3200

COMPUTER SYSTEM
SCOPE/COMPASS

REFERENCE MANUAL

CONTROL DATA
CORPORATION

Any comments concerning this manual should be addressed to:

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

September 1964
Pub. No. 60057700

CONTENTS

INTRODUCTION 1

This manual deals with machine language programming of the Control Data® 3200, 3100, and 3300 computers and the use of the monitor system, SCOPE, which controls job processing. SCOPE supervises the operation of the comprehensive library of utility programs provided by Control Data, including FORTRAN, COBOL, ALGOL, PERT, SORT and COMPASS, the flexible machine language assembler. The user may add programs to the SCOPE library through PRELIB, a SCOPE library program.

SCOPE loads and supervises the execution of user programs and provides mapping and debugging facilities, which include a selective dump routine called at the source or object language levels. Recovery from program failure is also provided by SCOPE.

This manual does not repeat the detailed information in the programming reference manual; when feasible, however, it summarizes pertinent information concerning hardware.

Description of problem oriented languages such as FORTRAN and COBOL will be found in separate manuals. The hardware assembler language, COMPASS, is described here.

SCOPE

SCOPE provides supervisory control of program execution. SCOPE includes processors for the following:

- Stacked or single job processing
- Loading relocatable programs
- Input/output control
- Interrupt control
- Debugging aids
- Library preparation and editing
- Forming and executing overlay programs
- Calling and executing utility routines

SCOPE OPERATION

The operator loads SCOPE from the library and provides information about date, time and the input/output configuration for the immediate system. SCOPE will process signal decks of proper structure or several decks from different programmers.

The programmer may select a library program for operation or load and execute his own subprograms. The operator may call library programs for execution.

JOBS

The sets of tasks assigned to SCOPE by the programmer are called jobs. A job may be stacked or non-stacked as indicated by a parameter in the JOB statement.

In a non-stacked job, SCOPE releases the system input/output, and when the job is complete, halts the computer. To process another job after a non-stacked job, SCOPE must be reloaded by the operator.

Stacked jobs afford greater convenience to the programmer and the operator, but they impose limits on the input/output requirements, which non-stacked jobs do not. Stacked jobs are processed in the order they appear on the standard input unit, except as otherwise directed by the operator. Regardless of the successful completion of the job, SCOPE will normally retain control of the system, log off the completed job and process another job in the stack.

Non-stacked and stacked jobs may be mixed on the standard input unit. More efficient use of the system results if non-stacked jobs are segregated, however.

A job is preceded by a JOB statement which follows a SEQUENCE statement. SCOPE control statements, object subprograms, input to library programs and input to object programs may appear after the job statement. The job terminates with the occurence of another SEQUENCE statement.

RUNS

A non-stacked job consists of only one run. A stacked job may consist of one or more runs. A run consists of the execution of a complete program under SCOPE control; there are two types of runs.

A library run consists of the operation of a library program under SCOPE control. The library run is chosen by a library name statement such as COMPASS, COBOL or FORTRAN.

The requested library program is loaded and operated. If errors are encountered during loading, the entire job is terminated. If errors occur during operation of the library program, subsequent programmer runs in the same job may be inhibited. A programmer run occurs when relocatable binary subprograms followed by a RUN statement are loaded during a job. A programmer run may be prevented by errors in loading subprograms or errors during previous library runs in the job. A programmer run may be successful or not. If a programmer run is inhibited or unsuccessful, the remainder of the job is not processed.

ASSEMBLER

The COMPASS assembly program converts programs written in COMPASS source language into a form suitable for execution under the monitor system. Source programs may consist of punched cards or BCD card images. The output from the assembler includes an assembly listing and a relocatable binary object program for immediate execution or punching.

PROGRAM STRUCTURE

Source programs may be divided into subprograms which are assembled independently. All symbols are local to the subprogram in which they appear, unless they are declared as external symbols. Locations which will be referenced by other subprograms are declared as entry points. When an entry point is found which matches an external symbol, linkage is established. The library tape is searched for routines with the names of the unmatched symbols. If unmatched symbols remain, the job is terminated and an error message produced.

SUBPROGRAM

Throughout the discussion of SCOPE and COMPASS, the term "subprogram" will be encountered; it refers to the smallest unit which can be handled by COMPASS or the SCOPE loader.

COMPASS

COMPASS is the comprehensive assembly system for CONTROL DATA computers. Operating under the SCOPE monitor system, COMPASS facilitates the writing of machine language programs through mnemonic instructions and symbolic addresses.

The COMPASS language includes the following features:

Address arithmetic	Constants, symbolic addresses, and arithmetic expressions may be used for addresses.
Character addressing	Symbols may be assigned to character locations and character designators may be suffixed to word addresses.
Preloaded data	Data areas may be specified and loaded with data in the source program.
Common assignments	Common areas may be designated to simplify communication between subprograms.

Data definitions	Integer, floating point, and BCD constants may be designated in familiar notation.
Library routines	Routines may be summoned from LIB during loading using source language statements.
Listing control	The format of the assembly listing may be controlled.
Diagnostics	Diagnostics for source program errors are included with the output listing.
Macro instructions	Sequences of instructions defined in a program will be inserted by the assembler whenever the macro name appears in an operation field.

COMPASS PROGRAMMING 2

In COMPASS source language, the programmer writes machine language instructions in mnemonic/symbolic form, specifies constants, exercises control over subprogram communication and controls the assembly process with a powerful set of pseudo instructions.

The programmer is assumed to be coding subprograms for assembly which are to be linked at load time and executed as a single unit. Thus, problems are expected to be solved by several subprograms or a program. This distinction is made for convenient discussion. The size of subprograms and the magnitude of the problems solved by a subprogram is at the discretion of the programmer.

SUBPROGRAM

A subprogram consists of an IDENT pseudo instruction and subsequent lines of code until and including and END pseudo instruction. The internal logical organization is at the discretion of the programmer so long as a few simple rules are followed. These rules are developed in the discussion of pseudo instructions.

Programmers using SCOPE and preparing subprogram in COMPASS or other source language must consider what happens when a subprogram is assembled or compiled, loaded and run.

When writing machine language programs, a programmer must not exceed the capabilities of the hardware. Similarly, restrictions are imposed by the manner in which SCOPE and COMPASS or a compiler process subprogram information. Subprograms compiled or assembled independently may be linked by the loader. Thus, it is impossible for COMPASS or a compiler to anticipate all of the requirements for loading and execution. The responsibility for the structure of the program when loaded and the responsibility for correct execution must remain with the programmer.

ASSEMBLY FOR STORAGE

Subprograms are assembled for an object machine with a storage element divided by convention into three areas:

Data area
Subprogram area
Common area

A fourth area is encountered only if octal corrections are entered for the object program. This area is called the program extension area.

The first three areas within a single source subprogram are defined at assembly time. Three pseudo instructions, DATA, COMMON and PRG, allow the programmer to use these areas freely. When more than one subprogram is loaded for execution at run time, the areas for storage of all subprograms must be considered. The data area is the same for all subprograms at run time and must be of sufficient length to contain all of the information assigned to it by all subprograms. The programmer must also organize the information to be stored in the data area by the loader so that it will not conflict with information destined for the shared data area from other subprograms.

COMPASS object code contains relocatable addresses, which are modified by a relocation factor during loading, to obtain the actual address in the computer memory.

When assembling subprograms, COMPASS assumes that the initial location in each of the three areas, data, common and subprogram has a relocatable address of zero.

Locations are assigned sequentially from zero unless the pseudo instruction ORGR is encountered. ORGR instructs COMPASS to assign the value in the address field of ORGR as the relocatable address of the following instruction and assign storage sequentially from that relocatable address.

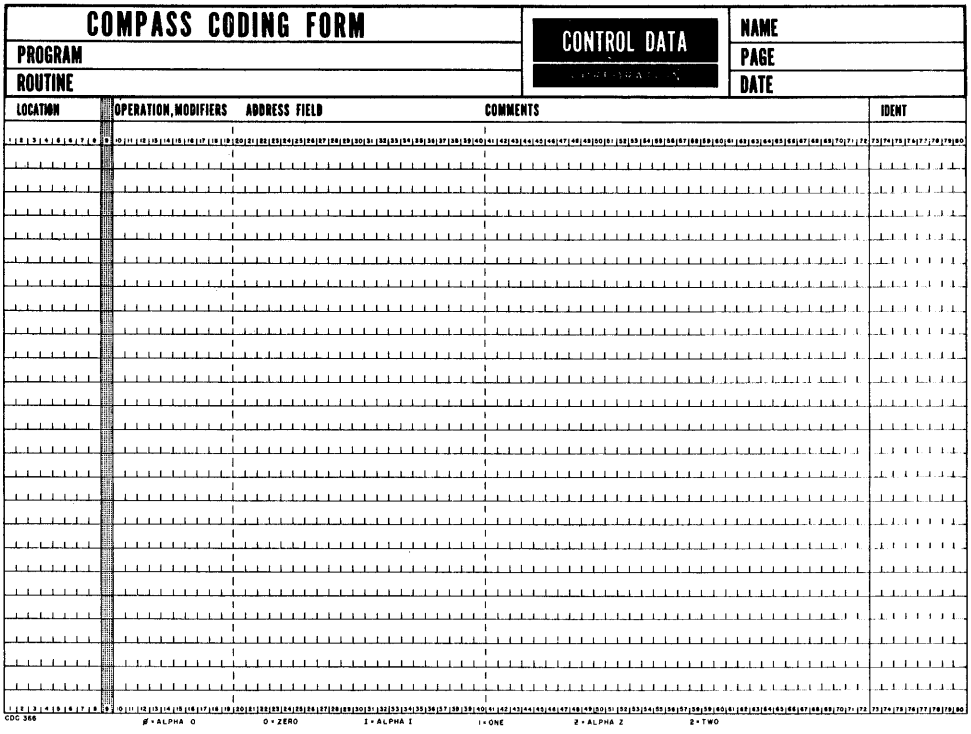
Each area recognized by COMPASS has its own address counter and the counter affected by ORGR depends on which counter COMPASS is using at the moment. The descriptions of the pseudo instruction DATA, COMMON, PRG and ORGR elaborate on this point.

The address counter used by COMPASS for a given area is the same throughout the subprogram. If set by ORGR, the particular counter remains set until a subsequent ORGR. All counters are initialized before assembling a new subprogram.

COMPASS INSTRUCTION FORMAT

Instructions in subprograms to be assembled by COMPASS are written on coding forms. The information on the coding form will be punched into cards or prepared on other suitable media for input to COMPASS. There is a one-to-one correspondence between columns on the coding sheet and card columns.

Each line on the coding sheet is normally punched into a single card. Each line of code is divided into four fields and all instructions are discussed in terms of what may be placed in these fields.



Field	Columns
location	1-8 inclusive, 9 always blank.
operation	begins in column 10 and continues until the first blank column.
address	may begin after the column terminating the operation field; it must begin before column 41. The address field terminates with the first blank, or column 73.
comments or remarks	are written between the end of the address field and column 73.
identification or sequence number	73-80 treated as a comment by COMPASS.

LOCATION FIELD

The type of instruction determines the legal content of the location field. For most instructions, the location field may contain a symbol or be blank. However, the use of symbols in the location fields of pseudo instructions is restricted; see the section on pseudo instructions for specific rules.

A location field symbol is composed of one to eight characters placed anywhere in the field. The first character must be alphabetic; subsequent characters may be alphabetic, numeric or a period. Imbedded blanks are illegal. An illegal symbol is flagged as an L error on the assembly listing.

The location field symbol may represent a 15-bit relocatable address, a 17-bit address, or a 15- or 17-bit nonrelocatable value. If the symbol represents an address, it is defined under control of one of the three address counters except for the special case of equating. The symbols defined under control of an address counter references the first word or character position occupied by the particular instruction.

When an asterisk appears in column 1, columns 2 through 72 are treated as a comment.

OPERATION FIELD

The operation field may contain mnemonic machine instruction codes or pseudo instruction mnemonics, with specific, related modifiers, macro instruction names, or the octal values, 00-77.

The field begins in column 10 and is terminated by the first blank. If column 10 is blank, an operation code of 00 is assembled. An illegal operation field is flagged as an o error on the assembly listing. Modifiers are separated from operation codes by commas.

The following are examples of acceptable operation fields:

```
BSS, C
BSS
LDA
INPC, INT, B, H
MACRO
74
```

ADDRESS FIELD

The address field of machine or pseudo instructions may contain one or more subfields separated by commas. The address field begins with the first non-blank character following the operation field and is terminated by a blank column or before column 73. It must begin before column 41. Address fields or subfields may contain symbols, constants or expressions.

The address field of a machine instruction has one or more subfields separated by commas. Machine instructions have implied subfields. A subfield may be assigned the value zero by giving only its trailing comma or, if it is the last subfield in the address field, by omitting both its content and the preceding comma.

SYMBOLS

A symbol in the address field may occupy the entire field or subfield or may be only one element in the field or subfield.

Any symbol used in an address field must be defined by appearance in the location field of an instruction in the subprogram, or be declared as external. A symbol in the address field is formed and expressed exactly like a location symbol. A symbol in the address field may be relocatable or non-relocatable.

A non-relocatable symbol is defined or equated to a value which may be a number of either 15 or 17 bits. The value assigned to the non-relocatable symbol will not be modified during loading.

A relocatable symbol represents either a 15- or 17-bit address. Relocatable addresses are values related to a memory area which will be incremented or decremented by the loader prior to storage of the instruction in which the address occurs.

Relocatable symbols are local or external to a subprogram and are equated to a 15-bit word address or a 17-bit character address. Relocatable symbols may be:

subprogram relocatable
external symbols
data relocatable
common relocatable

CONSTANT

The address field may contain signed or unsigned decimal or octal integers. If the sign is not present, the integer is understood positive. Octal integers are suffixed by the character, B.

SPECIAL CHARACTERS

Two special entries, a single or a double asterisk, may be made in address field.

* The special character, *, may be placed in the address field and used as any symbol. The * is interpreted as the momentary value of the COMPASS address counter in effect when the * is encountered. The * may result in either a 15-bit or 17-bit address. If the machine instruction consumes two words, * is the address of the first word.

** The special character, **, may be used as the only entry in a field or subfield. The ** yields a subfield containing a one in each bit position. Normally, the field represented by the double ** will be modified during execution of the program and the double asterisk provides a convenient way to ascertain the modification transpired.

LITERALS

If the address field or subfield of an instruction refers to an operand which may be a single or double precision value, the entry may be a literal expressed as: = mv.

The equal sign denotes that the field contains a literal; m denotes the mode of the literal; v is the literal. Single precision literals are expressed as above, double precision literals are expressed = 2mv; the digit, 2, preceding mode denotes a double precision field (48 bit) is to be established.

The mode of a literal may be decimal, octal or Hollerith.

Decimal Literals: = Dv
The value, v, of the literal is expressed in decimal. The decimal value is expressed in the same manner as described for address subfields in DEC and DECD pseudo instructions.

Octal Literals: = Ov
The value of the octal literal is written in the same manner as a subfield for an OCT pseudo instruction except that 16 octal digits may be stated to obtain a double precision constant.

Hollerith Literals: = Hv
The Hollerith literal is expressed as a string of either 4 or 8 characters. The column following a Hollerith literal must be blank or a comma.

During assembly, a literal is converted to binary and assigned a relocatable address which is substituted for the literal in the object code. Literals are assigned to contiguous storage locations at the end of the subprogram. Literals of the same value and size are not duplicated in the object subprogram. Each time COMPASS encounters a literal, the value is compared against the value of all previously assembled literals and if an identical value exists, the address of the previously assigned literal is substituted in the object code.

ADDRESS EXPRESSION

In an address field or subfield, symbols, the special character, *, and numeric constants may be combined with the operators, plus or minus, to form an address expression. The value of the expression is calculated by substituting the numeric value of the symbol and performing 15- or 17-bit arithmetic using the operators. External symbols, the double asterisk, and literals may not appear in an address expression.

If relocatable symbols are part of an address expression, the result of the evaluated expression must be relocatable within a single area. Subprogram, data or common relocatable symbols may be mixed:

$D_1-P_1+P_2-D_2+C_1-C_2$	non-relocatable value
D-C+C	positive data relocatable value
C-P-C	negative subprogram relocatable value

D, P and C are data, subprogram and common relocatable addresses, respectively.

In an expression containing relocatable symbols, the algebraic sum of the relocation indicators must be either an area relocation increment, an area relocation decrement, or no relocation designator and, therefore, a non-relocatable value.

The result of an address arithmetic expression depends on the number of bits assigned to the subfield in the object code.

ADDRESS OF ASSEMBLED INFORMATION

The hardware storage element consists of 24-bit words. Each word is divided into four 6-bit character positions. The entire word or any character in the word is addressable. Machine language instructions and pseudo instructions may require either a word or character address. The character address consists of 17 bits in positions 16-0 of the word in storage. The word address consists of 15 bits in positions 14-0. Word or character addresses may be expressed in the same manner using COMPASS symbolic addressing techniques.

Only the most significant 15 bits of a character address are relocated during loading.

FIELD DEFINITIONS FOR COMPASS INSTRUCTIONS

Symbol	Meaning	Literal	Expression	Absolute	Relocatable	External	Number of Bits	Bit Positions in Computer Word
b	Index designator 1 to 3		yes	yes			2	16-15 or 17
i	interval, 0 to 7		yes	yes			3	17-15
c	character count		yes	yes			7	17-23
c	or character		yes	yes			6	18-23
ch	channel designator		yes				2	
m	first operand word address	yes	yes	yes	yes	yes	15	14-0
n	second operand address	yes	yes	yes	yes	yes	15	14-0
r	first operand character address	yes	yes	yes	yes	yes	17	16-0
s	second operand character address	yes	yes	yes	yes	yes	17	16-0
v	register file address, 0 to 77_8		yes	yes			6	5-0
x	connect code or interrupt mask		yes	yes			12	11-0
y	15-bit operand or shift count	yes	yes	yes	yes	yes	15	14-0
z	17-bit operand	yes	yes	yes	yes	yes	17	16-0

ADDRESS SUBFIELDS

m and n

The address field entries m and n for machine instructions may be represented by a symbol, external symbol, literal, constant, expression or the special characters, * and **. The m and n subfields are generally operand addresses and always occupy bit positions 14-0 in the assembled instruction.

y

The y subfield may be represented by a symbol, external symbol, literal, constant, expression or the special characters, * and **. The y subfield represents an operand which occupies bit positions 14-0 in the assembled instruction.

r and s

Machine language instructions using a 17-bit character address contain r or s subfields which may be represented as a symbol, literal, constant, external symbol, expression, or the special characters, * and **. These subfields occupy bit positions 16-0 of the assembled instruction.

z

A 17-bit operand, z, may be represented by a symbol, constant, literal, expression or special characters, * and **. The z field occupies bits 16-0 of the assembled instructions.

b

The b subfield may be represented by a digit 1, 2, 3, a symbol equated to 1, 2, or 3, an expression whose value is 1, 2, 3, or **. The b subfield designates an index register. A b subfield is interpreted as follows:

If it is used with m and n subfields; mnemonic operation codes may be 1, 2, or 3, and occupies bit position 16-15 in the assembled instruction. If an octal operation code is used, b may be 0-7 occupying bits 17-15 in the assembled instruction.

If it is used with r and s subfields, b depends on the particular instruction; it is restricted by the instruction to only one digit and represents bit 17 of the assembled word.

i

The i subfield occurs in the MEQ and MTH instructions; it may be a symbol, constant or expression which results in a value from 1 to 8, or **. The i subfield occupies bit positions 17-15 in the assembled instruction.

In the following example ABLE = 100_8, INTERVAL = 1.

Coding:	Results (in octal)		
MEQ ABLE, INTERVAL	06	1	00100
MEQ ABLE, INTERVAL+1	06	2	00100
MEQ ABLE, 2	06	2	00100
MEQ ABLE, 8	06	0	00100
MEQ ABLE,**	06	7	00100

v

The v subfield in a machine language instruction denotes a location in the register file. It may be any sumbol, constant or expression which results in a value 0 to 77_8, or **. The v subfield occupies bit position 5-0 in assembled instructions.

x

The connect code for input/output units or the comparison mask for interrupt instructions is represented by x. May contain a symbol, constant, or expression which results in a value $0 \leq x \leq 2^{12} - 1$, or **.

ch

This subfield contains the channel designator for input/output instructions. May contain a symbol, constant or expression which results in a value $0 \leq ch \leq 3$, or **.

In the following examples, ABLE is equated to the value 0011_8 elsewhere in the program.

Coding:	Results (in octal)				
TMA ABLE	53	0	2	. . .	11
TMA 77B	53	0	2	. . .	77
TMA **	53	0	2	. . .	77
TMA ABLE+22B	53	0	2	. . .	33

c

The c subfield specifies the length of a character field or represents a search character.

MOVE: The c subfield may be a symbol or an expression which results in an absolute value from 1 to 128, or **.

SRCE or SRCN: The c subfield may be any symbol, constant, or ** which represents the 6-bit character code of the character for which the search is made.

In the following examples, ABLE is equated to the value 100_8 elsewhere in the program; BAKER to 00200_8.

Coding:		Results (in octal)
MOVE ABLE,BAKER,BAKER+100B	word 1	72000300
	2	40000200
	3	
MOVE 128,BAKER,BAKER+128	word 1	72000400
	2	00000200
	3	
MOVE 27B,BAKER,BAKER+27B	word 1	72000227
	2	13400200
	3	
MOVE **,BAKER,BAKER+100B	word 1	72000300
	2	77400200
	3	

In the following examples, A is defined elsewhere in the program as the octal value 21; ABLE and BAKER are defined as 00200_8 and 00100_8.

Coding:		Result (in octal)	
SRCE A,ABLE,BAKER	word 1	71	00100
	2	21	00200
	3		
SRCE 21B,ABLE,BAKER	word 1	71	00100
	2	21	00200
	3		
SRCE A+21B,ABLE,BAKER	word 1	71	00100
	2	42	00200
	3		

EVALUATION OF ADDRESS EXPRESSIONS

Address expressions are evaluated as a word address (15 bits) or a character address (17 bits). All address expressions are converted to binary numbers of modulus $2^{15}-1$ or $2^{17}-1$, and stored in the proper subfield. No size check is made for 15 or 17-bit subfields by COMPASS.

The location terms of all instructions except BCD, C, BSS, C and EQU, C are evaluated as word addresses.

	word address, 15 bits	character address 17 bits
subfield type	m, n, y,	r, s, z,
	i, x, v, ch, b	c

NON-RELOCATABLE SYMBOLS

Symbols defined as non-relocatable values are treated as integers. If the most significant bit of a non-relocatable value is one, the integer is assumed to be in complement form. If a 17-bit non-relocatable value is placed in an m, n or y subfield, it is reduced modulo $2^{15}-1$.

INTERCHANGE OF WORD AND CHARACTER ADDRESSES

A word address may be placed in a character address field or vice versa. If a symbol defined as a word address is placed in a subfield which consists of 17 bits, the assigned binary value is shifted left two places.

If a symbol defined as a character address is placed in a subfield which has only 15 bits, the 17 bit character address will be shifted right two places. If a one bit is lost by the shift, a T error occurs.

ASSEMBLY OF MACHINE LANGUAGE INSTRUCTIONS 3

The Control Data instruction repertoire for data processing, scientific and logical programming contains optional sets of BCD, floating point and double precision instructions for the hardware. All of these, including the optional commands, may be coded in the COMPASS language using convenient mnemonic codes and comprehensive symbolic programming techniques. The purpose of this discussion is to describe how machine language instructions are expressed in COMPASS, how COMPASS assembles them, and how they appear in the object program.

Control Data provides a set of simulation routines for the optional instructions. For any optional set not included at an installation, simulator routines may be placed on the library tape and called as subroutines. COMPASS will output the required XNL cards. Therefore, a programmer may use the mnemonics in the source subprogram as if the hardware were present.

The 24 bits of the instruction may be expressed as 4 fields:

operation code field, bits 23-18
designator field, bit 17
index or interval field, bits 16-15
address field, bits 14-0

Over half of the instructions are in this format, consisting of a mnemonic operation code; an indirect addressing indicator, or a condition or sign extension designator; an index register designator; and an address or operand. However, in order to obtain the power of the repertoire, it was necessary to introduce other formats.

To obtain the convenience desirable for symbolic programming and to exploit features of the computer, a flexible set of mnemonic codes and symbols was adopted for COMPASS.

There are eight formats for machine language instructions. Five of these formats require one computer word of 24 bits. Three are unique to the Block Class, and consist of two 24-bit computer words. In the machine reference manual, these are referred to as three-word instructions. When coding in COMPASS, the Block instructions are written as one line of code which will yield two 24-bit words in the object program. The programmer must produce the third word for the reject instruction on a separate line.

Instruction fields may be optional or mandatory. Limits exist on all fields in any instruction; an optional field may be expressed or not, as the programmer requires. The indirect addressing field, bit 17, and the b field, bits 16-15 are optional. Mandatory fields must be present and contain only stated options. The conditional modifiers for the AZJ instruction are an example of a mandatory field.

The codes used in the list of available instructions are explained below:

Term	Meaning	Bit Positions in Instruction
A	denotes the 24-bit A register	---
b	index register designator 1 to 3	16-15
B	denotes index register defined by B^b	---
x	connect code or interrupt mask	12-0
E	denotes the 48 (52)-bit E register	---
H	instruction modifier for INPC, OUTC indicating 6 or 12 bit I/O	18 (word 2)
i	increment or decrement, 0 to 7	17-15
I	instruction modifier denoting indirect addressing I present, indirect addressing is selected and bit 17 = 1 I omitted, direct addressing is selected and bit 17 = 0	17
ℓ	subscript representing lower half of the 48-bit E register, as E_ℓ	---
m	15-bit word address, first operand or jump address	14-0
M	actual operand or jump address as modified	---
n	same as m, second operand address	14-0
P	15 or (17)-bit P register	---
Q	24-bit Q register	---
r	17-bit character address	16-0
R	actual character address as modified	---
s	same as r, second operand address	16-0
S	instruction modifier denoting sign extension S present, bit 17 = 1, sign extended S omitted, bit 17 = 0, no sign extended	17
v	6-bit address in register file, 0 to 77	5-0
u	subscript representing the upper half of 48-bit E register, as E_u	---
y	15-bit operand or shift count	14-0
z	17-bit operand	16-0
c	denotes a character code or field in type Vb or Vc instruction	23-17 or 23-18
ch	denotes channel	23-21

Instruction Modifiers

EQ	equal
NE	not equal
GE	greater than or equal
LT	less than
I	indirect addressing
S	extend sign of operand to 24 bits
INT	interrupt on completion
A	conversion
B	backward read or write
H	half assembly or disassembly (12-24)
N	no assembly or disassembly
C	assign character address
NC	no conversion

OCTAL OPERATION CODE	SYMBOLIC INSTRUCTION FORMAT: MNEMONIC	SYMBOLIC INSTRUCTION FORMAT: ADDRESS FIELD INDEXING	OPERATION PERFORMED	INSTRUCTION TYPE
00.0	HLT	m	Unconditional stop RNI m	Ib
00.1	SL1	m	If key 1 is set, jump to m	Ib
00.2	SL2	m	If key 2 is set, jump to m	Ib
00.3	SL3	m	If key 3 is set, jump to m	Ib
00.4	SL4	m	If key 4 is set, jump to m	Ib
00.5	SL5	m	If key 5 is set, jump to m	Ib
00.6	SL6	m	If key 6 is set, jump to m	Ib
00.7	RTJ	m	$(P) \longrightarrow (m_{14-0})$, RNI m + 1	Ib
01.1-3	UJP,I	m, b	RNI m	Ia
02.0	no operation	(see 14.0)		Ia
02.1-3	IJI	m, b	If $(B^b) = 0$, RNI p + 1	Ia
			If $(B^b) \neq 0$, $(B^b) + 1 \longrightarrow (B^b)$, RNI m	
02.4-7	IJD	m, b	If $(B^b) = 0$, RNI p + 1	Ia
			If $(B^b) \neq 0$, $(B^b) - 1 \longrightarrow (B^b)$ RNI m	
03.0	AZJ,EQ	m	If (A) = 0, RNI m, otherwise RNI p + 1	Ib
03.1	AZJ,NE	m	If (A) ≠ 0, RNI m, otherwise RNI p + 1	Ib
03.2	AZJ,GE	m	If (A) ≥ 0, RNI m, otherwise RNI p + 1	Ib
03.3	AZJ,LT	m	If (A) < 0, RNI m, otherwise RNI p + 1	Ib
03.4	AQJ,EQ	m	If (A) = (Q), RNI m, otherwise RNI p + 1	Ib
03.5	AQJ,NE	m	If (A) ≠ (Q), RNI m, otherwise RNI p + 1	Ib
03.6	AQJ,GE	m	If (A) ≥ (Q), RNI m, otherwise RNI p + 1	Ib
03.7	AQJ,LT	m	If (A) < (Q), RNI m, otherwise RNI p + 1	Ib
04.0	ISE	y	If y = 0, RNI p + 2, otherwise RNI p + 1	Ia
04.1-3	ISE	y, b	If y = (B^b), RNI p + 2, otherwise RNI p + 1	Ia
04.4	ASE,S	y	If y = (A), RNI p + 2, otherwise RNI p + 1 sign extended	Ib
04.5	QSE,S	y	If y = (Q), RNI p + 2, otherwise RNI p + 1 sign extended	Ib
04.6	ASE	y	If y = (A), RNI p + 2, otherwise RNI p + 1 A = bits 14-0	Ib
04.7	QSE	y	If y = (Q), RNI p + 2, otherwise RNI p + 1 Q = bits 14-0	Ib

05.0	ISG	y	If 0 $\geq$, RNI p + 2, otherwise RNI p + 1	Ib
05.1-3	ISG	y, b	If $(B^b) \geq$, RNI p + 2, otherwise RNI p + 1	Ib
05.4-7	ASG,S	y	If (A) $\geq$, RNI p + 2, otherwise RNI p + 1 sign extended	Ib
05.5	QSG,S	y	If (Q) $\geq$, RNI p + 2, otherwise RNI p + 1 sign extended	Ib
05.6	ASG	y	If (A) $\geq$, RNI p + 2, otherwise RNI p + 1	Ib
05.7	QSG	y	If (Q) $\geq$, RNI p + 2, otherwise RNI p + 1	Ib
06.0-7	MEQ	m, i	(B') - i $\longrightarrow$ (B'); if (B') negative RNI p + 1 if (B') positive, test (A) = (Q) $\wedge$ (M), if true, RNI p + 2, if false, repeat sequence	Ib
07.0-7	MTH	m, i	(B^2) - i $\longrightarrow$ (B^2); if (B^2) negative, RNI p + 1; if (B^2) positive, test (A) $\geq$ (Q) $\wedge$ (M), if true RNI p + 2; if false, repeat sequence	Ib
10.0	SSH	m	Test sign of (m), shift (m) left closed one, if sign negative RNI p + 2; otherwise p + 1	Ib
10.1-3	ISI	y, b	If (B^b) = y, 0 $\longrightarrow$ (B^b) and RNI p + 2; if $(B^b) \neq$ y, $(B^b + 1 \longrightarrow (B^b)$, RNI p + 1	Ia
10.4-7	ISD	y, b	If (B^b) = y, 0 $\longrightarrow$ (B^b) RNI + 2; if $(B^b) \neq$ y, (B^b) - 1 $\longrightarrow$ (B^b), RNI p + 1	Ia
11.0	ECHA	y	y $\longrightarrow$ (A) bit 0-16	II
11.4	ECHA,S	y	y $\longrightarrow$ (A) sign extended	II
12.0-3	SHA	y, b	Shift A } left, magnitude of shift in k,	Ia
12.4-7	SHQ	y, b	Shift Q } right, complement of magnitude of shift in k	Ia
13.0-3	SHAQ	y, b	Shift AQ left, magnitude of shift in k right, complement of magnitude in k	Ia
13.4-7	SCAQ	y, b	Scale AQ	Ia
14.0	NOP		No operation (COMPASS assembled NOP)	Ia
14.1-3	ENI	y, b	Clear B^b, enter y	Ia
14.4	ENA,S	y	Clear A, enter y, sign extended	Ib
14.5	ENQ,S	y	Clear Q, enter y, sign extended	Ib
14.6	ENA	y	Clear A, enter y	Ib
14.7	ENQ	y	Clear Q, enter y	Ib
15.0	no operation			
15.1-3	INI	y, b	Increase index B^b by y, sign extended on y and B^b	Ia
15.4	INA,S	y	Increase A by y, sign extended	Ib
15.5	INQ,S	y	Increase Q by y, sign extended	Ib
15.6	INA	y	Increase A by y	Ib
15.7	INQ	y	Increase Q by y	Ib
16.0	No operation			
16.1-3	XOI	y, b	Enter selective complement of y and (B^b) into B^b	Ia
16.4	XOA,S	y	Enter selective complement of y and A into, A, sign of y extended	Ib
16.5	XOQ,S	y	Enter selective complement of y and Q into Q, sign of y extended	Ib
16.6	XOA	y	y V (A) $\longrightarrow$ A, no sign extended	Ib
16.7	XOQ	y	y V (Q) $\longrightarrow$ Q, no sign extended	Ib
17.0	No operation			
17.1-3	ANI	y, b	y $\wedge$ $(B^b) \longrightarrow B^b$	Ia
17.4	ANA,S	y	y $\wedge$ (A) $\longrightarrow$ A, sign of y extended	Ib
17.5	ANQ,S	y	y $\wedge$ (Q) $\longrightarrow$ Q, sign of y extended	Ib

17.6	ANA	y	y Λ (A) $\longrightarrow$ A, no sign extended	Ib
17.7	ANQ	y	y Λ (Q) $\longrightarrow$ Q, no sign extended	Ib
20	LDA,I	m, b	(M) $\longrightarrow$ (A)	Ia
21	LDQ,I	m, b	(M) $\longrightarrow$ (Q)	Ia
22	LACH	r, 1	(R) $\longrightarrow$ (A)	II
23	LQCH	r, 2	(R) $\longrightarrow$ (A)	II
24	LCA,I	m, b	$(\overline{M}) \longrightarrow (A)$	Ia
25	LDAQ,I	m, b	(M) $\longrightarrow$ (A), (M + 1) $\longrightarrow$ (Q)	Ia
26	LCAQ,I	m, b	$(\overline{M}) \longrightarrow (A)$, $(\overline{M + 1}) \longrightarrow (Q)$	Ia
27	LDL,I	m, b	(M) Λ (Q) $\longrightarrow$ (A)	Ia
30	ADA,I	m, b	(M) + (A) $\longrightarrow$ (A)	Ia
31	SBA,I	m, b	(A) - (M) $\longrightarrow$ (A)	Ia
32	ADAQ,I	m, b	(M, M + 1) + (A, Q) $\longrightarrow$ (A, Q)	Ia
33	SBAQ,I	m, b	(A, Q) - (M, M + 1) $\longrightarrow$ (A, Q)	Ia
34	RAD,I	m, b	(M) + (A) $\longrightarrow$ (M)	Ia
35	SSA,I	m, b	Where M contains a one bit, set the corresponding bit in A to one	Ia
36	SCA,I	m, b	Where M contains a one bit, complement the corresponding bit of A	Ia
37	LPA,I	m, b	(M) Λ (A) $\longrightarrow$ (A)	Ia
40	STA,I	m, b	(A) $\longrightarrow$ (M)	Ia
41	STQ,I	m, b	(Q) $\longrightarrow$ (M)	Ia
42	SACH	r, 2	$(A_{5-0}) \longrightarrow (R)$	II
43	SQCH	r, 1	$(Q_{5-0}) \longrightarrow (R)$	II
44	SWA,I	m, b	$(A_{14-0}) \longrightarrow (M_{14-0})$	Ia
45	STAQ,I	m, b	(A, Q) $\longrightarrow$ (M, M + 1)	Ia
46	SCHA,I	m, b	$(A_{16-0}) \longrightarrow (M_{16-0})$	Ia
47	STI,I	m, b	$(B^b) \longrightarrow (M_{14-0})$	Ia
50	MUA,I	m, b	(M) * (A) $\longrightarrow$ (Q, A)	Ia
51	DVA,I	m, b	(A) / (M) $\longrightarrow$ (A), Rem $\longrightarrow$ (Q)	Ia
52	CPR,I	m, b	(M) > (A), RNI p + 1 (Q) > (M), RNI p + 2 (A) $\geq$ (M) $\geq$ (Q), RNI p + 3 (A) & (Q) are unchanged	Ia
53.1-3	TIA	b	$0 \longrightarrow (A)$, $(B^b) \longrightarrow (A_{14-0})$	III
53.40-70	TAI	b	$(A_{14-0}) \longrightarrow (B^b)$	III
53.01	TMQ	v	(v) $\longrightarrow$ (Q)	IV
53.41	TQM	v	(Q) $\longrightarrow$ (v)	IV
53.02	TMA	v	(v) $\longrightarrow$ (A)	IV
53.42	TAM	v	(A) $\longrightarrow$ (v)	IV
53.(0+b)3	TMI	v, b	$(v_{14-0}) \longrightarrow (B^b)$	IV
53.(4+b)3	TIM	v, b	$(B^b) \longrightarrow (v_{14-0})$	IV
53.04	AQA		(A) + (Q) $\longrightarrow$ (A)	III
53.(0+b)4	AIA	b	$(A) + (B^b) \longrightarrow (A)$	III
53.(4+b)4	IAI	b	$(A) + (B^b) \longrightarrow (B)$	III

All other combinations of 53.00-77 are undefined and will be rejected by the assembler.

54	LDI,I	m, b	$(M_{14-0}) \longrightarrow (B^b)$	Ia
55.0		no operation		

55.1	ELQ		$(E_l) \longrightarrow (Q)$	Ib
55.2	EUA		$(E_u) \longrightarrow (A)$	Ib
55.3	EAQ		$(E_u, E_l) \longrightarrow (A, Q)$	
55.4		no operation		
55.5	QEL		$(Q) \longrightarrow (E_l)$	Ib
55.6	AEU		$(A) \longrightarrow (E_u)$	Ib
55.7	AQE		$(A, Q) \longrightarrow (E_u, E_l)$	Ib
56	MUAQ,I	m, b	(AQ) * (M) and (M + 1) ⟶ (AQE)	Ia
57	DVAQ	m, b	(AQE) / (M) and (M + 1) ⟶ (AQ) and remainder with sign extended in E. Divide fault, halts operation and program advances to next instruction.	Ia
60	FAD,I	m, b	Floating point addition of(M and M + 1)to (AQ) ⟶ (AQ)	Ia
61	FSB,I	m, b	Floating point subtraction of (M and M + 1) from (AQ) ⟶ (AQ)	Ia
62	FMU,I	m, b	Floating point multiplication of (AQ) and (M and M + 1) ⟶ (AQ)	Ia
63	FDV,I	m, b	Floating point division of (AQ) by (M and M + 1) ⟶ (AQ), remainder with sign extended to E	Ia
64†	LDE	m, 1	Load E with up to 12 numeric BCD characters from storage. Field length specified by contents of D register. Characters are read consecutively from least significant character (at address M + (D) - 1 until the most significant character (at address M) is in E. (E) is shifted right as loading progresses. The sign is acquired along with the least significant character.	II
65†	STE	m, 2	Store up to 13 numeric BCD characters from E. Least significant character stored at M + (D) - 1 continuing back to most significant character stored in M.	II
66†	ADE	m, 3	Up to twelve 4-bit characters (most significant character at address M) is added to (E). Sum appears in E. D register specifies field length.	II
67†	SBE	m, 3	Up to twelve 4-bit characters (most significant character at address m) is subtracted from E. Difference appears in E. D register specifies field length.	II
70.0-3	SFE	y, b	Shifts E in one character (4-bit) steps. Left shift: bit 23 = 0, magnitude of shift = lower 4 bits of Y = y + (B^b). Right shift: bit 23 = 1, magnitude of shift = lower 4 bits of complement of Y = y + (B^b).	Ia
70.4	EZJ,EQ	m	(E) = 0, jump to m; (E) ≠ 0, RNI p + 1	Ia
70.5	EZJ,LT	m	(E) < 0, jump to m; (E) ≥ 0, RNI p + 1	Ia
70.6	EOJ	m	Jump to m if E overflow, otherwise RNI p + 1	Ib
70.7	SET	y	Set (D) with lower 4 bits of y	Ib
71.0††	SRCE,INT	c, r, s	Search for inequality of character c in a list beginning at location r until unequal character is found, or until character location s is reached. $\leq c \leq 63$.	Vb
71.1†††	SRCN,INT	c, r, s	Same as 71.0.	Vb
72	MOVE,INT	c, r, s	Move c characters from r to s $1 \leq c \leq 128$	Vc
73.0†	INPC,A, INT,B,H	ch, r, s	6 or 12-bit character read from peripheral and stored in memory at given location.	Va

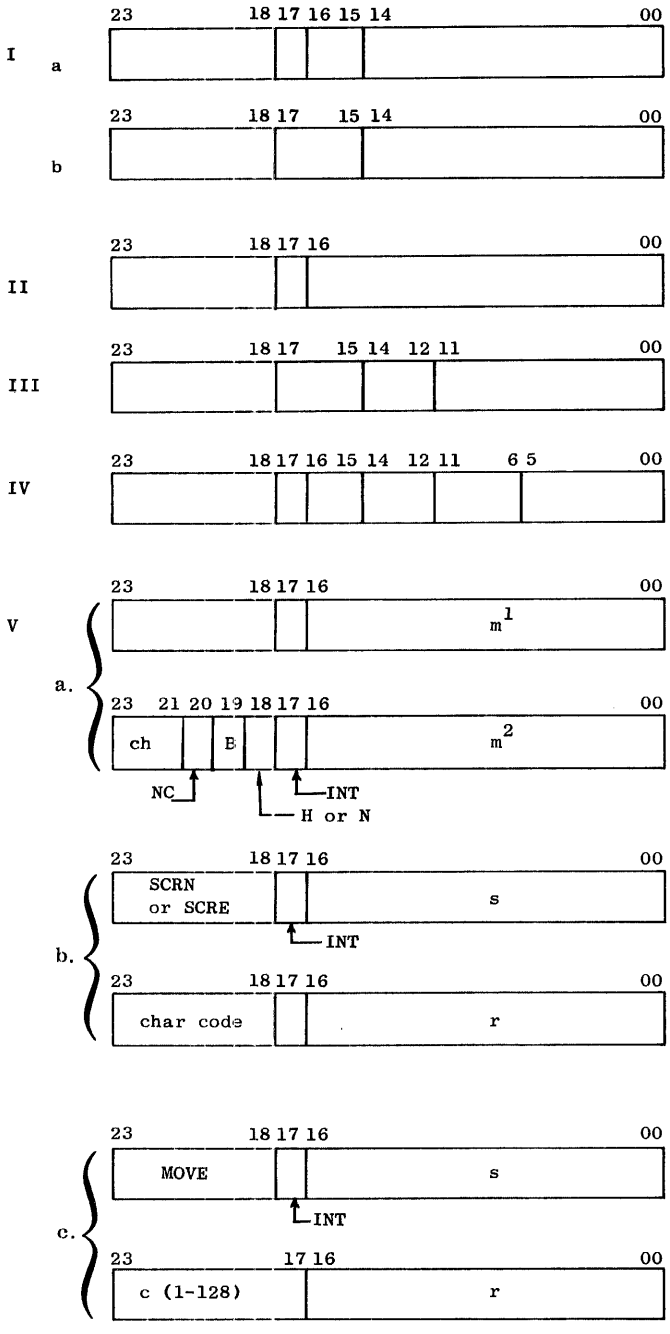
† 7-bit operation code, last digit is one in bit 17

†† 7-bit operation code and bit 17 on p + 1 = 0

††† 7-bit operation code and bit 17 on p + 1 = 1

73.1†	INAC,A,INT	ch	A cleared 6-bit character from peripheral to lower 6 bits of A.	Va
74.0†	INPW,A,INT, B,N	ch, m, n	Word address is placed in lower bits 0-14 15-16 zero, 12- or 24-bit words are read from peripheral and stored in memory.	Va
74.1†	INAW,A,INT	ch	A cleared 12- or 24-bit word read from peripheral (size depends on I/O channel) into lower 12 bits or all of A.	Va
75.0†	OUTC,A,INT, B,H	ch, r, s	Storage words assembled into 6 or 12-bit characters and sent to a peripheral device.	Va
75.1†	OTAC,A,INT	ch	Character from lower 6 bits of A is sent to peripheral device, (A) retained.	Va
76.0†	OUTW,A,INT, B,H	ch, m, n	Words read from storage to peripheral device.	Va
76.1†	OTAW,A,INT	ch	Word from lower 12 bits or all of A (depending on I/O channel) sent to a peripheral device.	V_a
77.0	CON	x, ch	If channel c busy, reject instruction, RNI p + 1	III
			If channel c not busy, 12-bit connect code sent on channel c with connect cable RNI p + 2	III
77.1	SEL	x, ch	Channel c busy, reject read from p + 1. c not busy, 12-bit function code sent on channel c with fn enable, RNI p + 2	III
77.2	COPY	ch	External status code from I/O channel c ⟶ lower 12-bits of A, contents of interrupt mask register ⟶ upper 12-bits of A. RNI p + 1	III
77.3	INS	x, ch	Sense internal status if 1 bits occur on status lines in any of same positions as 1 bits in mask, RNI p + 1. If no comparison, RNI p + 2.	III
77.3	CINS	ch	Interrupt mask and internal status to A.	III
77.4	INTS	x, ch	Sense for interrupt condition; if 1 bits occur simultaneously in interrupt lines and in RNI p + 1; if not, RNI p + 2	III
77.50	INTS	x	Interrupt faults defined by x are cleared.	III
77.51	IOCL	x	Clears I/O channel or search/move control as defined by bits 00-07 and 11 of x. Bits 08-10 are not used.	III
77.52	SSIM	x	Selectively sets Interrupt Mask Register; for each 1-bit in x, corresponding bit in register set to 1.	III
77.53	SCIM	x	Selectively clears Interrupt Mask Register; for each 1-bit in x, corresponding bit in register is set to 0.	III
77.54-56		no operation		
77.57	IAPR		Interrupt associated processor	III
77.6	PAUS	x	Sense busy lines — if 1 appears on a line corresponding to 1 bits in x, do not advance p. If p inhibited for longer than 150 usec, read reject from p + 1. If no comparison, RNI p + 2.	III
77.70	SLS		Program stops if Selective Stop switch is on; upon restarting, RNI p + 1	III
77.71	SFPF		Set floating point fault indicator.	III
77.72	SBCD		BCD fault flip-flop set 1.	III
77.73	DINT		Interrupt control is disabled.	III
77.74	EINT		Interrupt control is enabled, allows one more instruction to be executed before interrupt.	III
77.75	CTI		Set Type In / Set Type Out: beginning character address must be preset in location 23 of register file and last character address + 1 must be preset in location 33 of the file.	III
77.76	CTO			
77.77	UCS		Unconditioned stop. Upon restarting RNI p + 1	III

INSTRUCTION FORMATS



COMPASS PSEUDO INSTRUCTIONS 4

COMPASS contains a set of pseudo instructions for controlling the assembly process, converting constants, and reserving and assigning storage.

SUBPROGRAM CONTROL

Three pseudo instructions define a subprogram and provide control information for COMPASS. Information expressed in these pseudo instructions prepares control information for subsequent processing of the object program, including linking of subprograms.

IDENT

<blank> IDENT m

IDENT serves to control COMPASS and provide a subprogram name of 8 characters or less. The location field of IDENT should be blank. If a symbol is present, it will be ignored by COMPASS.

The address field contains the subprogram name; it may include as many characters as will fit into the field, but only the first eight are used as the subprogram name. They appear in the IDC card of the relocatable object subprogram deck and are printed as the title on the output listing unless, or until, a TITLE listing control pseudo instruction intervenes.

IDENT information is used by SCOPE as a reference when octal corrections or SNAP are included in a job. The subprogram name is not an entry point name and cannot be referenced in the source subprogram.

Instructions following IDENT are assembled using the subprogram address counter until the pseudo instructions DATA or COMMON intervene.

An O error flag is given if IDENT appears as other than the first instruction in the subprogram, and IDENT is ignored. If IDENT is not the first instruction, the job is terminated.

END

< blank> END m

END terminates the subprogram and produces a TRA card in the relocatable object subprogram deck. The location field should be blank; if present, it will be ignored by COMPASS.

A symbol in the address field is output to the TRA card as the symbolic transfer address. If the symbol is defined within the subprogram, the relocatable address is recorded on the <a> field of the TRA card. If a program is to receive control at the address denoted by the transfer symbol, the transfer address must be defined as an entry point during loading.

The final instruction in a COMPASS subprogram must be END.

FINIS

<blank> FINIS <blank>

FINIS signals that all subprograms have been assembled; it is the final instruction of a COMPASS input deck. The location and address fields are ignored by COMPASS. Normally FINIS immediately follows an END pseudo instruction.

COMPASS will write an end-of-file on all output units and if the unit allows, backspace over the end-of-file. However, COMPASS will recognize FINIS at any point even though it is in error and proceed as if END had occurred. If END is missing the job is terminated when control is returned to SCOPE.

PROGRAM STORAGE AREAS

The programmer may establish two storage areas to be shared by several subprograms. The common area may be shared by subprograms for information which is processed by the running program, or accumulated during the course of execution. Information may not be assembled in the common area. At the source language level the programmer may label, reserve or otherwise organize the common area but nothing more.

Information assembled for storage into the data area may consist of constants, message formats, masks and other information to be used by more than one subprogram. Both the common and data areas are shared by all subprograms during execution. The significant difference is that data can be prestored or loaded by the loader; common cannot. Three pseudo instructions indicate which of the two storage areas is referenced, or whether the program area is being defined.

During assembly, COMPASS initially uses the subprogram address counter. When the pseudo instructions DATA or COMMON are encountered, COMPASS assembles subsequent information into the appropriate area until another area assignment occurs. The pseudo instruction, PRG, returns control to the subprogram address counter.

If any instruction or pseudo instruction which results in binary output occurs while COMMON is in effect, an error indication is given and assembly continues as if PRG had occurred in the source subprogram. Any one of the three location counters may be set by the pseudo instruction ORGR. When COMPASS initiates use of a different area address counter, or the counter currently in

effect is reset by an ORGR, or is incremented by a BSS pseudo instruction, the current RIF card will be output.

PRG

<blank> PRG <blank>

The PRG pseudo instruction establishes the subprogram location counter during assembly. PRG specifies that all instructions which follow are to be assembled in a subprogram area; it restores the subprogram location counter for use by COMPASS. When IDENT is encountered, the subprogram counter is initialized and remains so until DATA or COMMON occurs. The location and address fields are ignored by COMPASS.

DATA

<blank> DATA <blank>

The DATA pseudo instruction specifies that all information following is to be stored or identified as part of the data area; it indicates use of the data area location counter. The location and address fields of DATA should be blank, and symbols contained in these fields are ignored by COMPASS.

Any instruction or pseudo instruction may follow DATA, provided that no reference is made to an external name and no location within the data area is declared an entry point to the subprogram. Once the DATA pseudo instruction occurs in a subprogram, the data area location counter is used for assembly until PRG or COMMON occur, or until the end of the subprogram.

COMMON

<blank> COMMON <blank>

COMMON organizes, labels and reserves space in the common area. The location and address fields of COMMON should be blank, if they are not, COMPASS ignores the field.

Information may not be assembled for storage in the common area; therefore, the only instructions which may follow COMMON are BSS, EQU, EXT, ENTRY, BSS, C, ORGR, IFT, IFN, IFF, IFZ, the output listing control instructions and PRG, DATA or END. If any other instruction is encountered, an error is flagged and assembly continues as if PRG had been encountered.

ORGR

< blank > ORGR m

The ORGR pseudo instruction controls the relocatable address for storage of instructions, constants, or the reservation of space in any of the three storage areas. The location field of ORGR is ignored by COMPASS and printed on the output listing. The address field may contain an expression which results

in a value for a relocatable address. Symbols must have been defined in the location field of a preceding instruction; and if relocatable, be assigned to the same area as the address counter currently in effect.

Example:

```
          IDENT   SAM
            .
            .
            .
          DATA
          BSS     2
MAC1      OCT     63
            .
            .
            .
          PRG
          ORGR    MAC1 + 16
            .
            .
            .
```

The above sequence would be incorrect because MAC1 + 16 would be in the DATA area and not under control of the subprogram address counter placed in effect by PRG.

If COMPASS is assembling into one area and an ORGR occurs with a different area relocatable symbol in the address field, an error results. All address counters remain unchanged, and COMPASS ignores the ORGR. The error flag is included on the output listing.

STORAGE RESERVATION

Full words or character positions may be reserved and labeled using the pseudo instruction BSS or BSS, C. The reservation is made in the area governed by the address counter COMPASS is currently using. The address field determines how many words or character positions are to be reserved.

BSS

<blank or symbol> BSS m

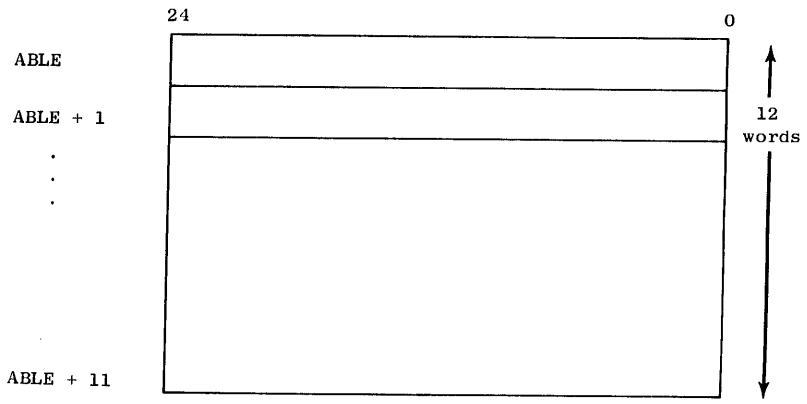
The BSS pseudo instruction reserves and labels a block of words in any area.

The location field may be blank or contain a symbol which is defined as the 15-bit relocatable word address of the first word in the block to be reserved by BSS. The assignment is made in the area governed by the address counter currently in effect for COMPASS assembly.

The address field, which specifies the number of words to be reserved, must contain a constant, a symbol, or an address expression which results in a non-relocatable value.

Example:

```
ABLE  BSS 12
```



The double asterisk (**) is illegal. The symbols in an address field must be defined in the location field of a preceding instruction. The result of an address expression must be a non-relocatable value.

If the address field is in error or is zero, no storage is reserved but a symbol in the location field has been defined. If the address field contains zero, and the instruction is immediately preceded by BCD, C or BSS, C, the next instruction which consumes space will be forced to a new word. To illustrate:

```
ALPHA   BDC,C  3,ABC     ALPHA
        BSS    0
        BDC,C  3,GHI     ALPHA + 1
```

A	B	C	

G	H	I	

```
ALPHA   BDC,C  3,ABC     ALPHA
        BDC,C  3,GHI     ALPHA + 1
```

A	B	C	G

H	I		

BSS,C

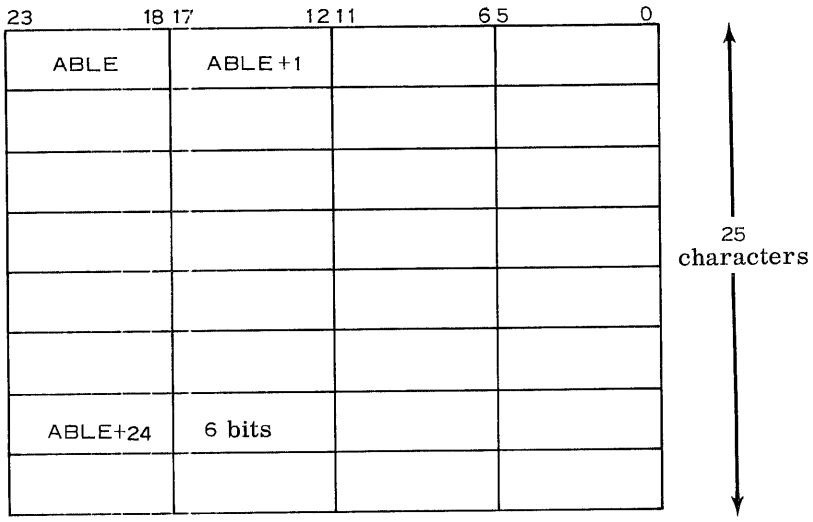
<symbol or blank> BSS, C m

The pseudo instruction BSS, C reserves and labels a block of character positions in the area governed by the location counter currently in effect for a COMPASS assembly.

The location field may contain a symbol or be blank. A symbol is defined as a 17-bit relocatable address of the first character in the block of characters to be reserved. The location symbol is defined as the momentary value of the current location counter. The address field specifies the number of characters

to be reserved. The address field must contain a constant, a symbol or an address expression which will result in a non-relocatable value. A zero address field does not reserve space, but the location symbol will be defined as above. When BSS, C is encountered, COMPASS will output the binary card it is building.

```
ABLE  BSS,C  25
```



SUBPROGRAM COMMUNICATION AND LINKAGE

Two pseudo instructions establish communication between subprograms. Programmers may define locations in a subprogram and declare them to be entry points. Symbols may be referenced within a subprogram even if they are not defined in that subprogram, provided they are declared as external symbols. Symbols declared external in one subprogram, will be declared as entry points in another subprogram. The resulting object subprograms will be loaded at the same time, but they need not be assembled at the same time.

A location to be referenced by another subprogram is declared an entry point with the ENTRY pseudo instruction. If a reference is made to a location in another subprogram, the symbol defining that location must be declared an external symbol in an EXT pseudo instruction. Using ENTRY and EXT , COMPASS produces EPT and XNL loader cards.

On the COMPASS assembly listing, instructions containing references to external names will have the usual format except the address field will be prefaced by X. These digits are the relocatable word address of a previous instruction in the subprogram area which references the external symbol. If it is the first or only reference to the external symbol, the address field listing will show X77777.

COMPASS places the subprogram relocatable address of the last instruction referencing the external symbol into the XNL loader card to begin the threaded list. If no reference is made to the external symbol in the subprogram, the XNL loader card will contain the address 77777_8, indicating there is no thread.

Each external name can be associated with two threaded lists; one for 15-bit addresses and one for 17-bit addresses. All references are chained in the threaded list with only the symbol in the EXT declaration appearing in an XNL card.

ENTRY

<blank> ENTRY $m_1, m_2, \ldots, m_n$

The location field of ENTRY should be blank. If a symbol appears, it is ignored by COMPASS. The address field contains one or more location names separated by commas; it may not contain blanks. The field terminates with the first blank or column 73.

Each subfield contains a symbol defined as a subprogram relocatable word address by appearance in a location field elsewhere in the subprogram.

EXT

<blank> EXT $m_1, m_2, \ldots, m_n$

Symbols referenced but not defined in the subprogram must be declared as external names in EXT pseudo instructions.

The location field should be blank; a symbol is ignored by COMPASS. The address field contains one or more subfields up to column 73; subfields are separated by commas and may not contain blanks. This field terminates with column 73 or the first blank column.

Each subfield contains a symbol which is output to an XNL loader card. The symbol must not be defined within the subprogram in which it is declared external; it may be referenced only from an instruction assembled into the subprogram area.

SYMBOL DEFINITION BY EQUATING

A symbol may be defined by equating it to another symbol, a constant, or an expression. The symbol may be defined as an absolute value, a relocatable word or relocatable character address. The symbol in the location field of the pseudo instruction is equated to the value of the address field. A symbol which is declared an entry point in the subprogram must not be equated to a symbol declared as external.

When the symbols are equated, they are identical and interchangeable; all symbols in the address field must have been previously defined by appearance in the location field of a preceding instruction or in an EXT declaration.

EQU

<symbol> EQU m

This symbol is equated to another symbol, a 15-bit word address or a 15-bit value. The symbol in the location field will be non-relocatable or relocatable as determined by the address field. If the location field does not contain a symbol, an error occurs.

The address field determines the definition of the symbol in the location field. It may contain:

An integer modulo $2^{15}-1$.

A symbol defined by appearance in the location field of a preceding instruction. The symbol in the location field is equated to the entry in the address field. If the symbol in the address field is relocatable into a given area, the symbol in the location field is also relocatable into that area.

An address expression containing symbols defined as above, and conforming to the rules for m subfields.

Expressions may not result in a complement relocatable value.

An entry point should not be equated to an external symbol. COMPASS will not log an error but when the object subprogram is loaded an error will result.

EQU,C

<symbol> EQU, C r

The symbol is equated to a 17-bit address, 17-bit value or another symbol. The symbol in the location field will be nonrelocatable or relocatable as determined by the address field. If the location field does not contain a symbol, an error occurs.

The address field determines the definition of the symbol in the location field. It may contain:

An integer modulo $2^{17}-1$.

A symbol defined by appearance in the location field of a preceding instruction. The symbol in the location field is equated to the entry in the address field. If the symbol in the address field is relocatable into a given area, the symbol in the location field is also relocatable into that area.

An address expression containing symbols defined as above and conforming to the rules for r subfields.

Expressions must not result in a complement relocatable value.

COMPASS ASSEMBLY OF CONSTANTS

Constants may be stated as octal, decimal or character in the source language. The constants may be single, double or variable precision of fixed or floating point format. Character constants may be placed into full words or character positions. Constants may be placed into bit positions of variable length fields.

The pseudo instructions are:

OCT	prepare octal constants
DEC	prepare decimal constants, fixed point
DECD	prepare double precision and/or floating point decimal constants
BCD	prepare binary coded decimal (internal BCD) word fields
BCD,C	prepare binary coded decimal character position fields
VFD	prepare variable fields

OCT

<symbol or blank> OCT $m_1, m_2, \ldots, m_n$

Expresses constants as signed or unsigned octal integers. The octal integer may consist of eight or less digits. As many constants as can be contained on a card may be expressed in the address field; they are separated by commas. The octal constants are assembled, right adjusted, for storage into consecutive locations. The address field of OCT is terminated by the first blank or column 73.

An optional binary scale factor may be stated by suffixing the constant by B and expressing the scale factor as a signed or unsigned decimal integer of not more than two digits. The magnitude of the constant after scaling must be less than 2^{24}.

The location field may be blank or contain a symbol which yields the 15-bit word address of the first constant in the address field.

The address field contains as many subfields as the card will contain, separated by commas. The subfields are of identical form at, and each may contain a signed or unsigned octal integer of not more than eight octal digits. The address field is terminated by the first blank column.

Example:

```
OCT 77777777,12345670,76543210
        octal result
word 1 | 77777777 |
     2 | 12345670 |
     3 | 76543210 |
```

```
OCT + 1, -57, 2040, -2
        octal result
word 1 | 00000001 |
     2 | 77777720 |
     3 | 00002040 |
     4 | 77777775 |
```

```
OCT 72B2
| 00000350 |
```

DEC

<symbol or blank> DEC $d_1, d_2, \ldots, d_n$

Constants may be expressed in decimal and converted for storage as single precision fixed point binary constants. A decimal and/or binary scale factor may be expressed for the 24-bit constant. The decimal constant may consist of a sign and not more than seven digits with a magnitude of less than 2^{23}. The decimal integer may be followed by a decimal or a binary scaling factor or both. If both are stated, they may appear in either order.

Examples:

1	decimal integer
+2	decimal integer
-38	decimal integer
1D5	decimal integer, decimal scale factor
73D-2	decimal integer, decimal scale factor
-6D+1B4	decimal integer, decimal and binary scale factors
200B-7	decimal integer, binary scale factor
36B+2D1	decimal integer, binary and decimal scale factors

The magnitude of the constant after scaling must be less than 2^{23}. The conversion is performed in three steps:

1. The decimal integer is converted to binary; the binary integer must be less than or equal to $2^{23}-1$ in magnitude.
2. The binary integer is multiplied or divided by 10^{d}; d is the decimal scaling factor. The magnitude of the result must be less than 2^{47}. If the decimal scaling factor is negative, a 47-bit fraction or mixed fraction is formed.
3. The result in step 2 is shifted the number of bits specified by the binary scaling factor. A negative factor produces a right shift; a positive scale factor causes a left shift to be performed. If non-zero bits are lost from the high order 24 bits of the result from step 2, an error is flagged. Low order bits of the intermediate result may legally be lost.

The location field of the DEC instruction may be blank or contain a symbol which is the relocatable word address of the first constant in the address field.

The address field may consist of as many subfields separated by commas, as the card can contain. The first blank terminates the address field and subsequent information is treated as remarks.

DECD

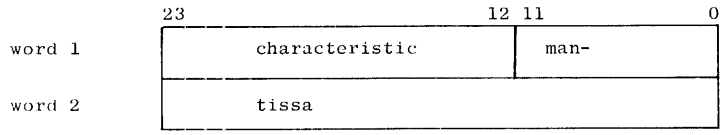
<symbol or blank> DECD $d_1, d_2, d_3, \ldots, d_n$

Decimal values may be stored as double precision fixed point constants or floating point constants. Either format requires 48 bits for storage. The format of the DECD pseudo instruction is the same as DEC except for the range of values and the point in floating point constants. The location and address fields are treated in the same fashion for DEC and DECD. A symbol in the location field references the first of the two words assembled as the result of DECD.

Fixed point constant format is identical to that of the DEC single precision constants, except that magnitudes may be larger. Up to 14 decimal digits may be specified, expressing a value whose magnitude is less than 2^{47}. Decimal and binary scale factors may be used as in the DEC pseudo instruction. The signed 48-bit binary result is stored into two consecutive computer words.

FLOATING POINT CONSTANTS

Floating point constants are stored as two 24-bit words. Floating point constants contain a decimal point. Floating point constants are stored as a 12-bit characteristic and 36-bit mantissa. Negative values are held in complement form.



Floating point constants may contain not more than 14 decimal digits and a decimal point which may appear anywhere within the constant. Binary scaling is not permitted. Decimal scaling is specified by suffixing the constant with a D followed by a signed or unsigned decimal scaling factor. In the absence of a sign, a positive value is assumed. The result after scaling must not exceed the capacity of the hardware (approximately $10^{\pm 308}$).

BCD

<symbol or blank> BCD n, $c_1 c_2 \ldots c_{4n}$

Characters are assembled for storage into consecutive computer words. They are stored as 6-bit binary coded decimal character codes into addressable character positions. The code is the internal BCD code.

The location field may be blank or contain a symbol which is established as the 15-bit relocatable word address of the first word in the field. The decimal integer, n in the address subfield contains the number of words to be used; n is separated from the characters to be converted and stored, by a comma. Four characters can be contained in a word, 4n characters may be punched in the card. If 4n characters cannot be held in the card before column 73, characters are converted through column 72. The character positions and/or words reserved by n, which cannot be expressed on the card, are filled with blanks. Any information on the card after 4n characters is treated as remarks.

BCD,C

<symbol or blank> BCD, C n, $c_1 c_2 \ldots c_n$

Characters may be assembled for storage into consecutive character positions using the pseudo instruction BCD, C. The characters are converted and encoded as for BCD; but the address, the manner of storage, and the statement of field length differ. The location field may contain a symbol or be blank. A symbol, if present, is established as a 17-bit character address.

The modifier, C, in the operation code denotes that character addresses and character string length are to be processed, not words. The card terminates after n characters or column 72, whichever occurs first. If n characters do not extend through column 72, information between the n characters and column 72 is treated as remarks; $1 \leq n \leq 2^{15}-1$. Character positions are filled from card columns until column 73. Character positions reserved but not expressed on the card are filled with blanks.

Characters are prepared for storage into consecutive positions. If the line of code immediately preceding BCD, C in the source program assigns character storage rather than word storage; the character string begins in the first available character position. If the line in the source program deals with word storage, BCD, C is assigned to the first character position in the first available word. The pseudo instructions which cause this variation are BSS, C and BCD, C.

If the number of characters declared by BCD, C does not fill an entire word, the unused positions in the trailing partial word are filled with zeros. If the next instruction which consumes space in the object program is BCD, C, the positions in the partial word are assigned to the leading characters to produce a packed field.

Example:

```
MOTCC BCD 9,ABCDEFGHIJKLMNOPQRSTUVWXYZ=-+0.)-0$BCD4, */,(1234567890
```

The characters in the above line comprise the complete BCD character set; including the blank.

Location	Content			
MOTCC	A	B	C	D
	21	22	23	24
	E	F	G	H
	25	26	27	30
	I	J	K	L
	31	41	42	43
	M	N	O	P
	44	45	46	47
	Q	R	S	T
	50	51	62	63
	U	V	W	X
	64	65	66	67
	Y	Z	=	-
	70	71	13	14
	+	+0	.	)
	20	32	33	34
	-	-0	$	*
	40	52	53	54
	b	/	,	(
	60	61	73	74
	1	2	3	4
	01	02	03	04
	5	6	7	8
	05	06	07	10
	9	0	6	6
	11	00	60	60

VARIABLE FIELD DEFINITION

<symbol or blank> VFD mn/v,. . .,mn/v

This pseudo instruction enters octal numbers, character codes, relocatable addresses or constants into variable length field. The field is assigned as a continuous string of bits of specified length. Information is placed into the field regardless of word length or character position. Unused bits in the least significant bit positions of the final word in the field are filled with zeros.

Each VFD instruction begins filling a new computer word. A symbol in the location field is defined as a relocatable word address. As many address subfields as can be contained on a single card are allowed; the address subfield terminates with a comma except when a blank terminates the entire VFD pseudo instruction. The card and the VFD pseudo instruction are terminated by a blank.

The mode parameter may designate one of four modes. The remainder of the subfield is restricted by the specified mode. Values are entered into variable fields right adjusted and character strings left adjusted.

The location field may contain a legal symbol or blanks. A symbol, if present, yields a relocatable word address. The address field contains subfields of the same general format.

- m mode indicator
- n positive decimal integer denoting the number of bit positions in the variable field specified by this subfield. The range of values for n varies with mode.
- / the virgule separates the description of the field mode and length from the statement of the field content.
- v represents the content of the variable field which varies according to mode and is restricted by declared length.

The address field may contain one or more subfields. The mode and length of the subfields may differ.

VFD MODES

The statement of variable field length and content varies according to the mode. Four modes may be expressed in a VFD address subfield.

- O Octal
- H 6-bit Hollerith (internal BCD)
- A Word address arithmetic
- C Character address arithmetic

OCTAL

VFD On/v

If the mode is octal, O, n may be 1 to 24 and v may be a maximum of 8 octal digits; the integer may be signed. If negative, the filed content is stored in one's complement form. The value is entered into the field right justified with leading bits inserted according to the sign and length. If the value exceeds the length of the field, an error is flagged and the field is set to zero. A binary scale factor may be supplied in the same manner as for an OCT pseudo instruction.

A octal subfield is terminated by a comma or a blank. If terminated by a blank, the address field is terminated and remaining information on the card is treated as a comment.

HOLLERITH

VFD Hn/v

The programmer may enter Hollerith information which is stored as a 6-bit internal BCD character codes; n must be expressed as a multiple of six. The subfield, v, terminates with the first comma or blank. If the subfield does not terminate after n/6 character an error results.

WORD ADDRESS

VFD An/v

The word address field consists of a constant, a symbol, or an expression formed by the rules for address field arithmetic. The VFD address field is terminated by the first blank column.

The location field may be blank, or contain a symbol.

n is the field length, restricted by the contents.

A type variable field length is 1 to 24.

If an expression yields a relocatable word address, the value must be entered into the computer word right justified to bit zero by the programmer. The expression is evaluated modulo $2^{15}-1$.

If the expression of the subfield results in a fixed value the field specified by n will be evaluated $2^{n}-1$.

CHARACTER ADDRESS

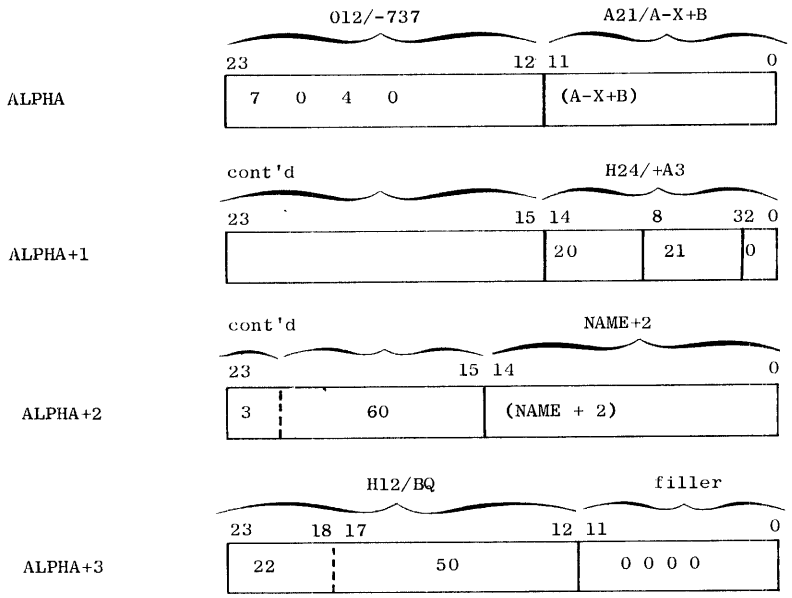
VFD Cn/v

The character address variable field is governed by the above rules, except that a minimum of 17 bits is required.

Example:

VFD a12/-737, A21/A-X+B, H24/+A3, A15/NAME+2, H12/BQ

A, X, and B are not relocatable symbols. Four words are generated, with the data placed as follows:



MACRO USE IN COMPASS 5

ASSEMBLY OF MACROS AND MACRO CALLS

A macro instruction is prepared with COMPASS pseudo instructions and macros are called by other pseudo instructions.

MACRO	define macro name and/or formal parameter
ENDM	terminate a macro prototype
IFT	if a given condition is true, assemble instructions when the macro is called
IFF	if a given condition is false, assemble instructions when the macro is called
LIBM	fetch the named macro prototypes from the library tape for use by the subprogram

Macros are called by:

<macro name>	assemble the defined macro with this name, using the parameters given at this point in the subprogram.

MACRO INSTRUCTION

When an operation is performed frequently in a program or in many programs, the instructions required to perform that operation may be coded once and defined as a macro instruction. The macro is defined by pseudo instructions and called by an assigned name. The same code is obtained and included in the subprogram each time the operation is desired.

Examples of operations which might be devised as macros include convert binary to decimal, delete blank characters or similar operations. Once defined, the macro name is used as any other operation code.

A library macro instruction may be placed in file two of LIB for general use; a programmer macro is defined by a programmer for use in a particular subprogram.

LIBRARY MACROS

The library macros may be unique to an installation; a list of macros should be available to the programmer. When a macro defined in LIB is to be used by a programmer, he must declare the macro name in the address field of a LIBM psudo instruction immediately following the IDENT pseudo instruction of the subprogram.

Library macros are placed on LIB using the MACRO control statement of PRELIB.

PROGRAMMER MACROS

The programmer may define macros for any subprogram immediately following the declaration of library macros, if any. Programmer macros are defined only in the subprogram in which they occur and may be referenced within that subprogram.

<symbol> MACRO $(p_1, p_2, \ldots, p_n)$

This statement indicates macro prototypes are to be processed, defines the macro name, and declares the formal parameters for the macro. The MACRO pseudo instruction does not consume space in the object program. Instructions and pseudo instructions follow until the pseudo instruction ENDM terminates the prototype.

Any location defined in the subprogram may be referenced within the macro; however, a location within a macro is local to the macro and may not be referenced from outside the macro. Reference may be made within the prototype to symbols external to the subprogram if they are declared by EXT pseudo instructions within either the macro or the subprogram. An EXT declaration within the macro remains in force for the entire subprogram.

If the EQU pseudo instruction appears within the macro instruction prototype, the symbol in the location field is considered local to the macro and treated as any location symbol in the macro.

The MACRO pseudo instruction must immediately follow the pseudo instructions IDENT, LIBM, or MACRO, except that comment cards (asterisk in column one) and the pseudo instructions REM or TITLE may intervene. If MACRO follows IDENT or LIBM, it defines a macro instruction; and the location field must contain a symbol which is the macro name. If MACRO follows a macro definition, it is a formal parameter continuation card; and the location field must be blank.

MACRO INSTRUCTION PROTOTYPE

The prototype code for a macro instruction begins with the pseudo instruction MACRO and terminates with ENDM. Any mnemonic or octal operation code or modifier in the hardware repertoire, any COMPASS pseudo instruction except IDENT, LIBM, ENDM, MACRO, or <macro name> may be used for the operation code field in the prototype.

The pseudo instructions ENTRY and EXT may appear within the prototype. Location symbols within a macro may not be declared as entry point names.

LOCATION SYMBOL IN MACRO PROTOTYPE

The location field of an instruction in the prototype may contain a location symbol of four characters or less; the location symbol is defined as local to the macro prototype in which it appears and will not conflict if used elsewhere in the subprogram. Parameters are not passed to the location field.

<symbol> MACRO $(p_1, p_2, \ldots, p_n)$

The first MACRO pseudo instruction must contain a legal symbol of 1 to 8 characters in the location field; the first character must be alphabetic. This symbol is the name of the macro instruction, not a location symbol. It may appear only in the operation field of instructions in subprograms for which the macro instruction is defined. The symbol in the location field should not be the same as any mnemonic code in the hardware repertoire, the pseudo instruction list, or the name of any other macro instruction defined for the subprogram in which this MACRO occurs. If this macro prototype is destined for LIB, the name should not conflict with any other macro on LIB.

Except as noted, the location symbol in the prototype is the same as any other COMPASS location symbol.

FORMAL PARAMETERS FOR MACROS

Formal parameters declared in the address fields of MACRO pseudo instructions are referenced in the macro prototype. Formal parameters represent elements of code to be defined when the macro instruction is used or called.

A formal parameter may represent any portion of an instruction or the entire instruction except for the location field. This flexibility is attained through the use of parentheses as delimiters in processing macro calls.

An address subfield or portion of a subfield may be expressed as a single formal parameter if that portion of the subfield is set off by a plus or minus sign, a comma, a blank or, in the case of VFD, a slash. A BCD or BCD, C pseudo instruction, however, may have only the n subfield expressed as a formal parameter.

When the macro instruction is assembled into a subprogram, the macro call contains actual parameters to be used when the macro instruction is executed. COMPASS transfers the actual parameter to the place at which the corresponding formal parameter was referenced in the prototype. Instructions in the prototype are assembled the same as other instructions in a subprogram.

The list of formal parameters must be enclosed in parentheses and individual parameters are separated by commas. Blanks may precede or follow the parameter but may not be imbedded.

The address field of a MACRO pseudo instruction must not extend beyond column 72. This does not limit the formal parameter list which may be continued on subsequent MACRO pseudo instruction cards provided that three rules are followed.

Formal parameter continuation cards must have a blank location field.

The operation field must contain the pseudo-instruction MACRO.

A formal parameter field and its terminal comma must be contained wholly on a single card prior to column 73.

Formal parameters may appear in an operation code or address field of the prototype which follows the MACRO pseudo instruction defining the macro name and/or formal parameter list. Symbols in the formal parameter list and the prototype are local to the macro instruction and identical symbols may be used elsewhere in the subprogram without conflict. The macro name should be unique within the subprogram.

MACRO CALLS

<symbol or blank> <macro name> $(p_1, p_2, \ldots, p_n)$

A subprogram may contain any macro instruction defined for the subprogram by LIBM and/or MACRO pseudo instructions and prototypes by referring to the macro name in the operation code. If the macro is defined for the subprogram, COMPASS will assemble and insert the macro code at that point.

The location field may be blank or contain a symbol which is the relocatable address of the first instruction that consumes space in the assembled macro. The address field of the macro name instruction provides constants, symbols, expressions or Hollerith literals. The actual parameters retain the sequence of the formal parameter list in the macro definition. When COMPASS assembles the macro, the actual parameters are transferred to the position at which the

formal parameters are referenced in the prototype. The address field of a single card terminates at column 72 or with a right parenthesis.

The actual parameter list need not be contained on a single card. It may be continued on subsequent cards with blank location field and the macro name operation code repeated. An actual parameter must be wholly contained on a single card. If the list is not closed by a right parenthesis, an error results.

The space consumed in the object program by a macro name instruction is determined by the prototype and by the conditional pseudo instruction IFZ, IFN, IFT, IFF within the prototype.

MACRO NAME ADDRESS FIELDS

The address field contains the list of actual parameters enclosed by parentheses. Parameters in the list are separated by commas. Single actual parameters may also be enclosed by parentheses within the list. This allows an entire instruction or several subfields of an instruction in the macro prototype to be expressed as a single actual parameter.

Single actual parameters may not include blanks or commas unless the entire actual parameter is enclosed in parentheses. If a single actual parameter is enclosed by parentheses, it may contain any character legal for the portion of the instruction it represents except a right parentheses. Refer to the examples. An actual parameter may be expressed as zero by providing only the comma. Actual parameters not expressed before the list terminates are assembled as zeros.

Actual parameters may not contain entries for location fields in the prototype. These fields will not be modified by COMPASS when assembling a macro instruction.

Examples of Macro Definition:

```
DIVIDE    MACRO     (P1,P2,P3)
          LDAQ      P1
          DVA       P2
          STQ       P3
          ENDM
```

The MACRO is tagged DIVIDE. The set of instructions following is a prototype of the instructions to be assembled when DIVIDE is called.

```
COMPUTE     MACRO     (P1,P2,P3,P4,P5,P6,P7,P8,P9)
            LDA       P1
            LDQ       P1
            ADA       P2

            SBAQ      P3
            VFD       P4/P5
            P4        **
            DVA       P6

            STQ       P7
            LDA       P7
            P8        *-P9

            ENDM
```

The above example shows how parameters may be specified in the operation field or the address field, or both, within the macro set of instructions. It also shows that a parameter may appear more than once in a set of instructions.

```
JAYSON      MACRO     (P1,P2,P3)
            LDA       P1
            ADA       P2
            STA       P3
            ENDM


            JAYSON ( (DOG,3), CHARLES, CAT)

Assembled:  LDA       DOG, 3
            ADA       CHARLES
            STA       CAT

TOSS        MACRO     (P1,P2,P3,P4)
            LDA       TOM, 3
            ADA       DICK
            STA       MARV
            UJP,P2
      TOM   P1
      DICK  DEC       P4
      MARV  P3
            ENDM
              .
              .
              .
BETTY       TOSS      ( (BDC 6,A), (I JOE, 2) )
```

Assembled:

```
BETTY       LDA     TOM, 3
            ADA     DICK
            STA     MARV
            UJP,I   JOE, 2
     TOM    BCD     6, A
     DICK   DEC     0
     MARV   00
```

LIBM

< blank> LIBM $name_1, name_2, \ldots, name_n$

Library macros are stored on LIB via the PRELIB system. LIBM instructs COMPASS to call a particular library macro from LIB. LIBM must be contiguous to the IDENT pseudo instruction or another LIBM pseudo instruction, otherwise an error results. However, comment cards with an asterisk in column one or the pseudo instructions REM and TITLE may intervene. LIBM does not consume space in the object program.

The location field of LIBM should be blank. A location symbol will be ignored by COMPASS but included on the output listing. Subfields in the address field contain the names of library macros separated by commas. The address field is terminated by the first blank or column 73.

The programmer may use as many LIBM pseudo instructions as required. However, a macro name must be wholly contained within a single subfield on a single card.

ENDM

< blank> ENDM <blank>

The prototype of a macro instruction is terminated by ENDM. The location field and address fields of ENDM should be blank. A location symbol is ignored by COMPASS but included on the output listing. If an entry appears in the address field, an error results.

CONDITIONAL PSEUDO INSTRUCTIONS

Assembly of instructions and constants from a source subprogram may be conditional as stated by the pseudo instructions listed below. COMPASS tests for the condition and includes subsequent lines of code depending on the outcome of the test.

IFZ	if zero
IFN	if non zero
IFT	if true
IFF	if false

The IFZ and IFN pseudo instructions may be used as desired in a subprogram. IFT and IFF, which compare a parameter string against stated variables, may occur only within a macro prototype. Macros are the only instance of such a parameter string which COMPASS can detect. In IFZ and IFN, symbols in the address field must be previously defined. Symbols in the third subfield of IFF and IFT must also be previously defined.

IFZ

<blank> IFZ m, n

An arithmetic expression may be stated and tested for zero to determine whether subsequent instructions should be included in a subprogram. The expression must conform to the rules for address expressions. A symbol in the location field is ignored by COMPASS but is included in the output listing.

The address field consists of two subfields.

m is an expression, the value of which is computed as any address expression and evaluated modulo $2^{15}-1$.

n contains an integer or an expression which results in a non-relocatable value.

If the expression in the m subfield results in zero, the following n lines of code are assembled into the object subprogram. If the m subfield yields a non-zero value, the n lines of code are skipped and do not appear in the subprogram. Symbols in the address field must be defined by appearance in the location field of a preceding instruction.

IFN

<blank> IFN m, n

This pseudo instruction is the same as IFZ except that n lines of code are assembled if the value in the m field is non-zero.

<blank> IFT m, p, n

IFT

Within a macro prototype, lines of code may be excluded or included in an object subprogram using the IFT pseudo instruction. The IFT pseudo instruction compares the first two subfields in its address field for literal equality. If the two character strings are equal, following lines of code are assembled. If they do not compare, the lines of code are excluded from the object program.

The location field of the IFT pseudo instruction should be blank. If a symbol is present it will be ignored by COMPASS but will be printed on the output listing. The address field has three subfields.

m	designates the first comparand
p	designates the second comparand
n	must result in a non-relocatable value denoting the number of lines of code to be assembled or excluded

The m and p terms may be character strings or formal parameters; the character string may not include slashes. If a character string is identical to a formal parameter, the string must be enclosed in slashes. Either m or p may be expressed as a character string.

The actual values used in the comparison are obtained by COMPASS as follows:

If the subfield is enclosed in slashes, the content of the subfield is used in the comparison.

If the subfield contains a formal parameter, COMPASS substitutes the corresponding actual parameter before the test is made.

If the subfield is not a formal parameter and is not enclosed in slashes, the character string is used as if slashes had appeared.

The n term must be a symbol, a constant, or an expression which results in a nonrelocatable value. Symbols in the address field must be previously defined.

If the m and p terms compare bit for bit, the n lines of code immediately following the IFT pseudo instruction are assembled into the subprogram. If the m and p terms are unlike, the n lines are skipped and not assembled by COMPASS.

Examples:

```
COMPUTE MACRO   (P1,P2,P3,P4,P5,P6)
        LDA     P1
        DVA     P2
        STQ     P3
        IFT     /P6/, P5, 2
        ENA     P4
        ENI     P6
        ENDM
```

The following sequence of instructions occurs within a subprogram and the call refers to the previously defined macro set.

```
        STA       TABLE
CAKE    COMPUTE   (B,C,A,LOC1,P6,56)
        LDAQ      QUANTITY
          .
          .
          .
```

The assembler would generate:

```
        STA       TABLE
CAKE    LDA       B
        DVA       C
        STQ       A
        ENA       LOC1
        ENI       56
        LDAQ      QUANTITY
```

Since the actual parameter substituted for P5 is identical to the character string "P6", the assembler includes the two instructions, ENA and ENI. The IFT instruction does not appear in the object subprogram.

```
        STA       TABLE
        COMPUTE   (B,C,A, LOC2, 54,56)
        LDAQ      QUANTITY
```

The assembler would generate:

```
        STA       TABLE
        LDA       B
        DVA       C
        STQ       A
        LDAQ      QUANTITY
          .
          .
          .
```

Since 54 is not equal to the characters enclosed in slashes in the IFT pseudo instruction, the assembler does not assemble the two instructions, ENA and ENI. Assembly continues with the next instruction from the input deck.

IFF

<blank> IFF m,p,n

The conditional pseudo instruction IFF functions the same as IFT except comparands are unlike, the n lines of code are assembled. If the m and p terms are identical, the n lines of code are excluded.

COMPASS OUTPUT LISTING 6

The output listings from COMPASS contain the octal relocatable storage addresses and relocatable and/or absolute contents of words in the object subprogram. The source code is printed side-by-side with the object code. Summaries of relevant information are printed before and after the subprogram code. The representation of the object code on a line of the listing depends on the type or class of machine instruction or pseudo instruction being printed.

OUTPUT LISTING FORMAT

The output listing format is shown in the accompanying table. The title of the listing is printed at the top of each page. A title is the subprogram name from the IDENT pseudo instruction or a title expressed via a TITLE pseudo instruction. The page number is printed in the upper right hand corner of each page of output.

Preceding the body of the subprogram are summaries of:

undefined symbols

doubly defined symbols

external names

entry point names

These are followed by storage summaries of

length of subprogram	5 octal digits
length of common	5 octal digits
length of data	5 octal digits

The subprogram is printed on subsequent pages; COMPASS provides several options for incorporating comments and remarks into the source program. Listing may be suppressed and resumed.

Following the source subprogram listing, single precision literals are printed three per line; double precision literals are printed two per line. The 5-digit octal relocatable address of the literal is followed by the 8 or 16 octal digits of the converted number. The number of errors detected in the subprogram is printed and the symbol reference table follows as the final COMPASS output.

Listing formats are given below:

Printer Columns	Content
COMPASS Subprogram Listing	
2-8	error code
9	blank
10-14	octal relocatable storage address
15	character position indicator or blank
16	blank
Machine Language Instructions	
17-24	8 octal digits of the word destined for storage, punched into RIF card. The representation is shown without regard to relocation.
25	blank
26-27	2-digit octal operation code
28	blank
29	content of bit 17 of assembled word; may be operation modifier. For 17-bit address instructions, the content of bit 17 is printed.
30	blank
31	alphabetic area relocation indicator P - subprogram area D - data area C - common area X - address is part of external string blank - absolute value, no relocation
32-36	5-digit octal value, either a relocatable word address or absolute value
37	blank
38	a. content of bits 16-15 of instruction which may be index designator b. content of bits 1-0, character position indicator for 17-bit address instructions
39-40	blank
Constants	
25-32	8-digit octal representation of constants assembled into 24-bit words

Printer Columns	Content
EQU	
30-34	5-digit octal relocatable address assigned to location term of EQU pseudo instruction
All Lines	
40-119	80-column image of source subprogram code line

LISTING CONTROL

The programmer may control COMPASS output listings with the following pseudo instructions. The listing control pseudo instructions are written as any COMPASS format instruction.

REM	insert remarks
NOLIST	suppress output listing
LIST	resume output listing
SPACE	space lines on output listing
EJECT	eject printer paper to top of next page
TITLE	begin succeeding pages with title given
asterisk (card column one)	print card columns 2-80 as a comment

REM

<any> REM <any>

Remarks may be inserted into the source program to appear on the output listing with this pseudo instruction. All fields except columns 9 to 13 of the operation code field may be used for remarks.

```
      THIS IS REM A REMARK PSEUDO-INSTRUCTION
```

NOLIST

<blank> NOLIST <blank>

This instruction suppresses listing of the subprogram until the pseudo instruction LIST appears in the source program. Regardless of NOLIST, lines in the source program containing errors will be listed. The location and address fields are ignored by COMPASS. The pseudo instruction will not appear on the output listing.

LIST

<blank> LIST <blank>
The pseudo instruction LIST resumes output listing after NOLIST has occurred. If LIST occurs without a preceding NOLIST, it is ignored. The pseudo instruction will not appear on the output listing.

SPACE

<blank> SPACE m
This pseudo instruction spaces the output listing; the designated number of lines are skipped on the printer paper. If as a result of SPACE, the printer begins a new page of output, printing resumes with the first line of the new page. SPACE instructs the printer to skip m lines or to the top of the next page, whichever is less. A symbol in the location field is ignored by COMPASS.

The parameter m, and unsigned decimal integer, designates the number of lines to be spaced. Zero through 32767 lines may be specified. However, the upper range would not exceed the number required to move the paper one page.

EJECT

<blank> EJECT <blank>
The pseudo instruction EJECT provides a more efficient way to move to the top of the next page. COMPASS will feed the current page through the printer and the line succeeding EJECT will be the first line of subprogram information on the new page. A symbol in the location field will be ignored by COMPASS. The address field must be blank or an error will result. Remarks may begin in column 41.

TITLE

<blank> TITLE <title>
The TITLE pseudo instruction describes a title to be printed at the top of each page of a listing.

If the first page of the listing is to be titled, TITLE must immediately follow IDENT. A symbol in the location field will be ignored by COMPASS. The contents of columns 20-72 of the address field contain the title. In the body of the subprogram, TITLE information replaces the present heading obtained from IDENT or preceding TITLE pseudo instructions. The first page following the TITLE pseudo instruction will have the new title. If the title is to head succeeding information, the pseudo instruction EJECT should be placed immediately following TITLE; the new page will have the inserted title.

COMMENT CARDS

When COMPASS detects a card with an asterisk in column one, the content of the card except for column one is printed on the output listing as a comment. No other action is performed.

COMPASS ERROR MESSAGES

If an error occurs in a line of code in a COMPASS source subprogram, an error flag is printed on the extreme left of the output listing. These flags indicate the field in which the error occurred and, in some instances, the exact error. The exact meaning of the flag depends on the particular instruction or pseudo instruction in which the error was detected. Generally, a field containing an error will be assembled as zeros. COMPASS provides nine error flags.

Error Flag

A Format error in address field.
Format errors are peculiar to each instruction.

C Attempt to assemble information into common.
The flag is given and the instructions are processed as if PRG was encountered.

D Duplicate Symbol.
The identical symbol has been used in more than one location field of the subprogram. The error flag is issued; the original definition the symbol holds the second and subsequent instructions using the symbol in the location field are assembled as if no symbol occurred.

F Full Symbol Table.
a. If the symbol in the location field would overflow the assembler symbol table, assembly proceeds after the F error flag is given; the F error will be issued for each additional location field symbol in the subprogram. All F flagged symbols are undefined and any reference to these symbols in address fields of the subprogram will yield a U error.

b. A COMPASS table other than the symbol table may produce this error flag but such instances will be rare.

L Location Field Error.
Illegal use of symbols in location fields yields an L error flag. The error may result from a symbol in the location field of an instruction which does not allow it, a missing symbol where one is required, or a format error such as imbedded blanks in a location symbol.

M Modifier Error.
Error in modifier subfield of operation code field.

O Operation Code Error.
An unrecognizable operation code occurred; the operation field is assembled as zeros.

U Undefined Symbol.
A symbol which appears in the address field of an instruction has not been defined in a location field. It may be missing from a legal location field or because of an F error.

T Truncation Error.
A symbol defined as a 17-bit character address is used in a subfield consisting of only 15 bits. The error is detected when the character address refers to character position 2, 3, or 4. The two least significant bits are truncated and the most significant 15 bits are used.

COMPASS CONTROL STATEMENT

To call COMPASS to assemble source subprograms the SCOPE $\begin{matrix}7\\9\end{matrix}$<library name> statement is used.

The COMPASS statement has the form:

$\begin{matrix}7\\9\end{matrix}$COMPASS, parameters

The statement has five optional parameters which may be expressed on the card; they are free field and are separated by commas. Parameters have the general form: option = logical unit number (u in the discussions given below).

The option must begin with a character I, P, X, L or R. Additional characters preceding the equal sign are ignored, thus L and LIST are the same parameter. If only the option is stated, COMPASS will make a standard assignment for the option.

PARAMETERS

INPUT = u
Source subprogram input unit; if the parameter or unit equation is absent, input from the standard input unit, INP, is assumed.

PUNCH = u
Punch option, u represents a logical unit number assigned to an output device. If the parameter is absent no punchable binary output will be produced. If only P appears, binary output will be produced on the standard punch unit, PUN.

XECUTE = u
Binary output for load-and-go option, u represents a logical unit number assigned to a magnetic tape unit. If the parameter is absent, no load-and-go tape will be written. If only X appears, binary output will be produced on the standard load-and-go unit, LGO.

LIST = u
List option, u represents a logical unit number assigned to an output device. If the parameter is absent, no listing will be produced. If only L appears, the listing will be produced on the standard output unit, OUT.

REFERENCE
If one of the COMPASS parameters is R, a symbol reference list is printed on the output listing. The symbol reference list in alphabetic order shows the symbol with its assigned address or value. The addresses are prefaced by an area relocation indicator.

SCOPE ORGANIZATION OF I/O 7

Under SCOPE, input/output devices are specified by logical unit numbers The logical units are organized according to function and the programmer or the operator assigns the logical unit to a particular type or unit of hardware through SCOPE control statements. Input/output operations may be independent of a particular configuration of hardware or type of input/output device. The logical unit may be assigned to any hardware unit, provided that equipment has the capability required by the logical unit.

Logical units are programmer units, scratch units, and system units. Programmer units are unrestricted as to use for any run in a job. Once defined, the definition of the programmer unit is fixed for the entire job. When a new job is encountered, all programmer units are released. Scratch units must be defined for each run and are released at the end of the run. Scratch units are assigned and used by library programs as required and may be used by the programmer.

System units are assigned to specific physical equitment within SCOPE but these assignments may be altered by the operator. System units are used for certain common functions; they are referenced by SCOPE and may be referenced by the programmer. When jobs are stacked, SCOPE protects system units from input/output requests which might destroy their contents or position the units to the detriment of the jobs. Protection is not provided when jobs are not stacked.

All logical units are referenced by a number, 1 through 63. Certain mnemonics are associated with the system units, they simplify discussion, but they are not interpreted by SCOPE.

At the option of the programmer, system units may be protected by SCOPE. However if protection is not desired, the programmer may so state in the JOB statement. Non-stacked jobs always have unprotected I/O. In the discussion of logical units, protection is assumed.

PROGRAMMER UNITS

These units are for general purpose use of the programmer. They are released at the end of the job unless saved by the programmer. Programmer units are numbered 1-49.

SCRATCH UNITS

Scratch units are for temporary use by the programmer. They are released at the end of the run. Scratch units are numbered 50-55.

SYSTEM UNITS

Selection of density on the following units, except LGO, is ignored when protection is active. The request containing the density statement is processed according to the usual procedures in all other respects.

Logical Unit Number	Mnemonic	Description
56	LGO	Load and Go. Object programs produced by assembly or compilation or transferred from another unit, may be stored on the LGO prior to loading and execution. LGO must be defined in each job. The unit is released after the object programs have been loaded; it may then be used as a scratch unit.
57	ACC	Accounting record. ACC is defined by installations keeping an accounting record. SCOPE does not use the unit.
58	CFO	Comment from operator. The operator communicates with SCOPE or other programs via CFO. Only READ requests are allowed for CFO. I/O protection assumes that CFO is the console typewriter, a buffered typewriter, or an on-line card reader. If CFO is equated to standard input, 60, protection is defined by that unit.
59	CTO	Comment to operator. The operator receives comments from SCOPE or other programs via CTO. Only WRITE requests are allowed on CTO. If CTO is equated to OUT, 61, protection is defined by that logical unit. CTO is assumed to be a console typewriter, a buffered typewriter, or an on-line printer; protection is based on this assumption.

Logical Unit Number	Mnemonic	Description
60	INP	Standard input. Protection for this logical unit prevents I/O requests which would destroy information or position the unit so that information pertinent to the current job would be lost. Specifically, if INP is assigned to magnetic tape, WRITE, REWIND, UNLOAD, SEFF, SEFB, WEOF and ERASE requests are prohibited. If the end-of-file status line is on, requests may be issued as shown:

Last Request	Request Allowed READ	READB	BKSP
READ	no	yes	yes
READB	yes	no	yes
BKSP	no	no	no

61	OUT	Standard output. OUT holds listable output from SCOPE and other programs. Protection is the same as specified for PUN below.
62	PUN	Standard punch. PUN receives punchable output from library programs such as COMPASS or FORTRAN. It may be used by programmers for other punch purposes. When a WRITE or ERASE request is issued for OUT or PUN, SCOPE checks for end of tape. When it is detected the reel is closed and the unit is unloaded. Closing a PUN reel consists of writing two end-of-file marks and the one-word BCD record, ER^^ . A message – "LOAD NEW XX" – is written on CTO. SCOPE waits for the unit to be reloaded. Protection for PUN and OUT assumes these units are assigned to magnetic tape. Only the programmer requests, WRITE, BKSP and ERASE are allowed; BKSP is allowed only following WRITE or ERASE.

INPUT/OUTPUT REQUESTS

Input/output requests are written in COMPASS programs as calling sequences for monitor routines controlled by a central input/output routine, CIO. CIO performs the following functions:

Selects an available channel.

Rejects a request if neither the unit nor any channel through which it may be accessed is available, or if an illegal function code is given.

Furnishes status for all requests.

Initiates input/output operations and returns control so that processing may continue while the operation is performed.

Responds to external interrupt and transfers control to a user specified interrupt subroutine.

The machine instructions for input/output operations should not be used when running a program under SCOPE.

CALLING SEQUENCE

Input/output operations are specified by entering an octal function code and other parameters into a calling sequence. The function code defines the operation.

Function Code	Request	Valid Unit References if Protection Selected
01	READ	1-58, 60, 63
02	WRITE	1-57, 59, 61, 62
03	READB	1-57, 60, 63
04	REWIND	1-57, 63
05	UNLOAD	1-57
06	BKSP	1-57, 60-63
07	SEFF	1-57, 63
10	SEFB	1-57, 63
11	WEOF	1-57
12	ERASE	1-57, 61, 62
13	STATUS	1-63
14	FORMAT	
	BCD	1-63
	Binary	1-63
	Low (200bpi)	1-56
	Medium (556bpi)	1-56
	High (800bpi)	1-56

The input/output operations are:

Function Code	Mnemonic†	Operation
01	READ	read n words starting at first word address
02	WRITE	write n words starting from first word address
03	READB	read n words backwards and store backwards starting at first word address + n - 1

The input/output operations are requested by the calling sequence:

L	RTJ	CIO
L+1	function code	logical unit, interrupt indicator
L+2	jump	reject address
L+3	mode	first word address
L+4		number of words
L+5		interrupt address
L+6	normal return	

The tape control operations are:

Function Code	Mnemonic†	Operation
04	REWIND	rewind
05	UNLOAD	unload
06	BKSP	backspace one record
07	SEFF	space forward past one end-of-file mark
10	SEFB	space backward past one end-of-file mark
11	WEOF	write end-of-file mark
12	ERASE	erase

† Mnemonics are used for discussion purposes only; they are not interpreted by SCOPE.

The tape control operations are requested by the calling sequence:

L	RTJ	CIO
L+1	function code	logical unit, interrupt indicator
L+2	jump	reject address
L+3		interrupt address
L+4	normal return	

The unit STATUS operation function code is 13. Unit STATUS is requested by the calling sequence:

L	RTJ	CIO
L+1	function	logical unit, dynamic flag
L+2	normal return	

When a STATUS request is made, SCOPE provides the status in the Q register and the unit condition in the A register.

The FORMAT selection function code is 14. Unit FORMAT selections are requested by the following calling sequence:

L	RTJ	CIO
L+1	14	logical unit, format code
L+2	jump	reject address
L+3	normal return	

PARAMETERS

CIO	must be declared as an external symbol in the source subprogram
Function code	octal number, 1-14, designating the function to be performed
Logical unit number	may be 1-63 depending on function code

Interrupt indicator — selects interrupt on normal or abnormal end of operation

- 0 — no interrupt
- 1 — interrupt on abnormal end of operation only, includes end-of-tape, end-of-file mark, load point, parity error, and lost data for magnetic tape and equivalent conditions for other hardware
- 2 or 3 — interrupt on end of operation, normal and abnormal

Mode — designates the recording mode in octal. If no mode is designated, binary mode is assumed and density is under control of the operator.

- 00 — do not select a new mode
- 40 — no density, even parity
- 41 — no density, odd parity
- 50 — low density, even parity
- 51 — low density, odd parity
- 60 — medium density, even parity
- 61 — medium density, odd parity
- 70 — high density, even parity
- 71 — high density, odd parity

First word address — symbolic word address of the first word in the input or output buffer area

Number of words — decimal number specifying the number of words to be transmitted

Reject address — symbolic word address; control returns to location L+2, if the request is rejected

Dynamic flag — if non-zero, the hardware unit is interrogated for status unconditionally (see STATUS requests)

Jump — any legitimate jump to the reject address

Interrupt — address of closed interrupt subroutine to which control is given when the specified interrupt occurs. If no interrupt is requested the normal return is written in location L+5 for function codes 1-3, and in location L+3 for function codes 4-12.

Code — tape format selection, any not listed below are illegal

1	BCD	4	Medium
2	Binary	5	High
3	Low		

NORMAL RETURNS

SCOPE initiates the operation specified by the calling sequence and returns control to the normal return so that processing can continue while the operation is performed. When control is given to the normal return, the Q register contains the status of the logical unit.

REJECT

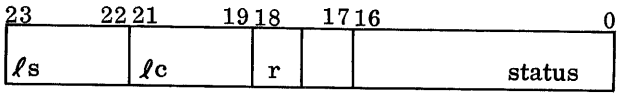
An input/output or tape control request can be rejected because the unit is unavailable, no channel is available, or the function code is illegal for the type of equipment. Upon reject for unit or channel unavailability, the A register is zero; for an illegal function code, the A register is non-zero. For any reject return, the Q register contains status of the unit.

INTERRUPT

If an interrupt address is specified and the interrupt indicator is non-zero, control transfers to the interrupt address at the end of the operation or upon abnormal condition interrupt. Before giving control to the interrupt address, SCOPE saves the A, Q and three index registers; and it enters the current condition and status of the unit in the A and Q registers. Control transfers to the interrupt address by a return jump. This address usually contains an unconditional jump. The programmer transfers control to SCOPE from the interrupt routine by returning through linkage established by the return jump. Upon regaining control, SCOPE restores the A, Q and index registers and returns control to the running program.

UNIT STATUS REPLIES

When a STATUS request is made or when control transfers to normal, reject, or interrupt address, the Q register contains information on the status of the unit.



ℓs — indicates logical status

00 — for STATUS request: unit is static; channel available

for reject return: hardware reject

for normal return (non-STATUS): unit is dynamic

01 — channel is not available for I/O or STATUS request processing

10 — previous operation is incomplete

11 — previous operation is complete but unit interrupt 15 is required for executing user interrupt request

ℓc — is the last channel to which unit was connected. It may or may not still be connected to this channel.

status of unit (as shown in the table) – If the unit is dynamic; current unit status at the time of the request is given. If the unit is static it reflects the results of the last completed operation.

r — is the retention code key obtained from first record of a labeled tape.

1 — tape contains a retention code in the label indicating the contents of the tape should not be destroyed.

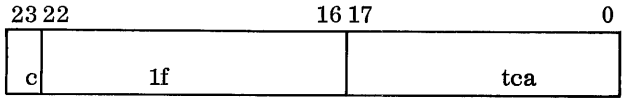
0 — tape may be used for output.

	STATUS BITS	MT (322X or 362X)	CR (3248/405)	CP (3245)	PR (1612)	PT (3691)	TY (3692)
XXX1	00	Ready	Ready	Ready	Ready	Ready	Ready
XXX2	01	Busy	Busy	Busy		Busy	Busy
XXX4	02	Write Enable					
XX1X	03	File Mark	EOF				
XX2X	04	Load Point					
XX4X	05	EOT	Hopper Empty			Tape Supply Low	
X1XX	06	Density (1=Med) (0=Low)					
X2XX	07	Density (1=High)					
X4XX	08	Lost Data	Fail to Read (06)	Fail to Feed			
1XXX	09	End of Operation					
2XXX	10	Parity Error	Reader Error	Compare Error			Parity Error
4XXX	11	Binary Mode	Binary Card (02)	Binary Mode		Binary Mode	
1XXXX	12		Stacker Full or Jammed (04)				
2XXXX	13	Int. Due, Ready or Busy (07)	Int. Due, Ready or Busy		Int.Due Ready, Not Busy		
4XXXX	14	(08) Int. Due to EOO	Int. Due to EOO		Int. Due to EOO		
1XXXXX	15	(09) Int. Due to EOO	Int.Due to Abnormal EOO		Int. Due to Abnormal EOO		
	16						
	17						

STATUS TABLE

UNIT CONDITION REPLIES

When a STATUS request is made or when control transfers to the normal interrupt address, the A register indicates the current condition of the unit.



c		is the condition
	1	unit is static
	0	unit is dynamic
ℓf		is the last function code (other than STATUS, 13) given for this unit.
tca		is the termination character address of data transmission contained in the Buffer Control Register.

INPUT/OUTPUT CONTROL

CIO controls the input/output channels. A channel is released for further operations as soon as possible; and hardware which can be connected to more than one channel is used to the best advantage.

Read, write, and read backward operations require the channel throughout execution. When an input/output operation is initiated the channel becomes busy. When an operation is completed, the channel is free and the unit status is recorded. If the user requests an end of operation interrupt; control passes to the specified interrupt subroutine; or if only an abnormal end of operation interrupt is requested, control transfers to the interrupt routine when an abnormal condition occurs.

TAPE CONTROL

The direction of tape motion after a backspace request depends upon whether the last read operation was read or read backward. If backspace is given after read backward, the tape moves forward one record. Other motion requests indicate true direction of tape motion and are not affected by read backward.

Tape control operations require the channel only during initiation of the function and do not cause the channel to be busy. If the user requests an interrupt at the end of a tape control operation, however, CIO considers the channel to be busy until the interrupt occurs.

STATUS

Status checking does not involve any action on the unit. It may require use of a channel to interrogate the unit if the unit is dynamic or dynamic STATUS is requested.

The unit is busy when it is performing an input/output or tape control operation. It is also considered to be busy if a tape control operation without interrupt has been initiated but its completion cannot be determined because all available channels have become busy through other operations. Status of the unit is given with respect to the user's ability to reference it.

For input/output operations, a unit is not busy after interrupt occurs. For tape control operations, a unit may be determined to be not busy during execution of a status request if a channel is available to sense the equipment status.

When status is requested, the current status is given if the unit is busy. When the unit is not busy, the status reply contains information about the operation last completed.

A user may demand that the unit be interrogated by setting the dynamic flag in a STATUS request. The current status is returned to the user. If a channel is available, the unit is connected (if not still connected as a result of a previous operation), and the unit status is copied. If no channel is available only the logical status is provided.

CAUTION: If the unit must be connected to obtain its status, the status lines carrying lost data signals are cleared by the hardware when a connect is issued.

EXAMPLES of input/output:

```
            EXT     CIO
             .       .
             .       .
             .       .
            RTJ     CIO

            01      12, 3

LOCATION    UJP     REJECT1

                    READAREA

                    9

                    INTROUT

    NEXT
```

CIO must be declared as an external symbol. A read operation (function code 01) is to be performed on logical unit 12. Nine words are to be read starting at READAREA. After the operation is initiated, control transfers to the normal return at NEXT and the A and Q registers contain information about logical unit 12.

If the unit or channel is unavailable or the function code is not valid, control transfers to LOCATION which contains an unconditional jump to REJECT1. When unit or channel is unavailable, the A register is zero. When the function code is illegal, the A register contains the function code, or 77777 if an attempt was made to use a 00 function code. The Q register contains the current condition of logical unit 12. If the request is honored, control transfers to the interrupt subroutine at INTROUT when the data transfer is completed. The A, Q and index registers are saved. At that time, updated current status for logical unit 12 is available in register Q and the condition in register A. A new mode is not selected.

```
          EXT     CIO
           .       .
           .       .
           .       .
          RTJ     CIO

          4       20

LOC       UJP     REJECT 4

GOON
```

CIO is declared as an external symbol. Rewind (function code 04) is initiated on logical unit 20. Since no interrupt is chosen (interrupt indicator is zero), no interrupt address is specified and the channel is available for another operation once rewind is initiated. After rewind is initiated, control transfers to the normal return at GOON and the Q register contains the status of logical unit 20. For reject, control returns to LOC; the A register contains zero for unavailable unit or channel and non-zero for illegal function code, the Q register contains the updated status of logical unit 20.

STATUS Example:

```
          EXT     CIO
           .       .
           .       .
           .       .
          RTJ     CIO

          13      25

          CONTINUE
```

STATUS (function code 13) is requested for logical unit 25. The updated status for logical unit 25 is in the Q register; the current condition of unit 25 is in the A register. Control returns to CONTINUE.

REJECTED I/O REQUESTS

Any input/output request except STATUS can be rejected. Conditions which return control to the reject address are the following:

illegal function code

illegal format request

channel busy

unit busy or not ready

hardware reject

interrupt not processed

The updated status words are in the A and Q register when the return is to the reject address. On a normal return, the registers contain the status of the unit prior to honoring the request.

When the reject occurs the programmer must have considered the particular unit in planning remedial action. The hardware requires different action depending on type and model. The remedial action must take this into account. The response to a reject may consist of several actions:

ignore the request and allow the program to proceed

take alternate action such as simulation of the request

instruct the operator to action that would enable the request to be executed, i.e., make the unit ready

terminate the program

TYPES OF REJECTS

On the return to the reject address the A register is a flag. If A contains a non-zero value, the reject was caused by an illegal function code and the illegal code is in A. If the illegal function code was zero, A contains 77777_8.

If A is zero the program must examine the ℓs bits of the UST word in Q.

ℓs	Meaning
00_2	Hardware reject. Any of the conditions noted for external rejects in the appropriate hardware reference manual occurred. Generally, the unit is (busy)*, which state is defined by the unit. It is a matter of timing and the type of indication supplied by the unit. This return usually indicates a hardware malfunction or disturbance of the control information used by EIO.
01_2	Channel busy. The channel is busy but the unit is free. The channel may be in use by another unit or SCOPE/CIO has been given inaccurate or inadequate information. The tables may have been altered by a program or the operator.
10_2	Unit busy or not ready. The channel is free but the unit is busy. The unit may be busy because some previous request is still in process. For example, the final data transmission is not complete or a rewind or backspace is in process. The unit may be not ready which may require operator intervention depending on the unit involved.
11_2	Both unit and channel busy. This reject occurs when the programmer has specified an interrupt subroutine for a previous operation on the same unit, and the interrupt subroutine has not been entered. The interrupt is available but the interrupt system has been disabled by entry to CIO or by the user. The only recovery is to enable the interrupt system or to terminate the job.

SCOPE CONTROL STATEMENTS 8

SCOPE is directed toward efficient control of hardware and programs by programmer and operator. SCOPE control statements specify the services for program execution; they allow the user to establish the peripheral equipment required and to call and operate library programs, or to load relocatable binary programs and operate them. SCOPE provides diagnostic routines and error checking.

SCOPE control statements consist of a statement name or mnemonic and parameters necessary to define the operation. Control statements may be read in sequence from the Standard Input unit, or may be serially presented by the operator. The operator enters them via the Comment From Operator unit which is usually the console typewriter. If a control statement has specific parameters and all are expressed, the final parameter may be terminated by a comma and comments placed following the parameter list. The restricted field control statements are:

SEQUENCE

JOB

XFER

LOAD

ENDSCOPE

SNAP

Examples:

$^{7}_{9}$SEQUENCE, 426, COMMENT

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

$^{7}_{9}$XFER, 20, THIS IS A COMMENT

SCOPE CONTROL CARDS

SCOPE control cards contain a 7, 9 punch in column one; there must be no other punches in column one. Columns 2 through 80 contain Hollerith information or blanks. The first information in the card must be the statement name; parameters are separated by commas. Except for these restrictions the card is free field.

Operator control statements are allowed once per job. Programmer control statements are inserted at appropriate points in the job deck. A flow diagram at the back of this manual shows the mandatory sequence of control statements in a programmer deck. In certain instances, functions obtained by various control statements are shown. The major purpose of the diagram is to show the proper position of control statements in programmer decks.

INPUT DECK STRUCTURES

Job input to SCOPE is on the standard input unit, INP. Assuming that INP is magnetic tape and that jobs are stacked, the input would consist of job decks each preceded by a SEQUENCE statement and terminated by an end-of-file. The SEQUENCE statement must be followed immediately by a JOB statement. A stack of jobs on INP appears as follows:

$^{7}_{9}$SEQUENCE, j

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

control statements

programmer decks

control statements

$^{7}_{9}$RUN, t, NM

data

EOF

$^{7}_{9}$SEQUENCE, . . .

$^{7}_{9}$JOB, . . .

:

EOF

$^{7}_{9}$SEQUENCE, . . .

$^{7}_{9}$JOB, . . .

:

EOF

$^{7}_{9}$SEQUENCE, . . .

$^{7}_{9}$JOB, . . .
:
:
EOF

$^{7}_{9}$ENDREEL (SCOPE will request next reel)

EOF

or

$^{7}_{9}$ENDSCOPE (no additional reels for INP exist)

EOF

The first statement on INP must be SEQUENCE. If JOB is not the next statement, the job is terminated and SCOPE moves INP until an end-of-file is detected. The following statement must be SEQUENCE, ENDSCOPE, or ENDREEL. If JOB follows SEQUENCE, SCOPE proceeds normally.

When a SEQUENCE statement is detected, SCOPE activates CFO to receive operator control statements.

Any of the statements listed below may follow the JOB statement:

EQUIP	UNLOAD
CTO	<library name>
XFER	<loader card>
REWIND	LOAD

Following LOAD and loader card, these statements only are allowed:

CTO	OCC
REWIND	SNAP
UNLOAD	RUN

SEQUENCE

$^{7}_{9}$SEQUENCE, j

The sequence statement assigns a number, j, from 1 to 999, to the job which it precedes. The SEQUENCE card is normally supplied by the computer operating facility; it is mandatory before each JOB card. An end-of-file must precede each SEQUENCE statement except the first on INP.

When the SEQUENCE statement is detected, the card is logged on CTO. End-of-file is written on OUT and PUN, if it is assigned, and all programmer units from the last job are released. Entry to the installation accounting routine completes the last job. The parameter j, is placed into ACCOUNTS. SCOPE requests operator control statements via CFO and processes them. The parameter, j, is the number referenced by the operator control statement, SEQUENCE.

Example:

```
EOF
7/9SEQUENCE, 4
7/9JOB, . . .
  .
  .
  .
EOF
7/9SEQUENCE, 88
7/9JOB, . . .
  .
  .
  .
EOF
7/9SEQUENCE, 3
7/9JOB, . . .
  .
  .
  .
```

JOB

$^{7}_{9}$JOB, c, i, t, NS / or, / ND / NP

- c — charge number, 0-8 characters
- i — programmer identification, 0-4 characters
- t — job time limit in minutes for entire job

  This includes idle time for setup as well as running time. The maximum time for job depends on the installation accounting routine.

NS indicates a single non-stacked job. If chosen, NP is implied. All system tape units are rewound and unloaded, making all I/O units available to the programmer.

NP suppresses system I/O protection

ND suppresses post-execution dump

Jobs submitted for processing under SCOPE require a JOB statement which supplies information to the installation accounting routine, identifies the programmer, and sets a job processing time limit. The JOB card must be immediately preceded by a SEQUENCE statement.

The c, i and t fields are mandatory. If one of the fields is blank, the comma delineating the field must appear, otherwise the job is terminated. The JOB card is written on OUT; and if the job is part of a stack, on CTO also.

If a job is terminated abnormally, a post execution dump of all non-system memory is written in octal on the standard output unit, unless ND appears on JOB card.

ENDSCOPE

7_9ENDSCOPE

The ENDSCOPE statement indicates that a SCOPE run is to be terminated. When ENDSCOPE appears on INP, the statement should follow the end-of-file terminating the last job of the stack. The card is logged on CTO, INP is unloaded. LIB is rewound and entry to the installation accounting routine is enabled to close the file. Finally, a double end-of-file and one BCD word of $ER_{\wedge\wedge}$ are written on OUT and OUT is unloaded. If PUN is assigned, it is treated in the same manner. If ACC differs from OUT or PUN, it is handled by the installation accounting routine.

When all action is completed, the computer halts. ENDSCOPE may be entered from CFO to terminate a SCOPE run arbitrarily.

ENDREEL

7_9ENDREEL

The ENDREEL statement terminates a reel of magnetic tape containing a job stack. Normally this statement is placed in the job stack by the operator during card to tape operations when preparing a tape to be used as INP. SCOPE requires an end-of-file immediately before and immediately after ENDREEL.

SCOPE accepts ENDREEL under the same condition in which it would accept SEQUENCE or ENDSCOPE. When ENDREEL is encountered on INP, SCOPE prints a message on CTO requesting the operator to mount the next reel of INP and halts the computer until the operator take action.

Example of stacked input on magnetic tape:

```
7/9SEQUENCE, . . .

7/9JOB, . . .

      job deck

EOF

7/9SEQUENCE, . . .

7/9JOB, . . .
   .
   .
   .
EOF

7/9ENDREEL

EOF
```

CTO

$^{7}_{9}$CTO, data

The programmer may provide instructions and messages to the operator with the CTO statement. The programmer may add Hollerith data following CTO. CTO cards may appear anywhere in a deck except next to the statements SEQUENCE, ENDSCOPE and ENDREEL, or between a SEQUENCE and a JOB statement. The CTO statement is logged on CTO and OUT.

REWIND

$^{7}_{9}$REWIND, u_1, u_2, . . . , u_i

The REWIND statement rewinds a magnetic tape to load point on logical unit u, 1 to 56.

This statement is copied on OUT and CTO. If u is not 1 to 56, or not magnetic tape, the request is ignored for that unit and the remainder of the statement is processed. Unit assignments are not altered.

UNLOAD

$^{7}_{9}$UNLOAD, u_1, u_2, u_3, . . . , u_i

Logical units, u, 1 to 56, may be unloaded by the programmer. The UNLOAD statement is handled in the same way as REWIND except that the unit is unloaded after it is rewound.

EQUIP

$^{7}_{9}$EQUIP, $x_1 = d_1$, $x_2 = d_2$, . . . , $x_i = d_i$

The EQUIP statement may be given prior to loading a program for a run. Refer to the SCOPE flow chart. It includes declarations, d, regarding logical units x_i (1 to 56).

HARDWARE DEFINITION

$^{7}_{9}$EQUIP, x = hh

This statement defines hardware type, hh, for a logical unit, x;

Mnemonic	Hardware Type
MT	magnetic tape
CR	card reader
PR	printer
CP	card punch
TY	console typewriter
PT	paper tape station

The designated logical unit is assigned to an available equipment of the specified hardware type. If hardware of the designated type is not available, a diagnostic will result. The job is terminated.

EQUATE LOGICAL UNITS

$^{7}_{9}$EQUIP, $x_1 = x_2$

Logical units, x_i, are equated by this statement. A system unit (57-63) may not be specified in the left side of the statement; if it is, the job will be terminated.

PHYSICAL UNIT ASSIGNMENT

$^{7}_{9}$EQUIP, x=hhCcEeUuu

c channel number 0-7, prefixed by C

e equipment number (controller), prefixed by E

uu unit number (device), prefixed by U

This form of the EQUIP statement offers the options shown. Blank spaces indicate parameters which may be omitted.

hh	Cc	Ee	Uuu
X			
X	X		
X	X	X	
X	X	X	X
	X	X	X
	X	X	

Use of a non-existent channel, equipment, or unit number will cause a diagnostic to be printed and the job to be terminated.

If more than one logical unit is to be assigned to the same I/O device, the first logical unit must be assigned to the device and the logical units equated. To assign PUN and LGO to the same physical tape the following statements could be used.

$^{7}_{9}$EQUIP, 62=MT C0EV03

$^{7}_{9}$EQUIP, 56=62

$^{7}_{9}$EQUIP, 62=MTC0EV03, 56=62 is also legal.

XFER

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

The logical unit number u, (1-56) may be undefined or defined as magnetic tape. If u is defined as equipment other than magnetic tape, a message is written on CTO and OUT, the job is terminated, and SCOPE proceeds to the next SEQUENCE statement on INP. If u is undefined, it is assigned to the first available magnetic tape, the assignment is logged on CTO and the system waits for the operator to continue.

SCOPE transfers information following the XFER statement and preceding the next control statement, from INP to a magnetic tape. The records are written in odd parity. SCOPE writes an end-of-file mark and backspaces over it when the succeeding SCOPE control statement is encountered.

Programmer data cards may be transferred from INP to another tape unit using XFER. A card or card image with a 7, 9 punch and no other punches in column one terminates the XFER operation; control is maintained by testing for a SCOPE control card.

Binary object subprograms may be transferred from INP to another tape unit using XFER. If this unit is a programmer or scratch unit, it must be rewound before it can be loaded.

Example:

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

data

$^{7}_{9}$SCOPE control statement

The binary information on INP is written in odd parity on logical unit 25 followed by an end-of-file mark.

LOAD

$^{7}_{9}$LOAD, u_1, u_2, u_3

The logical units, u_i, are logical unit numbers 1-56, previously defined by an EQUIP statement for this run. If omitted, the loader attempts to load from INP, 60. No more than three units may be specified.

This statement calls the loader to load binary subprograms into memory from programmer units, scratch units, LGO or INP. Only one LOAD statement may appear per run.

An end-of-file terminates loading from each unit. SCOPE loads from the units in the order in which they appear and then loads binary information from INP, if any exists. Since the loader operates only once per run, the LOAD statement must precede any binary object subprograms to be loaded from INP for the same run. Multiple files may be loaded from the same unit by repeating the unit designation in the LOAD statement.

If 56 (LGO), is specified as one of the three unit declarations, it will be rewound by SCOPE before the loader is entered. If programs are loaded from LGO, it will be released at the end of the run or the programmer may unload and save it. CTO statements may be used to inform the operator of proper handling. If LGO is not to be saved, it may be rewound and used as a scratch unit by the loaded program.

With or without a LOAD statement, a binary object subprogram on the standard input unit is loaded into memory when encountered by the monitor system, unless it is preceded by a control statement such as XFER, which specifies some other processing. All binary program cards before the next SCOPE control statement are loaded. If there is a LOAD statement for the same run, it must precede the binary object subprograms on INP.

Examples:

$^{7}_{9}$LOAD, 56, 3, 25

LGO, 56, is rewound by SCOPE and subprograms are loaded until an end-of-file mark is encountered. Subprograms are loaded from units 3 and 25 to an end of file.

LGO may be rewound and used as a scratch unit if it is not to be saved. Any binary subprograms on INP preceding a SCOPE control card are loaded following the tape loading operation.

$^{7}_{9}$LOAD, 2

Logical unit 2 is loaded until an end-of-file mark. Logical unit 2 is not moved except by loading. INP is loaded until a SCOPE control card is encountered.

$^{7}_{9}$LOAD

Binary subprograms on INP are loaded until a SCOPE control card is encountered. This use of LOAD is optional; the subprogram would be loaded if this statement did not appear.

Typical job deck with XFER:

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

$^{7}_{9}$EQUIP, $u_1 = d_1$, $u_2 = d_2$, . . . , $u_n = d_n$

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

binary object subprograms

data cards

$^{7}_{9}$COMPASS, assembly options

source subprograms
FINIS

$^{7}_{9}$REWIND, u_1, u_2, . . . , u_i

$^{7}_{9}$LOAD, u_1, u_2, u_3

binary object subprogram

$^{7}_{9}$RUN, t, NM

LIBRARY NAME STATEMENT

$^{7}_{9}$<library name>, parameters

<library name>	the program is in file two of LIB
parameters	Parameters to the called library program which are passed to the system by SCOPE.

Library programs may be called, loaded and executed by the programmer using this statement. The <library name> is the entry point to a system on the library tape such as COMPASS, FORTRAN, or COBOL. User programs may be placed on LIB and called in the same manner.

Example:

$^{7}_{9}$COMPASS, I, P, X, L, R

LOADER CARD AS CONTROL STATEMENT

The programmer may place binary subprogram cards in the appropriate place in his deck. These cards are placed on INP. No LOAD statement is required. When SCOPE detects a binary subprogram card, recognized by punches in at least one of the rows 3, 2, 1, 0, 11, 12 of column 1, the loader is called and reads INP until a SCOPE control card is encountered. Refer to Appendix A

RUN

$^{7}_{9}$RUN t, NM

t is the execution time limit in minutes, $0 \leq t \leq 999$. It is entered into the ACCOUNTS table for use by an installation accounting routine. It is not used by SCOPE. If omitted maximum time is assumed.

NM suppress the memory map that would otherwise be written on OUT. The map which lists memory allocations of a loaded program is written on OUT prior to execution of the program.

The RUN statement initiates program execution by transferring control to the object program in memory. The RUN statement is required for all programmer runs.

Example:

$^{7}_{9}$RUN, 2

the execution time limit is estimated 2 minutes. The memory map will be written on OUT.

The RUN statement follows the relocatable binary decks if the program is on INP or the LOAD statement if the program is on an input tape other than INP. The RUN control card is copied on OUT and CTO.

If the program runs successfully and the job is stacked, the next run (if any) in the job is processed. If the run is unsuccessful, the entire job is terminated. Non-stacked jobs may contain only one RUN statement. Non-stacked jobs end with an appropriate message and the computer halts.

OCC

$^{7}_{9}$OCC, location, octal correction, . . . , octal correction.

Octal corrections may be made to binary subprograms after loading. The OCC statements may be used to define and enter corrections or additions to subprograms by establishing a program extension area.

The program extension area is created after the subprogram area. Corrections to subprograms referring to the extension area, or instructions to be stored in the extension area may not be submitted until all subprograms have been loaded.

The parameters are free field. If the optional period is used to terminate the card, comments may follow it.

Location	Meaning
program name k	Corrections on this OCC card are loaded beginning with location k in the named subprogram.
Dk	Corrections are loaded beginning with location k in the Data area.
Xk	first occurrence: Define a program extension area of length k. Corrections on the first card with X in the location field are ignored. subsequent occurrences: Corrections are loaded beginning with location k of the program extension area.
+k	Continuation OCC cards. +k is an increment from the last location plus one corrected by the previous OCC statement.

Octal corrections, up to 8 octal digits, follow the location term and are set off by commas. Blanks may be included. Leading zeros may be omitted. They contain an octal value of up to 8 digits or an octal value and a relocation factor.

Each value is stored right justified in successive computer words. If a value of less than 8 digits is supplied, the computer word is zero filled.

Octal Correction	Meaning
octal correction	The correction replaces the contents of the memory location determined by the location field on the card and the position of this octal correction field on the card.
blank or contigous commas	Do not alter the location

octal correction relocation factor	Replaces the contents of memory determined by the location stated on the card and the position of this field on the card. The address portion of the octal correction is to be positvely relocated as dictated by the relocation factor.
octal correction minus sign relocation factor	The same as above except relocation is decremental
octal correction relocation factor C	The same as above except 17-bit arithmetic is applied to the address. C must follow all octal corrections containing character addresses to be relocated.
octal correction minus sign relocation factor C	Decremental character relocation.

Relocation Factor	Meaning
no relocation factor	Octal correction is to be stored as an absolute correction.
(Subprogram name)	Relocate the word address portion of the octal correction relative to the address of the first location in the subprogram named within the parenthesis.
D	Relocate the word address portion of the octal correction relative to the DATA area assigned for this run.
C	Relocate the word address portion of the octal correction relative to the COMMON area assigned for this run.
X	A program extension area has been assigned for this run and the word address portion of the octal correction must be relocated relative to the extension area.
*	Relocate the word address portion of the octal correction relative to the last subprogram named in any field of any preceding OCC or SNAP statements.
relocation factor C	If C follows any relocation factor, perform character address arithmetic (17 bits).

SNAP

SCOPE provides selective memory dumps during execution.

After a program has been loaded and before the RUN statement is encountered, the programmer may provide SNAP dump parameters on SNAP cards. The SNAPSHOT routine must have been loaded. This is accomplished by naming SNAPSHOT on an EXS loader control card. (See the loader section.) The SNAP statement provides parameters defining the instruction where the dump is to be taken, the area of memory to be dumped, and the dump format.

Addresses for the SNAP statement have the same general format as the OCC statement. OCC statements and SNAP statements are processed after loading and prior to the RUN statement. Conflicts between OCC statements and SNAP statements cannot occur if all SNAP statements are placed behind all OCC statements. This matter is discussed in greater detail in the section on debugging.

SNAP calling sequences are prepared from the SNAP statement. Each SNAP calling sequence consumes 7 locations in available memory.

An error in a SNAP statement causes the statement to be ignored. Execution is never inhibited as the result of an error in a SNAP statement. The format is described below.

$^{7}_{9}$SNAP, location, beginning address, ending address, mode, identification, comments

or

$^{7}_{9}$SNAP, k_c, b, e, m, id

Each parameter in a SNAP statement is separated by commas. Program names are always enclosed by parentheses in a SNAP statement.

The SNAP statement defines a location containing an instruction which will be replaced by a jump to a SNAPSHOT calling sequence. Additional parameters define the beginning and ending addresses of the area to be dumped, the mode of the dumps, and a dump identification. After the dump has been taken, the instruction originally replaced is executed and control returns to the program at the succeeding location.

Entries in location fields of SNAP statements are listed below:

Location	Meaning
<subprogram name>k	Replace location k of the subprogram with a return jump to a calling sequence to be generated.
Xk	The jump to a SNAP calling sequence is to replace the instruction in location k of the program extension area defined for this run.
Dk	The jump to a SNAP calling sequence is to replace instruction k in the DATA area.

The beginning address and ending address parameter are expressed in the same general manner. The ending address must always be greater than the starting address or the SNAP statement is ignored.

Address	Meaning
Dk	Dump begins or ends with location k in the DATA area.
Ck	Dump begins or ends with location k in the COMMON area.
Xk	Dump begins or ends with location k in the program extension area.
<subprogram name>k	Dump begins or ends with location k in the subprogram which had subprogram name in its IDC card.
k*	Dump begins or ends with location k of the subprogram last named in any field of any OCC or SNAP statement.

The dump may be in one of three formats and may include the register file or not.

Mode	Meaning
O	Print dump in octal
C	Print dump as 6-bit characters
F	Print dump in floating point format
R	Print register file in octal. R may be combined with any of the other options as OR, RC, etc. Refer to the section on debugging.

Identification	Meaning
	Zero through four BCD characters will be printed on the SNAP output to identify the dump.

OPERATOR CONTROL OF SCOPE 9

SCOPE yields control to the operator once per job as shown in the flow diagram. Immediately after a SEQUENCE card has been read and processed, SCOPE interrogates the CFO for operator control statements. These statements may consist of equipment changes or other manipulation of the I/O environment, the calling and execution of library programs, priority selection of a particular job for immediate processing, or termination of the SCOPE run.

An operator may manually interrupt SCOPE during job processing. If the user has not selected a manual interrupt and the operator presses the manual interrupt button, the current job is terminated and control is given to the operator after SCOPE has positioned INP past the next SEQUENCE statement.

OPERATOR CONTROL STATEMENTS

The operator may issue control statements to SCOPE through CFO which is usually a console typewriter. These statements consist of the statement name followed by commas and parameters if required. The statement is terminated by pressing the CLEAR or FINISH buttons. When the operator types a period and presses CLEAR, SCOPE begins processing the next job.

SEQUENCE

SEQUENCE	no parameters
or	
SEQUENCE, j	j is the sequence number of any job on INP.

With the SEQUENCE statement, the operator selects a job to be processed. SCOPE interrogates CFO for statements after a SEQUENCE card has been read from INP and logged on CTO. The operator may control job sequencing as follows.

To process the job with the sequence number printed on CTO, the operator types a period after issuing any other necessary statements, and return to SCOPE without using the SEQUENCE statement.

To repeat the last job processed, the operator types SEQUENCE, last job sequence number. SCOPE will backspace INP, for magnetic tape, and repeat the last job.

The operator may select any job on INP for immediate processing using the statement:

SEQUENCE, job sequence number.

The operator may issue SEQUENCE with no parameter on CFO; the job whose sequence number was logged on CTO will be skipped; and INP will be positioned to the next SEQUENCE statement on INP.

If the desired job is known to exist on INP in front of the job whose sequence number was last logged on CTO, and INP is magnetic tape, the operator may use the REWIND statement to rewind INP, 60, before issuing the SEQUENCE statement.

ENDSCOPE

ENDSCOPE

The operator terminates the SCOPE run with this statement. The action taken by SCOPE is identical to that for the programmer control statement ENDSCOPE.

REWIND AND UNLOAD

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

As many logical units, u_i, may be expressed as required. These statements for control of magnetic tape cause SCOPE to take the same action as described for the programmer control statements, REWIND and UNLOAD. The operator may refer to any logical unit, 1-63, without restriction.

AET

The operator interrogates or alters the available equipment table using the AET operator control statement (Appendix). This statement has three forms:

AET	When the operator uses AET without commas or parameters, the content of the entire table is printed on CTO in octal.
AET, a	$1 \leq a \leq 50_{10}$. When the a parameter is used, only entry a in the table is printed. †
AET, a, P	$1 \leq a \leq 50_{10}$. P may have two forms. This version of the AET statement alters entry a in the table as indicated by P.

†The length of the AET table, which may not be greater than 50_{10}, is established by each installation.

P, UP	set the s field, bit 17, in the AET word to operable.
P, DOWN	set the s field to inoperable.
P, RES	set the unit reserved for another computer.
P, FREE	clear the unit for use by this computer.
P	is an octal integer of 11 or 16 digits. Replace entry a in AET with the octal value. The octal digits are stored left justified into the AET entry. The driver address, the D field, may be specified by the octal integer or supplied by SCOPE or the LOADER.

CALL

CALL, library name, p_1, p_2, . . . p_n

The first parameter, library name, must be the name of an entry point in file two of LIB. The following parameters are not used by SCOPE but are passed to the called program. p_1, p_2, p_3, . . . , p_n vary according to the called program and the order and content are dictated by the library program.

The operator may call programs from LIB for execution using the CALL statement. When the CALL statement is encountered, protection of system I/O is voided, the message "OPERATOR" is written on OUT and a blank card (or card image) is written on PUN if it is assigned. Recovery dumps are suppressed. The called program is then operated as any library run. SCOPE requests additional statements via CFO when the library program has been executed.

The name used as a parameter in the CALL statement must be an entry point name defined in a program in file two of LIB.

EQUIP

The operator alters equipment assignments when SCOPE interrogates CFO for operator control statements. The operator uses the EQUIP, x=d statement in the same manner as does the programmer. The operator may reference any logical unit (1-63). The operators' designation of the EQUIP statement overrides loader or system assignment of I/O on LIB.

SCOPE informs the operator of the sequence of job processing, requests information and produces error messages on CTO. The operator is also advised of actions required for SCOPE to perform particular tasks.

AUTOLOAD

After SCOPE has been autoloaded, the operator must supply information to initialize the system. After SCOPE types a message, the operator replies with the requested information or acknowledges the message by pressing the FINISH button. If the CFO is not the console typewriter, a blank record signals completed action.

Scope Message	Operator Message
DATE	dd mm yy, press FINISH
TIME	hh cc, press FINISH
OUT	AET ordinal of OUT or INP, press FINISH. If the assignment for OUT or INP is not to be changed, it is only necessary to press FINISH.

dd day

mm month

yy year

hh hour, $00\text{-}24_{10}$

cc minutes, $00\text{-}60_{10}$

If date and time are not given, zero will be assigned by SCOPE.

SEQUENCE PRINT

SEQUENCE, decimal number. When a SEQUENCE card is encountered on INP it is logged on CTO. The TYPELOAD indicator will light. SCOPE waits for an operator control statement.

PROGRESS REPORTS

Messages appear on CTO to advise the operator about the job in process which do not require any action from the operator. These messages may enable the operator to perform such tasks as reloading tapes, removing output from the printer or a job deck from the card reader. The following programmer control statements are logged.

REWIND

UNLOAD

RUN

CTO

JOB, account number, name, time

When the ENDSCOPE statement is detected on INP or CFO, SCOPE reports this function on CTO.

When SCOPE encounters a CTO, comment to operator, statement on INP the statement is logged on CTO.

REPEAT MESSAGE

SCOPE will request the operator to repeat any message which it cannot decipher; the message is assumed to be in error. SCOPE will log REPEAT on CTO.

READY MESSAGE

After the unit assignments are logged, SCOPE outputs: READY? on CTO and waits for the operator to complete any required action. The operator responds when ready to process the job by pressing FINISH.

END CALL MESSAGE

When a library program is called via the operator CALL statement, the successful operation of the called program is reported on CTO as: END CALL.

Abnormal termination of the called program produces the message: ABNORMAL END CALL. The operator may issue additional control statements on CFO.

NON-STACKED JOB TERMINATION

When SCOPE has completed processing a non-stacked job, the operator is notified by one of two messages on CTO.

NORMAL END	run was successful
ABNORMAL END	run was unsuccessful

The operator must reload the system units and follow the autoload procedure to continue.

REPORT ON EQUIPMENT ASSIGNMENT

SCOPE will log equipment assignments on CTO

ll = HW Cc Ee Uuu

ll = logical unit number

HW = hardware type

c = channel

e = equipment

uu = unit

I/O UNIT NOT READY

If an assigned input/output unit is not ready when SCOPE requires it, this message is printed on CTO:

READY LUN logical unit number

SCOPE awaits operator action on the unit. When ready to proceed, the operator presses FINISH.

FAULTY I/O ASSIGNMENT

If SCOPE cannot execute an input/output requirement, this message will be logged on CTO and the computer halts.

CANNOT DRIVE logical unit number

The operator checks the physical unit to which the logical unit is assigned for operability and appropriate assignment, (i. e., INP is not the line printer if 60 were involved). To restart the computer the operator must press GO.

DATA TRANSMISSION FAULT

When control cards are read from INP, parity errors produce this message on CTO.

INP ERR

SCOPE logs this for operator information, and accepts the information.

PROTECTED TAPE

If a file-protected magnetic tape is designated for output, the computer awaits operator action after SCOPE has output a message on CTO.

ENABLE WRITE logical unit number

The operator must load a new tape or put the write ring in place, and then press FINISH.

END OF TAPE

If end of tape is reached on a logical unit, SCOPE reports:

LOAD NEW logical unit number

The computer waits for operator action. When the operator has loaded the new tape, he may continue by pressing FINISH.

INPUT/OUTPUT ERROR

If an input/output unit fails during operation, SCOPE prints a message on CTO:

I/O ERR error code LUN logical unit number

Error Codes, 2 characters

character 1	character 2
C connect	P parity error during I/O instruction transfer
S select	I internal reject
	E external reject
	Q undefined condition caused reject

UU undefined logical unit reference attempted

2 digits – protected unit violation

01	undefined unit
02	hardware reject of connect code not used
03	not used
04	protected function on LIB unit
05	protected function on PUN or OUT
06	protected function on INP
07	protected function on CTO
08	protected function on CFO
09	EOF detected and protected function on INP
10	hardware reject of WEOF or Unload on PUN or OUT after an EOT was detected

AET ERROR

If illegal entries are discovered in the AET table, the message AET ERR is logged on CTO and the computer halts. The error may be due to conflicting entries in AET or a program or hardware malfunction. The operator may initiate a simulated autoload of SCOPE by pressing GO. If the error indication persists, the operator should refer the problem to the system programmer who maintains LIB or the customer engineer.

ORGANIZATION OF MEMORY 10

The computer memory is organized by SCOPE into system memory and available memory. System memory contains those portions of SCOPE and input/output routines required for SCOPE during a job. This includes resident, which is always in memory, and routines selected by options such as system unit protection, recovery dump, and CIO routines for non-standard system I/O units. Resident occupies low memory and the other SCOPE I/O drivers reside in high memory.

AVAILABLE MEMORY ORGANIZATION

Available memory is organized by the loader from parameters received from resident. The contents of memory during SCOPE and loader operation are significant to the programmer only because the area used by SCOPE and loader is the area used for common storage; consequently, information may not be prestored in common during loading.

Available memory may be used by a program to be loaded and executed under SCOPE supervision. It may be a library program such as COMPASS, FORTRAN, COBOL, ALGOL, SORT or PRELIB; or it may be a group of subprograms for a programmer run.

Available memory is divided into three areas which must be considered when coding programs in the source language and preparing binary subprograms for input by loader. A fourth area, the program extension area, is defined if octal corrections add instructions to a subprogram. The four areas which must be considered are:

Subprogram area

Data area

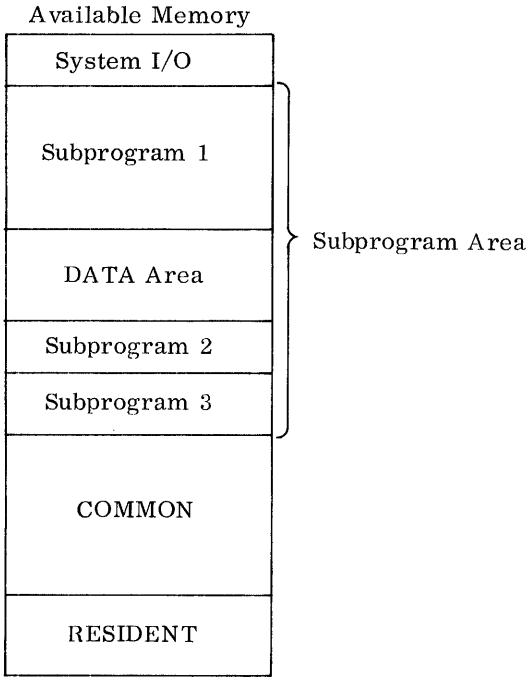
Common area

Program Extension area

The assignments for these areas are identified on a memory MAP which may be printed following loading. During loading, assignment and allocation of space to these areas is dynamic. The system performs the assignment using parameters, contained in the program decks, which were derived during assembly or compilation. SNAP statements consume available memory but do not appear on MAP. Program overlays are assigned to memory according to requirements discussed in the section on overlays.

SUBPROGRAM AREA The subprogram area for a run is defined by parameters obtained from the IDC cards from all of the subprograms loaded for the run. The total subprogram area is equal to the sum of all the subprogram lengths specified on all of the IDC cards encountered during loading for the run.

DATA AREA The data area is defined once per run by parameters obtained from the first IDC card encountered with a non zero length for a data area. The data area, when defined, is shared by all subprograms. The data area is outside the first subprogram which declares it; but if more than one subprogram is loaded, it is bounded by the subprogram area.



Since the data area is defined only once per run, the first subprogram loaded must reserve sufficient space to accommodate any data area used by any subprograms loaded for the run.

Generally, information will be stored into the data area by the loader. The absolute starting address of the data area is the same for all subprograms during the run. Therefore, in assembling instructions or constants for storage into the data area during loading, the programmer must consider all subprograms to be loaded. The programmer must guarantee the proper placement of information in the data area in the source subprogram. This may be accomplished in COMPASS through the pseudo instructions ORGR and BSS. Library routines may reference the same data area as all other programs. The programmer must consider this when defining and using the data area in the source subprogram.

For example, two subprograms, BAKER and GEORGE are to be loaded and executed at the same time. Program BAKER is loaded first. If program BAKER uses 100_8 locations in the data area and GEORGE uses 50_8 locations, the structure of the source programs might be:

```
IDENT     BAKER                    or          DATA
  .                                              .
  .                                              .
DATA                                             .
BSS 150B                                   100(8) locations of data
ORGR 0                                           .
  .                                              .
                                                 .
                                           BSS 50B
PRG                                        PRG
  .
  .
  .
END
IDENT     GEORGE
  .
  .
  .
DATA                                       DATA
BSS 150B                   or              ORGR 100B
ORGR 100B                                        .
  .                                              .
  .                                        50 locations of data
                                                 .
                                                 .
PRG                                        PRG
  .
  .
  .
END
```

The DATA area length declared in the IDC card of subprogram BAKER includes the area required by GEORGE.

The DATA area may also be organized as follows:

```
          IDENT     ABLE
            .
            .
            .
          DATA
BUFA      BCD       20,      this is the ABLE buffer
          BSS       20       space reserved for BAKER
          BSS       20       space reserved for CHAS
          PRG
            .
            .
            .
          END
```

```
                IDENT     BAKER
                  :
                DATA
or ORG 20       BSS       20          space reserved for ABLE
  BUFB          BCD       20,         this is the BAKER buffer
                BSS       20          space reserved for CHAS
                PRG
                  :
                END

                IDENT     CHAS
                  :
                DATA
                BSS       20          space reserved for ABLE
or ORG 20
                BSS       20          space reserved for BAKER
  BUFC          BCD       20,         this is the CHAS buffer
                PRG
                  :
                END
```

COMMON AREA

The common area is shared by all subprograms. Each time an IDC card is encountered, the length of the common area is examined. If the presently defined area has fewer locations than declared on the current IDC card, the length of common is extended to include the subprogram. The final length defined for common will be the greatest length declared on the IDC card of any subprogram loaded for the run.

Common has the same absolute starting address throughout the run. The programmer must guarantee the integrity of information in common during program execution to insure that information needed by one subprogram is not destroyed by another.

The area of available memory destined to be the common area during a run is occupied by the loader and loader symbol table prior to execution; information cannot be stored in the common area during loading. Compilers and assemblers for Control Data computers will not allow the production of object subprograms which would prestore common.

PROGRAM EXTENSION

The program extension area is defined by SCOPE from programmer OCC statements processed after loading. It follows the subprogram area. Rules pertaining to program extension areas are described in the section on debugging.

ASSIGNMENT OF AVAILABLE MEMORY

During loading, memory is allocated to the subprogram area, the data area, and the common area. Initially, all of non-system memory is available.

When the first IDC card is encountered, the subprogram and common areas are assigned for that subprogram and the data area if of non-zero length is assigned for an entire run.

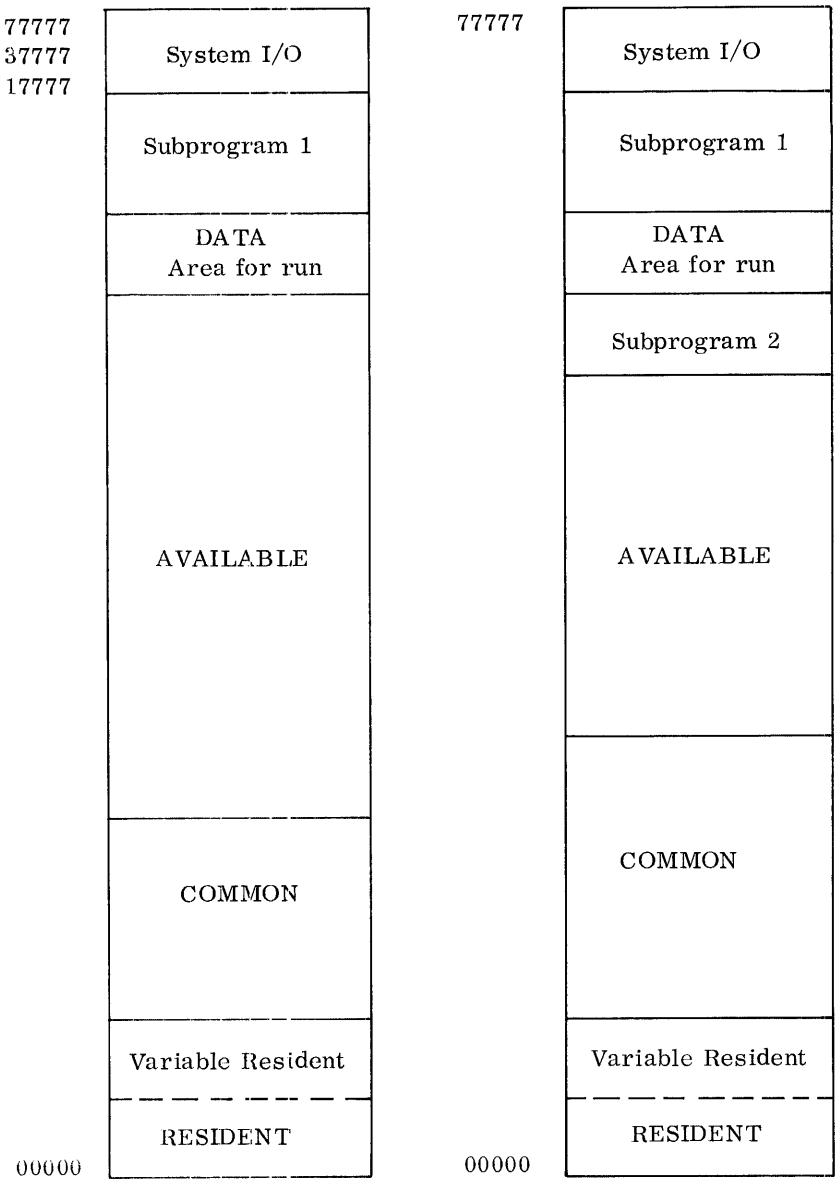
When the second subprogram is loaded into memory, the definition of a larger common will extend the length of common, but the data area will remain unchanged regardless of the length declared after the data area is defined. An error message will result if the data area declared in the card cannot be contained in the area defined for the run.

RELOCATABILITY

Addresses assigned by the compilers and assemblers are called relocatable; they do not identify actual addresses in the computer memory. They may be considered as sequence numbers or reference points. Relocatable addresses are relative to the beginning of the subprogram.

During coding in symbolic language the programmer has little interest in the ultimate assignment of absolute memory locations for use by his program. He organizes the program and dictates certain relationships between instructions and data. The compilers and assemblers record the relationships and produce relocatable object subprograms.

In a relocatable subprogram, the first location is given a sequence number or starting location of zero. Each successive instruction or data word in the subprogram is assigned an address one greater than its predecessor. By applying the same increment or decrement to these relocatable addresses or sequence



numbers, they may be assigned to any memory location desired during loading. It is important to retain the relationships established (or detected) during assembly or compilation.

By treating subprograms as unified blocks, and maintaining a constant interval between the internal parts, it is possible to leave the ultimate assignment of memory to the loader. Entry points and external symbols allow the loader to establish the correct linkage between subprograms.

Data and common storage are treated in a manner similar to the subprogram storage. The first location assumed by the assemblers and compilers is zero and the areas are addressed relatively thereafter. The loader relocates the addresses; the assemblers and compilers provide a separate relocation factor for each object program area.

RELOCATION OF SUBPROGRAMS

Assembler and compiler will determine which relocation factor is to be applied during loading. The loader determines which relocation factors to use by word count, or for RIF cards, by a relocation byte.

subprogram increment

common area increment

data area increment

subprogram decrement

common area decrement

data area decrement

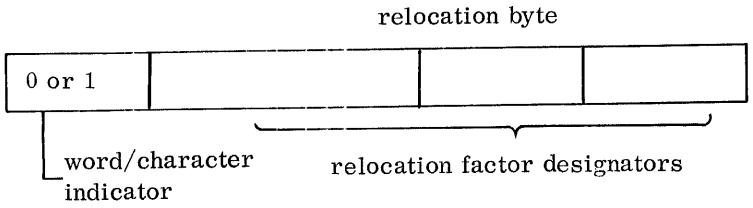
X, extension area increment (specified by programmer)

RELOCATION BYTES

The relocation byte determines whether the address in the word on the card is to be incremented or decremented, which area is involved, and whether to perform 15-bit or 17-bit arithmetic on the address.

RELOCATION BYTE OPTION

The relocation byte consists of 4 bits.



0 15-bit arithmetic
1 17-bit arithmetic

Relocation Byte	Relocation Factor
x000	not used; this code constitutes an error
x001	no relocation (absolute)
x010	subprogram increment
x011	common block increment
x100	data block increment
x101	subprogram decrement
x110	common block decrement
x111	data block decrement

SCOPE BINARY SUBPROGRAM 11

SCOPE provides a modular loader for relocatable binary object subprograms assembled or compiled by COMPASS, FORTRAN, COBOL, ALGOL and certain elements of the BASIC software package. The loader also prepares overlays from relocatable binary subprogram decks. The loader is called by SCOPE when a library name statement, a LOAD statement or a loader card is encountered on INP.

For user programmers the loader provides the following services:

Loads relocatable binary subprograms.

Establishes communication and linkage between independently assembled and/or compiled subprograms.

Loads and links library routines called by loaded subprograms.

Loads I/O drivers required for program execution from information supplied by the programmer or operator and stored in system tables.

Loads and links BCD and floating point simulator routines as required, without user action.

Prepares error messages and diagnostics for errors detected in the loader input.

Accumulates checksums for the binary information and compares this checksum against one in the binary cards to guarantee accurate loading of subprograms.

When loader is placed in control, it accepts both loader control cards and binary subprogram decks as input.

Both loader cards and loader control cards have 7, 9 punches and punches in rows 3-12 of column one. Rows 3-12 are called the word count field. The loader recognizes a card by the contents of this field.

LOADER CONTROL CARDS

Loader control cards provide specific information for the loader or direct the loader in processing the binary subprogram decks. Of the five loader control cards, three control preparation of overlay programs and two control linkage or input-output. Except for column one, which is binary, loader control cards contain symbolic information in 12-bit Hollerith.

Name	Function	Octal Word Count
MAIN	main overlay program	50
OVERLAY	overlay program element	51
SEGMENT	overlay program segment	52
EXS	external symbol declaration	55
LED	loader equipment declaration	54

These cards may appear at any position in the loader input. Sequence is important only when the success of an operation depends on the prior appearance of the control card. For example, if overlays are to be processed, MAIN must be the first card in the loader input.

LOADER CARDS

Name	Function	Octal Word Count
IDC	subprogram identification card	41
EPT	entry point name card	42
RIF	relocatable information card	1-40
XNL	external name card	43
LRL	local reference list card	45
TRA	transfer address card	44

OBJECT SUBPROGRAM STRUCTURE

To load subprograms, the loader must find an IDC card to obtain the length of subprograms, data and common areas for the subprogram.

The first card in a subprogram deck must be an IDC card, the last, a TRA card. If several subprograms are loaded, the TRA card for one subprogram should be followed immediately by the IDC card of the succeeding subprogram; LED and EXS cards, however may intervene except in file two of LIB. Overlay control cards always precede an IDC card.

Between the IDC and TRA card, any other loader cards may appear. The loader will accept these cards in any sequence but correct subprogram linkage and executability of the loaded program depends on a particular sequence within a subprogram. The descriptions of the loader cards, EPT, XNL, LRL, provide specific details.

TYPICAL LOADER INPUT STRUCTURE

A relocatable binary subprogram deck prepared by a compiler or assembler is in correct sequence for loading. If this sequence is disturbed, the following structure is reasonable for accurate loading:

IDC card

all EPT cards

all RIF cards

all LRL cards

all XNL cards

TRA card

Should two subprogram decks become intermixed, the decks are useless and the subprograms should be reassembled or recompiled.

LOADING LIBRARY ROUTINES

After all programmer decks are loaded, the loader searches file two of LIB to match external symbols with entry point names. The search begins from the current position of LIB. The loader scans LIB until an IDC card is found; the EPT cards which follow are examined. If a desired entry point name is found, the routine is loaded. If a non EPT card is found before a desired entry point name is detected, the loader skips to the next IDC card on LIB before searching for another entry point name.

Loading of library routines terminates when there are no undefined entry point names in the loader symbol table or when LIB has been rewound and searched to the end of file two without a routine being loaded.

CHECKSUM IN BINARY DECKS

The loader cards contain a checksum. With the single exception of the TRA card, the checksum pertains to all the information in the card except that in columns 3 and 4, the checksum field. The TRA card contains the checksum for all other loader cards encountered since the last IDC card except the TRA card.

Checksum errors result when the loader derived checksum does not compare bit for bit with the checksum in the card. This discrepancy may be due either to failures in hardware or the checksum may be in error. If the data in the card is correct and the checksum is in error, ignore the checksum.

If the ignore checksum field, row 8 of column 1, contains a punch, the checksum on the card is ignored, and no comparison is attempted. If the card is a TRA, the subprogram checksum is ignored. When any card originally produced in a subprogram deck has a punch in the ignore checksum field, the TRA card must have an ignore checksum punch as well. When a TRA card is inserted in a

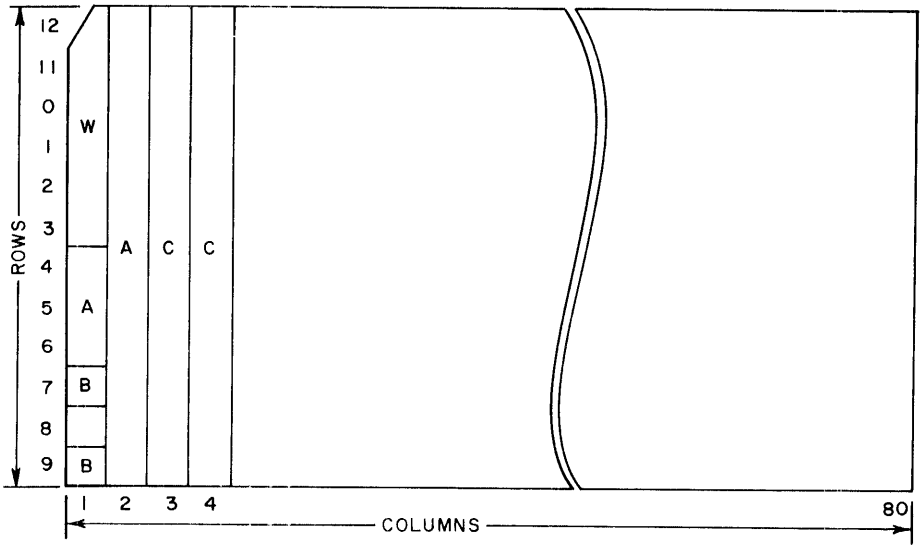
subprogram deck to facilitate loading several subprograms for execution, the ignore checksum field should always be punched.

LOADER CARD FORMAT

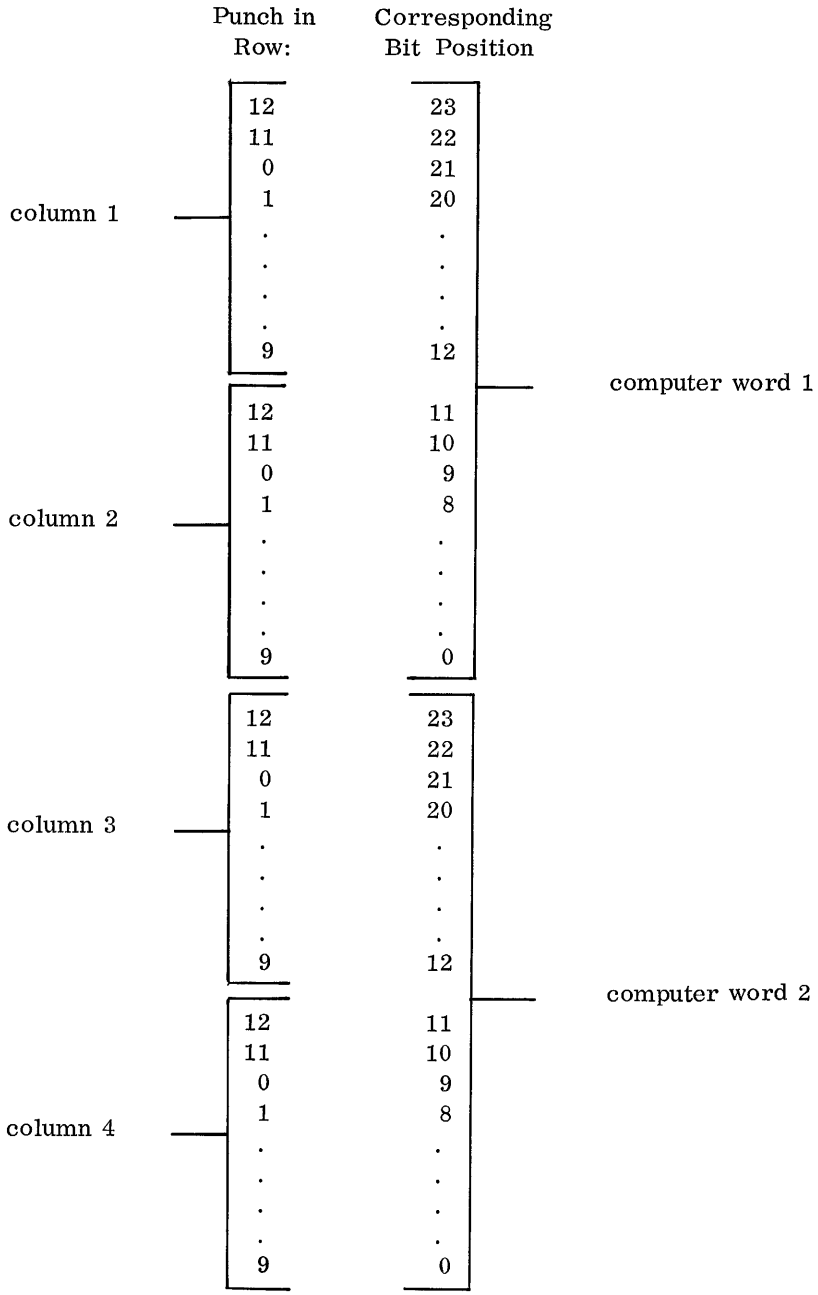
The SCOPE loader accepts cards with punches in rows 7 and 9 of column 1 and certain specific punches in rows 12, 11, 0, 1, 2, 3 of column 1.

The first four columns are the same for all loader cards. The first column identifies the card; the second provides an address or other information about storage; the third and fourth hold a checksum.

Mnemonic	Card Column	Rows	Computer Word Bit Position	Purpose
W	1	12, 11, 0, 1, 2, 3	23-18	word count (not zero)
B	1	7, 9	14 and 12	binary card indication
I	1	8	13	I = 1, checksum ignored I = 0, checksum must compare
A	1	4, 5, 6	17-15	relocatable address, sequence number or program length
	2	12, 11, 0, 1-9,	11-0	
C	3	12, 11, 0 1-9	23-12	24-bit checksum
	4	12, 11, 0, 1-9	11-0	



Two card columns of 12 bits each are contained in one 24-bit computer word. The content of row 12 of card column 1 is in bit 23 of the first computer word, row 9 of card column 1 is in bit 12, row 12 of card column 2 is in bit 11 of the first computer word and row 9 is in bit zero. If there is a punch in the card the corresponding bit position contains a one.



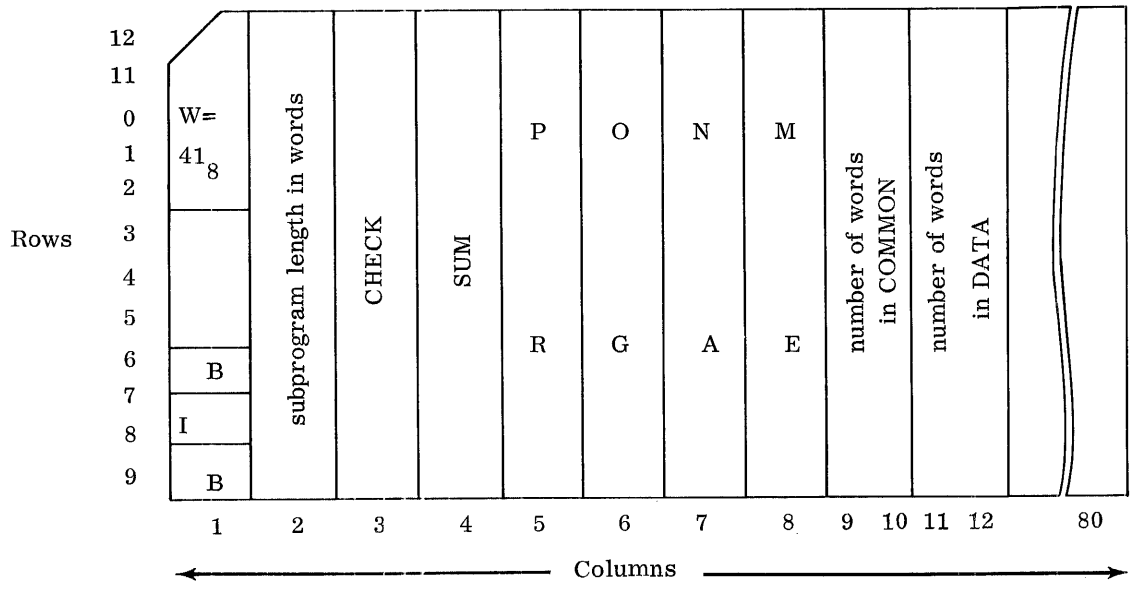
IDC CARD

The subprogram identification card is a binary card which identifies and provides information about the subprogram to the loader. The IDC card must be the first loader card in a subprogram deck.

The IDC card has an arbitrary word count of 41_8 and states the name of the subprogram, its length, and the sizes of the common area and the data area in words. The first subprogram loaded which specifies a non-zero data area defines the data area for the entire run. It is important to structure the loader input so that the data area is properly defined to accommodate all subprograms and information to be stored in the data area during loading and execution.

An IDC card is produced by an assembler or compiler. The name in the card is taken from the program name in a source language statement such as the IDENT card in the COMPASS language.

Card Columns	Computer Words	Meaning
1-2	1	$W = 41_8$, Subprogram length in words
3-4	2	C = checksum
5-8	3-4	subprogram name in 6-bit BCD†
9-10	5	common block length in words
11-12	6	data block length in words
13-80	7-40	unused



†The name is eight characters or less, left adjusted with trailing blanks added.

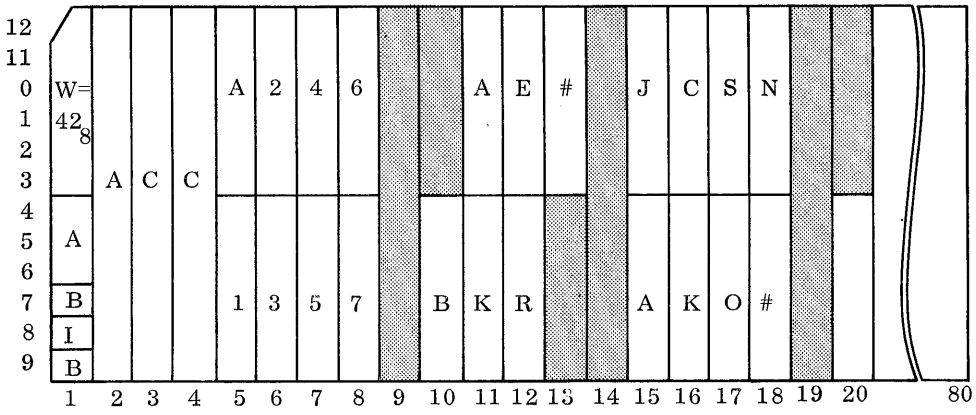
EPT CARD

Entry points in subprograms are declared by source language statements such as ENTRY in COMPASS: EPT cards result in relocatable binary decks. EPT card has an arbitrary word count of 42_8; it contains one or more entry point names and the equivalent relocatable addresses. Names are in 6-bit internal BCD and may be 8 characters or less. If names are less than 8 characters, a record mark, character code 72_8, follows the last character in the name. The name is followed by an 18-bit value; the rightmost 15 bits define the relocatable word address assigned to the entry point within the subprogram which will be relocated using the subprogram increment.

The A field contains a sequence number for programmer convenience; it is not used by the loader. An entry point name and its equivalent address must be wholly contained on a single card.

Card Columns	Computer Word	Meaning
1-2	1	$W = 42_8$. A = sequence number 1, 2, 3, ...
3-4	2	C = checksum
5-80	7-40	entry point names and locations

The EPT card shown contains three entry point addresses, A1234567, BAKER and JACKSON.



= record mark

b = blank

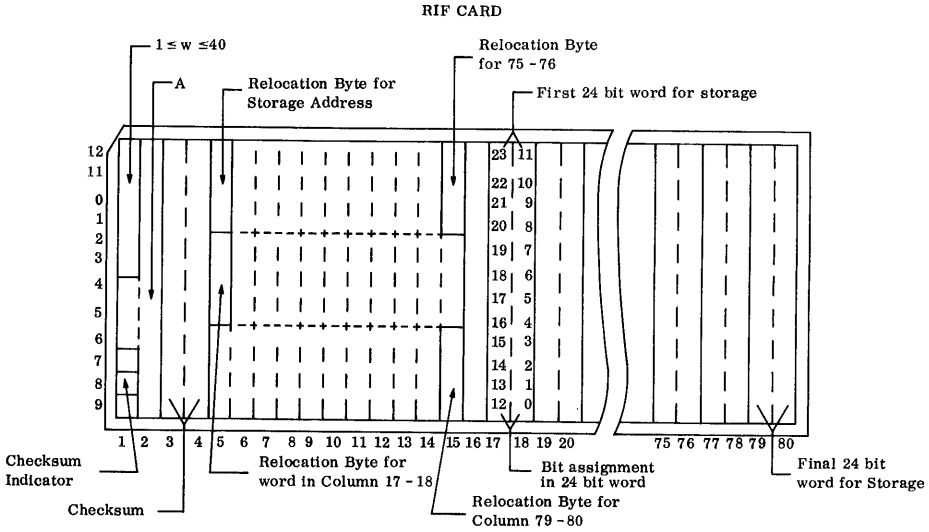
shaded portion - relocatable address

RIF CARD

Instructions and constants for storage in the subprogram and data areas are contained in relocatable binary information cards. The actual word count of an RIF card may vary from 1 to 40_8 depending on the number of instructions and constants in the card. The card is checksummed and has a relocatable load address which is always a 15-bit address.

The first instruction or constant in the RIF card is stored at the location referenced by the relocated word address. All other instructions and constants are stored in successive locations in memory. The load address may be in either the subprogram or data area and may have a related relocation byte of 0010_2 or 0100_2. The RIF card contains relocation bytes for the load address and each instruction and constant in the card.

Card Column	Computer Word	Meaning
1-2	1	W = 1 to 40_8; A = load address of first word.
3-4	2	C = checksum.
5-16	3-8	up to 33 relocation bytes; the first applies to load address and may have values 0010_2 or 0100_2. The other 32 relocation bytes apply to the address field of each machine word. Three bytes (12 bits) are unused. The first bit of each byte may be a 0 for a 15-bit address or 1 for a 17-bit address. The remaining three bits determine relocation factor.
17-80	9-40	may contain up to 32 words of data or machine language instructions to be loaded after the relocation factor is applied to the address portion of each word.



XNL CARD

Subprogram communication is established by the loader using information from the external name and linkage card. The XNL card has an arbitrary word count of 43_8 and is checksummed. The card holds symbolic names of 8 characters or less and a binary address related to the symbol. The symbol and address must be wholly contained on a single card. The symbolic name is that of an entry point in some other subprogram. The loader substitutes the address of the entry point whenever a reference is made in the subprogram to the external name. The XNL card results from source language statements such as the COMPASS pseudo instruction, EXT. The required entry point names may be in the permanent portion of the loader symbol table. They may be in EPT cards following the IDC card of a subprogram in file two of LIB, or they may be named in an EPT card of a subprogram loaded at the same time as the subprogram containing the XNL references.

The XNL symbol need not be referenced by the subprogram in which it is declared. For example, the declaration of SNAPSHOT would not be referenced until after the subprogram has been loaded and SNAP statements encountered. If no reference occurs, the related address in the card is always 77777_8. Any number of references with external symbols may occur in the subprogram. The compiler or assembler prepares a threaded list for the loader to use in establishing linkages. A symbol may be declared as external more than once in a single subprogram. The loader will link multiple declarations.

WORD AND CHARACTER EXTERNALS

An external symbol may be referenced from a 15-bit word or 17-bit character address. A character address is denoted when the three leading bits of the relocatable binary address are non-zero. Only the most significant 15 bits of a character address are filled by the 15-bit entry point address. The two low order bits remain unchanged.

An XNL card has the same general format as an EPT card.

Card Column	Computer Word	Meaning
1-2	1	W = 43_8, A = sequence number 1, 2, 3, . . .
3-4	2	C = checksum
5-80	3-40	External names and linkages

The name is 8 characters or less, if it is less than 8 characters, a record mark follows. Immediately following name or record mark are 18 bits of which the rightmost 15 specify the location of the instruction referencing the external symbol. If any of the three leading bits are non-zero the reference is from a 17-bit address, or string of 17-bit addresses. If several instructions reference the external symbol, a string is formulated and the address on the card provides the location of the first entry in the string. The address in the XNL card is the relocatable word address of the first instruction in the subprogram which references the XNL name. The address field of that instruction contains the relocatable address of the second instruction referencing the symbol, and so on, until the final reference. The address field of the last reference contains 77777_8 to terminate the string.

If no reference is made by an instruction in the subprogram, the address on XNL card will contain 77777_8. The number of XNL cards in a subprogram deck is not restricted.

The sequence of XNL cards in a subprogram deck is important if multiple references occur. An XNL card should not be the first card in a deck. If the external name is declared only once in the subprogram, the XNL card may appear anywhere between the IDC and TRA cards. If the external name appears more than once, (on two or more XNL cards) the string related to the first reference must be loaded before the XNL card containing the next declaration is encountered by the loader. Strings must not be interrupted by multiple declarations. A simple solution is to place all XNL cards immediately in front of the TRA card if the sequence established by the compiler or assembler has been disturbed.

LOCAL REFERENCE LIST CARD

The local reference list card is produced by an assembler or compiler, when a reference is made in the source program to a symbolic address not yet defined. The compiler or assembler builds a string of addresses at which a reference is made to the undefined address. This string is identical in construction to that used for external names. The LRL card is produced when the symbolic address is defined.

The LRL card has an arbitrary word count of 45_8 and is checksummed. The LRL card must follow RIF cards containing the string.

The LRL card contains the standard binary card information, a word/character flag, the relocatable subprogram address assigned to the local symbol which was initially undefined in the source program, the address of the first entry in the string and the length of the string list.

All addresses in LRL cards are relocated using the subprogram relocation increment.

LOCAL STRINGS

The LRL card and its related string are similar to the external name cards and the strings associated with the XNL cards. Since LRL cards, the string, and the definition of the address are contained wholly within a single subprogram, no symbol need be used. The first address in the LRL card is local to the subprogram for which a definition is to be made.

The first address is used similarly to the address in XNL cards following the the external names. This address is the relocatable address of the first location in the subprogram where a reference was made to the undefined local address. The second address is the relocatable subprogram address assigned when the symbol was defined during assembly or compilation.

The references to the second address constitute a threaded list or local reference string. The string is terminated when the string length count in the card reaches zero or when the string contains an address of 77777_8. If the zero length and the address of 77777_8 do not occur simultaneously, an error results and string processing is terminated.

Card Column	Computer Word	Meaning
1	1	standard binary card information, w = 45_8
2	1	word/character flag for address =0, word reference string ≠ 0, character reference string
3-4	2	24 bit checksum of LRL card
5-6	3	address of the string of references
7-8	4	relocatable subprogram address of first word of string in least significant 15 bits
9-10	5	number of references to address in the subprogram (length of string) in least significant 15 bits
11-80	6-40	not used

TRA CARD

The final card in a subprogram deck is a transfer card. It contains the subprogram checksum, but the information in the TRA card is not checksummed. The TRA card has an arbitrary word count of 44_8 and may contain a transfer symbol. The TRA card is mandatory for successful operation.

The transfer symbol designates the location of the first instruction to be executed under SCOPE. This starting address must be defined as an entry point. If the transfer symbol is defined as an entry point within the subprogram containing the TRA card, the assembler or compiler will punch the relocatable address in the A field of the TRA card.

When loading more than one subprogram during a run, several TRA cards will be encountered and it is possible to have two or more TRA cards terminating a subprogram deck. However, if more than two transfer symbols occur, or if no TRA card contains a transfer symbol, a loader error results; the execution of all subprograms loaded on the run is inhibited and the job is terminated. If execution of the program proceeds, SCOPE will pass control to the last transfer symbol encountered.

When subprograms are loaded from file two of LIB, the first TRA card encountered terminates loading of the subprogram. The loader will search LIB until an IDC card is found before loading can continue.

The above pertains to program execution initiated by the RUN statement. Programs called by the library program statement are initiated according to other rules.

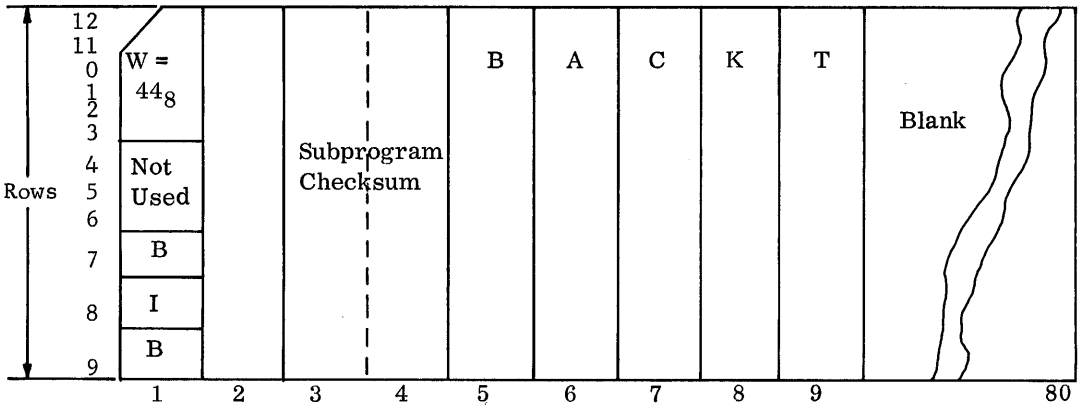
TRA FORMAT

The TRA card, which must be the last in a subprogram deck, contains the standard binary information in column 1 and has a special checksum. The checksum in columns 3 and 4 is a checksum of all of the other binary card checksums in the subprogram.

If the subprogram deck is changed by the addition, deletion or correction of checksummed cards or if the TRA card is altered to supply or delete transfer symbols, the "ignore checksum field", I, bit 8 of column 1, must be punched for correct loading and execution.

Card Columns	Computer Words	Meaning
1-2	1	W = 44_8. If TRA was produced by an assembler or compiler and the transfer symbol is defined within the subprogram, columns 1 and 2 contain the relocatable address assigned to the symbol. Otherwise, the A field is not used. Ignore checksum is frequently used.
3-4	2	subprogram checksum, refer to text
5-12	3-6	transfer symbol, 8 or less Hollerith characters terminated by a blank

TRA Card



The transfer symbol is BACKT.

The transfer symbol must appear in an EPT card loaded on the same run or appear in the permanent portion of the loader symbol tables. The transfer symbol is punched in Hollerith and may be punched on a keypunch.

LOADER CONTROL CARDS

Loader control cards provide information for the loader or direct the loader to perform special processing on the binary subprogram decks. Three of these cards control preparation of overlay programs and two control linkage and input/output.

Loader control cards are not checksummed and have an arbitrary word count of 50_8 or greater. Except for column 1, which is binary, loader control cards contain symbolic information in 12-bit Hollerith. Loader control cards may usually appear at any position in the loader input. The sequence is of importance only when the success of some operation depends on the prior appearance of the control card. An example is the use of LED in preparing overlays. The equipment declarations are time dependent during such processing.

EXS CONTROL CARD

External symbols may be declared in binary subprogram decks after assembly or compilation using external symbol cards. It is used to declare external symbols not declared in the source language, or to alter the linking of external symbols to entry points. The EXS card could be used to include SNAPSHOT or another debugging routine in memory for a run, eliminating the necessity of a source language declaration and simplifying the transition between debugging and production runs. An EXS card might also be used to equate several external symbols to a single entry point to facilitate testing of sections of programs.

EXS cards may appear in any position in the loader decks. On the LIB tape they may be placed only between the IDC and TRA cards for a subprogram. The EXS loader control card has an arbitrary word count of 55_8. Except for column one, the card is free field.

Card Column	Meaning
1	standard identification, 7/9 punches, $W_{\square} = 55_8$
2-8	two options:
	a. external symbol, external symbol, ... , external symbol
	b. external symbol, external symbol, ... , external symbol equal sign entry point name

EXS PARAMETER

When no equal sign appears, the external symbols are assumed to be entry point names in subprograms loaded for the run or entry point names of programs contained in file two of LIB. The external symbols are processed as if they came from XNL cards which had no related linkage strings. A program executed under SCOPE does not normally have access to the linkage information obtained by this use of EXS cards.

When one or more external symbols are equated to a single entry point, loading the program in which the single entry point name is defined will satisfy all of the external symbol declarations.

If an entry point name identical to an external symbol contained in such an EXS declaration appears after the EXS card, a duplicate symbol error will result. If the EXS declaration occurs during the loading process after the entry point name is defined, the normal relationship is overridden and the EXS declaration of equivalence prevails. However, the entry point name need not be defined prior to the EXS declaration.

LED CONTROL CARD

The loader equipment declaration card assigns logical units to specific hardware units or to hardware of a particular type. If a logical unit named in a LED card has been previously assigned, the LED declaration is ignored. Units assigned by LED cards need not be declared on EQUIP cards.

The LED card has an arbitrary word count of 54_8. Except for column 1, the card is in 12-bit Hollerith and it is free field. Fields are separated by commas. Any number of LED cards may appear in any position of the loader input except on LIB, LED cards must occur between an IDC and the first TRA following.

The LED card contains one or more hardware declarations. The last declaration must be wholly contained on the card.

Card Columns	Meaning
1	standard loader information, 7/9 punches $W = 54_8$

Each declaration may have the form:

LL hh or LL hh c e uu

LL	logical unit number $1 \leq LL \leq 56$
hh	hardware type, as encoded in AET
	01 magnetic tape
	02 card reader
	03 printer
	04 card punch
	05 typewriter
	06 paper tape station

c	channel number, 0 to 7
e	equipment number, 0 to 7
uu	unit designator, two octal digits

Imbedded blanks are ignored. The declaration LL hh means assigns the logical unit to an available unit of the designated type. Other assignments are specific as to type, controller and device. The last declaration must be wholly contained on the card.

LOADER ERRORS

The loader audits and evaluates input. Any error will inhibit execution of the loaded program. When errors are detected a message is written on OUT. If possible the card in error is identified. The general types of errors reported are:

checksum errors

format errors

symbolic address and linkage errors

deck and subprogram sequence errors

I/O errors due to faulty information, hardware failures or improper input formats

The loader detects most errors but it is necessary for the programmer to protect previously loaded information from destruction by information from subsequent RIF cards. If two RIF cards in a single subprogram overlap in storage, the loader will give no indication. Normally this will not occur unless the original deck sequence established by the compiler or assembler is altered.

It is also possible that information destined for storage in the data area from two subprograms could conflict so that information in one subprogram could be destroyed by information from subsequent subprograms. No error indication is given.

LOADER INPUT

The first card in a loader input deck is an IDC card, unless the input is for overlay preparation. In this case, the first card encountered must be a MAIN card. The general structure of the subprogram deck is:

IDC 41_8

EPT 42_8

RIF $1\text{-}40_8$

LRL 45_8

XNL 43_8

TRA 44_8

If a MAIN card is encountered during overlay preparation, a card sequence error, CS, will result. If loader encounters a SEGMENT card before an OVERLAY card a card sequence error, CS, results, the card is processed as if it were OVERLAY and execution is inhibited. If MAIN was not the first card and MAIN, OVERLAY or SEGMENT appear in the loader input, a card sequence error occurs and the card is unrecognizable.

The LRL card must appear after all RIF cards containing references to the string have been loaded. XNL cards need not occur in any particular sequence in the loader input unless the same symbol appears on more than one XNL card in the subprogram deck.

If the sequence of a single subprogram deck has changes from that established by the system that produced it, the following sequence may be used:

IDC

EPT

RIF

XNL

LRL

TRA

With the possible exception of RIF cards which are assigned to overlapping locations, this sequence will allow the subprogram to load correctly.

LOADER ERROR MESSAGES

The loader diagnoses certain errors during loading and prints error messages on OUT. The format of the message may vary depending on the type of error and the card format.

If the message includes the character ≠, a character code which occupied the corresponding position in the card could not be converted. If a message includes the term ≠UNCVRT≠, card columns 2 through 9 could not be converted to valid Hollerith information even though the word count indicates the card to be a loader control card.

The loader maintains a count of errors detected. SCOPE prints the error count on OUT. Any error inhibits execution of the loaded program.

If less than 63_{10} errors are detected, a true count is given.

If 64_{10} errors are detected, loading is terminated, SCOPE prints a 63_{10} count and the job is terminated.

LOADER ERROR CODES

Error Code/Name	Card Name	Word Count	Meaning
CF Card Format	TRA EXS	44_8 54_8	Illegal character on card
	MAIN OVERLAY SEGMENT	50_8 51_8 52_8	Incorrect or illegal punches in subfield
CS Card Sequence		0	Non-loader card has been read from a unit other than INP
	RIF EPT XNL LRL	$1\text{-}40_8$ 42_8 43_8 45_8	IDC card missing; no IDC card in front of loader cards following a TRA card
	TRA	44_8	A zero length subprogram on LIB
	varies		Unrecognizable binary card detected; card has non-zero word count; it may be an overlay control card if MAIN was not the first card in the deck
	IDC	41_8	IDC card detected which does not follow a TRA card. It is not the first IDC card in deck
	MAIN	50	Two MAIN cards in deck
	OVERLAY SEGMENT	51 52	a. Overlay control cards internal to subprogram b. Overlay control cards are contiguous. No subprogram decks intervened. A zero length element has been specified. c. A SEGMENT card was not preceded by an OVERLAY card. This SEGMENT card is treated as if it were OVERLAY. An error is counted to inhibit execution.

Error Code/Name	Card Name	Word Count	Meaning
CK Checksum Error	any loader card		Checksum error detected in a binary card
DS Duplicate Symbol	EPT	42_8	Entry point name appeared twice during loading
UD Undefined	XNL TRA EXS	43_8 44_8 54_8	Declared name is not a defined entry point in a loaded subprogram, in file two of LIB, or in the permanent portion of loader symbol table
TR Transfer Symbol	TRA	44_8	a. No TRA card contained a transfer symbol b. More than two TRA cards contained transfer symbols
RL Relocation Factor	RIF	$1\text{-}40_8$	a. Load address relocation byte is not 0010_2 or 0100_2 b. Load address relocation byte is 0100_2 and data area is undefined c. Data area is part of overlay element already on tape d. Relocation byte is zero
SE String Error			Count in LRL card and actual references detected by loader do not agree
SL String Loop	XNL	43_8	String of addresses has resulted in a loop
LX Equated Linkage Loop	EXS	54_8	External symbol equated to another defines a loop
MS Missing Subprogram			Entry point name does not exist in file two of LIB to correspond to library name statement
OV Memory Overflow			This error always terminates loading

Error Code/Name	Card Name	Word Count	Meaning
	IDC	41_8	Storage required for subprogram (common area or the data area, if this is first IDC) exceeds available memory
	OCC	00	During overlay processing, program extension area exceeds available memory
	Various		Symbol printed in error message could not be entered in loader symbol table without infringing allocated memory
			Space in memory does not allow library search. Loading is terminated
EOF End of File			Loader encounters end-of-file on INP. Loading is terminated.
EOT End of Tape			During overlay preparation a physical end-of-tape is detected on output tape for writing. Overlay processing is terminated
I/O Input/output Error			a. After five attempts to read a magnetic tape, loader unable to read the logical unit, number of which is stated in the error message b. Error on input unit which was not magnetic tape c. Unit specified in an overlay control card is not magnetic tape. Terminates loading and job d. An error occurred in writing a magnetic tape in overlay processing terminates loading and the job

ERROR MESSAGE FORMATS

Binary Card Error

PPPPPPPP	CC	WW	AAAAA

P - Subprogram Name
C - Error Code
W - Word Count
A - Card Address Field

Example:

PROGRAM1	RL	05	06421

Hollerith Card Error

PPPPPPPP	CC	WW	HHHHHHHH

P - Subprogram Name
C - Error Code
W - Card Type
H - Hollerith Columns 2-9

Examples:

PROGRAM1	CS	66	SPEC, 4, 2
PROGRAM1	CF	00	≠UNCVRT≠

This message indicates columns 2-9 of the card cannot be converted to valid Hollerith information

Symbol Error

PPPPPPPP	CC		SSSSSSSS

P - Subprogram Name
C - Error Code
S - Symbol

Examples:

PROGRAM1	DS	TAG4A
PROGRAM1	CF	TAG5M≠

If an unconvertible character is encountered in a Hollerith symbol, the CF diagnostic is printed with ≠ in position of the bad character

Miscellaneous Errors

PPPPPPPP	CCC	XX

P - Subprogram Name

C - Error Code

X - Other Number (if necessary)

Examples:

PROGRAM1	TR	03
PROGRAM1	I/O	60

OVERLAY PREPARATION 12

The SCOPE loader prepares overlay programs from relocatable binary subprogram decks. Overlays are composed of an executive or master control program called the main program and any number of parts called overlays and segments.

OVERLAY PROCESSING

Overlay processing allows programs that exceed available storage to be divided into independent parts which may be called and executed as needed. A program is divided into a main section and any number of overlays, each of which contains any number of segments. Main, overlay, and segment each contain subprograms. Only main, one overlay, and one segment may occupy storage at a given time.

The loader control cards, MAIN, OVERLAY, and SEGMENT, precede the relocatable binary subprograms which comprise the respective sections. After a source program is assembled or compiled, the decks are prepared and loaded. Each overlay or segment is written on an overlay tape as a separate record in absolute binary. The overlay tape is called in sections for execution. The absolute records do not require the relocatable binary loader to perform the usual relocating and linking functions.

If a segment is encountered before any overlay is encountered, that segment is treated as an overlay. However, the Card Sequence Error message will display the segment identification number relevant to this segment. Refer to the section on loader errors.

Initially, control is transferred to MAIN which resides in storage continuously; it in turn calls the overlays when they are needed. Segments are called only by an overlay. FORTRAN and COMPASS subroutines, available to call the overlays and segments during execution, must be included in the MAIN element. Once an overlay tape is created, it may be executed as many times as desired if the same equipment configuration and SCOPE resident programs are used with subsequent executions. Overlays occupy memory common to other overlays; segments associated with a given overlay occupy memory common to other segments of that overlay.

USE OF OVERLAYS

Segments may make reference to only those addresses which are:

1) defined within the segment
2) defined in the overlay with which the segment is associated and from which it is called, and/or
3) defined in the main program.

Overlays may reference only those addresses defined within the overlay or the main program.

The main program may refer only to addresses defined within itself. All entry point symbols referenced within the main program, an overlay, or a segment are defined for that program element and any elements subordinate to it.

Overlays and segments are stored on tapes from which they are called to be executed. The tapes are prepared by the loader as specified by loader control cards.

Two transfer addresses are recorded for each element loaded. If a transfer address does not appear, or if more than two are encountered, an error is indicated. The transfer addresses for the main program are returned to the calling program.

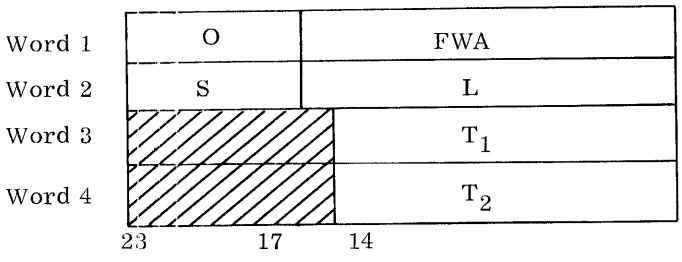
When the end of the program deck is reached, SCOPE reads the main program back into memory if it was stored on tape, prior to entering the user's program.

If an element written on tape contains the DATA block, the DATA block will retain its definition and may be referenced. The DATA block may not be loaded by subsequent elements subordinate to the element in which the DATA block was first defined.

OVERLAY TAPE FORMATS

Overlay tapes contain program overlays and/or segments as prepared by the loader. Each overlay or segment is preceded by a four word record which identifies the information record following. Information is recorded and read in binary mode with density unspecified.

The identifying record format is:



T_1	primary (last encountered) transfer address
T_2	secondary transfer address
O	number from $001\text{-}143_8$, identifying an overlay
S	number from $001\text{-}143_8$, identifying a segment of the overlay identified by †
FWA	first word address into which the information record is to be read
L	length of information record in words

Oand S are both 000 if the following record is the main program.

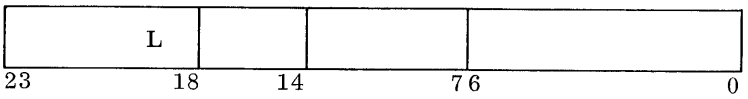
The information record consists of absolute binary words which are read into FWA through FWA + L - 1. The last information record on each overlay tape is followed by an end-of-file.

OVERLAY AND SEGMENT EXECUTION

Overlays and segments are called from tape by entering a library routine from the main program. The routine is called by:

```
place parameter in A
RTJ     EXECOVR
```

EXECOVR is declared to be external to the main program. The parameters in A are given in the format:



L	logical unit number used to address the tape on which the overlay or segment is stored.
I	identifying number of the overlay or segment to be entered. I is in the range of $001\text{-}143_8$; bits 7-14 are always zero.

EXECOVR locates the segment or overlay on the indicated tape, reads it into memory, and passes control to it by executing an RTJ to its primary transfer address, with the secondary transfer address contained in $A_{14\text{-}0}$. The segment or overlay returns control through the linkage established. When return is

† If the record is an overlay record, S contains 000.

made from a segment, EXECOVR returns control to the overlay which last called EXECOVR; if an overlay returns control, EXECOVR returns to the main program. The linkage to the main program, and the number of the overlay to be entered is saved by EXECOVR each time an overlay is requested. EXECOVR may not be called within a segment; a call made to EXECOVR within an overlay causes loading of a segment; a call made from the main program causes loading of an overlay.

If the requested overlay or segment cannot be located on the specified tape, or be read free of errors, the job is terminated.

EXECOVR must be declared as an external symbol in the main program and in each overlay which calls it. EXECOVR is loaded from the library tape as part of the main program.

MAPPING OF OVERLAY AND SEGMENTS

When an overlay or a segment has been prepared, a map of the memory is produced on the Standard Output unit, unless suppression is indicated by an S in the third field on the OVERLAY or SEGMENT card. The map shows:

The first word address of all subprograms comprising the overlay or segment.

All entry points defined within the overlay or segment.

The overlay or segment extension area, if one is assigned to contain corrections added to the overlay or segment.

The load map for each overlay and segment is printed on a new page. The first line of each page is headed with the following line:

" OVLAY nn SEG ss TAPE tt

nn Overlay identification number (decimal) of this overlay, or of last preceding overlay if this is the map of a segment.

ss Segment identification number (decimal) of this segment, or 00 if this is the map of an overlay.

tt Logical tape unit number (decimal) of the tape unit on which this element was written. Illegal unit designations will produce a Card Format Error message. The parameter tt will then appear as 00.

Similarly, illegal values of nn or ss will produce an error message, and the heading line of the corresponding map will show the illegal value as 00 for ss or nn.

OVERLAY CONTROL CARDS

The programmer prepares the loader control cards, MAIN, OVERLAY, and SEGMENT and places them in front of the subprogram decks. Corrections may be entered by OCC statements within the overlay input.

MAIN

MAIN must be the first card encountered by the loader when overlays are prepared. The subprograms following MAIN and preceding any other overlay control card constitute the main program. The MAIN card has an arbitrary word count of 50_8 and except for column one, the card is in 12-bit Hollerith. The MAIN card consists of the standard column 1 information and two fields of Hollerith information.

Card Column	Meaning
1	standard identification, 7/9 punches, W = 50_8
2	beginning of two fields described below
6-80	not used

The first field begins in column 2 and is terminated by a blank or a comma. This field contains the logical unit number $1\text{-}55_{10}$ of the magnetic tape on which an overlay program is to be written. If the MAIN program is not to be written on tape, field one may contain zero, two zeros, blank or a comma in column two.

The second field contains a P if OCC cards are contained in the overlay deck. If OCC occurs and P is not declared, OCC cards will be treated as card sequence errors, code CS.

OVERLAY

The OVERLAY card precedes each intermediate element of overlay programs. All subprograms which follow, and precede another overlay control card (or to the end of the deck) constitute the OVERLAY. The OVERLAY card has an arbitrary word count of 51_8 and is not checksummed.

The OVERLAY card defines the overlay, the logical unit on which the overlay is written and determines whether a map of the overlay will be prepared.

Card Column	Meaning
1	standard identification, 7/9 punches, W = 51_8
2	start of three fields

Three Hollerith fields separated by commas appear on the card beginning in column two. The first field specifies a logical unit number 1 to 55, on which the overlay will be written. This unit designation is mandatory. The second field contains a number, 1 to 99, which identifies the OVERLAY. The third field contains an S if the map for this overlay is to be suppressed.

SEGMENT

Overlay elements subordinate to OVERLAY are defined by SEGMENT cards. An OVERLAY may contain several segments. The subprograms following, to the next SEGMENT or OVERLAY card or the end of the program deck constitute a segment. Segments are subordinate to the overlay they follow.

The SEGMENT card has an arbitrary word count of 52_8 and is not checksummed. The SEGMENT card determines the logical unit on which the segment will be written, the segment identification and whether a map of the segment will be prepared.

Card Column	Meaning
1	standard identification, 7/9 punches, $W = 52_8$
2	start of three fields

The SEGMENT card contains three Hollerith fields separated by commas. The first field contains the logical unit number, 1 to 55, on which the SEGMENT is written. This field is mandatory. The second field contains a SEGMENT identification number 1 to 99. The third field contains an S if the map of the segment is to be suppressed.

PROGRAM DEBUGGING 13

Debugging aids may be used at the source language or object levels. SCOPE will produce a map at the programmer's option to show the assignment of subprograms, data, common, and the program extension area to memory. All entry point names and the assigned addresses are shown on the map as well.

An octal correction routine allows changing, modifying or correcting the program after it has been loaded. Selective dumps may be obtained by means of the SNAP control statement.

DUMP ROUTINES

The programmer may choose several options for printing the contents of console registers, the register file and part or all of the non-system memory. After programs have been loaded, the programmer may use the SCOPE control statement, SNAP, which will allow the printing of selected portions of memory during program execution. Calling sequences are available to specify a similar dump in the source language. Recovery dumps are also available should a program be terminated due to errors.

MEMORY ALLOCATION PRINT

The programmer may secure a map of memory allocated to a loaded program at the time the RUN statement is encountered. This map may be suppressed by a parameter of the RUN statement.

The map contains:

Heading	Category
SUBP	the name of each subprogram as stated in the IDC card and the absolute address of the first location of the subprogram in execution time memory.
ENTR	the entry point symbol as taken from EPT cards and the absolute address of each entry point declared in any subprogram loaded for the run.
COMM	the absolute addresses of the first and last locations in the common area.

Heading	Category
DATA	the absolute address of the first location in the data area.
PEXT	the absolute address of the first location in the program extension area.

Memory allocated to SNAP calling sequences is not included in the above areas and does not appear in the memory map.

Items in the map for subprograms are ordered from the lowest numbered address to the highest numbered address. Each item in these categories appears in the below format, where XXXXX represents a 5-digit octal word address: XXXXX 8 character name

The format for the data block and the extension area is:

```
Identifier
XXXXX
```

The format for the common block is:

```
COMM
XXXXX   XXXXX
```

Each MAP begins a new page of print. Categories are separated by one blank line. No lines are skipped within categories.

The information for the map is obtained from the loader symbol table.

SYSTEM DUMP ROUTINE

The programmer may use a dump routine to print the console registers, the register file and selected portions of non-system memory during a programmer run. The dump may be obtained in octal, character or decimal floating point formats. With the SNAP statement, the programmer may request the dump after the program has been loaded or may specify the dump in the FORTRAN or COMPASS source languages.

Each time it is called, the dump routine prints, on OUT, one line containing the dump identification of 4 BCD characters, the address of the calling sequence, the contents of the A and Q registers, the three index registers and the interrupt mask register. Except for the dump identification all of the above is printed in octal.

If the register file option is elected, the contents of all 64 registers are printed in octal following the console scoop.

The memory dump consists of printed lines in the mode indicated in the calling sequence. This is preceded by the octal absolute address of the location. If all words in a line are identical to the last word printed on the preceding line, lines are suppressed until one is found in which a difference occurs.

If the SNAP option is used, the dump routine prints the relocatable address of the location in the subprogram to the left of the absolute address of each print line. These columns are blank if the dump was not called by SNAP statements.

SOURCE LANGUAGE CALLS FOR SYSTEM DUMP

The calling sequence for the system dump routine in either FORTRAN or COMPASS is incorporated into the object subprogram. Each time the subprogram is operated, the dump routine will be in memory and dumps will be taken.

Except for relative address control, the format of the dump is the same for source time calls as for SNAP. The source language calls are shown below.

CALLING SYSTEM DUMP ROUTINE

The system dump routine has three entry points: FORTDUMP, called in FORTRAN programs; PROGDUMP, called in COMPASS programs; and SNAPSHOT, called by the SCOPE control statement, SNAP.

Source Calling Statement	Machine Language Calling Sequence
SCOPE Control Statement	SNAPSHOT Calling Sequence
7_9 SNAP, LC, B, E, M, ID	FWA
	m LWA
	Identification
	SRF
	*Snapped Instruction
	UJP RLA
	RTJ SNAPSHOT

COMPASS Statement	PROGDUMP Calling Sequence	
RTJ PROGDUMP	RTJ	PROGDUMP
M ID B E		FWA
	m	LWA
	BCD	1, <4 char. identification>
	*	

FORTRAN Statement	FORTDUMP Calling Sequence	
CALL FORTDUMP (B, E, M, ID)	RTJ	FORTDUMP
	77	**L(FWA)
	77	**L(LWA)
	77	**L(MODE)
	77	**L(ID)
	*	

Terms in SCOPE Calling Statements:

LC	specifies the location of the instruction at which the SNAP occurs
B	beginning address of the dumped region
E	ending address of the dumped region
M	mode indicator
ID	identifier

Terms in Machine Language Calling Sequences:

FWA	first word address to be dumped
LWA	last word address to be dumped
RLA	return linkage address to the program
SRF	subprogram relocation factor
Mode or M	mode indicator
ID	identification
*	indicates where control is returned
**L(x)	indicates the location of the word containing x

ERROR MNEMONICS

Mnemonic	Meaning
For SNAP and OCC statements	
*PN	Program Name
*BS	Common or data storage is undefined and referenced
*AD	Address or location field begins with illegal character
*8F	Octal field contains non-octal character
*XA	Extension area undefined or to small
*WR	Wraparound of location field address: exceeds core size
For OCC statements only	
*AN	Antecedent reference to a program (*) or loading address (+)
*RL	Relocation portion of correction field contains illegal character

Mnemonic	Meaning
For SNAP statements only	
*OV	Overflow of memory will occur if this SNAP is loaded
*IM	Illegal Mode field on SNAP card
*RG	Range of area to be snapped has beginning address larger than ending address

SNAP

The format of the SNAP control statement is shown in the section on SCOPE control statements. SCOPE processes the SNAP statement and builds a calling sequence. The instruction at the designated location in the subprogram is exchanged for a jump to the calling sequence and saved for execution after the dump is taken. SNAP prints the dump on OUT as indicated by the parameters. The system dump routine must be loaded by the use of the loader control card EXS unless called by some source language option.

The location at which the SNAP occurs must not be altered during execution or by OCC statements. When SNAP and OCC statements are intermixed, an OCC statement must not destroy the jump to the SNAP calling sequence.

In choosing instructions at which to request SNAP dumps, the programmer must follow certain rules.

1. Do not SNAP at instructions involving more than one word, such as search, move, skips and certain I/O instructions.
2. Do not SNAP conditional tests or jumps.
3. Do not SNAP indirectly addressed instructions.
4. Do not SNAP instructions which will be modified by program execution.
5. Do not exchange any of the following instructions with a jump to the SNAP calling sequence.

Octal	Mnemonic	Octal	Mnemonic
10	ISI	07	MTH
10	ISD	10	SSH
04	ASE	52	CPR
04	QSE	77.0	CON
04	ISE	77.1	SEL
05	ASG	77.2	EXS
05	QSG	77.3	INS
05	ISG	77.4	INTS
06	MEQ	77.6	PAUS

6. Do not exchange any instruction within the range of the following instructions with a jump to a SNAP calling sequence.

Octal	Mnemonic	Octal	Mnemonic
71	SRCE	74	INAW
72	MOVE	74	OUTC
73	INPC	75	OTAC
73	INAC	76	OUTW
74	INPW	76	OTAW

7. Avoid SNAP calls within a loop.

SYSTEM DUMP PRINT MODES

The mode parameter establishes the format of the dump. The symbolic form is used in SNAP statements; the octal form is used with PROGDUMP and FORTDUMP.

Symbol	Octal	Mode
O	1	octal
C	2	character
F	3	floating point
R	4	register file
OR or RO	5	octal and register file
RC or CR	6	character and register file
RF or FR	7	floating point and register file

Examples:

7/9 SNAP, Lc, B, E, M, ID

The SNAPSHOT card is put behind a binary deck. SNAPSHOT entry is called to obtain relative addressing from COMPASS code as follows:

```
          IDENT     PROG
FIRST     BSS       0
          UJP       SNPDMP
          EXT       SNAPSHOT
                    BUFY
          1         BUFY+24
          BCD       1, PRG1
                    FIRST
          NOP       0
          UJP       *+2
SNPDMP    RTJ       SNAPSHOT
```

To take a dump after the instruction is executed at fifth location of subprogram SUB1, use the following SNAP card; assume BUFY at location 0441_8 in SUB1.

7/9 SNAP, (SUB1) 6, 441*, 465*, O, SNP1

The area between BUFY and BUFY+24 is dumped in the octal mode. SNP1 identifies this card.

PROGDUMP calling sequence:

```
RTJ       PROGDUMP
          BUFY
1         BUFY+24
BCD       1, PRG1
```

RECOVERY DUMP

If a job is terminated abnormally, a post execution dump of non-system memory is written in octal on OUT unless the programmer has specified in the JOB card that it is to be suppressed. The dump consists of console conditions, the register file, and non-system memory. The dump is preceded by interrupt and trapped instruction information. The format of the dump is that described in the systems dump routine if octal and register file options are selected. The recovery dump is a separate routine.

OCTAL CORRECTION OF LOADED PROGRAMS

When subprograms have been loaded and control returned to SCOPE, the programmer may enter octal corrections for the loaded program. The SCOPE control statement OCC provides for replacing single or several contiguous instructions in the program. The format of the OCC statement is discussed in the section on SCOPE control statements.

The corrections loaded by the OCC statements may replace one or more instructions or constants in a subprogram, or the programmer may use OCC to define a program extension area.

If an OCC is directed to the same location as a SNAP statement and follows the SNAP statement in the SCOPE input, the SNAP is lost. No error is indicated and the SNAP calling sequence still consumes available memory.

DEFINITION OF PROGRAM EXTENSION AREAS

If a program extension area is used, it must be defined by an OCC statement in the following form:

$$\begin{matrix}7\\9\end{matrix}\ \text{OCC, Xk}$$

X indicates the definition of the extension area and k is an octal integer indicating the length of the extension area. If the length, k, exceeds available memory, the area is adjusted to the size of available memory. A message is printed on OUT and execution is allowed if this is the only error in the run. If loading of corrections exceeds the SCOPE defined area, it is handled as memory overflow. If the first OCC statement to name X in the location field is of incorrect format, it is ignored; an error indication is given; an arbitrary extension area of 777_8 locations (which may be adjusted to fit available memory) is defined. The deck is processed to the RUN statement and execution is inhibited.

Octal corrections can be made into three areas of memory; subprogram, data area and program extension areas. Since compilers and assemblers output in the relocatable mode, it is further necessary to use address modifiers to find memory locations. For example; if several subprograms are loaded, the absolute address of a variable in a particular subprogram is not known. The programmer must place the name of the subprogram and the relative address on the octal correction card.

$^{7}_{9}$OCC, (PROG1)70, 20000100*

A correction is to be entered at address 00070 relative to subprogram PROG1. The * tells SCOPE to relocate address 00100 relative to subprogram PROG1.

$^{7}_{9}$OCC, (SUB1)77, 20000100*, 40000101(SUB2).

Put 200XXXXX in location SUB1+77 of subprogram SUB1; relocate 00100 relative to subprogram SUB1. Put 400XXXXX in location SUB1+100 of subprogram SUB1; relocate 00101 relative to subprogram SUB2.

$^{7}_{9}$OCC (SUB1)20, 00000036, 0000036, 000036, 00036, 0036, 036, 36.

Put the octal value 00000036 into locations 20, 21, 23, 23, 24, 25, 26 of subprogram SUB1; since values are right justified, 00000036 and 36 both go into memory as 00000036.

$^{7}_{9}$OCC, X20.

$^{7}_{9}$OCC, X, 20000100(SUB1), 40000101*, 20000102*, 40000103*.

$^{7}_{9}$OCC, +, 20000400(SUB2), 40000401*, 20000402(SUB3), 40000403*.

$^{7}_{9}$OCC, X10, 2000620(SUB4), 40000621*, 20000622(SUB5), 40000623*.

The first X assigns 20 locations to the program extension area. The next X puts 200ZZZZZ into location 1 of the extension area; ZZZZZ is the relocated address relative to subprogram SUB1. 400ZZZZZ goes to location 2 of the extension area, 200ZZZZZ to 3, and 400ZZZZZ to 4; all ZZZZZ addresses are relocated relative to subprogram SUB1. The + card continues stacking information in the program extension area; the addresses of the first two corrections are relocated relative to subprogram SUB2, the last two are relative to subprogram SUB3.

The last card starts loading the information pertaining to subprogram SUB4 and subprogram SUB5 into program extension area location 10.

$^{7}_{9}$OCC,D,5,10,15,20,25,30,35,40.

$^{7}_{9}$OCC,+,45,50,55,60,65,70.

$^{7}_{9}$OCC,D20,75,100,105,110,115, , 125, , 135.

The first two cards will put the 14 octal values 5-70 in successive data area locations starting with zero.

The last card will put the 5 octal values 75-115 in successive locations starting with data area 20. Data area 25 will be unchanged, 26 will hold 00000125, 27 will be unchanged and 30 will hold 00000135.

$^{7}_{9}$OCC,(SUB1)70,01000010X,20000005C, 40000007D

$^{7}_{9}$OCC,+,20000007(SUB1)C

Put instruction 010XXXXX into location 70 SUB1; XXXXX is modified relative to address 10 of the program extension area. Put into SUB1+71 200XXXXX; XXXXX is modified relative to the fifth common address; SUB1+72, etc. Put 20XXXXXX into SUB1+73; XXXXXX is a character address modified relative to subprogram SUB1.

ERRORS IN SCOPE DEBUG STATEMENTS

When SCOPE is processing SNAP and OCC statements, errors may result from incorrect format or memory overflow. SNAP card errors do not prevent execution; errors in OCC cards do, except when too large an extension area is declared. If SNAP calling sequences overflow memory, execution is inhibited. If the system dump routine is not in memory and SNAP statements occur, the job is terminated.

ERROR MESSAGES

If a SNAP statement is used and the system dump routine is not in memory, the run is terminated and this diagnostic is printed:

***NOSD

RUN ABORTED

If an extension area is defined incorrectly, the following is printed:

***Xnnn

nnn is the 3-digit octal length of the SCOPE defined extension area.

The format for all other error diagnostics is the same.

*mn, COL nn

mn is the mnemonic

nn is the actual column number on the card

at which the error was detected. In certain cases the column number will be the last in a field.

APPENDIX SECTION

APPENDIX A
BINARY CONTROL CARDS

Control cards which control SCOPE and related systems may be considered in three categories:

Executive/monitor

Loader

PRELIB

Control cards for executive/monitor are binary cards with a word count of zero. Loader cards are binary cards with a non-zero word count. PRELIB control cards are of the identical format of executive/monitor control cards. PRELIB also uses loader cards but in a unique manner.

A control card for any unit is a binary card defined by the hardware when a 7, 9 punch is detected in column one. This card is read into memory and examined for punches in rows 3, 2, 1, 0, 11, 12, of column 1; they contain the word count.

Word count is derived from the relocatable binary information card, RIF, which may contain a variable amount of information for storage in one or more computer words. In the RIF card this is an actual word count. In other control cards this number is an arbitrary value used to tell the system how the information on the card is to be used.

All legal control cards and the word count are shown in the following table.

Intra SCOPE Communication

PRELIB is called by the PRELIB card and is in control until abnormal termination or until two FILE cards have been encountered.

Loader receives control from executive/monitor once per run to load I/O drivers needed immediately by SCOPE. During any loading operation, I/O drivers are loaded automatically. For programmer runs the loader receives control from executive/monitor when a binary card with non-zero word count is read from INP or when a LOAD statement is encountered. Loader retains control until abnormal termination or until a binary card with a word count of zero is read from INP, or when loading from the library when the called program and all externally linked subprograms have been loaded. Loader returns control to the POSTLOAD portion of executive/monitor.

BINARY CONTROL CARDS

Name	Octal Word Count	Description	Check-sum	Sub-system	Page
CTO	0	comment to operator	no	EXEC	
DELETE	0	delete subprograms	no	PRELIB	
ENDREEL	0	end of one reel of standard input	no	EXEC	
ENDSCOPE	0	terminate SCOPE run	no	EXEC	
<library name>	0	call library program	no	EXEC	
EPT	42	entry point name	yes	LOADER	
EQUIP	0	equipment declaration	no	EXEC	
EXS	55	external symbol declaration	no	LOADER	
FILE	0	terminate file	no	PRELIB	
IDC	41	subprogram identification	yes	LOADER	
INSERT	0	insert subprograms	no	PRELIB	
JOB	0	declare programmer's job	no	EXEC	
LED	54	equipment declaration	no	LOADER	
LOAD	0	load subprograms	no	EXEC	
LRL	45	local reference list	yes	LOADER	
MACRO	0	load macro symbolic deck	no	PRELIB	
MAIN	50	main program declaration	no	OVERLAY	
OCC	0	load octal corrections	no	EXEC	
ORIGIN	0	establish starting location	no	PRELIB	
OVERLAY	51	declare program overlay	no	OVERLAY	
RECORD	0	declare record	no	PRELIB	
REPLACE	0	replace subprograms	no	PRELIB	
REWIND	0	rewind magnetic tapes	no	EXEC	
RIF	1-40	relocatable binary information	yes	LOADER	
RUN	0	execute object program	no	EXEC	
SEGMENT	52	declare overlay segment	no	OVERLAY	
SEPOINT	0	declare system entry point	no	PRELIB	
SEQUENCE	0	declare job sequence	no	EXEC	

Name	Octal Word Count	Description	Check-sum	Sub-system	Page
SNAP	0	establish snapshot parameters	no	EXEC	
TRA	44	subprogram transfer	no†	LOADER	
UNIT	0	declare PRELIB input unit	no	PRELIB	
UNLOAD	0	rewind and unload magnetic tape	no	EXEC	
XFER	0	copy data from standard input	no	EXEC	
XNL	43	declare external name and linkage	yes	LOADER	

†The TRA card contains a subprogram checksum but it is not checksummed.

APPENDIX B
INSTALLATION ACCOUNTING

SCOPE contains an ACCOUNTS table which holds information pertinent to installation accounting. This table receives information from the operator and from the programmer control statements SEQUENCE, JOB and RUN. SCOPE maintains the ACCOUNTS table but does not perform any accounting function. Rather, at appropriate times, SCOPE provides linkage with an installation accounting routine which may perform whatever accounting a particular installation deems desirable. The installation may elect to have a complex routine which restricts job times and prepares detailed reports; a simple routine which merely retrieves accounting information for use at a later time; or no accounting routine.

Calls to Accounting Routine

SCOPE provides linkage with the installation accounting routine through the SEQUENCE, JOB, ENDSCOPE and RUN control statements. SCOPE will update the accounts table before or after the installation accounting routine is executed.

Sequence Statement

processes the accounting record for the preceding job.

Job Statement

for a non-stacked job closes the accounting file prepared by the accounting routine prior to the unloading of the ACC output unit. When control is returned to SCOPE, the current ACC is unloaded and ACC is equated to CTO. For a stacked job, a new record is opened.

Endscope Statement

The installation accounting routine is entered to close out the records for the preceding job and to close the file prior to unloading ACC.

Run Statement

The accounting routine is entered to set up timing restrictions.

Accounting Routine Calling Sequence

The SCOPE call to the accounting routine has the general form:

```
RTJ     JOBACC

    parameters
```

The parameters indicate what control card SCOPE is processing.

Structure of Accounts Table

The ACCOUNTS table consists of 12 words which contain the date, local and machine times, sequence number of the job in process, the accounting and identification information from the JOB statement, and the times declared on the JOB and RUN statements.

Accounts

+0, +1	ddmmyybb	
+2, +3	local time in BCD machine time in binary	when SCOPE was initiated
+4	sequence number of current job in BCD	
+5, +6	c field of JOB statement in BCD	
+7	i field of JOB statement	
+8, +9	run time flags	
+10	machine time at beginning of job, in binary	
+11	run time limit in BCD	

APPENDIX C
SCOPE TABLES

C

The SCOPE input/output control routine, CIO, uses the tables described on the following pages.

Available Equipment Table (AET)

Unit Status Table (UST)

Channel Status Table (CST)

Running Hardware Table (RHT)

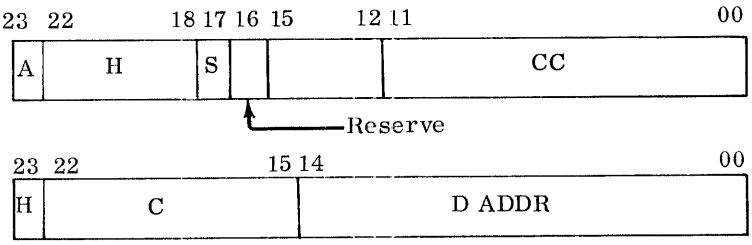
CIO also addresses CIT.

AVAILABLE EQUIPMENT TABLE

The Available Equipment Table (AET) contains up to 50 two-word entries, depending on the equipment configuration used. Each entry contains the information necessary for CIO to use the particular hardware unit described in that entry.

The operator may use the AET control statement to alter AET.

The word length of the table is recorded in bits 14-0 of the location immediately preceding the table. The table has the following format:



A - 0 unit has not been assigned
1 unit has been assigned

H - Numeric code, $00\text{-}37_8$, to describe each hardware type

R - Reserve
0 unit is reserved by another computer
1 unit is available to this computer

S - 0 unit is operable
1 unit is not operable

E - 0 no action
1 unit of this hardware type to be assigned

CC - a 12-bit connect code for each unit, the actual machine code to be inserted in the connect instruction

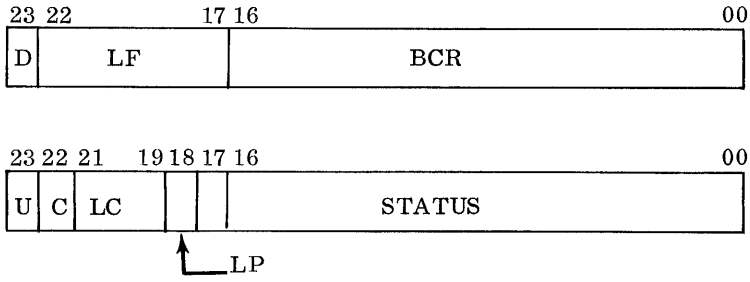
C - an 8-bit code in locations 22-15 corresponding to the channels 0-7, which are available for use by this unit. A bit set ON indicates channel availability. If this field contains all zeros, there are no channels to which the unit may be connected. CIO will pass control directly to the driver for processing of the I/O request.

D ADDR - driver address for the hardware type of the particular unit.

UNIT STATUS TABLE

Each two-word entry in the UST contains the current status of a hardware unit.

UST entry:



D - significant only to the I/O system. 1 unit is static, the last preceding operation on the unit is completed.
0 unit is dynamic.

LF - last function code (other than status, 13) given for this unit

BCR - input/output buffer termination address (contained in the buffer control register). If the unit is busy, this may not be current.

U - the unit busy indicator
0 unit is free
1 unit is busy

C - channel busy indicator
0 channel is free
1 channel is busy

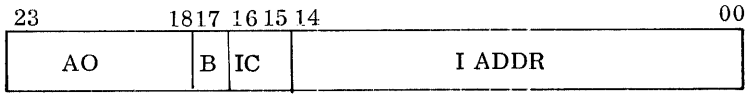
LC - channel last used by this unit

STATUS - status replies received from the I/O equipments. If the unit is busy, current unit status at the time of the request is given. If the unit is free, status reflects the results of the last completed operation.

CHANNEL STATUS TABLE

The Channel Status Table, CST, defines channel status, I/O status and interrupt selections.

This table consists of up to 8 one-word entries. Each table entry reflects the busy or not busy status and user requested interrupt status of a particular channel. The format of the table is:



AO - AET ordinal of last hardware unit to use this channel

B - 0 channel not busy
1 channel busy

IC - 00 no interrupt selected by the user
01 abnormal termination (EOT, EOF, LP, PARITY) interrupt selected by user

10 or 11 end of operation (includes abnormal termination interrupt selected by user)

I ADDR - user's interrupt subroutine address

UNNING HARDWARE TABLE

iis table is a reference for operating hardware units and SCOPE logical units. The Running Hard-
ire Table, RHT, consists of 63 entries of two octal digits each. The table is arranged by logical
it number; entry one refers to logical unit one, entry 60 refers to INP, and so forth. Each entry
ntains the position number in AET and UST of the physical unit to which the logical unit is assigned.
the entry is zero the logical unit is unassigned. The entry for logical unit number zero is not refer-
ced by CIO.

APPENDIX D
INTERNAL INTERRUPT CONTROL

D

SCOPE controls the processing of internal interrupts. The Central Interrupt Control routine, CIC, detects the interrupt condition and transfers control to the appropriate user routine. The conditions include those defined by the interrupt mask register plus manual interrupt.

The address of all interrupt routines must be stored in the Central Interrupt Table (CIT) which is maintained within CIC. This table contains the following:

Symbolic Location		Content	Explanation
CIT	+0	Interrupt flag	+0 = no interrupt occurred
	+1	(A)	Contents of these registers when last interrupt occurred. Registers are restored from here on exit from CIC.
	+2	(Q)	
	+3	(B1)	
	+4	(B2)	
	+5	(B3)	
	+6	Real Time Clock	Initially contains ABNORMAL. Set by the user selecting the corresponding interrupt.
	+7	Arithmetic Overflow	
	+8	Divide Fault	
	+9	Exponent Overflow	
	+10	BCD Fault	
	+11	Search/Move	
	+12	Manual	
	+13	Associated Processor	
	+14	Channel 0	Initially contains ABNORMAL. Address is set by CIO when interrupt is selected. Table is extended for each pair of channels added to the hardware configuration, up to a total of 21 entries.
	+15	Channel 1	
	+21	Channel 7	

In addition to storing the subroutine address in CIT, the user also sets the appropriate bit in the interrupt mask register.

When an interrupt occurs, CIC determines if the instruction about to be executed is a Disable Interrupt command. If so, control returns to the interrupted program. If the instruction is not DINT, CIC determines the interrupt subroutine to be entered. The address in CIT is replaced by the address of the system entry point, ABNORMAL; and a return jump to the specified interrupt subroutine is executed. Only the address portion of the CIT entries 6-21 applies; the operation portion (for 6-11) contains the interrupt mask bits.

Within his interrupt subroutine, the user determines the particular condition which caused the interrupt, clears the condition when processed, and finally, clears the appropriate bits (internal) in the interrupt mask register.

When interrupt occurs, CIC sets the interrupt flag and stores the contents of the registers in CIT. After the user subroutine is executed and before return to the point of interrupt, CIC resets the flag and restores the registers. The user may access or alter the content of the registers by referencing the appropriate CIT entries.

ORDERING OF LOW NUMBERED MEMORY

Octal Location	Opera-tion	Bit	Address	Meaning
00000	NOP		BOOT or ABNORMAL	set by EXEC
00001	UJP		ABNORMAL	protect memory overlap
00002	xx	0	*****	power loss cell
00003	00	0	ABNORMAL	Halt on power loss
00004	xx	0	*****	program interrupt address
00005	NOP	0	xx***	NOP, receives code
00006	01	0	yyyyy	Jump to CIC
00007	xx	X	xxxxx	Unused
00010	xx	0	*****	program address of trapped instruction
00011	77	7	3xx**	Disable interrupt instruction holds trap code
00012	01	0	zzzzz	UJP to process trapped instruction

The symbols have the following meaning:

x = not used; may contain any bit combinations convenient to the system. These areas are reserved for use by SCOPE.

yyyyy = address of entry to CIC.

zzzzz = address of OPTBOXS, or ABNORMAL if no trapped instructions are indicated. This address is set by the loader.

* = octal digit (3 binary positions) set by the hardware.

INTERRUPT PROCESSING

The following sequence of events occurs to process interrupts:

1. When the hardware recognizes an interrupt, it stores the interrupt code in location 5, the return address to the point of interrupt in location 4, and disables further interrupts.
2. The hardware begins the interrupt processing sequence at location 5 and jumps to CIC from location 6.
3. CIC performs the following operations:
   a. Saves registers A, Q, B1, B2, B3, in CIT + 1 through CIT + 5.
   b. Sets interrupt flag, CIT + 0, to -0.

c. Determines from code in location 5 which of 16 jumps in CIT + 6 through CIT + 21 to execute.

d. Based upon decision in c, executes appropriate interrupt subroutine.

e. Upon return from interrupt subroutine, resets interrupt flag, CIT + 0, to zero.

f. Restores registers A, Q, B1, B2, B3, from locations CIT + 1 through CIT + 5.

g. Enables interrupts, and performs an indirect jump through location 00004 (UJP, I 4).

PROTECTION OF ROUTINES

Certain routines in the main program and the interrupt subroutines must be protected from an interrupt occurring during their execution. To do so, the first executable instruction in the routine must be a Disable Interrupt Control, DINT, instruction.

Furthermore, before exiting, the interrupt flag, CIT + 0, must be tested to determine whether interrupt control had already been disabled by an interrupt. If interrupt control was not disabled, the last executed instruction before the jump must be an Enable Interrupt Control Instruction. SCOPE routines are coded in this manner.

TRAPPED INSTRUCTIONS

Systems programs simulating the trapped instructions gain control through location 12. These routines disable and enable interrupt control as described above for protection of routines. They will not cause an interrupt; however, they recognize any interrupt condition. The SCOPE loader is responsible for recognizing requirements for trapped instruction simulation and establishing linkage to the required routines.

APPENDIX E
PRELIB CONTROL STATEMENTS

PRELIB is a library program; PRELIB control statements are contained on cards in the same format as SCOPE control cards.

PRELIB prepares or updates a system library tape, LIB, which consists of two files. The first file contains SCOPE, including loader, overlay processor, and so forth. File two contains the library programs such as COMPASS, FORTRAN, SORT, PRELIB and the system macros, library subroutines, user programs and non-system I/O drivers.

The use of PRELIB and the structure of the control deck is dictated by which file is being maintained. The input deck consists of PRELIB control cards, loader cards and, for file two, Hollerith cards generally of COMPASS format.

RECORD STATEMENT

$^{7}_{9}$RECORD, m

RECORD is used only for file one maintenance. The RECORD statement defines the beginning of a new record in file one of LIB. RECORD must follow the REPLACE statement prior to the appearance of any loader cards in the deck. RECORD must be followed immediately by an ORIGIN statement except that any number of SEPOINT statements may intervene.

The parameter, m, may be the name of an entry point previously defined by PRELIB. It may be an entry point name followed by a plus or minus and not more than 5 octal digits. m may be also expressed by octal integer of 5 digits or may be omitted.

The length of programs following the RECORD, m statement and the related ORIGIN cards must not exceed the capacity of the available portion of the PRELIB work area.

UNIT STATEMENT

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

UNIT may be used to maintain file one and two. The UNIT statement allows PRELIB input from a unit other than INP. PRELIB reads program decks on the designated unit, u, to an end-of-file. The logical unit number should not be 60, INP, and may be further restricted by equipment assignments for the current job. The unit may contain only loader cards and must not contain PRELIB control cards. When the end of file is read, input is resumed from INP. The card images on the unit must conform to the rules for PRELIB input deck structures.

SEPOINT STATEMENT

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

SEPOINT cards are used only for file one maintenance. The system entry point statement names entry points to SCOPE. The entry point name becomes a part of the permanent/system portion of the loader symbol table. A programmer may refer to such entry points by declaring the entry point names external symbols in his source program.

p is the name of an entry point defined in an EPT or EXS card in the file one input for PRELIB. If p is not defined in an EPT or EXS card, the address defined for ABNORMAL will be assigned. If ABNORMAL is undefined 77777_8 will be assigned. A diagnostic will result if p is not assigned.

Only one system entry point name may be declared per card. Any number of SEPOINT statements may occur between the REPLACE statement and the FILE statement terminating file one of the LIB modification deck. There are no restrictions as to sequence of SEPOINT statements.

ORIGIN STATEMENT

$^{7}_{9}$ORIGIN, m

ORIGIN is used only for file one maintenance. The program following ORIGIN in the PRELIB deck will be located at the absolute address expressed by the ORIGIN parameter, m; the parameter has the same meaning and is expressed in the same manner as for the $^{7}_{9}$RECORD statement.

Any number of ORIGIN cards can be used in the PRELIB deck. One ORIGIN must immediately follow each RECORD or SEPOINT. Optional ORIGIN cards must follow a TRA card and must precede an IDC card except that SEPOINT cards may precede or follow any ORIGIN card.

FILE STATEMENT

$^{7}_{9}$FILE

There are no parameters for the FILE statement. Every PRELIB input deck must contain two FILE statements. If FILE immediately follows the first statement, copy file one on the new LIB and process file two input.

To terminate the modification deck for either of the two files of LIB, the FILE statement is used. When the input for modification of file one has been read, the FILE statement instructs PRELIB to process the intermediate output.

When processing is complete, the new file one for the updated LIB is written on the tape unit designated by the parameter on the PRELIB statement. The information for modification of file two is then read and processed by PRELIB. When the second FILE statement is encountered, PRELIB writes a second end-of-file terminating the library portion of LIB and returns control to SCOPE.

DELETE STATEMENT

$^{7}_{9}$DELETE, PF, PT

A single subprogram or a group of contiguous subprograms may be removed from file two of LIB using DELETE. Any number of DELETE statements may be in the file two deck. The following rules govern the parameters for DELETE:

PF is the name in the IDC card of the first subprogram to be deleted from file two of LIB.

PT is the name in the IDC card of the last subprogram to be removed from file two of LIB.

PF blank, PT not blank. The first and all subprograms to and including PT will be removed from file two.

PT blank, PF not blank. PF and all subsequent subprograms are deleted to the end of file two.

PF and PT blank. File two is deleted.

PF equals PT. Only one subprogram is deleted.

PT occurs before PF on old LIB. Parameter error. No subprograms are deleted and a diagnostic is produced.

PF does not occur on old LIB. Same as above.

PT does not occur on old LIB. Same as above.

INSERT STATEMENT

$^{7}_{9}$INSERT, P

Subprogram decks may be inserted into file two of LIB. The subprograms may be on INP or another unit. If they are not on INP, a UNIT statement must immediately follow INSERT.

P names the subprogram after which the new subprograms are to be added. If the INSERT parameter, P, is blank and the statement in which the blank occurs is the first card in the PRELIB file two modification deck, the new subprograms will be added at the beginning of file two.

If any other INSERT statement has a parameter of blank or P does not occur on the old LIB, the job is terminated after a diagnostic.

REPLACE STATEMENT

REPLACE deletes and inserts subprograms. The statement has two meanings and two formats:

File One

$^{7}_{9}$REPLACE

This statement deletes the entire first file and a new file one is read into a working area for processing by PRELIB. The input for the new file one which follows immediately consists of file one control cards and relocatable subprogram decks. File one input is terminated by a $^{7}_{9}$FILE statement.

File Two

$^{7}_{9}$REPLACE, PF, PT

PF name in IDC card of first subprogram is to be deleted from file two.

PT name in IDC card of last subprogram to be deleted from file two.

Subprograms to be added to LIB may be on INP or another unit. The UNIT statement must immediately follow REPLACE in the PRELIB input.

When PF is encountered in the old LIB the subprograms on INP or u are written on the new LIB. Old LIB is then searched for PT and all intervening subprograms, including PT, are deleted.

PF and PT operate the same as in the DELETE statement.

MACRO STATEMENT

$^{7}_{9}$MACRO, P, u

Hollerith card images may be written in file two of LIB. COMPASS uses such images to assemble macros. Other uses may be defined by other programs. Hollerith decks are preceded in the file two PRELIB input by the MACRO statement.

P is the name of the Hollerith input deck and the name placed in the pseudo-IDC card on LIB; P must be present. u is the logical unit number of the tape containing P. If P is on INP, u is blank.

When the MACRO statement is encountered, PRELIB write a special card on the new LIB identical in format to an IDC card, but with a word count of 70_8. Cards are copied from INP or u until an END card identical in format to a COMPASS END card is encountered. Each Hollerith card copied on LIB has a word count of 71_8. The card images are converted to BCD, and are packed two per 40-word record. Card columns 77 through 80 of the Hollerith cards are lost.

If an end-of-file is reached before an END card, a diagnostic is produced and the job is terminated.

In editing the resulting tape, the named pseudo-programs may be deleted or replaced, or otherwise used as a reference point. Individual card images within a macro program may not be edited without replacing the entire program. Any number of cards may be included in a macro program on LIB.

OTHER PRELIB INPUTS

In addition to PRELIB Control Statements, PRELIB will accept all loader cards. OCC cards may not be input for either file one or file two maintenance.

If a subprogram is called, the loader searches file two of LIB until an IDC card is found. The loader examines the EPT cards following the IDC card until the first non EPT card. When the first non EPT card is reached the loader skips to the next IDC card and begins the search again.

LIB is searched until the end-of-file. If no subprograms were loaded, the loader produces error flags and diagnostics for any undefined EXT entries in the loader symbol table and transfers control to SCOPE. If subprograms were loaded, LIB is rewound and file two is searched again. The process is repeated until all called subprograms on LIB have been loaded.

PRELIB Control Cards

		LIB File
DELETE	Delete Subprogram	F2
FILE	Terminate LIB File	F1, F2
INSERT	Insert Subprogram	F2
MACRO	Load Hollerith Cards on LIB	F2
ORIGIN	Originate Subprogram Absolutely	F1
RECORD	Begin Record on	F1
REPLACE	Replace File One of LIB Replace Subprogram on LIB	F1 F2
SEPOINT	Define System Entry Point	F1
UNIT	Load from Unit	F1, F2

Card formats are identical to SCOPE control cards

Detail Cards for PRELIB

Word Count (octal)	Name		LIB File
1-40	RIF	Relocatable Binary Information	F1, F2
41	IDC	Program Identification	F1, F2
42	EPT	Entry Point Name	F1, F2
43	XNL	External Name	F1, F2
44	TRA	Transfer	F1, F2
45	LRL	Local Reference List	F1, F2
50†	MAIN	Main Overlay Program	F2
51†	OVERLAY	Overlay Program	F2
52†	SEGMENT	Overlay Segment	F2
54†	LED	Loader Equipment Declaration	F2
55†	EXS	External Symbol	F1, F2
70		Dummy IDC for MACRO	F2
71	<BCD>	Identifier for BCD card image for MACRO	F2

†Card in Hollerith except for column 1

APPENDIX F TYPICAL DECK STRUCTURES

F

ASSEMBLY AND EXECUTION

SCOPE monitor system performs the following functions:

Maintain continuity of processing

Loads and links subprograms and library subroutines

Manipulates I/O equipments as directed

Initiates program execution

Communicates with the operator

Transfers records to a logical unit

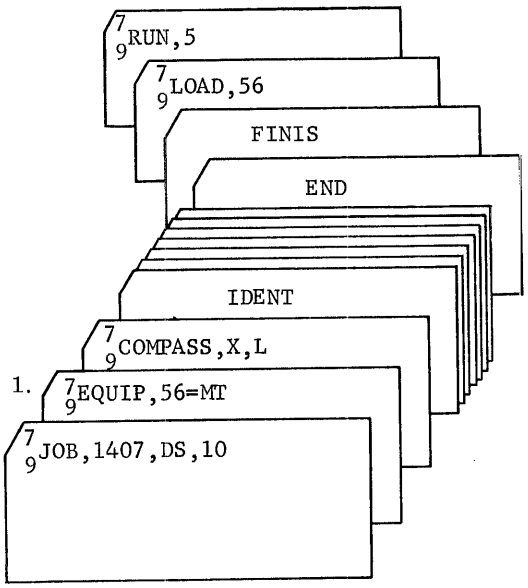
Assigns I/O devices as directed

The arrangement and content of the control statements indicate to SCOPE the manner in which a job is to be processed. Jobs are submitted on the standard input unit; control statements may be entered via the standard input unit or the comment from operator unit.

A job is a closed unit of processing for a single account. A run is a single program or library program execution.

The examples illustrate the deck arrangement for assembly and execution of COMPASS programs. Non-stacked jobs are not considered.

A subprogram must begin with an IDENT card and terminate with an END card. Any number of subprograms may follow the 7/9COMPASS card. The FINIS card must terminate the deck of subprograms to be assembled.



Execute

Load binary object program from unit 56

End of assembly deck

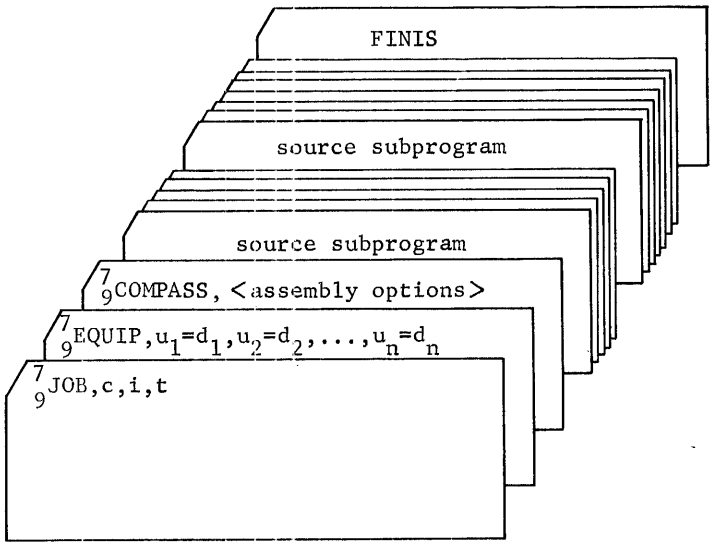
COMPASS source program. Each must begin with an INDENT card and terminate with an END card

Define logical unit 56, the standard load-and-go unit, as magnetic tape

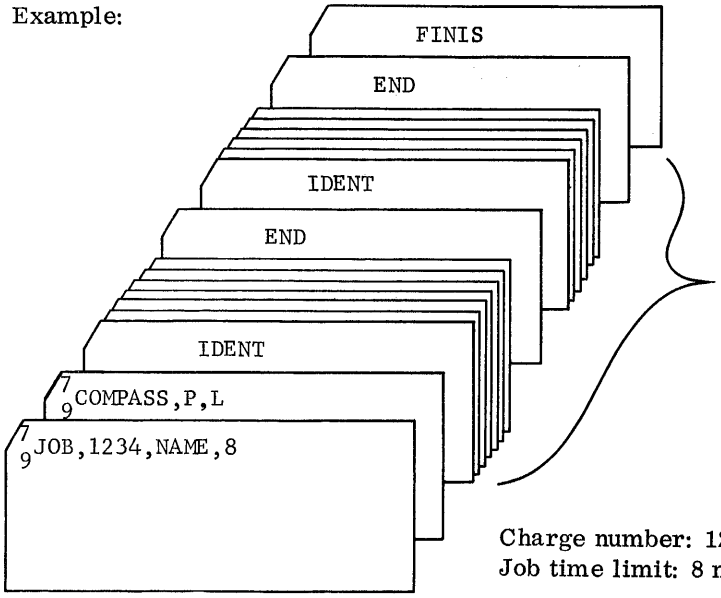
1. Assemble and write the binary object program on load-and-go unit (56), and list the source and object program on the standard output unit.

ASSEMBLY ONLY Assemble a COMPASS program or subprogram.

General Structure:



Example:



FINIS card signals end of assembly. Punch starting in column 10.

2 COMPASS source language subprograms, each terminated by END.

Charge number: 1234. Programmer: NAME
Job time limit: 8 minutes.

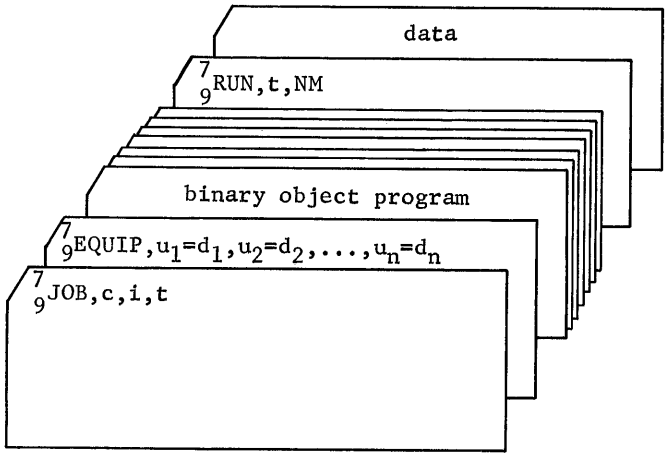
7/9 COMPASS, P, L

Source input on logical unit 60. Binary deck punched on logical unit 62. Source and object programs listed on logical unit 61.

The EQUIP statement is not required because system units 60, 61, and 62 are defined on the system library.

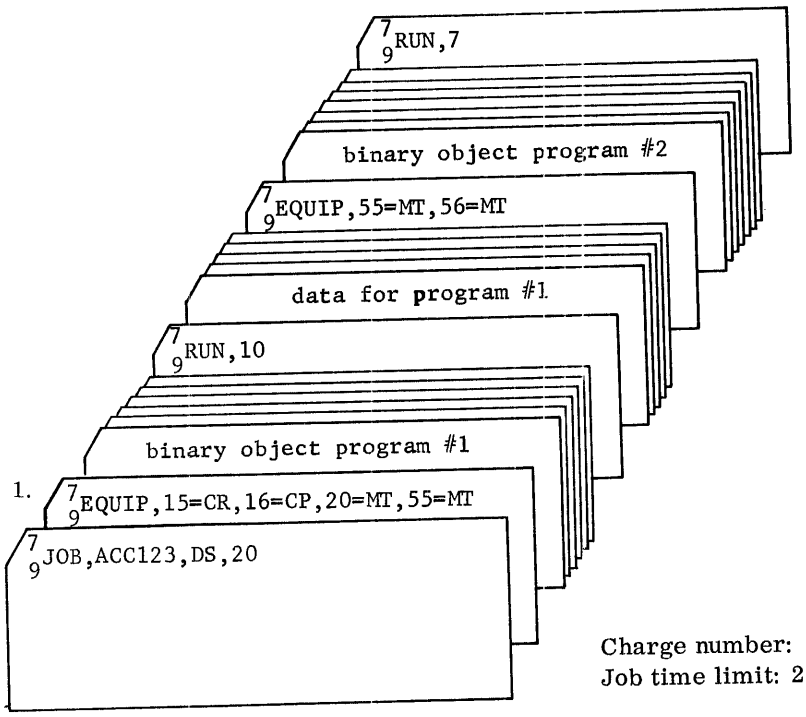
EXECUTION ONLY Execute directly from standard input unit.

General Structure:



If data is included on the standard input unit, it must follow the RUN card. Several executions may be performed in the same job. Programmer units must be defined prior to their use; scratch units must be defined prior to each run in which they are used.

Example:



Execution time limit: 7 minutes. Map on standard output unit

Scratch units 55 and 56 defined as magnetic tape for second run.

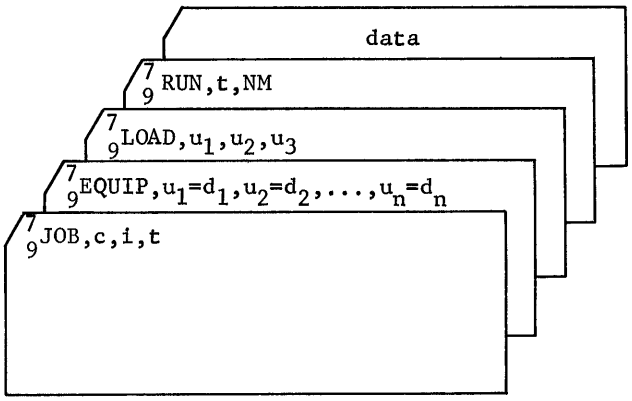
Execute the first object program. Time limit: 10 minutes. Memory map is written on standard output unit.

Charge number: ACC 123. Programmer: DS. Job time limit: 20 minutes.

1. Logical unit 15 defined as card reader, 16 as card punch, 20 as magnetic tape. These programmer units remain defined for the job. Scratch unit 55 is defined as magnetic tape for the first run only; it is rewound at end of first run.

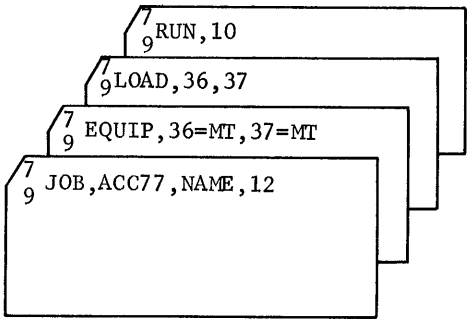
EXECUTE FROM A PROGRAMMER UNIT

General Structure:



Subprograms may be loaded from different programmer units, but only one LOAD card may appear. Equipment used in LOAD must be defined. If data is to be included on the standard input unit, it follows RUN card.

Example:



Execute

Define programmer units 36 and 37 as magnetic tape.

Job time limit: 12 minutes

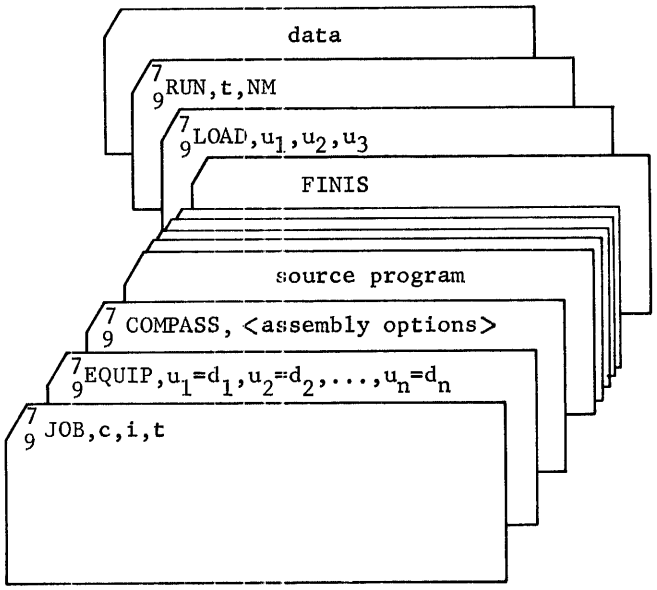
7/9 EQUIP

Load binary subprograms from logical unit 36 until end-of-file. Load binary subprograms from logical unit 37, until end-of-file.

ASSEMBLY AND EXECUTION

Assemble a COMPASS program and execute it immediately.

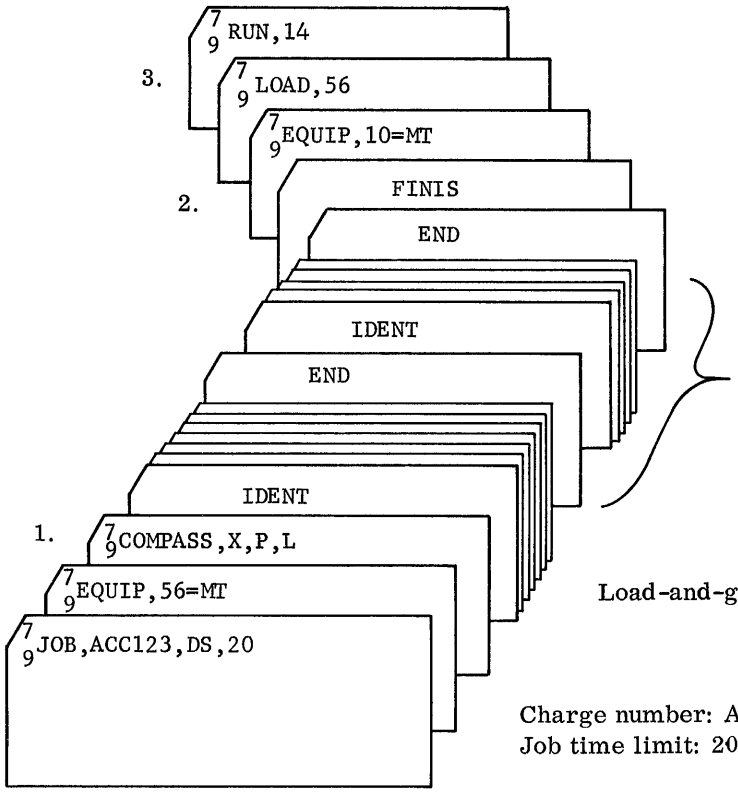
General Structure:



If the data is to be included on the standard input unit, it must follow the RUN card. For assembly only, any number of subprograms may follow the COMPASS card terminated by FINIS.

Although the load-and-go unit is a system unit, it must be defined by an EQUIP card.

Example:



Execution time: 14 minutes, map on standard output unit.

COMPASS source language subprograms, each terminated by an END card.

Load-and-go unit, 56, is defined as magnetic tape.

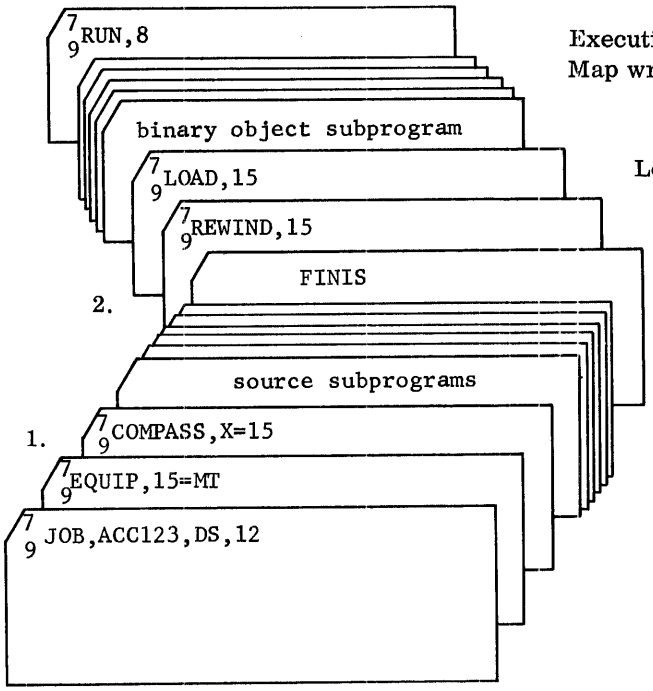
Charge number: ACC 123. Programmer: DS.
Job time limit: 20 minutes.

1. Source input on logical unit 60. Binary object program written on load-and-go unit 56. Binary deck punched on unit 62. Source and object programs listed on unit 61.

2. FINIS card signals end of assembly. Punch starting in column 10. End-of-file mark is written after last subprogram on unit 56.

3. Rewind load-and-go unit, 56 and load binary subprograms from it until end-of-file. It is available as a scratch unit in the following run.

Example:



Execution time limit: 8 minutes.
Map written on standard output unit.

Load binary subprograms from unit 15.

Rewind programmer unit 15.

Programmer unit 15 is defined as magnetic tape.

Charge number: ACC 123. Programmer: DS.
Job time limit: 12 minutes.

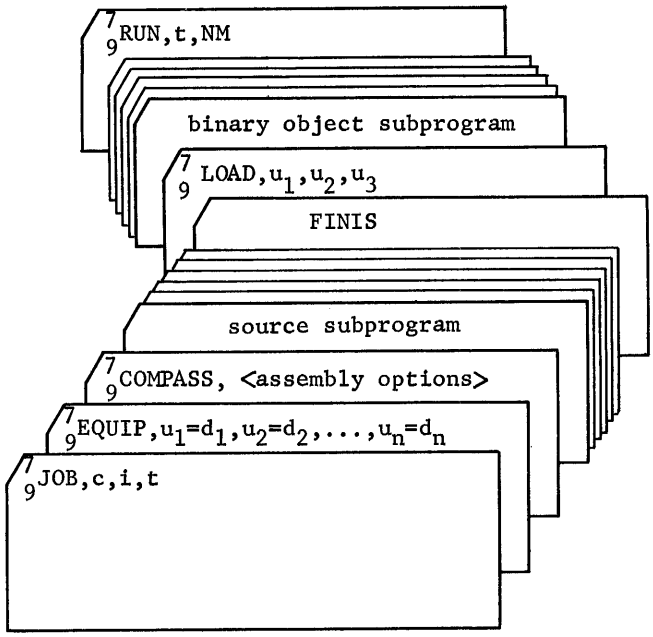
1. 7/9 COMPASS, X = 15
   Source input is on logical unit 60. Binary object subprograms are written on programmer unit 15.

2. FINIS card signals end of assembly. Punch starting in column 10. An end-of-file mark is written after the last subprogram on unit 15.

ASSEMBLE AND EXECUTE

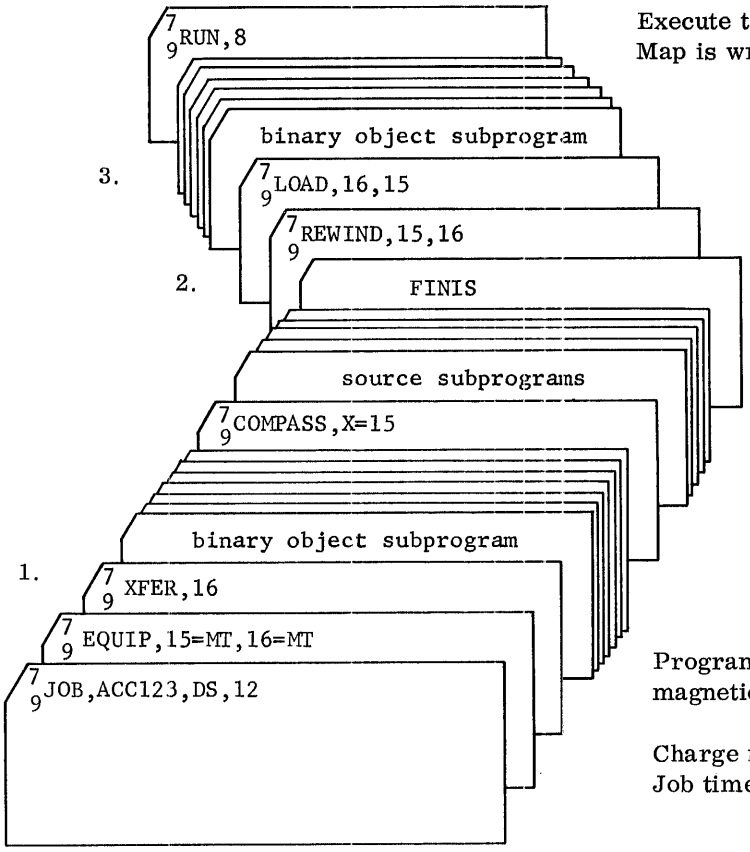
Include previously assembled binary decks in the program.

General Structure:



After assembly is completed, the subprograms may be loaded in any order. Binary object subprograms on the standard input must follow the LOAD card. Linkage does not occur until all subprograms are loaded. If a programmer unit is chosen for the load-and-go option, it must be rewound by the REWIND statement before it is loaded.

Example:



Execute time limit: 8 minutes.
Map is written on standard output unit.

Source input is on logical unit 60.
Binary object subprograms are written on programmer unit 15.

Programmer units 15 and 16 are defined as magnetic tape.

Charge number: ACC 123. Programmer: DS.
Job time limit: 12 minutes.

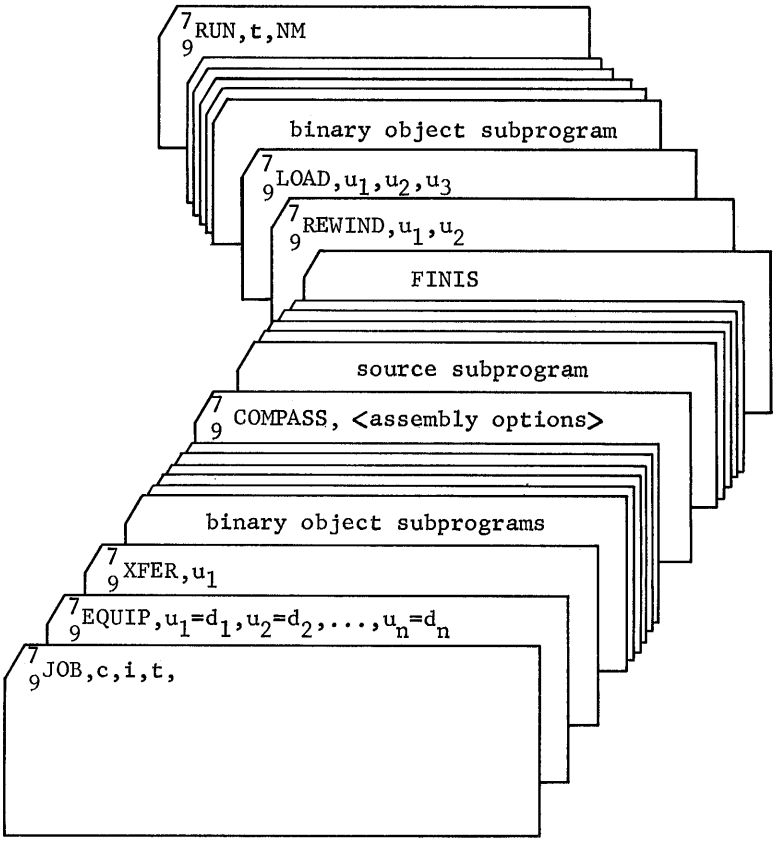
1. Transferred to logical unit 16. Binary information up to COMPASS control card is transferred to unit 16; end-of-file mark is written and back-spaced over.

2. FINIS signals end of assembly. Punch starting in column 10. An end-of-file mark is written after the last subprogram on unit 15. COMPASS source language subprograms, each terminated by an END card.

3. Load binary subprograms from unit 16, until end-of-file; then from unit 15 until end-of-file. Rewind programmer units 15 and 16.

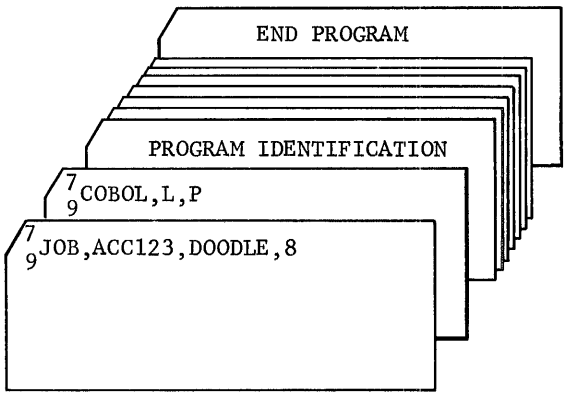
Binary object subprograms may be transferred from the standard input unit to another logical tape unit. If this unit is a programmer or scratch unit, it must be rewound before it can be loaded.

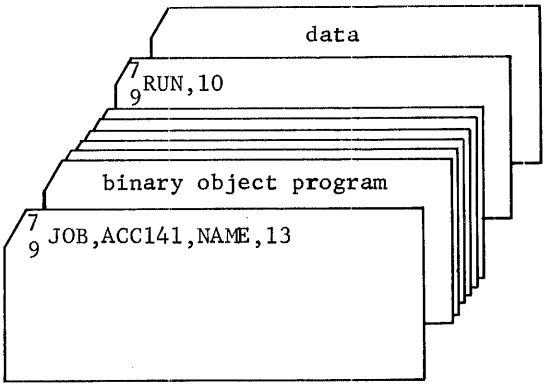
General Structure:



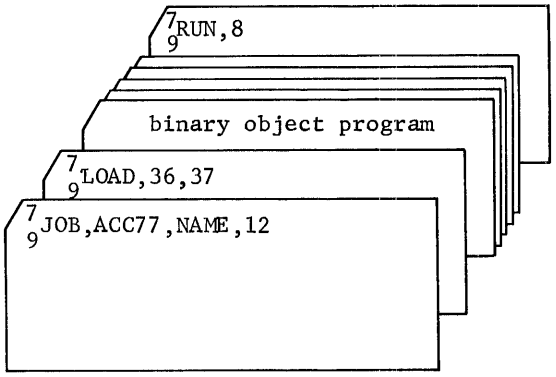
Compilation of a Single COBOL Program

The COBOL source program and binary object program are listed on OUT. The binary object program is also punched on PUN.



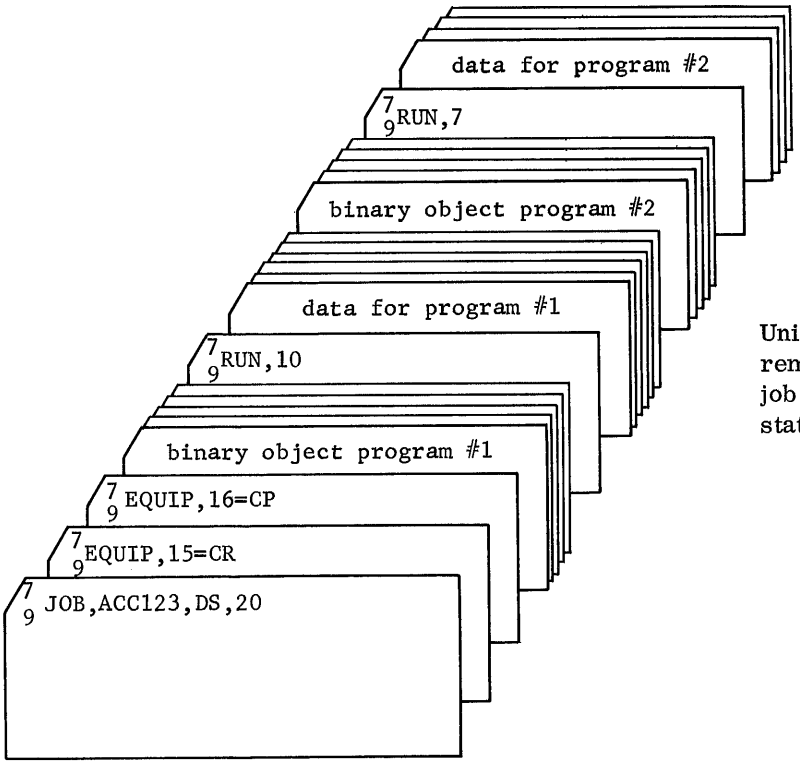


Execute directly from the standard input unit.



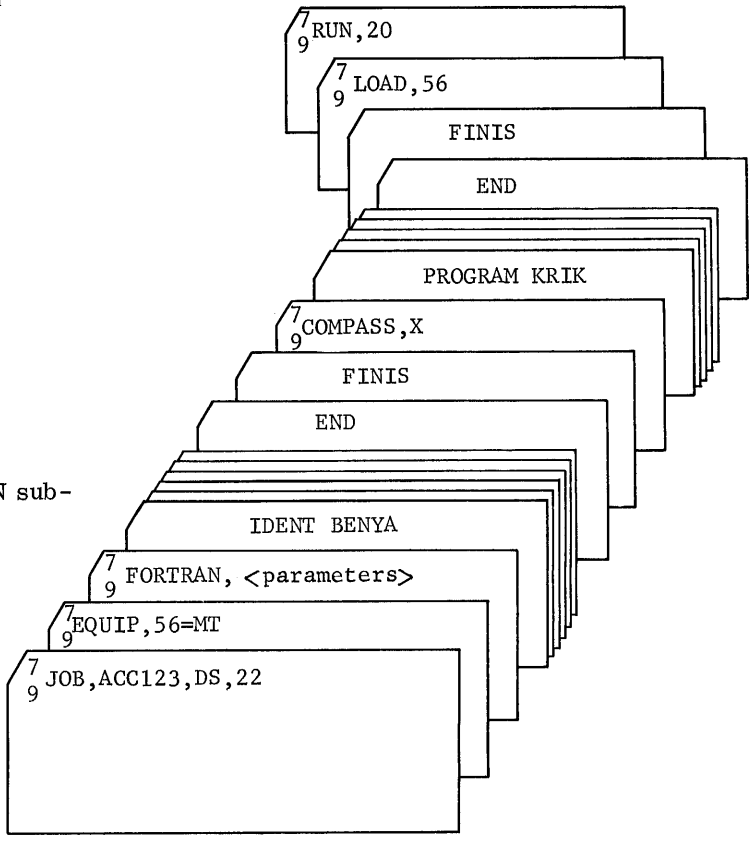
Load from two programmer units and INP.

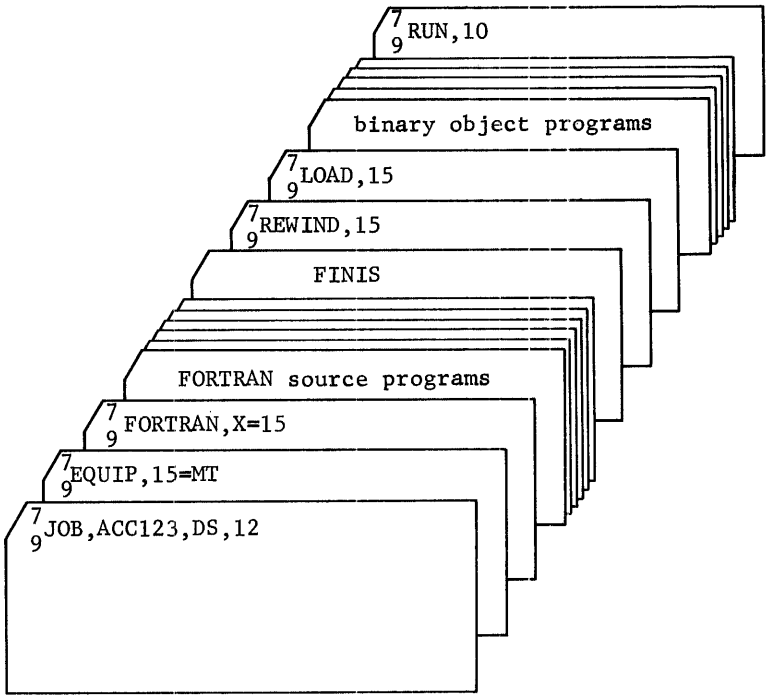
Sequential execution using EQUIP statements



Unit assignments by EQUIP statements remain in effect for the duration of the job or until changed by another EQUIP statement.

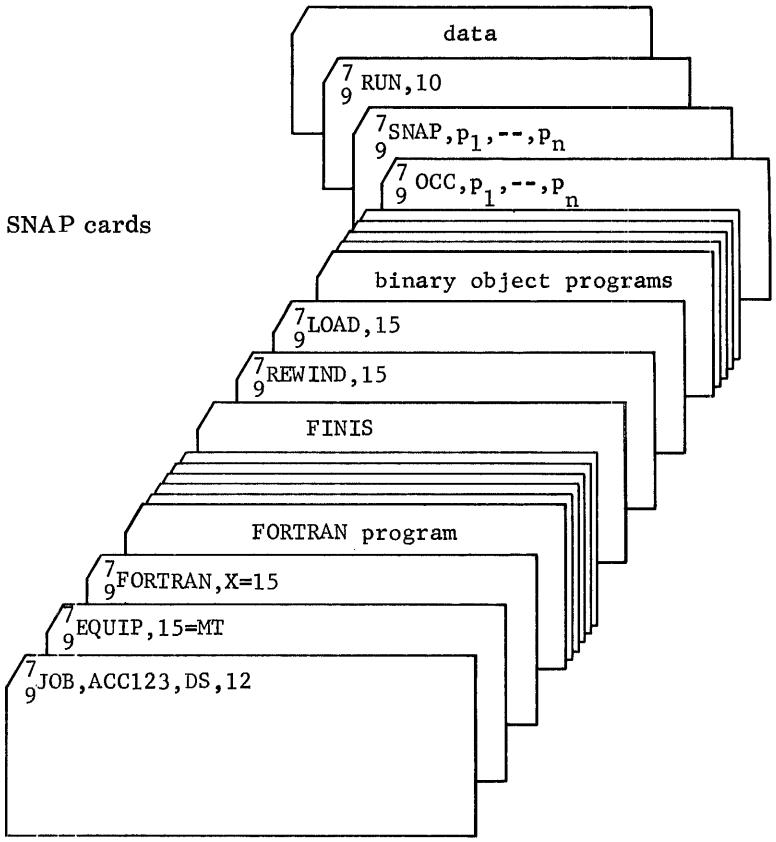
Assemble COMPASS and FORTRAN subprograms. Execute the program.





Compile and load a FORTRAN program. Load a binary subprogram from INP and execute.

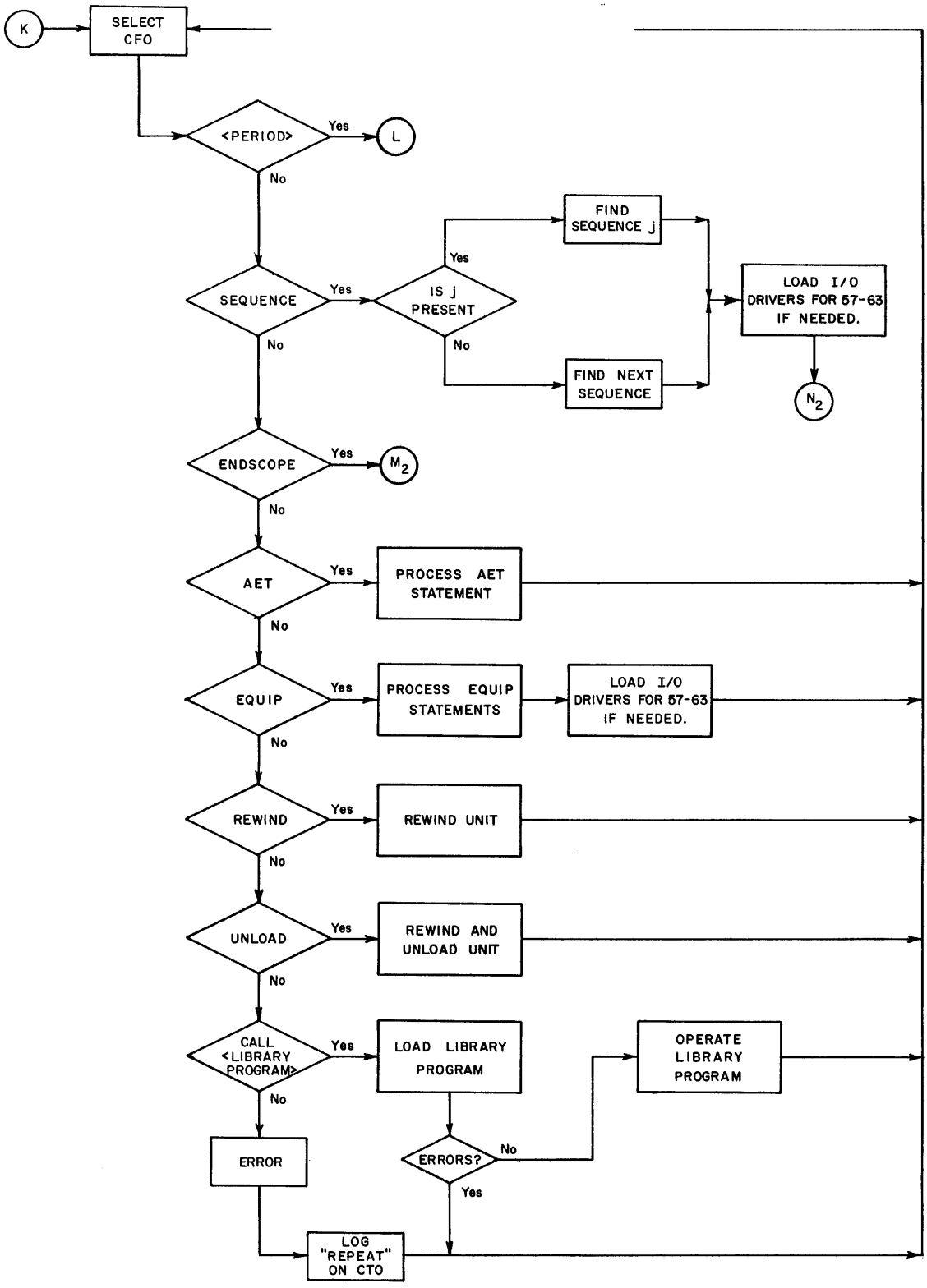
Use of debugging aids. OCC and SNAP cards precede RUN.



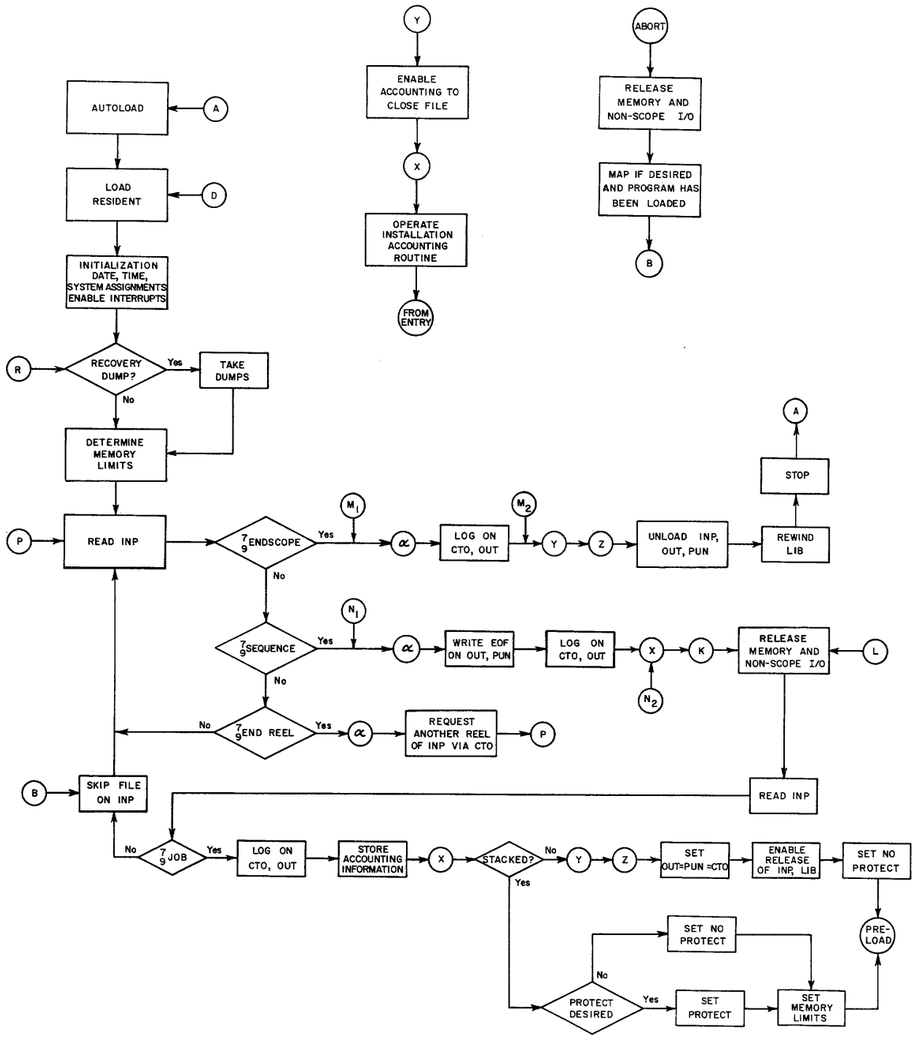
APPENDIX G
STANDARD LABELS 3000 SERIES MAGNETIC TAPE

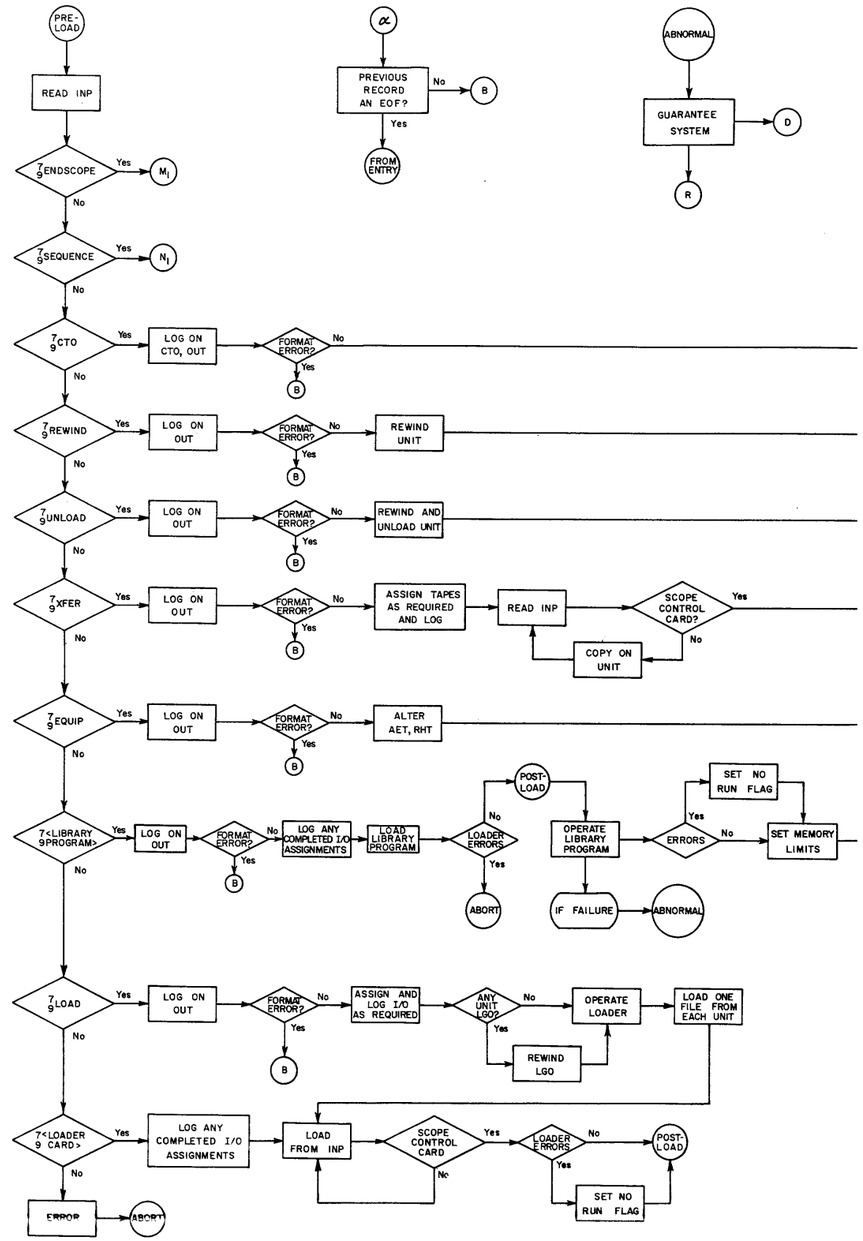
Character Position	
1	recording density (2, 5, or 8, indicating 200, 556, or 800 bpi)
2-3	unique label identifier - ()
4-5	logical unit number - 2 BCD digits
6-8	retention cycle - 3 BCD digits
9-22	file name - 14 alphanumeric characters
23-24	reel number - 2 BCD digits
25-30	date written - Month, Day, Year in BCD (MMDDYY) File ID
31-32	edition number - 2 BCD digits
33-80	user supplied information

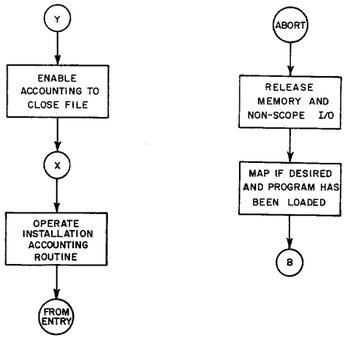
APPENDIX H

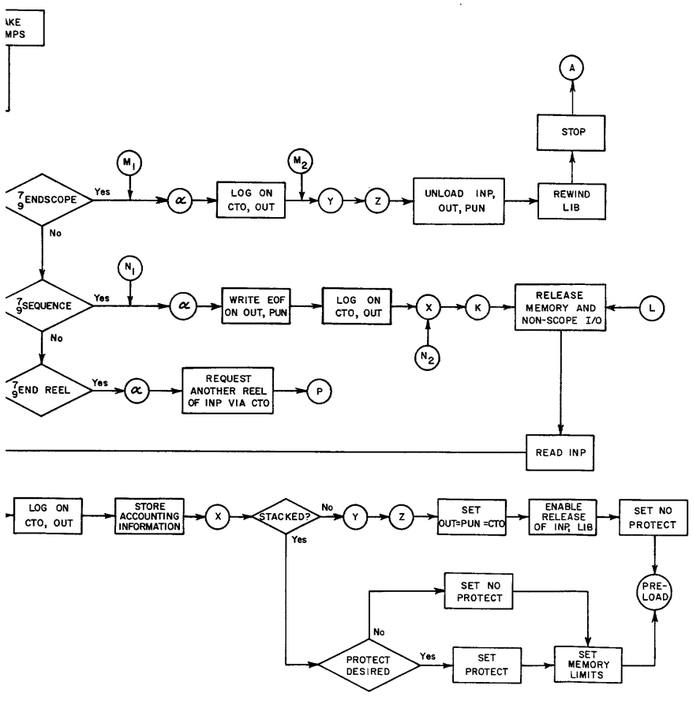


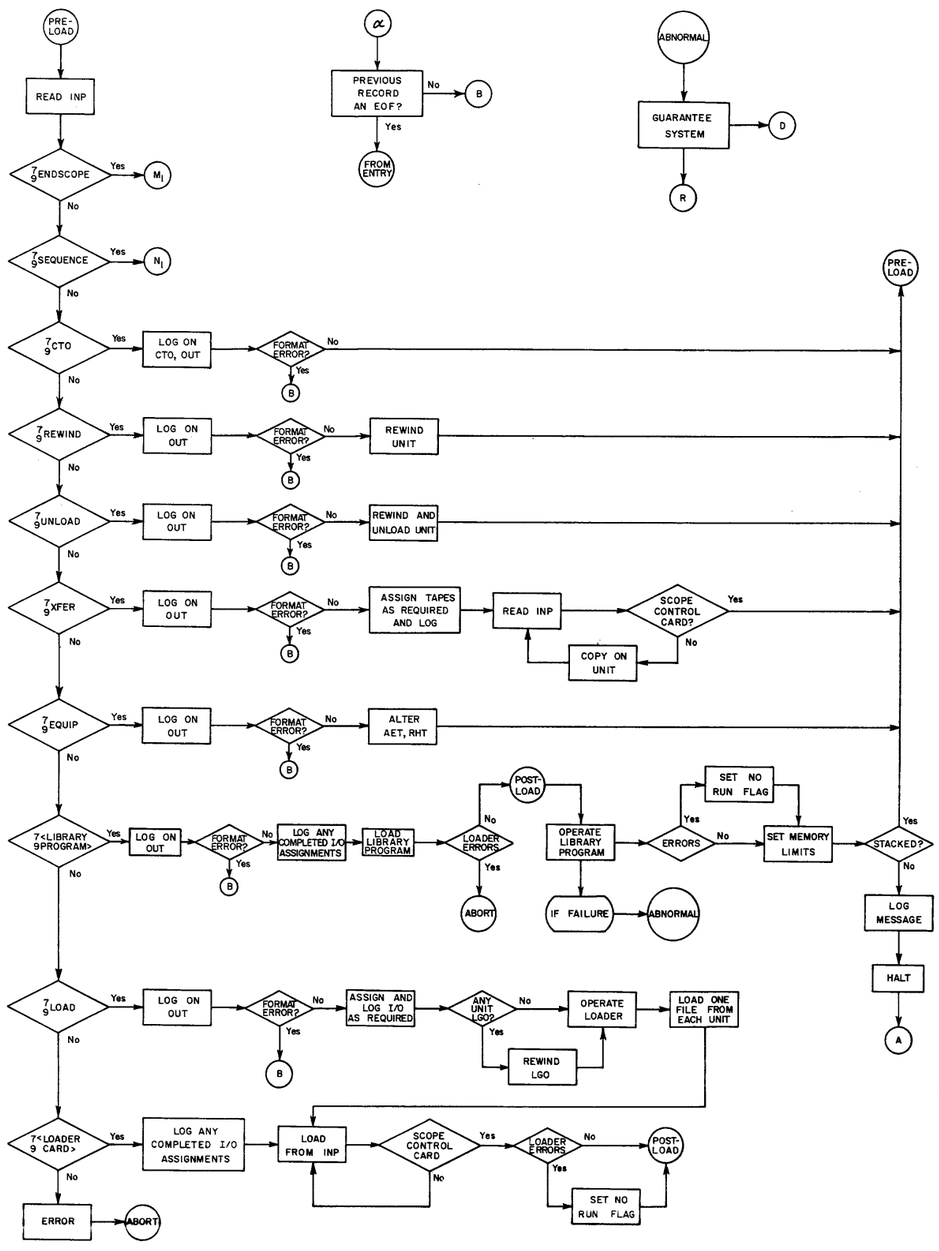
OPERATOR CONTROL STATEMENTS

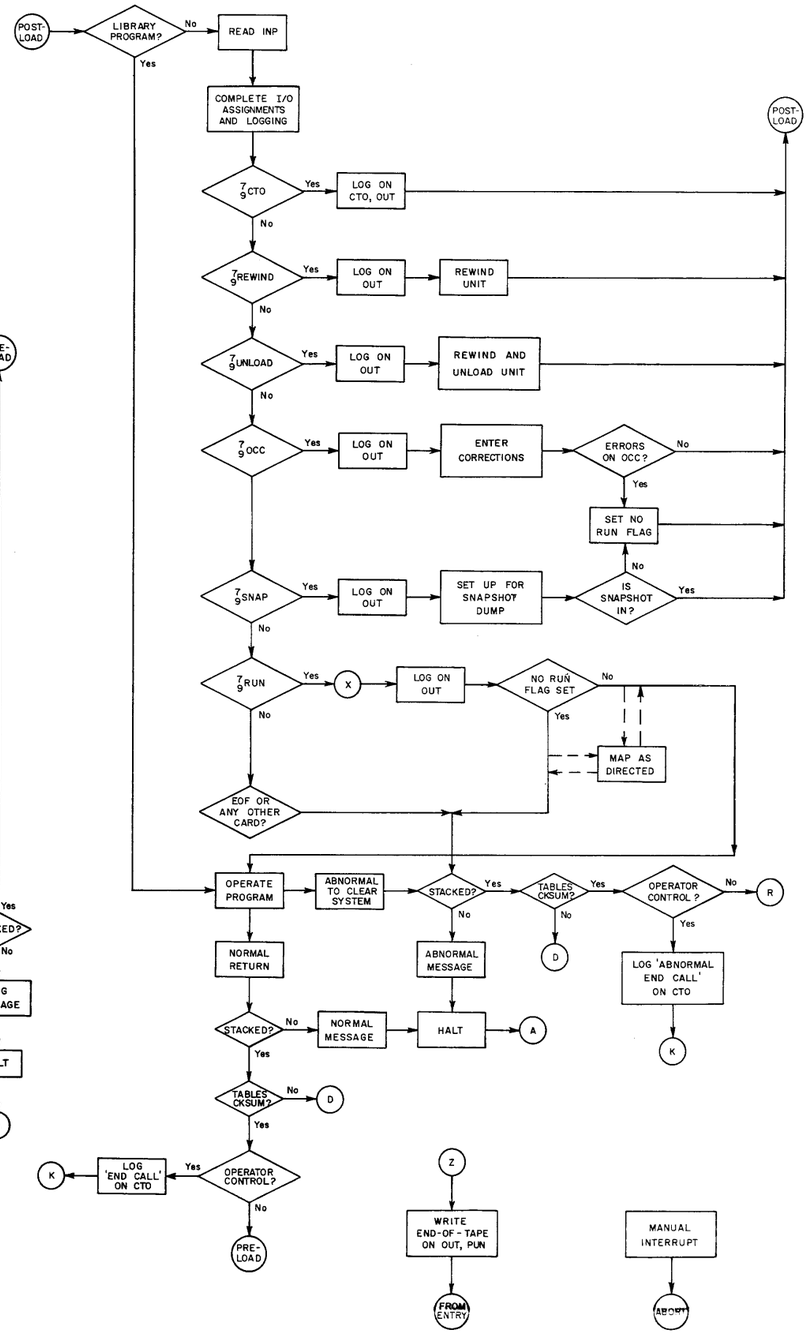

AUTOLOAD
A
LOAD RESIDENT
D
INITIALIZATION DATE, TIME, SYSTEM ASSIGNMENTS ENABLE INTERRUPTS
R
RECOVERY DUMP?
Yes
TAKE DUMPS
No
DETERMINE MEMORY LIMITS
P
READ INP
7 9 ENDSCOPE
Yes
M1
α
LOG ON CTO, OUT
M2
Y
Z
UNLOAD INP, OUT, PUN
STOP
A
REWIND LIB
No
N1
7 9 SEQUENCE
Yes
α
WRITE EOF ON OUT, PUN
LOG ON CTO, OUT
X
K
RELEASE MEMORY AND NON-SCOPE I/O
L
N2
No
7 9 END REEL
Yes
α
REQUEST ANOTHER REEL OF INP VIA CTO
P
No
B
SKIP FILE ON INP
READ INP
No
7 9 JOB
Yes
LOG ON CTO, OUT
STORE ACCOUNTING INFORMATION
X
STACKED?
No
Y
Z
SET OUT=PUN =CTO
ENABLE RELEASE OF INP LIB
SET NO PROTECT
PRE-LOAD
Yes
SET NO PROTECT
No
PROTECT DESIRED
Yes
SET PROTECT
SET MEMORY LIMITS
Y
ENABLE ACCOUNTING TO CLOSE FILE
X
OPERATE INSTALLATION ACCOUNTING ROUTINE
FROM ENTRY
ABORT
RELEASE MEMORY AND NON-SCOPE I/O
MAP IF DESIRED AND PROGRAM HAS BEEN LOADED
B


PRE-LOAD
READ INP
7 9 ENDSCOPE
Yes
M1
No
7 9 SEQUENCE
Yes
N1
No
7 9 CTO
Yes
LOG ON CTO, OUT
FORMAT ERROR?
No
Yes
B
No
7 9 REWIND
Yes
LOG ON OUT
FORMAT ERROR?
No
REWIND UNIT
Yes
B
No
7 9 UNLOAD
Yes
LOG ON OUT
FORMAT ERROR?
No
REWIND AND UNLOAD UNIT
Yes
B
No
7 9 XFER
Yes
LOG ON OUT
FORMAT ERROR?
No
ASSIGN TAPES AS REQUIRED AND LOG
READ INP
SCOPE CONTROL CARD?
Yes
No
COPY ON UNIT
Yes
B
No
7 9 EQUIP
Yes
LOG ON OUT
FORMAT ERROR?
No
ALTER AET, RHT
Yes
B
No
7<LIBRARY 9PROGRAM>
Yes
LOG ON OUT
FORMAT ERROR?
No
LOG ANY COMPLETED I/O ASSIGNMENTS
LOAD LIBRARY PROGRAM
LOADER ERRORS
No
Yes
ABORT
POST-LOAD
OPERATE LIBRARY PROGRAM
ERRORS
Yes
No
IF FAILURE
ABNORMAL
SET NO RUN FLAG
SET MEMORY LIMITS
Yes
B
No
7 9 LOAD
Yes
LOG ON OUT
FORMAT ERROR?
No
ASSIGN AND LOG I/O AS REQUIRED
ANY UNIT LGO?
No
Yes
REWIND LGO
OPERATE LOADER
LOAD ONE FILE FROM EACH UNIT
Yes
B
No
7<LOADER 9 CARD>
Yes
LOG ANY COMPLETED I/O ASSIGNMENTS
LOAD FROM INP
SCOPE CONTROL CARD
Yes
No
LOADER ERRORS
No
Yes
POST-LOAD
SET NO RUN FLAG
No
ERROR
ABORT
α
PREVIOUS RECORD AN EOF?
No
B
Yes
FROM ENTRY
ABNORMAL
GUARANTEE SYSTEM
D
R









3200 SCOPE GENERALIZED FLOW

CONTROL DATA SALES OFFICES

ALAMOGORDO • ALBUQUERQUE • ATLANTA • BILLINGS • BOSTON • CAPE CANAVERAL • CHICAGO • CINCINNATI • CLEVELAND • COLORADO SPRINGS DALLAS • DAYTON • DENVER • DETROIT • DOWNEY, CALIFORNIA • HONOLULU HOUSTON • HUNTSVILLE • ITHACA • KANSAS CITY, KANSAS • LOS ANGELES MADISON, WISCONSIN • MINNEAPOLIS • NEWARK • NEW ORLEANS • NEW YORK CITY • OAKLAND • OMAHA • PALO ALTO • PHILADELPHIA • PHOENIX PITTSBURGH • SACRAMENTO • SALT LAKE CITY • SAN BERNARDINO • SAN DIEGO • SEATTLE • ST. LOUIS • WASHINGTON, D.C.

ATHENS • BOMBAY • CANBERRA • DUSSELDORF • FRANKFURT • HAMBURG • JOHANNESBURG • LONDON • MELBOURNE • (REGAL ELECTRONICA DE MEXICO, S.A.) • MILAN • MONTREA OSLO • OTTAWA • PARIS • STOCKHOLM • STUTTGART • SYDNE TOKYO (C. ITOH ELECTRONIC COMPUTING SERVICE CO., LTD.) ZURICH

8100 34th AVE. SO., MINNEAPOLIS, MINN. 55440

CONTROL DATA CORPORATION

Pub. No. 60057700