the library, a return jump is executed to the lower half of the second word, which consists of SLJ O HOWLER, an alarm routine that indicates to the operator that the driver was not found in the library.

RUNNING HARDWARE TABLE

The RHT table consists of all AET entries required to process the current job. Entries are ordered according to logical unit number and the table is fixed length, 64 entries. RHT is created for each job from the I/O list in the MCS card, and from the standard unit subtable in the AET table. For each job, entries 0 and 50 through 55 in the RHT contain the AET entry for each standard unit.

When an MCS card is processed, the AET table is scanned for unassigned equipment. As an entry is found that corresponds to the description in the I/O list on the MCS card, this entry is transferred to an RHT entry position corresponding to the logical unit number in the I/O list.

When the entry is transferred, two modifications take place, and the associated equipment driver is loaded into the high portion of core storage. When the routine that processes the MCS card I/O list finds an appropriate AET entry, the AEDNT table is searched to find the name of the equipment driver for that entry. If the driver is not in storage, the driver is loaded from the library tape. The AET entry is then transferred to the RHT, with bits 14-0 now containing the absolute address of the entry point of the associated equipment driver routine. The assignment key for that entry is set to the appropriate value in the AET, and the I/O list processor proceeds to the next logical unit number.

Assignments are processed in the following order:

1. All unit numbers assigned as both I and O units
2. All S units
3. All I only units
4. All O only units
5. All E units

If an error is encountered during I/O list processing, the job is terminated. The RHT is set for each job and cannot be altered during the job.

OPERATOR CONTROL B

The operator and the monitor communicate with each other through a central control routine. The operator supplies information on the typewriter; the monitor supplies information on the typewriter and the comment-to-operator unit which may be any output device connected to the computer, but will usually be the typewriter.

The operator may supply information to the monitor under three conditions:

When the monitor requires information to continue processing.

When the program being processed requires information to continue running. (See Chapter 5.)

When the operator wishes to interrupt normal processing to impose a new requirement on the system or to supply equipment information.

The information supplied by the operator is a control statement, a message of less than 33 characters, that is always initiated in the same way. The monitor types an asterisk (*) on the typewriter when it is ready to receive an interrupt from the operator. To type a control statement, the operator

1. Strikes the carriage return key to interrupt processing.
2. Waits until the monitor types a period to indicate readiness to accept a message.
3. Types the message and terminates it with a period and carriage return.

CONTROL MESSAGE ERRORS

If the message does not contain a terminal period, or if an apostrophe (') is typed before the terminal carriage return is typed, the message will be ignored; the monitor will type a carriage return and a double asterisk (**) to so indicate. The operator may return to step 1 above to retype the message.

If the routine requested is not available, the letter "u" will be typed followed by a carriage return and a double asterisk (**). The operator may return to step 1 above to type another message.

If the message line contains more than 33 characters including the terminal period, it will be ignored. The control routine will type a CR followed by "too long", another CR, and a double asterisk (**). The operator may return to step 1 to type another message.

OPERATOR CONTROL STATEMENTS

Operator control statements which may be typed are listed below:

Flag setting statement

f, a /n, . . . (See Chapter 5)

Operator program communication statement

com, . . . (See Chapter 5)

JOB SEQUENCER STATEMENTS

Initiation statements may be given only after initial loading of the monitor or after a monitor run has been terminated.

m.	Rewind standard input unit and start processing.
m,xxx.	Find job xxx (operator's identification number) on the standard input unit and start processing.
m,n.	Skip to the next job on the standard input unit and start processing.
m,xxx,n.	Find job xxx (operator's identification number) on the standard input unit, skip to the next job and start processing.
m,t.	Backspace the standard input unit to the beginning of this job and start processing.

Termination statements may be given whenever the operator communication routine is ready to accept an interrupt; they are equivalent to an END MONITOR INPUT card in the job deck.

mstp.	Terminate the current monitor run immediately.
mstp,t.	Terminate the current monitor run at the end of the current job.
mstp,xxx.	Terminate the current monitor run at the end of the job xxx (operator's identification number).

AET TABLE CONTROL

These statements may be used to display and update the Available Equipment Table and to reassign the standard units.

aet,o,n.	Contents of the AET will be written on the unit specified by AET entry n, a decimal number.
aet.	Contents of the AET will be written on the comment-to-operator unit.
aet,i,n.	Contents of the AET will be replaced by the AET read from the unit specified by n, a decimal number.
sio.	Entries of the AET assigned as standard input/output units will be written on the comment-to-operator unit.
aet,n/m.	Current contents of AET entry n will be replaced by m; m may be a 16-octal digit quantity or one of the following alphabetic codes that modifies bits 47-43 in entry n, a decimal number.

n	not to be used
i	use as input only
o	use as output only
io	use as input and output
d	unit down

sio,$n_1/m_1,n_2/m_2 \ldots n_4/m_4$.

This statement changes standard unit assignments. Standard input/output unit n_i will be assigned the unit specified by AET entry m_i. n_i is a code or a number from the following list.

Code	Number	Description
1	0	library unit
i	50	standard input unit
0	51	standard output unit
p	52	standard punch unit
ic	53	comment-from-operator unit
oc	54	comment-to-operator unit
a	55	accounting unit
1s	56	standard scratch unit 1
2s	57	standard scratch unit 2

Operator-program communications are described in Chapter 5.

Additional operating procedures are described in the CO-OP Monitor Operator's Guide, publication no. 509.

PART I BINARY LOADER C

The monitor binary loader is used for loading programs and monitor library routines, including subsystems, compilers and assemblers. A program may consist of one or more subprograms. Subprograms and library routines are in the form of relocatable binary cards or card images. Subprograms may be loaded from punched cards or magnetic tape; library routines are loaded from the currently defined system library tape.

The binary deck of a subprogram contains six types of cards generated by a compiler or assembler. The deck may also contain one or more of the loader control cards described in chapters 2 and 6. All cards except the loader control cards are punched in column binary format. Column binary cards for the 1604 are punched four columns for each 48-bit word, starting with columns 1-4 which represent the first word on the card. Row 12 of column 1 represents bit 47 of the first word; row 9 of column 4 represents bit 0 of the first word. One card contains 20 words.

The six card types described below are generated by compilers and assemblers that operate under monitor control.

IDC IDC Subprogram identification card. The first IDC card in a subprogram deck contains the FWA and the LWA + 1 of the subprogram, followed by the subprogram name in BCD. The name of the subprogram is arbitrary and need not correspond to the symbolic entry points within the subprogram. If a CODAP-1 subprogram contains an I/O pseudo instruction, the logical unit numbers in the I/O m-term appear on the first IDC card. Additional IDC cards are used to contain the names and lengths of numbered and labeled common blocks declared in the subprogram. IDC cards are generated by the IDENT and BLOCK pseudo instructions in CODAP-1.

RBD Relocatable binary program card contains the machine language code of the subprogram. One or more RBD cards contain the machine instructions for the subprogram.

EPT

Entry point table card contains the symbolic name (in BCD) of each entry point defined in the subprogram and the corresponding relocatable address within the subprogram. Entry points may occur anywhere within the subprogram. EPT cards are generated by ENTRY pseudo instructions in CODAP-1.

EXT

External symbol table card contains all symbols declared as external to the subprogram. These are the symbolic entry points of other subprograms and library subroutines. EXT cards are generated by EXT pseudo instructions in CODAP-1.

LAT

Linkage address table card gives the relocatable address within the current subprogram of an instruction which references an external symbol. LAT cards are generated by EXT pseudo instructions in CODAP-1.

TRA

Transfer card indicates the end of a subprogram; one is required at the end of each subprogram. The last subprogram to be loaded must have two cards at the end of the subprogram deck to indicate the end of the program. Only one of the TRA cards in the program deck may contain the name of the entry point to which control will be released when loading is complete. Columns 2 through 8 of the TRA card are blank with the named transfer entry point in Hollerith beginning in column 9. TRA cards are generated by END cards in CODAP-1.

These cards make up a subprogram deck in relocatable format. The card formats are described in detail in Part II. Cards are ordered within a single subprogram in the same sequence as the card type descriptions above. A subroutine on a library tape does not contain EPT or EXT cards.

The loader uses the information on these cards to generate an absolute binary object program in storage. The object program contains only absolute addresses. The subprogram is assigned to a block of storage determined by the loader, and all relocatable addresses and external symbols are assigned absolute addresses during the loading process. At the same time all labeled and numbered common blocks are assigned in the order given in the memory layout in Chapter 1.

LOADING SEQUENCE

Subprograms are loaded in the following sequence:

1. The subprogram is loaded.
2. A search is made for all external symbols listed on the EXT cards for that subprogram. The search is made in the following order:
    a. Checks if used as entry point of previously loaded subprogram.
    b. Checks if this is an entry point of a library subroutine.
    c. Checks if this is an entry point of the resident monitor. If no entry point is found, the external symbol is held in a table so that subsequent subprograms can be checked for this symbol. If no corresponding entry point is found, an error is flagged and the job is terminated.
3. All library routines named by the subprogram external symbols have space in core reserved for later loading by the system.
4. The next subprogram is then loaded, and the process repeated.
5. The library subroutines are loaded.
6. All external symbols referenced in each subprogram are replaced by the appropriate absolute addresses before the loading process is terminated.

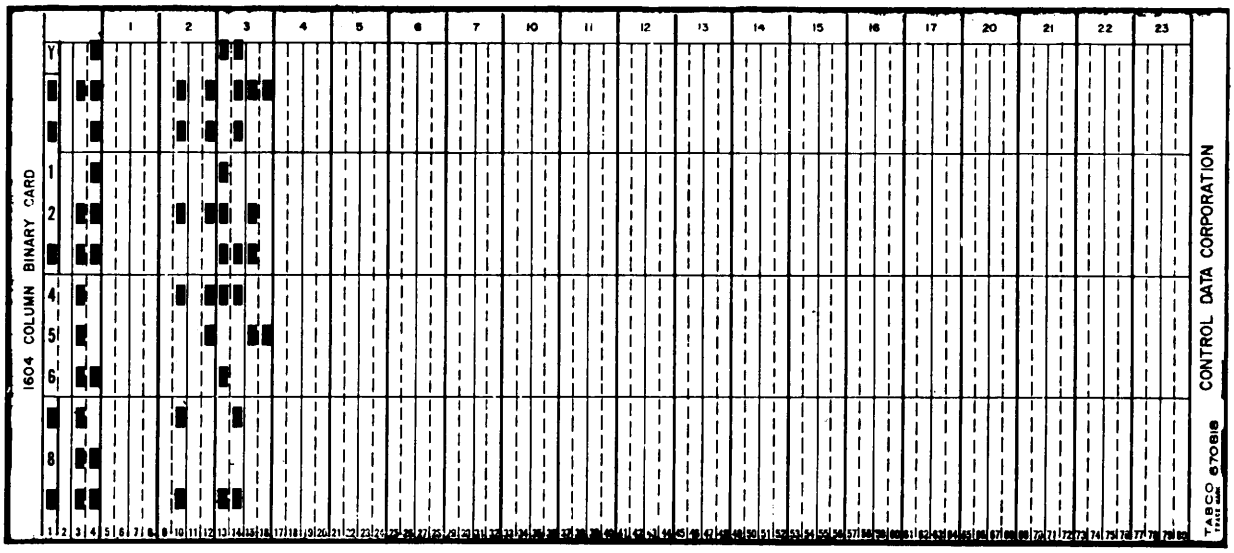
PART II BINARY CARD FORMAT

The binary cards interpreted by the loader are in column binary format. Columns 1 and 2 contain punches identifying the card, as well as punches for indicating addresses and control and sequence information. Octal addresses are those that appear on the assembled listing.

IDC CARD (First)

The first IDC (31) card in a program deck contains the name and first and last word addresses of a subprogram in the following format.

Column	Content
1	11,0,3,7,9. ($11,0,3=31_8$)
2	blank
3 and 4	checksum (ignored if row 8 in column 1 contains a punch)
5,6,7,8	blank
9 and 10	relocatable octal FWA of the subprogram (blank if no labeled common in subprogram)
11 and 12	relocatable octal LWA + 1 of the subprogram
13,14,15,16	BCD program name of the subprogram (generated from IDENT card in a CODAP-1 subprogram)
29-80	programmer assigned I/O unit numbers generated from the I/O pseudo instructions in CODAP-1 with a maximum of 63 I/O unit numbers.

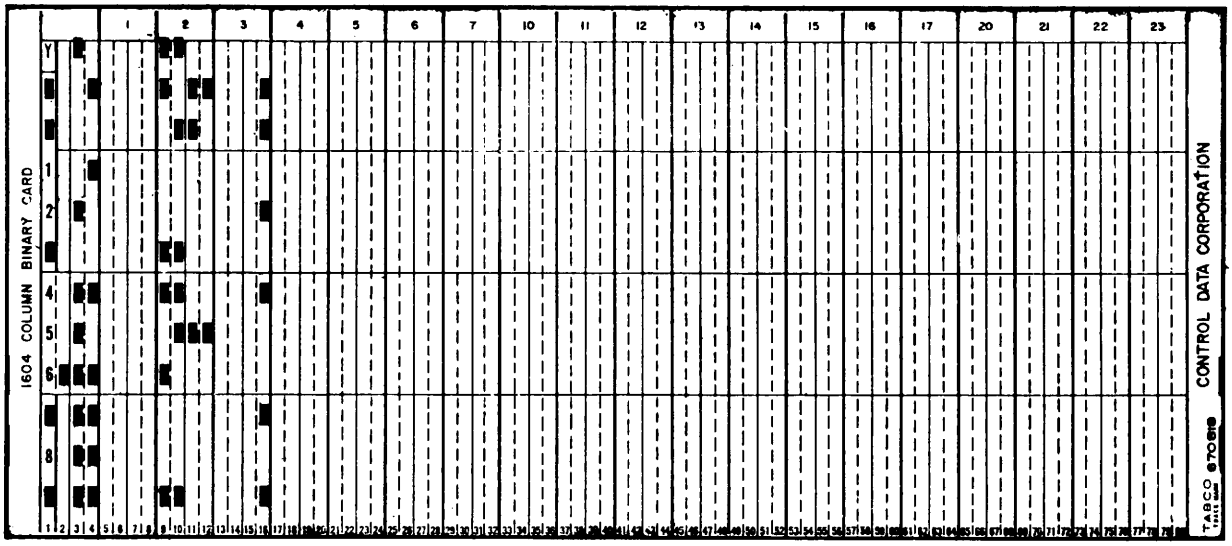


First IDC Card

IDC CARD (Second and Following) As many IDC cards as necessary are used in a binary deck.

Column		Content
1		11,0,3,7,9
2		octal sequence number:
	Card 2	0010 (in row 6)
	Card 3	00020 (rows 5 and 6)
	Card 4	00030 (rows 5 and 6)
		Additional IDC cards follow the same sequence. The high order digit of the sequence number is punched in rows of 5 and 6 in column 1.
3 and 4		checksum (ignored if row 8 in column 1 contains a punch)
5,6,7,8		blank
9,10,11,12		BCD name of first common block in subprogram (The first BCD character indicates numbered or labeled common, and is generated by the BLOCK pseudo instruction.)
13 and 14		blank

15 and 16	octal length of previously named block. This number starts in rows 7 through 9 in column 15 and continues in column 16. Remaining columns on the card repeat the scheme for column 9 through 16 for each common block.



Second IDC Card, containing one block name

RBD CARD

An RBD card contains part or all of the instructions in a subprogram. These instructions are in column binary format, with a group of four columns representing a complete word. Columns 1 through 4 represent the first 48-bit word, columns 5 through 8 the second 48-bit words, and so on. Row 12 of the first column represents bit 47, and row 9 of the fourth column represents bit 0.

First Word (Columns 1 through 4)

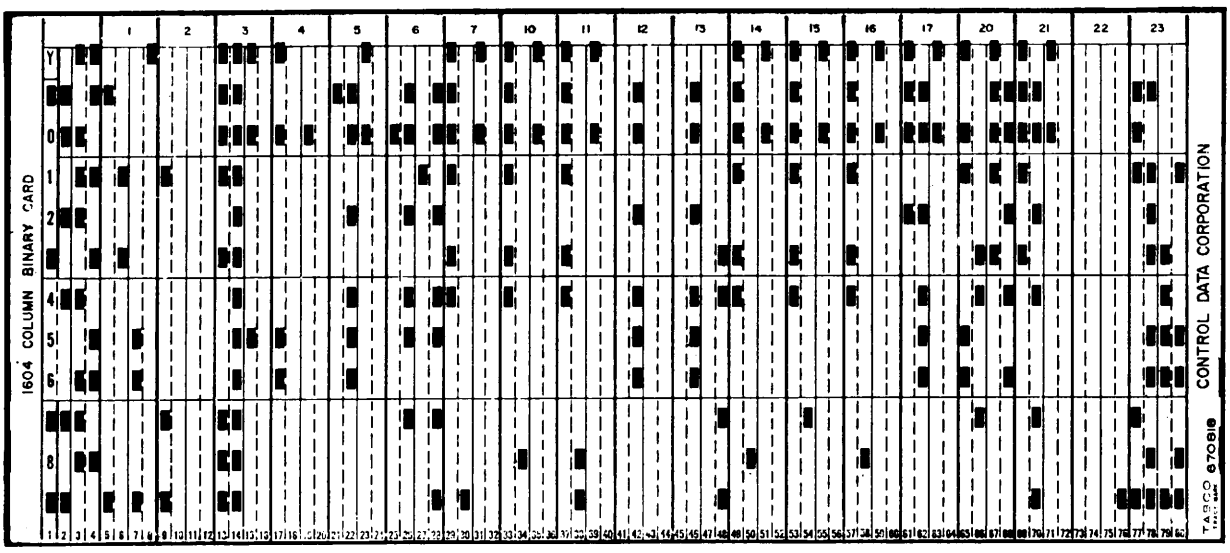
Column	Content
1	rows 12 through 3, six rows contain the word count, a two-digit octal number indicating the number of instruction words on the RBD card.
1	rows 4 through 6, three rows represent the high order octal digit of the relocatable address of the first instruction word on the card.

1	rows 7 through 9: 7 and 9 contain a punch. If the checksum is to be ignored, 8 is also punched.
2	the remaining four octal digits of the relocatable address of the first machine word on the card. The low-order digit is in rows 7, 8 and 9.
3 and 4	checksum

Second Through Fifth Word (if necessary)

These words comprise the relocation bit field. The relocation bits indicate which instruction contains relocatable program or common block addresses. Only as many words are punched as required. The relocation bits are described below.

The remainder of the RBD card contains machine instructions; the number of instruction words are indicated by the word count in column 1.



RBD Card

Relocation Bits

The relocation bits on a card are grouped in sets or bytes and the length of each byte depends on the number of common blocks in the subprogram. Bytes are

counted by column starting in row 12 in column 5. As many bytes are included as there are instructions on the card. Dummy bytes are included for BSS storage words and constants within the program. A byte may be split between two columns but not between words. The first byte of the relocation bit field is an extra byte for the starting address punched in columns 1 and 2 (column 5, row 12).

The length of each byte is computed as follows; the number of words used to contain the relocation bytes is indicated by the right hand column.

No. Common Blocks	Byte Length	No. Relocation Words
none	1 bit	1
1 or 2	2 bits	2
3 through 6	3 bits	3
7 through 14	4 bits	3
15 through 30	5 bits	4
16 through 62	6 bits	4

The value of the byte indicates if the address is absolute. If it is to be relocated, it indicates whether it is to be relative to the absolute address of the first word of the subprogram or relative to the absolute starting address of a common block. The absolute addresses are contained in a table within the loader in which the common blocks are listed in the order of appearance on the IDC cards.

Byte Value (Octal)	Absolute Address Used for Modification of Instruction Address
0	No modification
1	FWA of subprogram
2	FWA of 1st common block
3	FWA of 2nd common block
4-75	FWA of 3rd through 64th common block

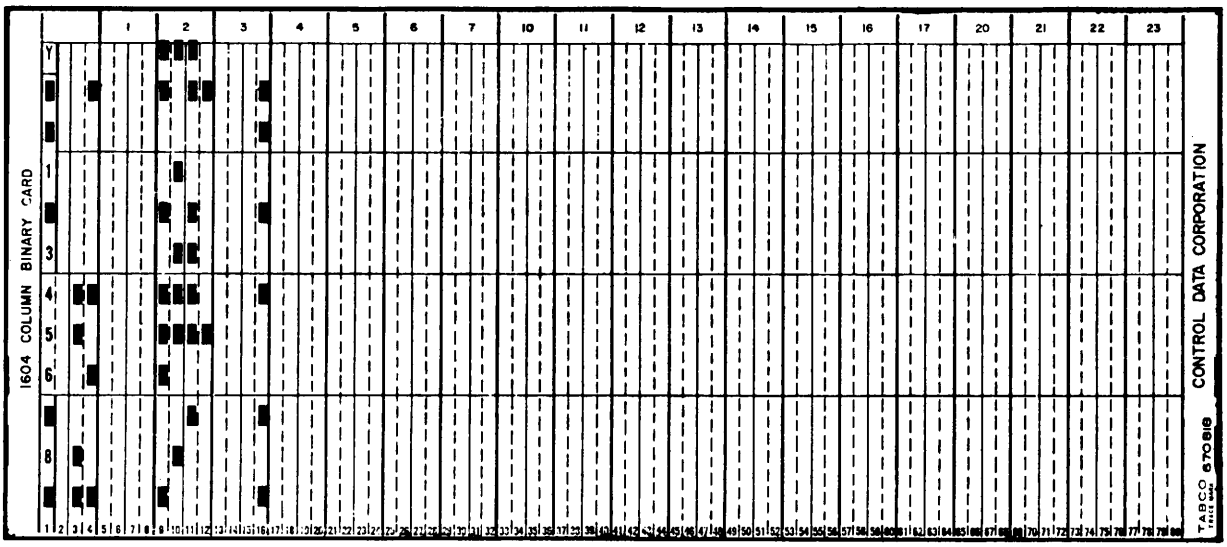
EPT CARD

The EPT (32) cards contain the BCD names of all the declared entry points in the subprogram and their relocatable addresses.

Column	Content
1	11,0,2,79 punches (11,0,2=32_8)
1	rows 4, 5 and 6 contain high order octal digit of EPT card sequence number
2	low order 4 digits of sequence number 00000., 00010., 00020., etc.

3 and 4	checksum of EPT card
5,6,7,8	blank

Each successive group of eight columns, starting with column 9, contains an 8-character BCD entry point and its relocatable address within the subprogram. The BCD name is punched in the first four columns, and the address is punched in columns 7 and 8, and columns 5 and 6 are blank.



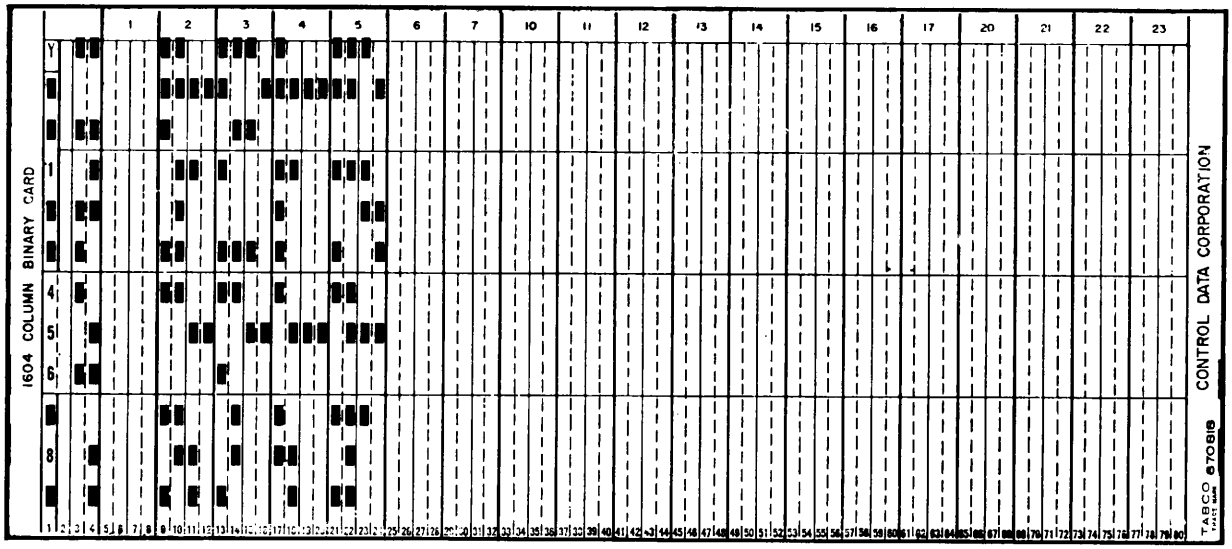
EPT Card

EXT CARD

The EXT (33) cards contain the BCD names of all declared external symbols in the subprogram.

Column	Content
1	11,0,2,3,7,9 punches ($11,0,2,3=33_1$)
1	rows 4, 5 and 6 contain the high-order octal digit of the EXT card sequence number
2	low-order four digits of the EXT card sequence number which is sequenced the same as EPT cards
3 and 4	checksum
5,6,7,8	blank

The remaining columns starting with 9 contain the BCD names of the external symbols, 8 characters for one name, and four columns contain one name.

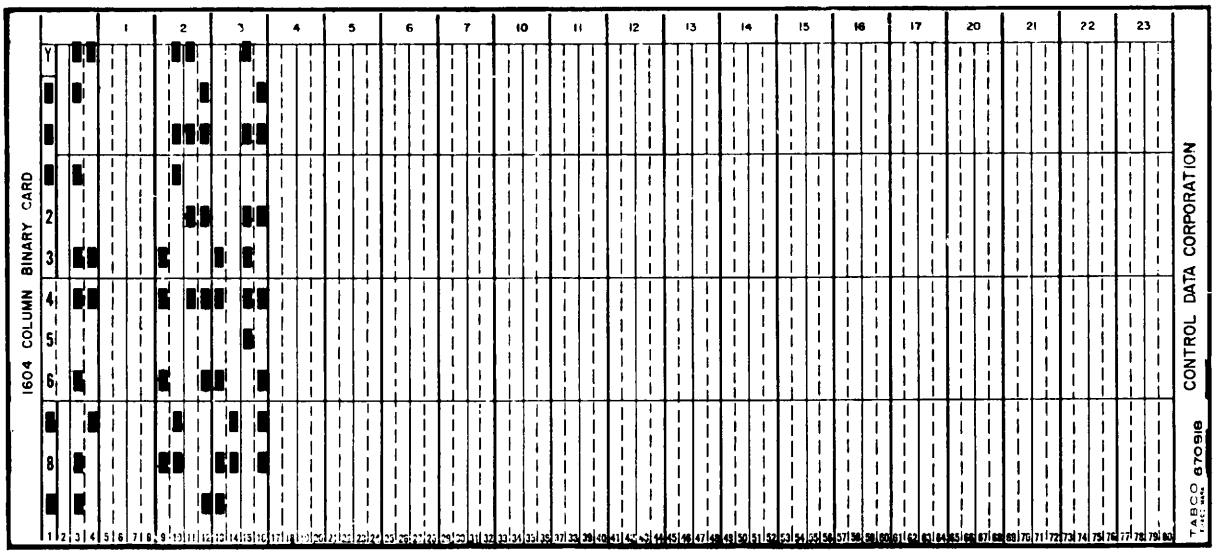


EXT Card

LAT CARD

The LAT (34) card contains the relocatable addresses in the subprogram that contain references to external symbols. Each address is represented as 16 bits, with the high-order bit acting as an upper-lower instruction indicator.

Column	Content
1	11,0,1,7,9 punches ($11,0,1=34_8$)
1	rows 4, 5 and 6 contain the high-order octal digit of the LAT card sequence number
2	the low-order four digits of the sequence number, which is sequenced the same as EPT card
3 and 4	checksum
5,6,7,8	blank
9-80	16-bit relocatable addresses which contain external symbols. Three addresses are contained in every four columns. The high-order bit is the upper-lower indicator (0 for upper, 1 for lower).



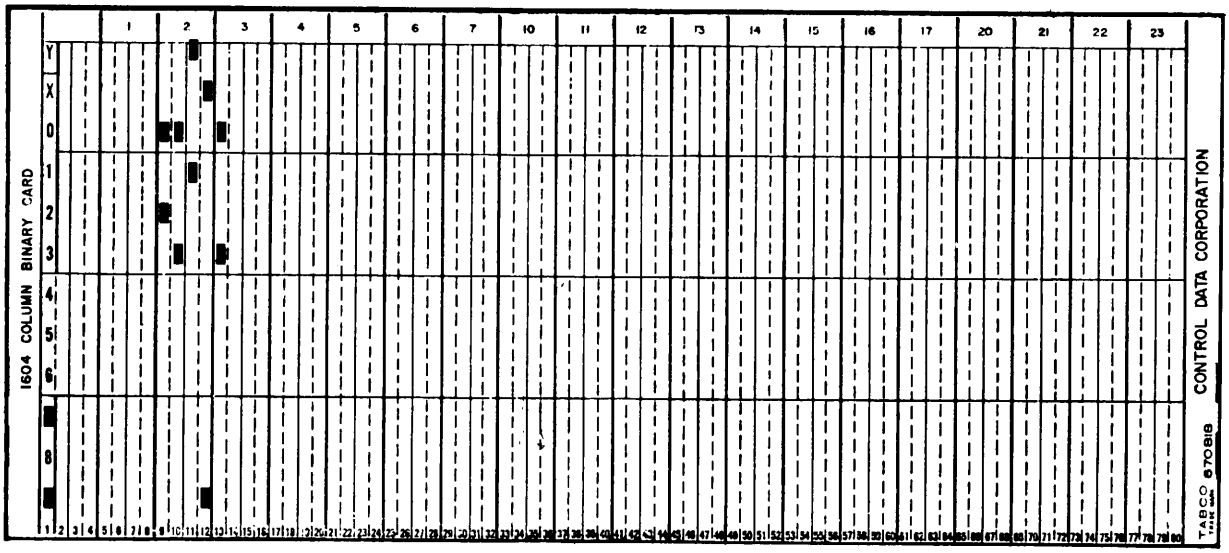
LAT Card

TRA CARD

The TRA card is the last card in a subprogram. Two successive TRA cards indicate the end of an object program. One TRA card in an object deck must contain the name of the transfer address which must be declared as an entry point.

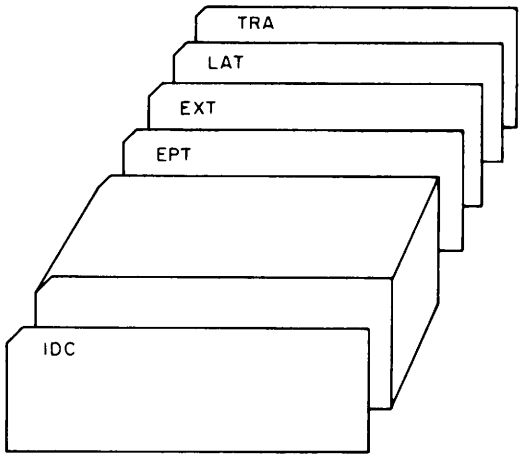
An unnamed TRA card has 7, 9 punches in column 1 and columns 2-80 are blank.

A named TRA card has 7, 9 punches in column 1 and columns 2-8 are blank. The transfer address in Hollerith (maximum of 8 characters) is punched starting in column 9. The remainder of the card is blank.



TRA Card with Transfer Address

SAMPLE BINARY SUBPROGRAM DECK



TRA card (one or two)

LAT card (one or more)

EXT card (one or more)

EPT card (one or more)

RBD cards

IDC card (one or more)

PATCHING OBJECT PROGRAMS D

Relocatable binary program (RBD) decks can be corrected by providing a binary correction deck when the object program is loaded. This correction or patch deck must be the output of a CODAP-1 assembly, and the correction information must be written in CODAP-1 language. The instructions in the correction deck, which are loaded after the object program, replace the original instructions in the locations specified on the PATCH control card. The object correction deck has the standard RBD format.

PATCH CONTROL CARD

The PATCH control card indicates the locations in the preceding object program that are to be corrected by the deck following the PATCH card. It has the following format:

```
11
 0
 7PATCH, p + n
 9
```

Column 1 contains 11,0,7,9 punches, PATCH starts in column 2 and a comma in column 6 is followed by subprogram name, p, plus or minus an octal constant n. The binary object deck following the PATCH card is loaded at the location indicated by n relative to the start of the subprogram p. For example:

```
11
 0
 7PATCH, POPOV+122
 9
```

The relocatable binary subprogram which follows this card will be loaded at the location determined by the location of POPOV plus 122 octal.

The octal integer on the PATCH card is determined from the assembly listing. This integer enables the loader to reference the first word of the area to be patched. The choice of this octal integer is dependent upon the particular patch required as illustrated in the following examples.

ASSEMBLING A PATCH

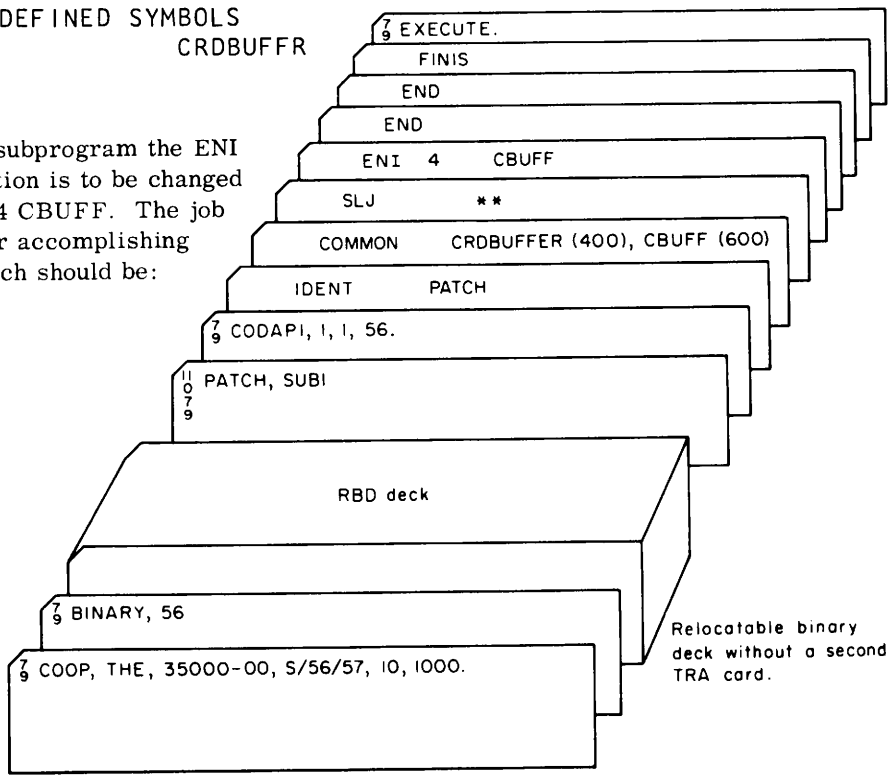
A patch may be assembled from symbolic coding by CODAP-1. The rules governing these patch assemblies are:

1. The patched subprogram must not occupy a greater number of words in memory after the patch is included than before.
2. The instructions to be patched must be contained in an integral number of computer words.
3. ORG and ORGR pseudo instructions may not be used in a symbolic patch.

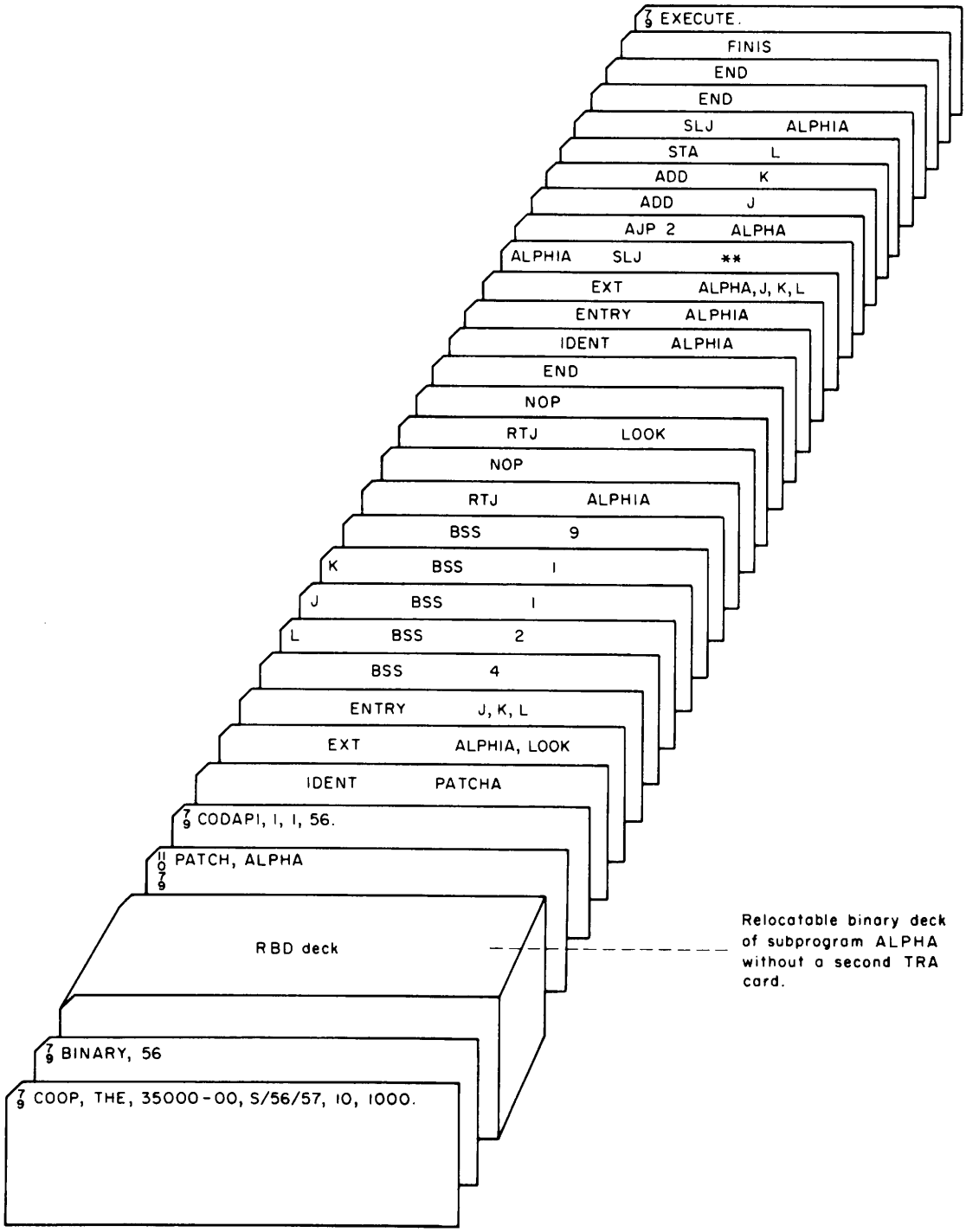
The listing of subprogram SUB1 shown below illustrates the patch capability.

```
                                                    IDENT      SUB1
RANGE                    FWA     -   LWA+1
                         01750        01754
ENTRY POINTS
                         01750        SUB1
EXTERNAL SYMBOLS
                         00001    SCANNER
                         01750+                  ENTRY      SUB1
00000+                                           BLOCK
00000+                                           COMMON     CRDBUFFR(400),
                                                            CBUFF(600)
00620+                                                      CBUFF(600)
00001                                            EXT        SCANNER
01750+   75   0      77777    SUB1               SLJ      0 **
         50   6      00620+                      ENI      6 CBUFF
01751+   75   4     X00001                       RTJ      0 SCANNER
         50   0      00000
01752+   52   5      01750+   +                  LIU      5 SUB1
         51   5      00001                       INI      5 1
01753+   56   5      01750+                      SIU      5 SUB1
         75   0      01750+                      SLJ      0 SUB1
                                                 END

NO DOUBLY DEFINED
NO UNDEFINED SYMBOLS
NULLS                CRDBUFFR
```

In this subprogram the ENI instruction is to be changed to ENI 4 CBUFF. The job deck for accomplishing this patch should be:



Any number of instructions may be changed by using the BSS or BES pseudo instruction to properly position the instructions relative to the original program. Following is a listing of subprogram ALPHA.



If a patched subprogram requires more words than the original, the additional instructions may be included in a new subprogram which is entered by a return jump patched into the original subprogram. In the listing of the sample subprogram ALPHA, the change PATCHA shown below requires more words than the original. The job deck for accomplishing this patch should be as follows:

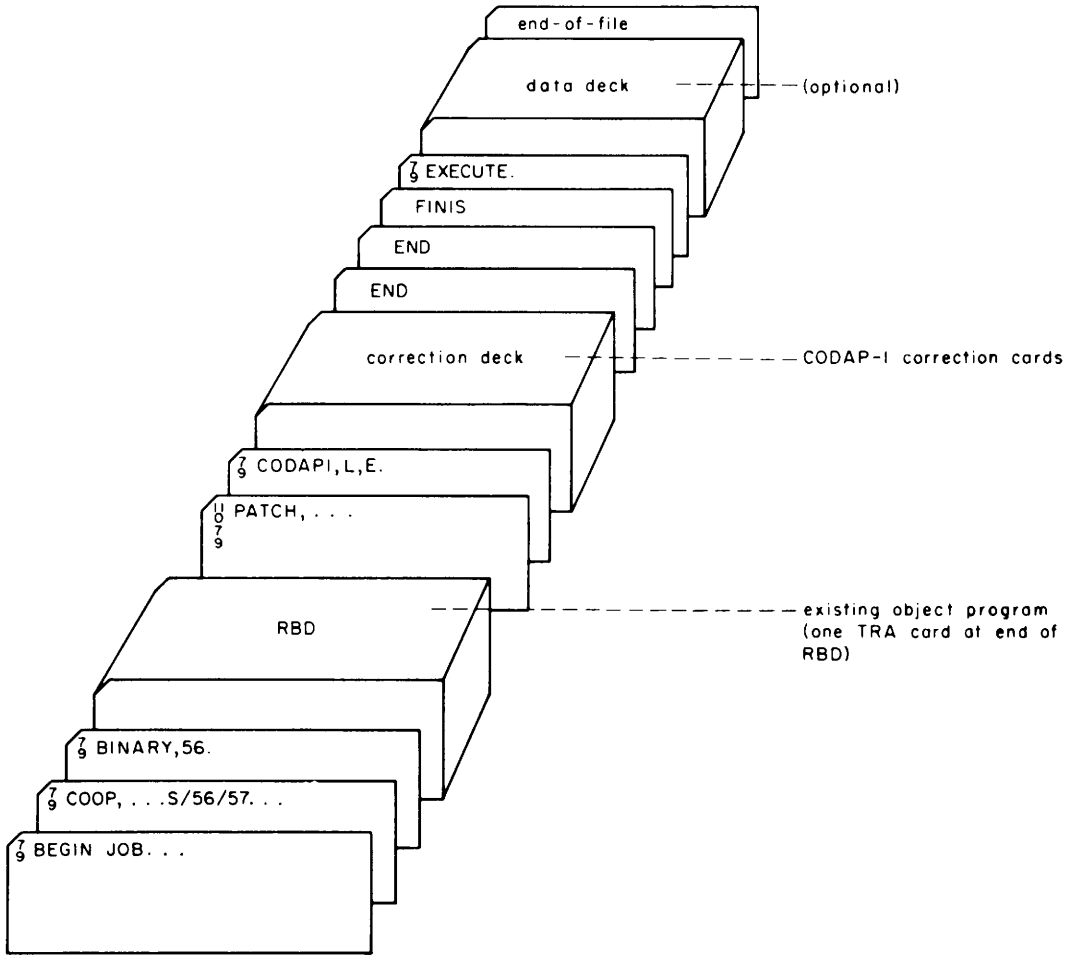
```
                                                    IDENT        ALPHA
RANGE                      FWA   -   LWA+1
                           00003       00026
ENTRY POINTS
                           00013       ALPHA
EXTERNAL SYMBOLS
                           00001       LOOK
                           00002       LOOKUP
00000+                                 BLOCK
00000+                                 COMMON       11
00001+                                 COMMON       JJ
00002+                                 COMMON       KK
                           00013+      ENTRY        ALPHA
00001                                  LIB          LOOK
00002                                  LIB          LOOKUP
00003+                           UPDATE1 BSS          1
00004+                           L       BSS          1
00005+     00  0   00000        I       DEC          50
           00  0   00062
00006+     00  0   00000        J       DEC          75
           00  0   00113
00007+     00  0   00000        K       DEC          100
           00  0   00144
00010+     00  0   00000        TERM1   DEC          3600
           00  0   07020
00011+     40  0   00000        LARGE   OCT          4000000000000000
           00  0   00000
00012+     00  0   00000        UPDATE  OCT          2
           00  0   00002
                   00000        CLK     EQU          0
                   00070        TERM11  EQU          70B
                   00071        TERM    EQU          71B
                   00072        TERM21  EQU          72B
00013+     75  0   77777        ALPHA   SLJ          **
           12  0   00010+              LDA          TERM1
00014+     24  0   00012+              MUI      0   UPDATE
           20  0   00003+              STA      0   UPDATE1
00015+     12  0   00011+              LDA          LARGE
           15  0   00003+              SUB          UPDATE1
00016+     20  0   00000               STA          CLK
           12  0   00003+              LDA          UPDATE1
00017+     20  0   00071               STA          TERM
           20  0   00070               STA          TERM11
00020+     20  0   00072               STA          TERM21
           12  0   00005+              LDA          I
00021+     14  0   00006+              ADD          J
           14  0   00007+              ADD          K
00022+     20  0   00004+              STA          L
           75  4   X00001              RTJ          LOOK
00023+     10  0   00005               ENA          5
           14  0   00005+              ADD          I
00024+     20  0   00004+              STA      0   L
           75  4   X00002              RTJ          LOOKUP
00025+     75  0   00013+              SLJ          ALPHA
           50  0   00000
                   00000               END          ALPHA

NO DOUBLY DEFINED
NO UNDEFINED SYMBOLS
NULLS                    11          JJ                    KK
```

Because J, K, and L are local variables in subprogram ALPHA, they must be declared as entry points so that they may be referenced by ALPH1A.

BSS pseudo instructions in the patch provide definitions for J, K, and L which agree with their definitions in ALPHA.

JOB DECK FOR PATCHING

Program correction cards can be compiled and inserted in binary object decks with PATCH load control cards described in Appendix D. CODAP-1 must be used as the source language of the correction deck. More than one patch may be used.



Compile and Execute with a Patch Deck

USER ENTRY POINTS IN RESIDENT MONITOR

E

Some of the entry points in the resident portion of the CO-OP monitor are available for program use. Most of them are entries to special purpose subroutines in the monitor.

CLNBCD*

This routine converts a 20-word Hollerith card image to a 10-word BCD image. The Hollerith card images must have a 7,9 punch in column 1 and be stored four columns per word in column binary format.

Calling sequence:

```
        ENA     Y_HOL
        ENQ     Y_BCD
L       RTJ     CLNBCD*
L+1     error return
L+2     normal return
```

Y_{HOL} is the location of the first word of the 20-word Hollerith card image. Y_{BCD} is the location of the first word of the area to contain the 10-word BCD image. The 7,9 punch in column 1 is converted to the BCD code 00. If an illegal Hollerith punch is found in colums 2 through 80 of the card image, the illegal character is converted to the BCD code 00, and the error return is taken as soon as column 80 has been converted. If no error is detected, the normal return is taken.

BCDBN*

This routine converts a word of up to 8 BCD numeric characters to its equivalent positive binary integer. The word to be converted is placed in the A register before entering the routine. If less than 8 characters, the word is right justified with leading 00 fill.

Calling sequence:

```
L       RTJ             BCDBN*

L+1     normal return
```

On return, the A-register contains the converted binary integer. No error return is provided.

EXIT*

This entry point can be used for normal termination of a job.

Calling sequence:

```
SLJ   0   EXIT*   or RTJ   EXIT*
```

No recovery dumps occur when EXIT* is used, however, a SNAP dump planted at XITDUMP* will be executed.

ERROR*

This entry point is used by the monitor for abnormal job terminations because of an error condition. A dump will be made according to the recovery key on the MCS card. If a SNAP dump has been planted at ERRDUMP*, it will be executed first. A programmer may use this entry point by using the calling sequence RTJ EXIT*.

ERRDUMP*

This name, used as the P_1 parameter on a SNAP control card, will cause a snapshot dump if the job is terminated through ERROR*. The N_1 parameter on the SNAP card is omitted.

RECRET*

This routine permits the user to regain control for 60 seconds after ERROR* has been entered and any requested SNAP post mortem dumps have been performed. The following calling sequence is a flag to the monitor to indicate that if an ERROR* exit occurs, control is returned to the location indicated for 60 seconds.

```
L       RTJ     RECRET*

        ZRO     (location to which control is to be transferred)
```

RECLIM*

This routine sets limits of program and data regions for extended recovery options.

Calling sequence:

```
        LDA     L(DLIM)
        LDQ     L(PLIM)
L       RTJ     RECLIM*
L+1     Return
```

If DLIM = 0, return is with the currently defined data limits in the A register and program limits in the Q register. If DLIM $\neq$ 0, the data limits will be set according to the contents of the A register and the program limits according to the contents (PLIM) of the Q register. Limits are defined by a word containing the first word address and the last word address of the region in the upper and lower address positions.

MEMREC*

This routine is used to reserve and release internal storage areas and to obtain the limits of memory not yet reserved. It is used primarily by the binary loader, but may be used by the programmer.

Calling sequence:

```
L       RTJ     MEMREC*
L+1     ZRO     A
        ZRO     B
L+2     normal return
```

The parameters A and B are zero when MEMREC* is for determining available memory. The subroutine exits to the next instruction in L+2, with FWA and LWA of unassigned memory in the A register. The upper address portion of A contains the FWA and the lower address contains the LWA. The contents of the Q register will be zero.

If a program needs additional space, a portion of occupied memory no longer needed may be released. To release a specific area, it is necessary to first find the currently reserved portion of memory by using a MEMREC* calling sequence with A and B set to 0. Any amount of storage area may be released by executing another return jump to MEMREC*, with parameters A and B set to the complement of the FWA and LWA of the block to be released. The only restriction is that the FWA of the block to be released must be one

greater than the LWA in unreserved memory. In other words, a block of memory cannot be released unless it is contiguous to the unreserved portion of memory.

To reserve unassigned areas of memory, parameters A and B are set to the FWA and LWA of the area to be reserved. This block must be entirely within unassigned memory.

On return from MEMREC* the A register always contains the FWA and LWA of available memory, and the Q register contains an error code if an error was detected.

Code	Error
Q = 0(or positive number)	No error
Q = -0	In L+1, FWA was greater than LWA
Q = -1	Attempt to reserve memory when either FWA or LWA fell outside available memory
Q = -2	Attempt to release a block of memory which was not adjacent to a previously assigned block

DATE*

This entry point consists of the BCD coded date in columns 17-24 of the BEGIN JOB card. To obtain the date in the A register, use LDA DATE* or the equivalent.

LIBREW*

This subroutine will rewind the currently defined library tape to the local point.

Calling sequence:

```
RTJ    LIBREW*
```

LOADER*

This entry point is to the relocatable binary loader; the A and Q registers contain the required parameters.

Calling sequence:

```
L       RTJ             LOADER*

L+1     Return
```

The parameter in the Q register, if non-zero, must be a library subroutine entry point. The A register consists of subparameter R in the upper address and subparameter N in the lower address, with zero in the remaining bits. If the A register is positive, a memory map of all subroutines and subprograms loaded during the current call of the loader is produced on the standard output unit. If the A register is negative, the memory map is suppressed and the A register parameter is complemented before extracting subparameters R and N.

The subparameter N denotes the logical unit number of the current input unit. If it is a library tape, N must have been assigned to a magnetic tape unit.

The following paragraphs describe the loader execution as a function of the input parameters.

R = 0

Q register = Entry Point Name

Subparameter N must agree with the last defined logical unit number for the library tape. The library subroutine containing the entry point name and all library subroutines referenced by it are loaded from library tape N. Any TRA card containing a symbolic name is ignored during this loading. The A register will contain in the lower address the relocated address of the entry point name; the remainder of the A register will be zero. The original contents of Q are destroyed.

Q register = 0

Subparameter N is the logical unit number of an input tape. The loader will begin reading cards from unit N. Two successive TRA cards will cause a normal return from the loader. An end-of-file on an I/O unit other than the standard input unit will cause loading to continue at that point from the standard input unit. An end-of-file on the standard input unit produces an error. Undefined external symbols are assumed to be entry points of subroutines on the last library tape defined before they are finally classed as undefined symbols. The contents of the A register on return from the loader will be as follows:

a) TRA card entry point from input only

UA = 00000

LA = relocated address of that entry point

b) TRA card entry point from both the input unit and the library tape

UA = relocated address of input TRA

LA = relocated address of library subroutine TRA

c) TRA card entry point from a library subroutine only

UA = 00000

LA = relocated address of that entry point

d) No TRA card entry points

UA = 00000

LA = 00000

A register bits not included in UA or LA are always zero.

R = 1

The Q resister is ignored; subparameter N defines the library tape. Unit N is rewound and the directory loaded from it. All standard I/O drivers defined in RHT are loaded. On return, the contents of the A register are zero.

R = 2

Subparameter N must agree with the last defined library tape number. All non-standard I/O drivers as defined in RHT are loaded.

Q-register = Entry Point name

The library subroutine containing the entry point and all library subroutines referenced by it are loaded from library tape N. The lower address of the A register will contain the relocated address of the entry point name; the remainder of the A register will be zero. The original contents of the Q register are destroyed.

R = 3

The Q register is ignored. Subparameter N defines the library tape. Unit N is rewound and the directory loaded from it. On return, the contents of the A register are zero.

Detected errors are indicated by a line of output on the standard output unit in the following format:

```
LOADER ERROR XX          YYYYYYYY      ZZZZZZZZ
```

XX is a two-letter mnemonic code denoting the type of error, YYYYYYYY is an eight-character quantity containing a symbolic name, and ZZZZZZZZ is an eight-digit octal quantity denoting the upper half of the first word of a binary card or the upper instruction of the loader entry point, LOADER*. See Chapter 6 for loader errors.

RELOAD*

This routine calls the loader if the loader in resident has been destroyed. This would occur when RELOCOM or LIBRARY loader control card is used.

Calling sequence:

```
RTJ    RELOAD*
```

The A register contains two parameters.

$A_U = 3$

A_L = logical unit number of current library tape

All subsequent loader calls are through LOADER*.

CLOCK CONTROL F

A real-time clock is a valuable programming aid. It is convenient for automatic time logging and imposing time limitations on programs executed under a monitor system. The real-time clock on the Control Data® 1604 is automatically incremented every 1/60 second. Since incrementing is performed in the accumulator, interrupt on arithmetic fault is used to indicate the end of a specified time period. The clock may cause an interrupt after a specified time by providing a proper initial value. For example, if an interrupt is required after two minutes the initial setting is $(2^{47} -1)-(2x60x60)$. An arithmetic fault occurs when the time period has elapsed.

Three functions are performed during clock operation in the monitor.

1. Job sequencer timing includes imposing a time limit on the job and maintaining an elapsed time record. The time limit is obtained from the MCS card.
2. CO-OP Control System Timing is the time limit specified on the EXECUTE or EXECUTER card.
3. Programmer timing includes reading and setting the clock at the request of the programmer.

Two clock times are kept in the monitor: the over-all job time and a time interval set by the routine SETCLK*. This routine is used by the programmer to obtain the remaining over-all time and to set time intervals within his program. When a time interval set by the programmer has elapsed, control returns to an interrupt routine provided by the programmer. Only one time interval can be in force at any one time. The elapsed portion of the current time intervals is maintained in a table.

The first entry in this table is the setting from the Job Sequencer. Initially this setting will be equal to the time indicated on the MCS card and may be cleared only by the monitor. If the time is exceeded, the contents of the 1604 real-time clock is saved to indicate the elapsed time for the job.

A time limit on an EXECUTE card is treated as a programmer time interval set by SETCLK*.

SETCLK* CALLING SEQUENCE

Prior to entering the SETCLK* subroutine, the address of an interrupt subroutine is placed in the low 15 bits of the Q register, and the time interval, t, is placed in the A register. The interrupt subroutine will be executed when the time interval, t, has elapsed. The time interval, t, is expressed as a binary integer representing the interval in sixtieths of a second.

The calling sequence to SETCLK* is:

```
L       RTJ             SETCLK*
L+1     Error Return
L+2     Normal Return
```

Regardless of the return or the initial value of t, the remaining over-all job time is set in the A register by SETCLK*.

READING THE MONITOR CLOCK

To read the current value of the monitor clock, the parameter, t, is set to 0, and a return jump is made to SETCLK*. The routine will exit through the normal return L+2, with the remaining over-all job time in the A register.

SETTING TIME LIMITS

Time limits are entered in SETCLK by setting, t, to the time limit desired; t may be either a positive or negative value (bit 47 is 0 or 1). If t is greater than zero, one of three possibilities will occur:

The error return (L+1) is taken if t is greater than the over-all time remaining for the job.

The error return is taken if a previous time interval set by SETCLK* has not elapsed. This interval would have been set by a previous program call to SETCLK* or by the time limit on the EXECUTE card.

If no previous time interval was in effect, the new interval t is set into the clock table by SETCLK*, and a normal exit is made. When the time interval t has elapsed, the interrupt subroutine will be executed.

If t is less than zero, t is first complemented, and one of the following actions will occur:

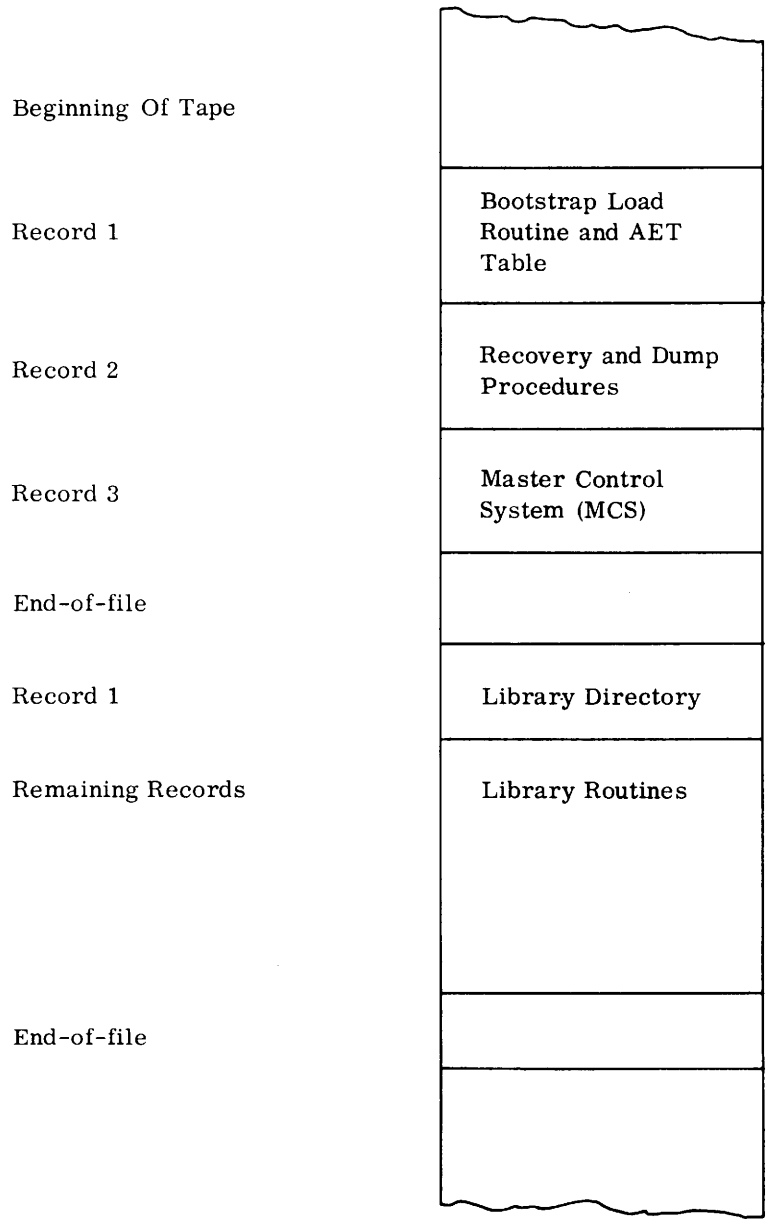
If the complement of t is greater than the remaining over-all time, the error return is taken.

If the complement of t is less than the over-all time remaining, any previous time interval is cleared and replaced by the new (t complement) time interval. The previous interrupt will be replaced by the new interrupt entry point in Q.

CLEARING A PREVIOUS TIME INTERVAL

A previous time interval set by SETCLK* can be cleared by entering SETCLK* with t=-0. No interrupt routine will be executed.

LIBRARY TAPE LAYOUT G



RESERVED ENTRY POINT LIST H

The words in this list are the entry points to the monitor resident and library routines as indicated. These words should not be used as entry points in user programs. If these words are encountered as external symbols by the loader, and have not been previously defined as entry points in a user subprogram or subroutine, the library routine containing that entry point will be loaded.

Not all systems library tapes contain all the systems listed below; conversely, new systems may be added to the list. Each installation should examine their systems tape directory in order to obtain a complete and correct reserved entry point list.

CODAP	COBOL	COBOL	COBOL	COBOL	COBOL	COBOL
CODAP2	CBLDG1	DBINT2A	FILL1	LRS	OEDIT	RP1.1
CODAP=	CBLDG2	DBINT3A	FILL2	MOVE01	OPEN	RP1.2
COSY	CBLERR	DBINT3B	FILLUP	MOVE05	OPER1	SCMUDV1
	CEDIT	DISPLAY	FILP4	MULT10	OPER1SUB	SEG
	CLOSE	DIVD10	FNT	MULT01	OPER2	SEQ
COBOL	COBOL	DIVD01	FNTAVSP	MULT07	OPER2SUB	SEQB
	COBOLMC	DIVD07	GOIF	MULT08	OPER3	SFINC
ACCEPT	COMP01	DIVD08	GRABC	MULT09	OPER3SUB	SHIFT1
ADD10	CONDFLG	DIVD09	IF	NEXTWORD	OSR	SIZDEFX
ADDRESSO	COPY	DMYNTRY	IF01	NILE	PAIL44	SIZETRIG
ADD01	COPYBIT	DNT	IF05	NLE	PAIL44X	SIZFXDE
ADD05	COPYFLAG	DNTAVSP	IMAKER	NOLDFLG	PERFTABL	SSW
ADD07	CRACKED	EPE	JUMPADD	NONZRC	PERMCON	SUB10
ADD08	CRDBR	EPECLNUP	L2R	NUMCONVO	PIAT	SUB01
ADD09	CURNLEVL	ERRTRIG1	LASTWORD	NUMOCT	PM	SUB07
ADVBLNK	CURNLOC	EXAM1	LDANDGO	NUMOCTX	PNCHFGX	SUB08
ALL60	CURNQUAL	EXAM2	LEAP	NUMREC	PRINT	SUB09
ALLIT3	CURNTWD	EXAM3	LEVEL1	NXTWORD	PUNCH	TESTCEL1
ALLIT4	CURNTWDA	EXAM4	LGFLAG	NXTWRD	PUNCHFLG	TESTCELL
AND/OR	D14ZZ1A	EXAM01	LGFLAGX	OADDSUB	QUALBILD	THISWORD
ATTN	DATAMAP	EXE	LIBPREP	OADSB1	QUE	TRACARD
BDITNT1	DATMPFLG	EXECLNUP	LIBSRCH	OADSB2	RDPROC	TRIVFLAG
BITE1	DB1A	FDPM	LISTFLAG	OBJLOC	READ	TSTPRMCN
BITE2	DB1B	FEGEN1	LITDROP	OBJSTR	RECT	UDT
BLKA	DB2A	FEGEN2	LK8	OB/ZR	REFTAB	UDTAVSP
BLKB	DB2B	FEGEN3	LN9COM	OCC	REFTABX	UDTLOOK
BUFAR	DBDEPON	FEGMSKLA	LN9COMX	OCCBEG	RESTRAC	VALPROC
CATBIT	DBINT1	FENCE	LNACOM	OCCFNL	RFINC	VLFILL
CATBITX	DBINT1A	FET	LNACOMX	OCCLIM	RNDFXDE	VLMASK
CATCHEX	DBINT2	FETX	LOOKUP	OCCM	RNT	VLMUDV1

COBOL	COBOL	COBOL	COBOL	COBOL
VLMUDV2	WRITE	XFER2	XFER3S	XPRINTX
VLMUDV3	WSBIT	XFER2A	XFER4	XPUNCHX
VLXFER	WSBIT1	XFER2R	XFER4A	XQUEX
VS	XCPC	XFER2S	XJMPX	XRECTX
WORDCODE	XCURWDX	XFER3	XLDGOX	XTHSWDX
WRDCRK	XDNTX	XFER3A	XLOOKUPX	XWRDCRKX
WRDPRNT	XFER	XFER3R	XNUMCON	ZSCP

FORTRAN-63	FORTRAN-63	FORTRAN-63	FORTRAN-63	FORTRAN-63
FERROR=	Q1Q03300	Q1Q10310	Q3Q10040	Q8QEXITS
FTN	Q1Q03310	Q1Q10320	Q3Q10140	Q8QFILES
LENGTHF	Q1Q03320	Q1Q10330	Q3Q10240	Q8QFORMS
Q0Q06200	Q1Q03330	Q1Q10400	Q3Q10340	Q8QFTSEN
Q0Q06300	Q1Q04100	Q1Q10410	Q3Q10440	Q8QFTSET
Q1Q00100	Q1Q04200	Q1Q10420	Q7QFLOAT	Q8QGINTY
Q1Q00200	Q1Q04210	Q1Q10430	Q7QLDCC3	Q8QGLIST
Q1Q00210	Q1Q04220	Q2QLOADA	Q7QLDDC3	Q8QGNSEN
Q1Q00220	Q1Q04300	Q3Q00040	Q7QLDIC2	Q8QGNSET
Q1Q00300	Q1Q04310	Q3Q00140	Q7QLDID2	Q8QGNTAB
Q1Q00310	Q1Q04320	Q3Q00240	Q7QLDLC4	Q8QGOTTY
Q1Q00320	Q1Q04330	Q3Q00340	Q7QLDRC2	Q8QGTTAB
Q1Q00330	Q1Q05100	Q3Q01040	Q7QLDRD2	Q8QHUNCH
Q1Q01100	Q1Q05200	Q3Q01140	Q7QLODCC	Q8QIBJOB
Q1Q01200	Q1Q05210	Q3Q01240	Q7QLODDC	Q8QIFORM
Q1Q01210	Q1Q05220	Q3Q01340	Q7QLODIC	Q8QINBFI
Q1Q01220	Q1Q05300	Q3Q02040	Q7QLODID	Q8QINBFO
Q1Q01300	Q1Q05310	Q3Q02140	Q7QLODLC	Q8QINBIN
Q1Q01310	Q1Q05320	Q3Q02240	Q7QLODRC	Q8QINBOT
Q1Q01320	Q1Q05330	Q3Q02340	Q7QLODRD	Q8QINGIN
Q1Q01330	Q1Q10000	Q3Q03040	Q8QBACKS	Q8QINGOT
Q1Q02100	Q1Q10010	Q3Q03140	Q8QBCDDI	Q8QIOSEN
Q1Q02200	Q1Q10020	Q3Q03240	Q8QBINTY	Q8QIOSET
Q1Q02210	Q1Q10030	Q3Q03340	Q8QBOTTY	Q8QLOADA
Q1Q02220	Q1Q10100	Q3Q04040	Q8QBUTAB	Q8QLPSEN
Q1Q02300	Q1Q10110	Q3Q04140	Q8QCKSLT	Q8QLPSET
Q1Q02310	Q1Q10120	Q3Q04240	Q8QDPI/O	Q8QLUNCH
Q1Q02320	Q1Q10130	Q3Q04340	Q8QENBIN	Q8QLURCH
Q1Q02330	Q1Q10200	Q3Q05040	Q8QENBOT	Q8QREWND
Q1Q03100	Q1Q10210	Q3Q05140	Q8QENGIN	Q8QSENLT
Q1Q03200	Q1Q10220	Q3Q05240	Q8QENGOT	Q9QEVALL
Q1Q03210	Q1Q10230	Q3Q05340	Q8QENTRY	Q9QEVALB
Q1Q03220	Q1Q10300			

FUNCTION ROUTINES (FORTRAN-63)

ABSF	EXPF	MAX0F	Q8QSIGNF	XINTF
Q8QABSF	Q8QEXPF	MAX1F	SINF	Q8QXINTF
ACOSF	FLOATF	MIN0F	Q8QSINF	XMAX1F
Q8QACOSF	Q8QFLOAT	MIN1F	SQRTF	XMAX0F
ASINF	INTF	MODF	Q8QSQRTF	XMIN0F
Q8QASINF	Q8QINTF	Q8QMODF	TANHF	XMIN1F
ATANF	ITOJ	POWRF	Q8QTANHF	XMODF
Q8QATANF	Q8QITOJ	Q8QPOWRF	XABSF	Q8QXMODF
COSF	Q2Q07000	Q2Q07111	Q8QXABSF	XSIGNF
Q8QCOSF	ITOX	RANF	XDIMF	Q8QXSIGN
CUBERTF	Q2Q07101	RANFGET	Q8QXDIMF	XTOI
Q8QCUBER	Q8QITOX	RANFSET	XFIXF	Q2Q07110
DIMF	LOGF	Q8QRANF	Q8QXFIXF	Q8QXTOI
Q8QDIMF	Q8QLOGF	SIGNF		

I/O DRIVERS

BFT
CARDPNCH
CARDREAD
PRINTD
PRINTS
PTI
PTLOG
PUPT
RW1615
R1617
TYPE I/O

RESIDENT

BCDBN*	LIBREW*	
CBF*	LMSRCH*	
CBF1*	LOADER*	
CBF2*	LOK1*	
CBF7*	MEMREC*	
CHKSTD*	MODIRET*	
CLBNBCD*	MRW600*	
DATE*	OPCOM*	
DETECT*	READ*	
ERROR*	RECLIM*	
EXIT*	RECRET*	
EXSEN*	RELOAD*	
FCH*	RELOCOM*	
FLAGTST*	REMOVE*	
GETCH*	SELECT*	
ILMF*	SETCLK*	
IOSLECT*	SIEOF*	
LIBPSIT	WRITE*	

MISCELLANEOUS SYSTEMS and PROGRAMS

COOP	MAP
COPY*	OVERLAY
DUMP	PLT
ELT	SEGMENT
ERROR	SETCLK
ERROR2	SNAP
IO	SNAPTABL
LOADMAIN	VERIFY*

SORT

MERGE
S/MERGE
SORT
SORTN
SORTP
VGEN

BCD CODES

I

Character	Code (Octal)	Character	Code (Octal)
A	61	2	02
B	62	3	03
C	63	4	04
D	64	5	05
E	65	6	06
F	66	7	07
G	67	8	10
H	70	9	11
I	71	&	60
J	41	-	40
K	42	(blank)	20
L	43	/	21
M	44	. (period)	73
N	45	$	53
O	46	*	54
P	47	, (comma)	33
Q	50	%	34
R	51	#	13
S	22	@	14
T	23	¤	74
U	24	0 (numerical zero)	12
V	25	record mark	32
W	26	0 (minus zero)	52
X	27	0 (plus zero)	72
Y	30	group mark	77
Z	31	tape mark	17
0	12		
1	01		

TYPEWRITER CODES

J

CHARACTERS UC	CHARACTERS LC	CODE	CHARACTERS UC	CHARACTERS LC	CODE
A	a	30	X	x	27
B	b	23	Y	y	25
C	c	16	Z	z	21
D	d	22	)	0	56
E	e	20	*	1	74
F	f	26	@	2	70
G	g	13	#	3	64
H	h	05	$	4	62
I	i	14	%	5	66
J	j	32	¢	6	72
K	k	36	&	7	60
L	l	11	1/2	8	33
M	m	07	(	9	37
N	n	06	_	-	52
O	o	03	?	/	44
P	p	15	"	'	54
Q	q	35	°	+	46
R	r	12	.	.	42
S	s	24	:	;	50
T	t	01	,	,	40
U	u	34	÷	=	02
V	v	17	tab	tab	51
W	w	31	space		04
Backspace		61	Carriage Return		45
Lower Case		57	Upper case		47

INDEX