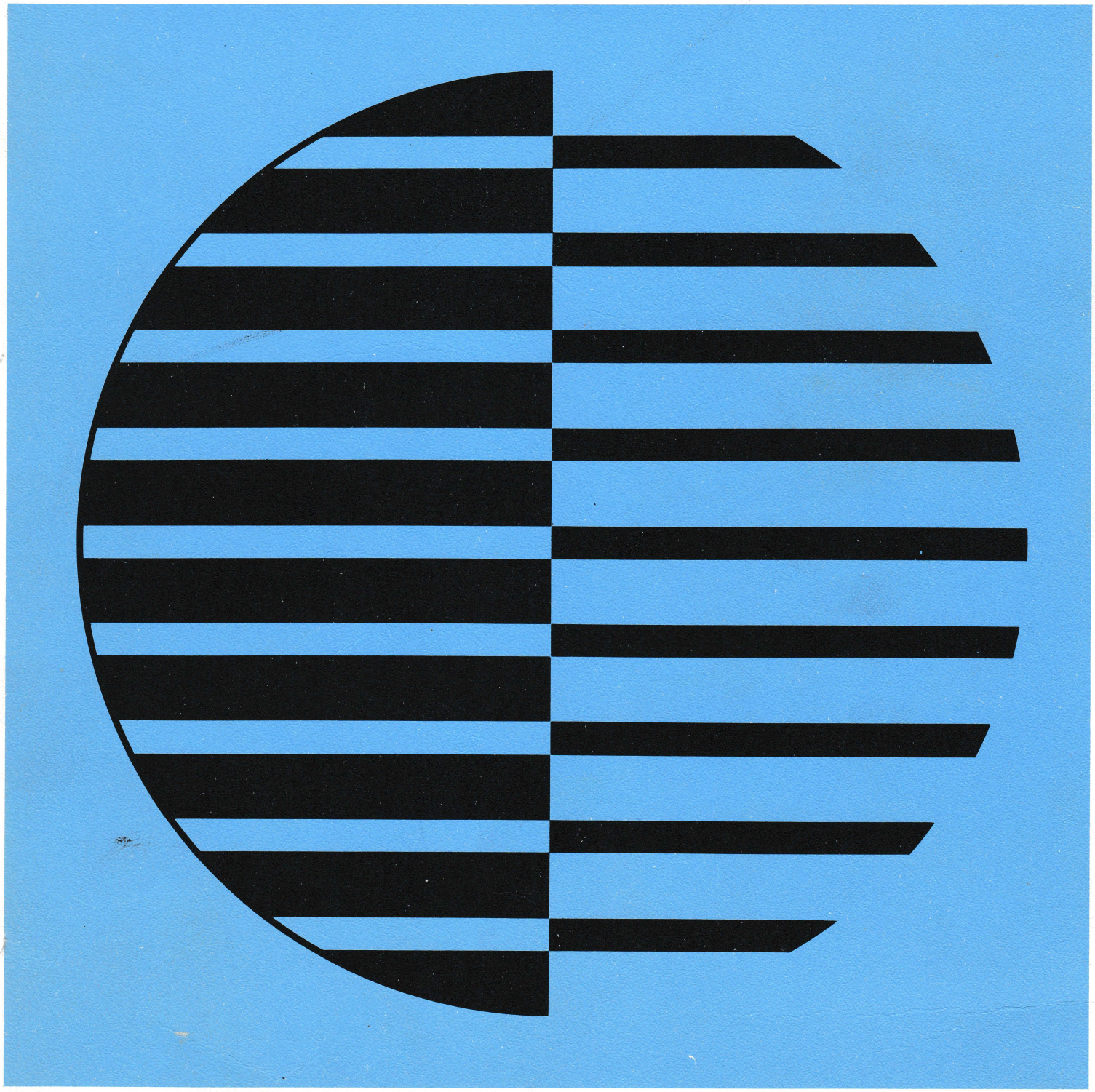
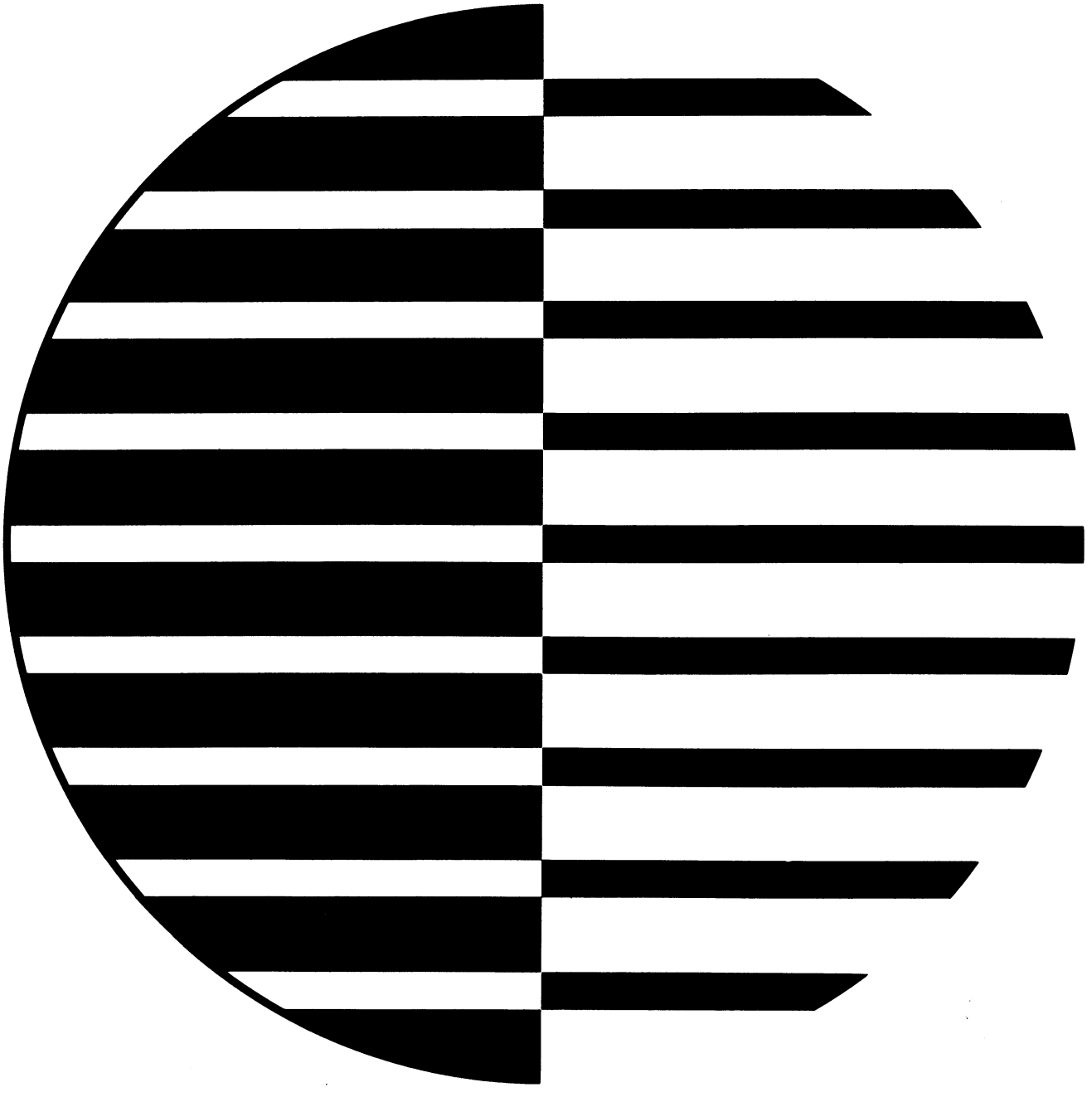
CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
COMPASS Reference Manual



CONTROL DATA®
6400/6500/6600 COMPUTER SYSTEMS
COMPASS Reference Manual



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

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

April, 1967
Publication No. 60190900

CONTENTS

INTRODUCTION

6400/6500/6600 COMPASS, a comprehensive assembly system, provides a symbolic program language for the CONTROL DATA®6400/6500/6600 computers. COMPASS is designed for efficient utilization of all computer resources and maximum flexibility in program construction.

The COMPASS language allows all hardware functions of the 6400, 6500, and 6600 computers to be expressed symbolically. In addition, many features are included which enable the programmer to control the assembly process itself. These include:

- Free-field source format
- Assembly-time access to symbol table information
- Programmer control over local and common code blocks
- Macros (both programmer and system defined)
- OPDEF, a special macro form for redefining machine mnemonics
- Micro coding

COMPASS operates under control of the SCOPE monitor system; a minimum SCOPE configuration permits full use of all COMPASS features.

PROGRAM STRUCTURE AND ORGANIZATION 1

1.1 CENTRAL AND PERIPHERAL PROCESSOR CODING

A COMPASS subprogram consists of either central processor (CP) code or peripheral processor (PP) code. The machine instructions for the two processors are different, but most of the pseudo instruction set described in Chapter 3 is used in both CP and PP subprograms. The pseudo instructions may differ in specification, significance, or result, according to whether the subprogram is a CP or PP subprogram; these differences are noted in the pseudo instruction description. CP and PP machine instructions may not be intermixed within a subprogram.

1.2 SUBPROGRAM STRUCTURE

The programmer or COMPASS assigns to each subprogram one or more blocks into which all code is assembled. These code blocks may be local or common. A local block contains code accessible to the subprogram only; a common block contains code accessible to all subprograms loaded together. A program may use a maximum of 252 local and common blocks in addition to those defined by the assembler.

As assembly proceeds, all locations and references to locations within a block are considered relative to the start of that block. COMPASS maintains the origin of each block, the current position within each block, and the final length of each block. The programmer may manipulate origin, location, and position counters to control his position. At the end of assembly, COMPASS assigns an origin, relative to the start of the first program block, to each local block in the order in which its name was introduced. The length of a subprogram is the sum of the last origin counter values of all local blocks.

Refer to the USE pseudo instruction in Chapter 3 for rules on programmer control of block usage.

1.2.1 LOCAL BLOCKS

Code within local blocks is accessible only to the subprogram itself. Three local blocks are pre-defined by COMPASS in every subprogram; they need not be declared by the programmer:

Absolute block, used for all absolute code

Zero block, used by COMPASS when no programmer-assigned block is specified in a relocatable CP assembly

Literals block which contains all literal data values

The zero block is the nominal block for all relocatable subprograms; the absolute block is the nominal block for all absolute subprograms as well as the block for absolute origins in relocatable subprograms. PP subprograms are always absolute; CP subprograms may be absolute or relocatable. All code in a subprogram will be in either the zero block or the absolute block, unless the programmer requests or uses another block. The programmer may refer to the zero block in a USE statement, but he may not refer to the absolute block except with the ORG statement.

All data literals are assigned to the literals block, which may not be referenced by the programmer. The literals block, at the end of assembly, is assigned an origin at the end of the zero block.

The programmer may define and use additional local blocks with the USE statement. Named local blocks are considered extensions of the zero block; they are assigned origins by COMPASS at the end of the zero block (after any literals), in the order in which they were declared.

1.2.2 COMMON BLOCKS

Code assigned to common blocks is accessible by all subprograms loaded together. Common blocks are assigned origins by the loader at load time (unlike local blocks which are assigned origins by COMPASS at assembly time). They may be labeled common or blank common. Labeled common includes blocks designated with a numeric name.

Data may be pre-loaded into labeled common, but not into blank common. Space may be reserved in blank common by using only the BSS or ORG pseudo instructions.

1.3 COUNTERS

Origin, location, and position counters are maintained by COMPASS to define the location of code and the current position within a word. These counters may be reset by certain pseudo instructions, and their values may be tested at any point.

1.3.1 ORIGIN COUNTER

The origin counter is maintained by COMPASS to indicate the location where instructions will be placed by the loader. For each block, the origin counter starts at zero relative to the block origin, or at the last known size of that block if it has been previously used.

The origin counter is incremented by one for each completed word of assembled data. Its value may be reset with the ORG pseudo instruction. The current value of the origin counter is returned when the special element *O is used in an instruction.

1.3.2 LOCATION COUNTER

The location counter, under normal circumstances, has a value identical to the origin counter and gives definition to location symbols. It may be adjusted, however, to differ from the origin counter if succeeding data is to be executed in a memory area different from its assigned load time area. For example, a block might be loaded in ECS, subsequently moved, and executed in another area. In that case, the location counter should reflect the location at which execution will occur.

The location counter may be reset with the LOC pseudo instruction. The current location counter value is returned when either of the special elements * or *L is used.

1.3.3 POSITION COUNTER

This counter maintains a position within a 60-bit or 12-bit word of assembly. As each code-generating instruction is encountered, the position counter is updated to reflect the next available bit position. The position counter contains the number of the high-order bit of the field, numbered 59 (sign) to 0. When used in CP instructions, it will have a value of 59, 44, 29, or 14. In PP instructions, 11 is the normal value. Other values are possible in VFD instructions.

The current position counter value is returned whenever the special element $ is used in an instruction.

1.4 FORCING UPPER

In central processor assemblies, assembled data is packed sequentially into a 60-bit word in bytes of 15, 30, or 60 bits (except for VFD). If there is not room in a partially filled 60-bit word for the instruction or data currently being evaluated, the remainder of that word is filled with 15-bit no-operation instructions (46000_8), and the current instruction is assigned the first position in the next word.

COMPASS also forces upper when any of the following occurs:

- A symbol or + appears in the location field of the current statement
- Current instruction is RE, WE, or XJ, unless there is a minus sign in the location field
- Current instruction is one of the pseudo instructions END, LOC, BSS, BSSZ, DATA, or DIS. ORG also forces upper in the block which it references (ORG, section 3.2.2).

Forcing upper is automatic after JP, RJ, PS, and an EQ or ZR with a single address (the unconditional EQ or ZR). The ECS instructions WE and RE must appear in the upper 30 bits of an instruction; and when executed successfully, execution continues at the beginning of the next 60-bit word. The lower half of the WE or RE word presumably contains a jump to an error routine to be taken if the WE or RE is rejected. COMPASS will not force upper after WE or RE.

In a PP assembly, there is no forcing upper, and the location field entry + will be ignored except on a VFD line, in which case the position counter will be reset to the beginning of a PP word.

Automatic forcing upper after JP, RJ, PS, EQ, and ZR as well as forcing upper on RE, WE, or XJ can be negated by using a minus sign in the location field of the next instruction. When a minus sign appears, the current line will be assembled into the next position large enough to contain it.

COMPASS LANGUAGE CODING 2

2.1 SOURCE STATEMENTS

2.1.1 FORMAT

A COMPASS program consists of a sequence of symbolic statements; each statement contains a maximum of four fields in the order listed below. The format is essentially free-field.

Location Field must begin in column 1 or 2

Operation Field may begin in any column from 3 to 35

Variable Field must begin before column 36

Comments Field may begin after the termination of the variable field, or no earlier than column 36 if the variable field is empty.

Columns 73-90 may be used only for comments; they are typically used for sequencing.

Fields are separated by one or more blanks. Blanks are interpreted as field separators except when embedded in the comments field, in character data items, or in a parenthesized macro parameter.

A statement may be a comment or an instruction; it may contain as many as ten 90-column lines. Column 1 indicates the type of line being introduced: an asterisk identifies a comment statement; a comma indicates that this line is a continuation of the previous line and part of the same statement; any other character, including blank, indicates the beginning of a new statement.

A comment statement may be introduced in two ways: by an asterisk in column 1, or by a line which is blank in columns 1-35. The comment lines are listed in the assembler output and they have no other effect on assembly.

A line introduced by a column 1 comma is considered a continuation of the preceding line. A maximum of 9 continuation lines are permitted. Column 2 of each continuation line is interpreted as an immediate continuation of column 72 of the preceding line. The break between lines need not be a field or subfield separator; even a symbol may be split between the two lines. If more than 9 continuation lines appear, the 10th and subsequent lines are considered comments.

A line with a location field entry but no operation field entry creates a word of zeros, and is equivalent to the instruction:

```
loc   BSSZ   1
```

Example of a standard format for source lines:

Column	
1	Blank, asterisk, or comma
2-9	Location field, left justified
10	Blank
11-16	Operation field, left justified
17	Blank
18-	Variable field, terminated by 1 or more blanks
36-	Comments field

2.1.2 CATENATION AND MICRO SUBSTITUTION

Any line not containing a column 1 asterisk is examined for the two special characters → and ≠ before COMPASS attempts any other interpretation. COMPASS immediately takes the following action on the line: the → character indicates that the two adjoining columns are to be catenated (joined); the ≠ mark indicates micro substitution (Chapter 5). The line which has changed as a result of catenation or micro substitution may be any type: a comment line, an instruction, or a continuation of an instruction. Micro substitution might itself cause a continuation line to be produced.

2.1.3 TYPES OF STATEMENTS

Statements processed by COMPASS fall in three different categories:

- A normal statement which is assembled and which perhaps produces output.
- A statement which is bypassed because of a conditional instruction test which failed.
- A statement which is part of a definition: those lines contained between a MACRO and its ENDM, between a DUP and its ENDD, between a RMT and its terminating RMT.

The important difference to be noted is that statements which are part of definitions are not examined for the two special marks ≠ and →. This then implies that this editing occurs when macros, duplications, etc. are expanded rather than at the time they are defined. Thus, for example, one could not create an ENDM which would terminate a macro definition by using micro substitution or catenation.

Catenation and micro substitution do occur on lines which are being skipped.

2.2 ELEMENTS OF AN INSTRUCTION

Location Field

The location field may be blank or may contain one of the following:

Symbol

Name

+

-

Operation Field

The operation field must be present, and may contain one of the following:

Central processor operation code

Peripheral processor operation code

Pseudo instruction

Macro name

Variable Field

The contents of the variable field are dictated by the operation code. The variable field for COMPASS machine instructions consists of 1, 2, or 3 subfields, separated by commas. A subfield in CP instructions may contain register names separated by the operators + - * /. COMPASS must determine the octal value of the instruction from these operators, so they may not be substituted by any other characters.

Comments Field

This field is optional; it may contain any combination of characters. The catenation mark (→) and the micro mark (≠) produce the same results in the comments field as in any other field. (See 2.1.2.)

2.3 SYMBOLS

A symbol is a string of 1 to 8 characters representing a value. Symbol value is determined according to its use as follows:

- In the location field of a machine instruction and certain pseudo instructions, the value assigned to the symbol is the current value of the location counter.
- In the location field of an EQU or SET pseudo instruction, the value in the address field is assigned to the symbol.
- In a list of external symbols, the symbol is defined and a value is assigned in a subprogram external to this one.
- By default: if the symbol is preceded by =S or =X and has no other definition, COMPASS will define it.

Absolute symbols may be defined with the EQU or SET pseudo instructions, or as location symbols in code with an absolute origin; they are assigned a 21-bit value. Relocatable symbols are assigned a value which is relative to an unknown base address either in common storage or within the subprogram. For the purpose of symbol definition, relocatable symbols may exist in absolute code; this is true in blocks other than the zero block since COMPASS does not know the size of the zero block (therefore the origin of other blocks) until the end of assembly.

A symbol may not include any of the following characters:

* / , + - or blank

The first character may not be numeric, $ or =. Other special characters must be used with care. In CP programs, a decimal point will produce a register name if the decimal point is the second character and A, B, or X is the first. When any of the characters $. =) (are used in symbols which might appear as a macro parameter or in the body of a macro definition, unexpected results might occur. Refer to macro definition rules in Chapter 4.

A symbol in a CP assembly may not be An, Bn, or Xn, where n is a single digit between 0 and 7.

Examples of legal symbols:

A	A10	A1.75
A=B	AAAAAAAA	A(B)
ABCDEF.3	A$$$	.01

Some symbol names are further restricted if they are used as the following:

Subprogram names

External symbols

Entry points

Common block names

These are called linkage symbols since they are used by the loader. Such symbols must begin with a letter (A-Z), and may not exceed 7 characters in length. For PP subprograms, subprogram names may begin with a letter or a number, and may not exceed 3 characters.

2.4 REGISTERS

Register names are symbolic representations of the 24 operating registers of the computer. Register names are pre-defined in central processor COMPASS assemblies; they may not be redefined in the program. They are of two forms:

- The first form is represented by An, Bn, or Xn, where n is a single digit between 0 and 7. Any other term for n will be interpreted as a symbol, and not a register name.

- The second form is represented by A.n, B.n, or X.n, where n may be a single symbol or an integer. If the value of n is not 0-7, its value will be truncated to the low-order 3 bits and a warning flag will be issued.

Register names of either type are considered ordinary symbols in a peripheral processor assembly.

Examples:

A1	Accumulator register 1
A10	Symbol, not a register name
A.1	Accumulator register 1
A.10	Accumulator 2; produces a warning flag ($10_{10}=12_8$ which truncates to 2)

The following produce equivalent results:

```
SB3    A2+ALPHA        SUM     SET     3
                       SUB     SET     2
                               SB.SUM  A.SUB+ALPHA
```

2.5 DEFERRED SYMBOL DEFINITION

Definition of a symbol may be deferred until end of assembly. At that time, COMPASS will define all deferred symbols that have not been programmer-defined by conventional methods.

Deferred symbols may be indicated in an address expression by the form:

=Ssymbol a normal relocatable symbol

=Xsymbol an external symbol

The form =Ssymbol results in the following:

- If the symbol is not defined in the subprogram, it represents a location which COMPASS assigns at the end of the zero block. All subsequent references to that symbol, whether preceded by =S or not, are to that assigned location. Any symbol so defined may not be used where a previously defined symbol is required.

- If the symbol is defined in the subprogram, COMPASS will not define it again as a deferred symbol. The programmer-defined value of the symbol will be used instead.

The form =Xsymbol results in the following:

- If the symbol is not defined in the subprogram, the symbol is assumed to be external as though it were declared in an EXT pseudo instruction. It must conform to the rules for linkage symbols.

- If the symbol is defined in the subprogram, it represents whatever value the programmer has assigned. COMPASS will not define it again as a deferred external symbol.

If a symbol appears as both =S and =X, or as =X in an absolute assembly and has no other definition, it will be undefined and will produce an error.

2.6 NAMES

A name is a special kind of symbol which names one of the following items:

block

macro

instruction bracket

micro

These names do not conflict with ordinary symbols since they are used differently. Since names may not be used in address expressions, the rules for forming them are less strict than for ordinary symbols. A name may be formed of any combination of 1 to 8 characters except blank or comma.

Examples of legal names:

2	3A	A+B*C
X*Y/Z	$+A	=48
2+6	*LA$+SF	1.5

2.7 ABSOLUTE DATA

Absolute data is used in literals, LIT and DATA pseudo instructions, and in address expressions as constants. COMPASS supplies a format for data specification which is common to all these usages, with minor exceptions.

Data item describes an absolute item which produces one or more full-word values. The following are data items:

A subfield of the DATA pseudo instruction

A subfield of the LIT pseudo instruction

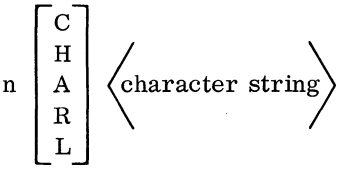
A literal (the portion which follows = if the item is not =Ssymbol or =Xsymbol)

Address constant describes a constant which may appear in an address expression. A constant may be up to a 60-bit value. Constants appear in machine and pseudo instruction subfields, including VFD.

Absolute data may be character data or numeric data; numeric data may be octal, decimal, single-precision floating point, double-precision floating point, fixed point.

2.7.1 CHARACTER DATA

Following is the general format for each character data item whether it is used as a data item or an address constant:



n is a character count. The character string is justified within the given field length as follows:

C	Left justified with zero fill. Two zeros are guaranteed at the end of the string even if it requires allocating another word.
H	Left justified with trailing blanks
A	Right justified with leading blanks
R	Right justified with leading zeros
L	Left justified with trailing zeros

If a minus sign precedes n, the entire character string is complemented.

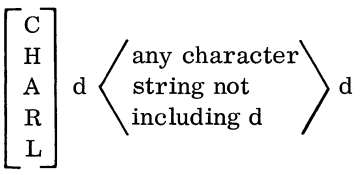
Field length for a character string is determined according to the following rules:

- In data items (DATA, LIT, literals), the characters are left/right justified within a 60-bit (CP) or 12-bit (PP) word.
- In any EQU or SET address field, characters are left/right justified within an 18-bit field.
- In a VFD, characters are left/right justified within the field size specified by the VFD subfield.
- As a constant in an address expression, characters are left/right justified in a field which is equal in length to the address size (18 or 6 bits in CP; 18, 12, or 6 bits in PP).

Characters specified with the C option in address expressions are handled the same as under the L option; the two trailing zeros are not guaranteed on C character strings in address fields.

COMPASS interprets the character string in a character data subfield according to the value of n (see general format above).

(a) If n is missing, the programmer may specify delimiters for the character string:

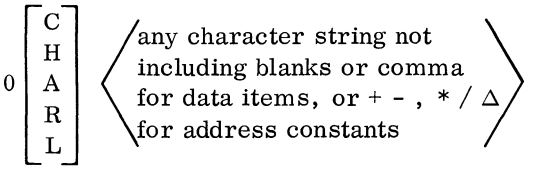


d is any single character. All characters between the first and second occurrence of d are considered the character string; thus, the string may include any character except d.

This form or character specification is restricted to data items, (a DATA or LIT subfield or a literal), since address items beginning with an alphabetic character are considered symbols rather than constants.

A minus sign may precede C, H, A, R, or L to complement the character string.

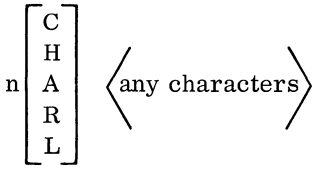
(b) If n is zero, the character string is considered ended when a subfield terminator is encountered:



A blank or comma terminates this character string if it is used in LIT, DATA, or a literal (a data item). Blank, comma, +, -, *, or / terminates an address constant in this format.

When used as an address constant, the string may not exceed ten characters.

(c) If non-zero, n is the count of characters which follow C, H, A, R, or L.



For address constants, $(1 \leq n \leq \frac{\text{field length}}{6})$. n may be of any value for data items. When n is preceded by a minus sign the character string is complemented. For the C designation, the two added zeros are not included in the character count n.

If the character count for a data item is greater than the number of columns remaining in the line, including maximum allowable continuation cards, the character string is terminated at the last column position.

(d) Empty Character Strings:

In either case (a) or (b) above, it is possible to generate "empty" character strings. For example:

```
H++     0L
```

As address constants, empty character strings are valid and have a value of zero. As literals, they are illegal and will produce an error. As an item in DATA or LIT, they are legal and produce no values. In LIT, however, one or more of the items listed must be non-empty.

Examples of Character Data Use	Data Produced (octal)
SA1 X3+3RCIO	5213031117
SB6 X0+1L$	6260530000
VFD 30/0HIOTA,6/1RA,24/0AX+1	11172401550155555531
SA1 =H+LEFTΔJUSTIFYΔWITHΔBLANKS+	14050624551225232411 06315527112410550214 01161323555555555555
SA1 =0CTENCHARCTS	24051603100122032423 00000000000000000000
LIT RA+-*/(A,6L)$=Δ,.,0C0,0L,20HLITERALS	
	00000000004546475051 52535455565700000000 33000000000000000000 14112405220114235555 55555555555555555555
DATA L*ERRORΔINΔPDQΔ*,15B,10HΔΔΔΔΔΔΔΔΔΔ	
	05222217225511165520 04215500000000000000 00000000000000000015 55555555555555555555
SX3 1R → .+1	7130000060
VFD 42/0LOUTPUT,18/1	17252420252400000001

2.7.2 NUMERIC DATA

Numeric data items define values. A data item may consist of the following parts:

		Specified with			
(1)	Sign	e	+	-	
(2)	Pre-radix	O	D	B	e
(3)	Integer	e	n		
(4)	Fraction	.e	.n		
(5)	Scale(base 10)-single precision	E	En	E±n	
(6)	Scale(base 10)-double precision	EE	EEn	EE±n	
(7)	Binary scale (base 2)	S	Sn	S±n	
(8)	Binary point position	P	Pn	P±n	
(9)	Post-radix	O	D	B	e

where e indicates empty or not present, and n is a numeric string.

(1) Sign: A + or - may appear as the first character of a data item. If no sign is present, + is assumed.

(2) Pre-radix: Alternative to post-radix. D indicates the value section is expressed in decimal notation; B or O indicate octal notation. The radix pertains only to the value section — integer and fraction. At most, one radix specification may be included in a numeric data item.

(3,4) Value Section: A string of digits identifies an integer value; a decimal point identifies a floating point value. If (8) or (9) appears in the value section when the radix is octal, it is considered an error.

The modifier section follows the value section. The modifiers (E or EE, S, P, post-radix) may appear in any order, but a given modifier may appear only once.

(5,6) Decimal Scale: A modifier of the form E+n or EE+n defines a power of 10 scale factor. E denotes a single precision value; EE a double precision value. The sign is optional; if omitted, + is assumed. The scale value is a decimal integer (regardless of the nominal base). The effect of this scale is to multiply the number by 10 raised to the specified value. The range of the scale value is limited to +32767. Both fixed and floating point numbers may be scaled. If the scale specifier EE is used with a fixed point number, it will still produce a fixed point number in single precision.

(7) Binary Scale: A modifier of the form S+n defines a power of 2 scale factor. The sign is optional; if omitted, + is assumed. The scale value is a decimal integer. The effect of this scale is to multiply the number by 2 raised to the specified value. The scale value must not exceed 32767 in absolute value. Both fixed and floating point numbers may be binary scaled.

(8) Binary Point Position: A modifier of the form P+n places the binary point in a floating point number to represent an unnormalized floating point number. The sign is optional; if omitted, + is assumed. Placing the binary point is equivalent to fixing the exponent.

With a P scale, the exponent is adjusted to a value of -(P scale factor). Thus, a number with P-6 will have a biased exponent of 2006_8, and P10 will have an exponent of 1765_8. The value is shifted accordingly.

Another way of explaining P scale:

The number is aligned so that the binary point occurs to the right of the n^{th} bit (counting from low order). The exponent will be adjusted accordingly. Thus a P0 number is an unnormalized integer in floating point notation.

P scales may be specified only for floating point numbers of single or double precision. To avoid an error indication, P scaling must result in the high-order significant bit being within the fraction portion of the number.

(9) Radix: D, O, or B defines the radix of the value section. D defines radix 10; O or B defines radix 8. Either a pre-radix or a post-radix may appear, not both. When radix is not specified, the base of the number is derived from the BASE pseudo instruction.

The valid ranges for numbers are restricted by the hardware, although scale factors may exceed valid ranges. The number

```
1.0E400S-1200
```

yields a number which is approximately 5.8×10^{38}, and is in range of the floating point representation.

All scaling calculations are performed in 144-bit precision and rounded to 96-bit precision. For single precision, an additional rounding is performed to yield 48-bit precision.

The value section may contain no more than 32 significant digits if octal, or not exceed 7.9×10^{28}. Extra significant digits may cause erroneous results.

In PP assemblies, only fixed point values are permitted.

Examples of numeric data (assume decimal radix):

7	0000	0000	0000	0000	0007
-9	7777	7777	7777	7777	7766
+B13	0000	0000	0000	0000	0013
14BS1	0000	0000	0000	0000	0030
24BE-1	0000	0000	0000	0000	0002
1.0	1720	4000	0000	0000	0000
1.0EE1	1723	5000	0000	0000	0000
	1643	0000	0000	0000	0000
1.0E+1P0	2000	0000	0000	0000	0000
3.2P1S-5E1	1776	0000	0000	0000	0002
0.0151E+01	1715	6314	6314	6314	6315
0.1P47	1720	0063	1463	1463	1463
-D19	7777	7777	7777	7777	7754
-E	7777	7777	7777	7777	7777
DEES	0000	0000	0000	0000	0000
	0000	0000	0000	0000	0000

2.8 LITERALS

A literal may be defined as a read-only constant. All such constants should be specified as literals for maximum efficiency. A literal is stored by the assembler at the end of the zero block, and the address of that data is substituted in the instruction referring to the literal. Literal usage eliminates duplication of read-only data item values without the user searching for duplicate values.

Literals are specified in an address expression by the format =n where n is a character or numeric data item (as described in previous sections). The first appearance of a value in a literal causes COMPASS to enter that value into a literal table. The contents of this table entry are used when subsequent reference is made to that particular value in a literal.

Example:

```
SB2    =1
SB3    =1RA
SB4    =2
```

The first statement creates a word in the literal table containing the value 0000000000000000001. The address of that entry is then used in the address field of both statements one and two. (The literal in statement two specifies a right justified character A, which also has the value 1.) The third statement creates an entry in the literal table with the value 0000000000000000002, and the address of that entry is in the address field of statement three.

COMPASS also permits symbolic reference to literal table entries. Data values can be entered into the literal table and symbols associated with them via the LIT pseudo instruction. These entries may then be symbolically referenced. The code sequence below will produce the same results as the example above.

```
A    LIT  1,2
     SB2  A
     SB3  A
     SB4  A+1
```

Data items listed in a LIT variable field always appear in the literal table in the order listed. Literal data values may be character or numeric and are specified just like data items, as follows:

Type	Format
Character	
Delimited by subfield end	=0[C H A R L] ⟨character string⟩ Δ ,
Delimited by character count	=n[C H A R L] ⟨n characters⟩
Delimited by delimiter	=[C H A R L] d⟨any character string not including d⟩d
Numeric	
Octal	=Bnumeric
Octal	=Onumeric
Decimal	=Dnumeric
Octal	=numericB
Octal	=numericO
Decimal	=numericD

numeric may be an integer, fixed point, or floating point data item, and a + or - may immediately follow the =. If no B, O, or D appears, the base is assumed to be that currently in use.

2.9 CONSTANTS

A constant is a string of characters which specify an octal, decimal, or character value. Constants may be used in address expressions of machine and pseudo instructions. The size of the constant depends on the size of the subfield in which it is used.

To be recognized as a constant, the item must begin with a numeric character but otherwise follows the general rules for data items. If no B, O, or D appears, the base will be that currently in use.

Example of address constants:

```
    SA1  X1+1R
XY  EQU  3HXXX
    VFD  60/0RMESSAGE,30/3LCIO,30/0R0
    SA2  0L(Z)
```

2.10 SPECIAL ELEMENTS

The character * represents the value of the location counter at the beginning of the field. The characters *L and * are equivalent.

The character *O represents the current value of the origin counter.

The character $ stands for the position counter value. When used in an instruction which does not generate code, such as a conditional, the value of $ will usually be either 59, 44, 29, or 14 in CP assembly, or 11 in PP assembly. $ reflects next available bit position and is also one less than the number of bits still available in a word. The value of $ may be other than 59, 44, 29, 14, or 11 if the previous instruction in the block was a VFD.

2.11 ADDRESS EXPRESSIONS

An address expression may appear as a subfield in the variable field of a machine or pseudo instruction. An address expression consists of terms joined by the operators + and -. A term consists of elements joined by the operators * and /.

2.11.1 ELEMENTS

An element, the basic component of an address expression, is one of the following; all of which have been previously described.

symbol

constant

special element *, *L, *O, or $

deferred symbol: =Ssymbol or =Xsymbol

2.11.2 TERMS

A term is a combination of elements and a term operator * (multiplication) or / (division). A term may consist of any number of elements, each joined by * or / and it must begin with an element. Two successive elements are illegal. However, ** is legal since only one of the asterisks is considered an operator. The last element in a term may be omitted; COMPASS will then provide an element with zero value.

Examples:

```
A       A*B     72**
*L/2    $
```

2.11.3 EVALUATION OF EXPRESSIONS

An expression is composed of terms joined by the additive operators + and - There may be one single term or a number of terms. If two or more of the additive operators appear together, a term with the value zero will be assumed between them.

A literal (=n) may be used as a term only if it is the last term in the expression. (This avoids confusion regarding the use of +n at the end of a literal.) Examples of expressions:

```
A       +-A*A     A*B-72**
$-29    1+=1      *+3
1+=3.14159EE
```

In evaluating an expression, each element is replaced with its 60-bit value. Constants are replaced with their value; address elements which are 21-bit quantities (literals, *, symbols) have their signs extended to 60 bits.

Within a term, calculation is performed from left to right. The following rules pertain:

In division, the integral part of the quotient is retained and any remainder is discarded. Thus, 5/2*2 results in 4.

Division by zero results in the value zero and no error.

Only one relocatable or external element may be used in a term, thus **A is illegal in a relocatable assembly if A and * are relocatable.

To the left of a division or as the divisor in a division, only absolute values may appear.

Once terms are evaluated, they are combined, left to right, into an expression. As a result of the calculation, only the following forms are permitted:

Absolute value

External value±constant

±Relocatable±constant

Terms may cancel relocation values. For example, if A, B, and C are all defined as program relocatable symbols, relative to the same base, then 3*A-B-C is a permissible expression resulting in single program relocation.

PSEUDO INSTRUCTIONS 3

Pseudo instructions are grouped here according to general function. Their appearance in a subprogram is governed by the following rules:

A. Operations required:

IDENT must be the first line

END must be the last line

B. When the following operations are used, they must appear before any operations listed at D. They must also appear before a macro call or the pseudo operation HERE if either generates an operation listed at D.

ABS

PERIPH

C. The following operations may appear anywhere between IDENT and END:

MACRO, its definition, and ENDM

Comments lines

LIST, EJECT, SPACE, TITLE, ERR, LCC, XTEXT, RMT and its bracketed code

HERE and XTEXT, provided the code generated does not include operations listed at D.

A macro call, provided it does not expand into any of the operations listed at D.

D. The first appearance of these operations makes illegal the subsequent appearance of any operation listed at B.

USE, LOC, ORG
MICRO
Any machine instruction
ENTRY, EXT
EQU, SET
BSS, BSSZ
DATA, VFD, REP, DIS
DUP, ENDD, STOPDUP
All conditional pseudo instructions

3.1 ASSEMBLER CONTROL

The mode of assembly is controlled by these instructions.

3.1.1 IDENT

The first operation of every subprogram must be IDENT.

Location	Operation	Variable
ignored	IDENT	1, 2, or 3 subfields

IDENT must occur only once in each subprogram. Its occurrence at any other time is considered an error. If it is omitted, an error will result. The first variable subfield must contain a linkage symbol which becomes the name of the subprogram only and is not defined in the assembly. For relocatable assemblies, the second and third subfields are ignored.

In absolute assemblies, the second subfield defines the first word address of the absolute binary program image. During assembly, data may be origined at a location higher than this base origin address, but not below it. This first word address does not serve the same function as an ORG nor does it replace the need for an ORG to set the origin counter value.

In absolute CP subprograms, the third subfield contains the entry address. Refer to Chapter 6 for an explanation of assembler binary output and the significance of the IDENT subfields.

In the absence of a TITLE instruction in the assembly, the IDENT variable field is used for the main subprogram title.

3.1.2 END

END is required as the last operation of each subprogram.

Location	Operation	Variable
symbol or blank	END	blank or a linkage symbol

This operation terminates a subprogram deck. It causes the assembler to terminate any counter, conditional assembly, macro generation or code duplication in progress. Any waiting remote text is assembled; all local blocks are assigned an origin relative to the program origin in the order in which they were first introduced. If there is a symbol in the location field of END, it is defined as having a relocatable value of the total subprogram length, or last word address + 1. Total subprogram length includes the

length of the literals block. A symbol in the variable field of END is considered a transfer address and is relevant only for relocatable assemblies.

3.1.3 ABS

A non-relocatable CP program may be assembled with this instruction:

Location	Operation	Variable
ignored	ABS	ignored

ABS declares the program to be absolute; if used, it must appear at the beginning of the assembly. The assembler will assign all blocks an origin relative to absolute zero. Although the output is absolute, relocatable symbols may exist during assembly; any literal or any symbol defined in a block other than the zero block is considered relocatable. This is a relevant consideration for symbol definition, storage allocation, and the IF pseudo instruction.

In absolute assemblies, the following are illegal: ENTRY, REP, and EXT.

3.1.4 PERIPH

PP code is assembled with this instruction:

Location	Operation	Variable
ignored	PERIPH	ignored

PERIPH declares the program to be a PP program and absolute. The rules stated under ABS apply; in addition, LCC is illegal.

Within PERIPH assemblies, the register names of CP assemblies are treated as normal symbols. The use of any CP instruction will cause an O-error.

3.1.5 BASE

With BASE, the programmer can change the mode of numeric data.

Location	Operation	Variable
ignored	BASE	O or D

A variable field symbol beginning with the letter O denotes octal assembly mode; a symbol beginning with D denotes decimal mode. Any other entry will be error-flagged, and assembly will be decimal.

In succeeding lines, all numeric address constants and data items consisting of digits with no O, D or B prefix or suffix are subject to the specified mode control. Under BASE O, for example, the constant 15 is considered 15_8, as is 15B; and constant 15D is evaluated as 17_8. Under octal base, any numeric item without a D prefix or suffix containing an 8 or 9 is flagged as an error. Decimal base is always assumed if no BASE is encountered.

All numeric items are under base control. For example, under octal, base, VFD 60/-1 defines a 48-bit field. In numeric data items, only the value portion is under base control; scale factors and binary point position are always considered decimal values. A second subfield on the IDENT line is also evaluated as decimal number unless specifically designated octal.

3.2 COUNTER CONTROL

These pseudo instructions control the three counters.

3.2.1 USE

USE declares a block into which succeeding instructions are to be placed.

Location	Operation	Variable
ignored	USE	block name

Upon encountering USE, the assembler places succeeding assembled values in the block named in the variable field. The first appearance of a block name in USE causes a force upper, subsequent USE statements for that block do not. The values of the current origin and position counters are saved to indicate the last known length of the block being assembled. An indication as to whether the next instruction is to be forced upper is also saved. If the block name in the USE statement is enclosed in slashes, that block is a common block, and subsequent uses of that name in USE need not be enclosed in slashes. If the block name is never enclosed in slashes, it is a local block.

The following notations may be used to set the origin of data:

```
USE      Data origin is in zero block
USE  0   Data origin is in zero block
USE  //  Use blank common block
USE  *   Use block in effect prior to previous USE
```

The zero block, the nominal subprogram block, contains the entire subprogram if no other USE is encountered.

If the blank common block is named in a USE statement, BSS and ORG are the only instructions which allocate storage that may follow USE; not even BSSZ is permitted since it presets the block to zero.

If the block named in USE is new to COMPASS (not named in a previous USE), the origin and location counters are started at zero relative to the block origin, and the position counter is set to the beginning of a new word. Block type is considered local unless the block name is enclosed in slashes.

If the block name has previously appeared in a USE operation, or is the zero block, the origin, location, and position counters are started at their last known values.

If the last instruction assembled under this block was one which forces the next instruction upper, it will be forced upper. For example:

```
GAMMA   RJ      ALPHA
        USE     DATA
SAM     DATA    1.0
        USE     *
        SA3     SAM
```

The SA3 instruction will be forced upper.

If the last instruction did not indicate a force upper, forcing upper is determined by the instructions which follow USE. With this facility, one may, for example, pack partial-word bytes into a table which resides in a block other than the one currently being used. For example:

```
        .
        .
        .
USE     /TABLE/
VFD     6/CODE
USE     *
        .
        .
        .
USE     /TABLE/
VFD     6/1RX,18/ADDR
USE     *
        .
        .
        .
```

The value of the location counter is not saved, so if LOC has been employed, caution must be exercised to produce the desired results.

If the block name appearing in USE is the one currently in effect, the only effects are to force the location counter to agree with the origin counter and to record this block as the last known block for use with a subsequent USE *.

The assembler maintains a record of USE and ORG pseudo-operations. Each use of one of these pseudo operations (except USE *) adds an entry to this record. Each use of USE * restores the most recent entry and removes it from the list. In this way, a push-down list is maintained. Only the last 50 entries are maintained. When the list is exhausted (more USE * instructions than entries), the zero block is used.

Any symbol used as a block name has definition as a block name only, and may be defined elsewhere without ambiguity.

3.2.2 ORG

With ORG, the origin and location counters may be reset.

Location	Operation	Variable
ignored	ORG	address expression

The ORG instruction causes the location and origin counters to be reset to the value stated in the address field. As in USE, the current origin, location, and position counters are saved. ORG starts the assembly at the upper position of a word.

When ORG is used, care should be taken concerning block size. At the end of assembly, the size of each block is considered to be the current value of the origin counters of each block. Since ORG may reset this value, the size of a block may be less than it appears to be. For instance, total size of block GAMMA is 1, not 100, in the following example:

```
      USE/GAMMA/
G10   BSS    100
      USE *
      :
      :
      ORG  G10
      RJ   ALPHA
      USE  *
      :
```

The use of * in the variable field of ORG has no effect other than to force the current block upper. The USE pseudo instruction must be used to return control to the last-used block.

The expression in the variable field of ORG must not contain symbols not yet defined; the expression may not result in a negative relocatable value.

3.2.3 LOC

The location counter may be set with this instruction.

Location	Operation	Variable
ignored	LOC	address expression

The location counter is set to the value of the variable field expression, but the origin counter is not reset. The location counter value is normally the same as the origin counter, since instructions are executed normally at the location into which they were loaded. LOC allows the location counter to be adjusted so that code may be loaded into one place, and executed at another. The location counter will be reset to origin counter value whenever a subsequent USE or ORG is encountered.

Symbols in the variable field expression of LOC must have been previously defined. LOC causes the next instruction to be forced upper. The use of LOC * has no effect other than to cause a force upper.

3.3 LINKAGE CONTROL

Names to be passed to the loader for subprogram linkage are declared with these instructions. They are valid for relocatable code only.

3.3.1 ENTRY

An entry point name is passed to the loader with this statement.

Location	Operation	Variable
ignored	ENTRY	symbols separated by commas

The linkage symbols listed in the variable field are declared to the loader as entry points. Each symbol must be defined in the assembly as a non-external symbol.

3.3.2 EXT

This instruction declares symbols external to the subprogram.

Location	Operation	Variable
ignored	EXT	symbols separated by commas

The linkage symbols listed in the variable field are passed to the loader as external symbols. These symbols must not be defined within the subprogram.

3.4 STORAGE ALLOCATION

These pseudo instructions cause adjustment of both the location and origin counters. All operations force upper.

3.4.1 BSS

An area of storage is reserved with this statement:

Location	Operation	Variable
symbol or blank	BSS	absolute address expression

A location field symbol is defined as the current value of the location counter. The expression in the address field is evaluated and the location and origin counters are incremented by that amount. Symbols in the expression must have been previously defined. If there are errors in the address expression, no space will be reserved, but a force upper will occur. BSS 0 allocates no storage but does cause a force upper.

3.4.2 BSSZ

BSSZ reserves an area of zero-filled words in storage. The specification of BSSZ is exactly like that of BSS, and the effect is the same, except that allocated storage is preset to zeros at load time. BSSZ 0 causes a force upper and no storage allocation.

3.5 SYMBOL DEFINITION

These operations permit the direct definition of symbols.

3.5.1 EQU

Location	Operation	Variable
symbol	EQU	address expression

The symbol in the location field is defined as having the same value as the address expression. Once defined, the symbol retains that definition throughout assembly. An undefined symbol may not appear in the variable field expression. =Ssymbol and =Xsymbol may not be used in the address field unless the symbols have been defined by some other conventional method. The address expression may result in an absolute, relocatable, or external value. If there are errors in the address field, the location symbol of the EQU will not be defined, and a warning flag will be issued.

3.5.2 SET

Location	Operation	Variable
symbol	SET	address expression

SET redefines the value of the location symbol to the value of the variable field expression. Such symbols are called redefinable and may be defined only with the SET instruction; they have this definition only until reset. Symbols in the address field expression must have been previously defined. The same rule about =S and =X as noted for EQU pertains to SET. The address expression may result in an absolute, relocatable or external value. If there are any errors in the address field, the location symbol will not be redefined, and a warning flag will be given. A SET-defined symbol may not be referenced before it is first defined by a SET.

3.6 DATA GENERATION

With these instructions, data items may be included in the subprogram.

3.6.1 DATA

The DATA operation is used to declare data items — numeric and character.

Location	Operation	Variable
blank or symbol	DATA	absolute data items

If a location symbol is present, it is defined as the current value of the location counter. The data items may be octal, decimal, or display code characters, and will be full-word values. They are separated by commas and terminated by a blank. Refer to Chapter 2 for rules on the specification of data items.

The DATA pseudo instruction forces upper. Literals may not be used in the variable field list.

3.6.2 DIS

DIS is supplied in COMPASS as a convenient means of writing display code lines.

Location	Operation	Variable
blank or symbol	DIS	word count, and a character string

The expression in the first subfield must result in an absolute value which is a word count. COMPASS extracts n·10 characters beyond the comma following the address expression, and packs them as they occur, into n words. If the statement ends before n·10 is satisfied, the remainder of the words requested will be filled with blanks (55_8). (For PP, n·2 is the character count.)

If the count subfield is missing or has a zero value, the character string must be bounded by delimiters. The comma must always be present. The first character after the comma is the delimiter. All characters between the delimiter and its next occurrence are packed into as many words as are necessary. Two zeros are guaranteed at the end of the character string, even if it means that COMPASS must allocate another word to accommodate them. If the delimiter character is not encountered again, the end-of-statement terminates the character string.

The DIS pseudo instruction forces upper.

3.6.3 LIT

Absolute values are entered into the literal table with the LIT statement.

Location	Operation	Variable
blank or symbol	LIT	up to 100 words of data items

A location symbol is defined as the location of the first mentioned value. The data items are separated by commas and terminated by a blank. Data items are entered in the literal table in the order in which they are specified. Duplications in data items may occur in the literal table if there are duplicate values in the LIT variable field; but if all data items listed for one LIT are identical to an existing sequence in the literal table, they will not be duplicated. Subsequently defined literals (defined either with LIT or the =n form) will not be duplicated in the literal table if they exist in a LIT-declared sequence.

The specification of data items in the LIT variable field is the same as for DATA. No = is used before LIT-declared literals. At least one of the data items must be non-empty.

3.6.4 VFD

Fields of binary data are generated with the VFD statement.

Location	Operation	Variable
blank, +, -, or symbol	VFD	a list of subfields separated by commas

When plus or a symbol appears in the location field, data begins in a new word. A symbol is given the new value of the location counter. A minus sign in the location field causes the position counter to be positioned at the next quarter word boundary in a CP assembly, or at a new word in a PP assembly.

The subfields are of the format n/v where n is a bit count of field length and may be any single element previously defined and absolute. It must be positive and may not exceed 60. The value expression, v, consists of any valid address expression. If a non-absolute value occurs (relocatable or external), it must be within a field that is at least 18 bits long and ends at bit 0, 15, or 30.

Absolute data items follow all rules indicated in Section 2.7 and are right or left justified within the field length, n.

Example:

```
ALPHA    SET    15
TABLE    VFD    36/4CTAB1,6/9,18/TABLOC
         VFD    30/*-1,30/5Hbbbbb,ALPHA/-0
         VFD    $/0,1/1
```

Word 1	24010234000011	TABLOC
Word 2	0000 TABLE	555555555
Word 3	77777	000000000000001

VFD leaves the position counter pointed at the next available bit position. If the next instruction is also VFD with no location field entry, or if USE is the only intervening instruction, values are packed into words with no padding or forcing upper.

A plus or minus in the location field of a VFD in PP has the effect of forcing the VFD data to begin at the next full word boundary.

3.6.5 REP

REP defers data generation until load time. It is valid only in relocatable assemblies.

Location	Operation	Variable
ignored	REP	up to 5 subfields separated by commas

Information is passed by the assembler to the loader. This replication control is used when a block of storage is to be set to a given series of values, yet is not to be represented in its duplicated state in the COMPASS binary output. First, a set of data is placed in consecutive locations, established by the programmer with normal assembler techniques. Then the loader is instructed to move blocks of data in storage. Each subfield consists of a letter, S, D, C, B, I, followed by a slash, followed by a non-external address expression. The letters indicate the following:

S Source address

D Destination address

C Repetition count

B Code block size

I Increment

The operation at load time is to move B words from location S to location D, B words from location S + I to location D + I, B words from S + 2I to location D + 2I, etc. This operation is repeated C times. An omitted specification, except S, is passed to the loader as zero. Only one specification of each type may appear. The loader will make the following assumptions, in the order shown, if a subfield is zero:

B = 1

I = B

C = 1

D = value of S subfield plus value of B subfield

The assembler error-flags the REP instruction if the value of S is zero, and does not pass REP to the loader.

At load time, REPs are deferred until all other loading is finished.

3.7 CONDITIONAL OPERATIONS

These pseudo instructions control the conditional assembly of code: a number of succeeding instructions is assembled only if the condition stated is true. When a value of an address expression is involved, only previously defined symbols may be used, and the result must not be relocatable. If undefined elements are used, the expression has a zero value, the conditional is error-flagged, and assembly proceeds with the next instruction.

The number of instructions to be assembled or skipped may be controlled by a line count or by brackets (an IF to a matching ENDIF). If a count is used, it must be the last subfield of the variable field. If the count field is missing or zero, the assembler looks for a bracketing ENDIF, and assembly resumes with the instruction immediately following the appropriate ENDIF.

If there is an instruction bracket name, the corresponding ENDIF is the first one encountered which has the same name as the IF or no name. If there is no instruction bracket name on the conditional instruction, and no line count is given, the first ENDIF encountered, with or without a name, terminates the bracket. Instruction brackets have significance only if coding is not to be assembled. An ENDIF encountered when code is being assembled is ignored. An END card terminates the skipping process. During the skipping process, macros are not expanded. Thus, an ENDIF which would have appeared in the macro expansion is not detected.

The four kinds of conditional pseudo instructions are as follows:

Tests of the comparative value of two address expressions

Tests of assembly environment

Tests of the attribute of a single symbol or address expression

Tests of the value of character strings

3.7.1 IF: COMPARE EXPRESSION VALUES

Conditionals of this type are all of the format:

Location	Operation	Variable
blank or instruction bracket name	IFxx	2 or 3 address expressions separated by commas

xx is EQ, NE, GT, GE, LT, or LE. The value of the first address expression is compared to the second. The third subfield is the count of lines to be assembled if the value comparison is satisfied.

IFEQ: Succeeding code is assembled if the value of the first subfield is equal to the second.

IFNE: Succeeding code is assembled if the value of the first subfield is not equal to the second.

In IFEQ and IFNE tests, all information pertinent to the value of the two address expressions is compared for equality. Not only must the expressions have the same numeric value, but they must have equal attributes. For example, both must be common relocatable, program relocatable, absolute, external, or register names.

IFGT: Succeeding code is assembled if the value of the first subfield is greater than the second.

IFGE: Succeeding code is assembled if the value of the first subfield is greater than or equal to the second.

IFLT: Succeeding code is assembled if the value of the first subfield is less than the second.

IFLE: Succeeding code is assembled if the value of the first subfield is less than or equal to the second.

In the last four tests, only the values of the expressions are compared. Relocation and other attributes are not tested for equality.

3.7.2 IF: TEST ASSEMBLY ENVIRONMENT

The two conditionals in this class are of the format:

Location	Operation	Variable
blank or instruction bracket name	IFPP or IFCP	A single optional address expression

The test IFPP will be satisfied if this is a PP assembly; IFCP will be satisfied if this is a CP assembly. The variable field expression results in a count of lines to be skipped if the test is not satisfied.

3.7.3 IF: TEST SYMBOL ATTRIBUTE

Conditionals of this class are all of the format:

Location	Operation	Variable
blank or instruction bracket name	IF	2 or 3 subfields, separated by commas: attribute mnemonic; symbol or address expression; address expression

The attribute mnemonic is SET, ABS, REL, REG, EXT, COM, LOC, or DEF. The second subfield is a single symbol or an address expression, depending on the mnemonic used. The third subfield is an address expression resulting in the line count. The line count and its preceding comma may be omitted if ENDIF is used.

For all attributes listed, the negative attribute may be specified by preceding the attribute mnemonic with a minus sign.

The following tests are made:

SET Satisfied (true) if the symbol in the second subfield has been previously defined by the SET pseudo instruction. -SET is satisfied if the symbol is defined by any method other than SET. The second subfield must be a single symbol.

ABS Satisfied if the address expression is absolute (not relocatable or external). -ABS is satisfied if the expression has other than an absolute value.

REL Satisfied if the address expression is common or program relocatable. -REL is satisfied if the address expression is other than program or common relocatable.

REG Satisfied if any symbol in the address expression is a register name. -REG is satisfied if no symbol is a register name.

COM True if the expression is common relocatable. -COM is true if the expression is not common relocatable.

EXT True if any symbol in the address expression is an external symbol. -EXT is true if there is no external symbol.

LOC Satisfied if the expression is program relocatable. -LOC is true if the expression is not program relocatable.

DEF Satisfied if all symbols in the expression have been defined. -DEF is satisfied if any expression symbol has not yet been defined.

The attributes listed, except -DEF and REG are known to the assembler only after the symbols in the expression (or the single SET symbol) have been defined. For example, if a common block name has not yet been declared in a USE pseudo instruction, a test for COM on that name will fail. Any test on an undefined symbol, except for DEF, will result in an error.

3.7.4 IFC

This is a conditional assembly option to test equality of two character strings.

Location	Operation	Variable
blank or instruction bracket name	IFC	2 or 3 subfields separated by commas: a relational mnemonic, 2 delimited character strings, and an optional address expression

Relational mnemonics:

EQ or -NE	equal
NE or -EQ	not equal
GT or -LE	greater than
GE or -LT	greater than or equal
LT or -GE	less than
LE or -GT	less than or equal

The delimited character strings are of the format:

dccc...ccdccc...ccd

d is any character. Characters between the first and second d constitute the first character string; characters between the second and third d constitute the second character string.

The optional third subfield is an address expression which results in line count. It must be preceded by a comma. If ENDIF is used, the line count and its preceding comma may be omitted.

The test is performed in the following manner: each character in the first string is compared with the corresponding character in the second string, progressing from left to right, until an inequality is found or both strings are exhausted. If one string is shorter than the other, the short string is padded with a character which is smaller than any other character in the string.

The truth condition is then evaluated based on the relative magnitudes of the strings.

Example:

ABCABC$	is equal
ABABC$	is less than
$$$	is equal
$A$$	is greater than
Z8$	is less than

The collating sequence is given in Appendix D.

3.7.5 ENDIF

ENDIF terminates the range of a conditional assembly operation.

Location	Operation	Variable
blank or instruction bracket name	ENDIF	ignored

ENDIF terminates an instruction bracket. At any other time it is ignored. An ENDIF with no name terminates any conditional in effect. A named ENDIF terminates a conditional with the same bracket name, or a conditional with no name. ENDIF is ignored if it appears within a range controlled by line count.

3.8 LIST CONTROL

These instructions control the listing format and have no other effect on the assembly.

The listable output from a COMPASS assembly normally contains the following:

Heading information	Program length, origin and length of each block, entry points, external symbols.
Assembly text	Listing of the line and assembly results of each line assembled (not skipped) that came from the input device (not generated by RMT, DUP, XTEXT, or a macro expansion). For generative pseudo instructions DATA, DIS, VFD, only one line is listed. Any line with an error flag is listed. Each line with the instruction LIST is listed.
Assembler statistics	Size of unused storage, a count of statements actually generated in the assembly, if non-zero, a count of references discarded because of restricted core storage.
Error directory	Explanation of each error as well as each page on which it occurred. If no errors occur, the error directory is suppressed.
Reference table	List of each symbol, its definition, and for each reference, the value of the origin counter at the place of reference.

Primary list control is specified on the COMPASS control card. When L=0, only the heading information, assembler statistics, error-flagged lines and the error directory are listed. When L is other than 0, more extensive listings than those noted above may be specified with the LIST pseudo instruction.

3.8.1 LIST

This instruction controls the listable output from COMPASS, and is relevant only if listings are being produced.

Location	Operation	Variable
ignored	LIST	list control options, separated by commas

Each option is represented by a single letter. If the letter appears alone, the option is turned on. If the letter is preceded by a minus sign, the option is turned off.

The L and R options are normally on, all others are normally off.

L List Control

This is the master list control. When off, only error-flagged lines and the LIST pseudo instruction are listed. The accumulation and listing of the reference table is unaffected by this option.

R Reference Accumulation and List

When this option is off, no references will be accumulated. If off at the end of assembly, the reference table listing will be suppressed. To be assured of a complete reference listing, R should never be turned off.

G Code Generation List

When this option is on, lines which result in code generation will be listed regardless of other list controls (except L). In this way, the code generated from macro calls may be listed without listing the entire macro expansion. The following operations are controlled by G: machine operations, DATA, BSS, BSSZ, VFD, DIS.

A Assembly List

Normally (A off), when a → or ≠ mark appears in a line that would be listed, the line appears with the → and ≠ marks in it exactly as presented to the assembler. With the A option on, however, the line will also be listed with catenation marks removed and micros substituted.

C Control Card List

EJECT, SPACE, and TITLE are listed when this option is on. Their effect is not changed.

D Detail

The following items are controlled by this option: second and subsequent lines of VFD, DATA, DIS; code assembled remotely when HERE or END causes its assembly; a list of literals and deferred symbols at the end of the assembly.

E Echoed Lines

This option controls the listing of lines generated by DUP. When on, all iterations of duplicated code are listed.

F IF-skipped Lines

This option controls the listing of lines skipped by IF-type instructions.

M Macro

This option controls the listing of lines generated by macro calls. This does not include system macro list control.

S Systems Macros

The S option controls the listing of lines generated by systems macros.

X XTEXT Lines

The X option controls the listing of lines generated as a result of an XTEXT pseudo instruction.

The list options A, C, D, E, F, M, S, and X cause a line to be listed only if all the options which apply to it are on. For example, if a DUP appears within a macro, its expansion will be listed only if both M and E are on. If a systems macro call is made within XTEXT text, its expansion will be listed only if X and S are both on. If the marks → or ≠ appear in external text inside a DUP bracket, the lines will be listed with → and ≠ removed only if A and X and E are all on, etc.

3.8.2 EJECT

EJECT is the operation field entry; location and variable fields are ignored. EJECT causes this and succeeding lines to begin on a new page, after the page headings.

3.8.3 SPACE

Location	Operation	Variable
ignored	SPACE	address expression

The assembler spaces the listing the number of lines specified by the address field expression. If it exceeds the number of lines remaining on the page, an eject occurs, and listing resumes after the titles are printed on the next page.

3.8.4 TITLE

With this instruction the programmer can establish titles for subprogram listings.

Location	Operation	Variable
ignored	TITLE	character string

The character string starts at the column immediately following a blank after the E of the operation code. The first TITLE instruction in a subprogram defines the primary title which will appear on every page. Subsequent TITLE instructions generate subtitles. Except for the first TITLE, this instruction causes a page eject. If no title is desired, a card containing only the word TITLE will achieve the desired effect, for title and subtitle. If no TITLE is specified in a subprogram, the variable field of the IDENT line is used as the main title.

3.9 CODE DUPLICATION

3.9.1 DUP

DUP permits replication of a sequence of lines.

Location	Operation	Variable
blank or instruction bracket name	DUP	1 or 2 address expressions separated by a comma

A specified number of lines following DUP are assembled the number of times specified by the value of the first address expression. Each assembly is identical to the first one. The lines to be assembled may be indicated in one of two ways: by an instruction bracket (DUP to an appropriate ENDD), or by a line count on the DUP instruction, which is the second address expression.

If the iteration count is zero, the effect is to not assemble code, or in effect to skip the indicated number of lines.

Indefinite duplication of code is specified by an unobtainable iteration count and the STOPDUP statement. ENDD or line count is still necessary. Any operation otherwise legal is permissible within the range of DUP, except END. A comment card with a column 1 * will not be counted in the line count if one is given and will not be duplicated.

3.9.2 ENDD

ENDD terminates the range of a DUP if a line count is zero or not used.

Location	Operation	Variable
blank or instruction bracket name	ENDD	ignored

ENDD should follow the last line to be duplicated by DUP. An ENDD with no location field entry terminates any DUP in effect, including any inner DUP. An ENDD with an instruction bracket name terminates a DUP with the same name or a DUP with no name, and every inner DUP.

ENDD is ignored if it appears anywhere except as a DUP terminator.

3.9.3 STOPDUP

STOPDUP may be used to stop the duplication process:

Location	Operation	Variable
ignored	STOPDUP	ignored

If the iteration count of DUP has not been exhausted when STOPDUP is encountered, the effect is to stop the duplication when the current iteration is finished.

STOPDUP is normally used after a conditional operation which, when satisfied, indicates that no more duplications are needed. Once STOPDUP is encountered, code is assembled to the proper ENDD or to the end of line count, then no more iterations will occur.

STOPDUP is ignored outside a DUP range.

3.10 REMOTE ASSEMBLY

RMT generates symbolic instructions for assembly at a later time or place; it supplements the USE facility. Code following USE is assembled when it is encountered; code following RMT is not assembled until the programmer calls for it to be assembled. COMPASS stores the code, unassembled, until it is called. Symbols, macro definitions, micros, and block names defined within a remote section do not become defined until the remote section is actually assembled.

3.10.1 RMT

RMT introduces the section of symbolic instructions to be saved for later assembly.

Location	Operation	Variable
ignored	RMT	ignored

All instructions between the first and second RMT statements are saved for later assembly. Any instructions, except RMT, may be contained within RMT sections as long as their use is legal when the remote lines are assembled. COMPASS takes no note of remote code at the time it is saved, except to recognize a second RMT instruction, which acts as an off switch. Alternate appearances of RMT act as on/off switches. However, within remote sequences, macro calls, catenation or micro substitution may specify RMT sequences, since expansion and substitution occur at assembly time and not at remote definition time.

3.10.2 HERE

When HERE is encountered, all saved remote code is assembled. HERE also clears the remote retention table such that the code is not called again. The instruction consists simply of the operation field entry HERE. Other fields are ignored. If, in the assembly of remote sequences, RMT pairs occur, the bracketed lines will be saved for later assembly when another HERE or END is encountered.

In the absence of USE within the remote sequence, the remote code is assembled under whatever block is in effect at the time HERE is encountered.

If HERE does not occur in a subprogram, any waiting remote lines are assembled when END is encountered but before END is processed. Any remote lines which might have been saved as a result of this last remote assembly will be lost.

3.11 LOADER CONTROL: LCC

Loader Directives may be included in a relocatable source program only. They are passed along in the binary output file for subsequent loader recognition. Loader directives are specified by LCC.

Location	Operation	Variable
ignored	LCC	any string of non-blank characters

All characters in the variable field from the first non-blank to the first encountered blank are considered the directive. They are moved to the first position (column 1) of a loader table in packed display code. COMPASS does not edit the directive. Illegalities are recognized at load time by the loader.

All loader directives appear before any of the binary output for a subprogram. For loader directive formats, refer to SCOPE documents.

3.12 ERR

ERR introduces a fatal error into the subprogram to inhibit subsequent loading.

Location	Operation	Variable
ignored	ERR	ignored

The appearance of ERR in a subprogram has no effect on any other code. It may be used in conjunction with a conditional assembly pseudo operation to force an error into the assembly based upon an assembly time test. This can be used effectively to check for illegal macro parameters.

3.13 EXTERNAL TEXT

The pseudo instruction XTEXT provides a method of introducing lines from a file other than that being used for input.

Location	Operation	Variable
file name	XTEXT	blank, or a record name

COMPASS will gain access to the file named in the location field and search for the named record on that file. The contents of that record, to an END card or end-of-record, will be brought into the subprogram for assembly at the point where XTEXT is encountered. The text may contain anything legal for assembly, including macro definitions, which might be classed as library macros.

If the record name is not supplied in the XTEXT variable field, COMPASS will rewind the file and read the first record in the file. If the record name is given, the file must be an indexed file with named records. If either the file or the record cannot be found, an error flag is issued. The file must be a standard coded file exactly like an input file. Text brought in by XTEXT will not be listed (except for lines with assembly errors) unless the X list option is turned on.

MACROS 4

A macro is a sequence of code that may be called whenever needed by a single instruction — a macro name. A macro name in the operation field of a statement (a macro call) results in the macro code sequence being assembled at that point in the program. The macro call may also contain parameters which are substituted for defined parameters in the macro code sequence. The use of a macro requires two steps: defining the macro sequence and calling the macro.

4.1 MACRO DEFINITION

A macro definition consists of three parts:

Macro Heading	MACRO pseudo instruction which states the name of the macro and identifies its substitutable parameters. The LOCAL pseudo instruction may also be used to identify local parameters.
Macro Body	Symbolic instructions which constitute the macro code sequence.
Macro Terminator	ENDM pseudo instruction which terminates the definition.

A macro definition may appear anywhere in a subprogram according to the rules for pseudo instructions given in Chapter 3. The macro definition must appear, however, before that macro is called.

A macro may be redefined at any time. The latest definition of a macro name applies to a macro call. For any redefinition, including redefining a mnemonic, a warning flag is issued but the new definition is valid.

4.1.1 MACRO HEADING

The macro heading line has two forms:

(1) MACRO, Standard Form

Location	Operation	Variable
macro name	MACRO	up to 63 parameters

The location field contains the name of the macro being defined; it may be any legal name except END, LOCAL, or ENDM; it may be the same as other program-defined symbols since it has meaning only in the operation field. For example, the symbol ABC may co-exist with a macro named ABC. A macro name may be identical to a machine or pseudo instruction mnemonic. This results in a redefinition of the mnemonic to be the macro definition. For example, definition of a macro name SB3 overrides the machine mnemonic SB3; an SB3 in the operation field of a subsequent statement will be interpreted as a macro call. If SB3 appears in the macro body it also is interpreted as a macro call and an infinite macro expansion may occur. Once a mnemonic has been redefined as a macro, there is no way of returning that name to mnemonic status. The macro may be re-defined, however, to produce equivalent results by using a VFD.

The variable field of the MACRO line contains the name of substitutable parameters in the order in which they occur on the macro call instruction. Each is a symbol of one to eight alphanumeric characters, the first of which must be alphabetic. Parameter names are separated by any one of the special characters:

```
. , + - * / ) ( $ =
```

and the list is terminated by a blank. These special characters have no meaning other than as separators.

ENDM, LOCAL, or END may not be used as parameter names. Parameter names may occur more than once in the parameter list but second and subsequent appearances are ignored. Parameter names beginning with a number are ignored. The total number of unique parameter names plus LOCAL symbols may not exceed 63 for any one macro definition.

Such notations as the following are permitted; all are equivalent:

```
SUM   MACRO   X=Y+Z+X
SUM   MACRO   X(Y+Z)
SUM   MACRO   X=Y+Z
SUM   MACRO   X, Y, (Z+X)
```

The following are also equivalent:

```
RAO   MACRO   X
RAO   MACRO   X=X+1
```

(2) MACRO, Alternate Form

Location	Operation	Variable
blank	MACRO	2 or more subfields

This form is identified by the blank location field of the MACRO line. The macro name is the first subfield of the variable field. Subsequent subfields are the substitutable parameters, listed with the same rules as for the normal MACRO header form. The first of these substitutable parameters, which must be present in the alternate form macro, has special meaning. It is called the location argument since the location field entry of the macro call will be its substituted value.

Example:

```
          MACRO   TABLE, TABNAM, VALUE1, VALUE2,
TABNAM    VFD     60/VALUE1, 60/VALUE2
          ENDM
```

The macro is named TABLE, its substitutable parameters are TABNAM, VALUE1, and VALUE2. TABNAM is the location argument. TABLE might be called with an instruction like this:

```
SPVAL     TABLE   1.0, 2.0
```

which will result in the expansion

```
SPVAL     VFD     60/1.0, 60/2.0
```

If it had been called with this instruction:

```
          TABLE   1.0
```

the expansion would be

```
          VFD     60/1.0, 60/
```

since the location argument and VALUE2 are null.

The location argument must be present on the MACRO line, or a warning flag will be given and the definition will be ignored. The following examples of definition headers, then, are illegal:

```
          MACRO   ABC
          MACRO   ABC, , FP
```

One or more LOCAL pseudo instructions may immediately follow the MACRO line of either form.

Location	Operation	Variable
ignored	LOCAL	list of symbols

The listed symbols may be separated by any one of the special characters

, + - * / .) ($ =

Therefore a local symbol may not contain any of those characters.

The symbols are to be considered local to the macro, or known only within the macro definition. If a substitutable parameter name appears in the LOCAL list, it is ignored. The total number of local symbols plus substitutable parameters may not exceed 63. For each local symbol defined within the macro, the assembler creates a symbol which will not likely be accidentally duplicated elsewhere by the programmer, and substitutes this created symbol for each use of the declared symbol. They appear as ↯nnnnnn, where n is unique for each local symbol in a subprogram. The symbol A, for example, if it is declared local to the macro, may co-exist with another symbol A defined elsewhere in the subprogram.

Created symbols will be substituted for local symbols wherever they appear in the macro except on comments lines with an * in column 1. Created symbols will not be listed in the symbol reference table.

All symbols defined within the macro which are not local are global; they are accessible outside the macro definition, but local symbols are not.

A local symbol may be passed to inner macro definitions or inner macro calls.

Example:

```
ABC   MACRO   A,B
      LOCAL   C
C     BSS     10
        .
        .
XYZ   MACRO   D
      SA1     C
        .
        .
      ENDM
```

If the representation of C is ↟000010, when COMPASS defines the macro XYZ (when ABC is called), it is as if the definition were:

```
XYZ   MACRO  D
      SA1    ↟000010
       .
       .
       .
      ENDM
```

Note the difference, however, between the above examples and the following:

```
ABC   MACRO  A,B
      LOCAL  C
C     BSS    10
       .
       .
       .
XYZ   MACRO  D
      LOCAL  C
      SA1    C
       .
       .
       .
      ENDM
```

When XYZ is defined, it looks like the following to COMPASS:

```
XYZ   MACRO  D
      LOCAL  ↟000010
      SA1    ↟000010
       .
       .
       .
      ENDM
```

The symbol ↟000010 will be replaced with another invented symbol, and the reference to C in the SA1 instruction will not result in a reference to the C of the outer macro.

Thus, like substitutable parameters, invented symbols will replace LOCAL-named symbols wherever they appear in a macro definition, including inner macro definitions and inner macro calls.

4.1.2 MACRO BODY

The first line following MACRO which is not a LOCAL statement or a comment is the start of the macro body. The macro body consists of a series of symbolic instructions. Within these lines, in any field, may appear the name of a substitutable parameter listed on the MACRO line. To be recognized as such, the parameter must be bounded by two of the following characters:

= . ≠ - * / $, →) (Δ

Beginning of statement (column 1 or 2) or end of statement is also a delimiter. The character → may be used to catenate a substitutable parameter name with some other item, or to flag a parameter name in a place where it might not otherwise be recognized as such (is not bounded by any of the other special characters listed above). Each → in the definition is removed when the macro is called, and the items it connects are catenated. For example, if the parameter P1 is substituted in the expansion by A2, and P2 by A, then

S→P1 P1+1R→P2 becomes SA2 A2+1RA

As this example indicates, the substitutable parameters may appear in any field of a statement in the macro body. The only place where parameters will not be noticed and substituted is in a comments line with an * in column 1. Comments cards within a macro definition are ignored and not reproduced when the macro is called.

Any instructions except END may appear within a macro definition including other macro definitions and/or macro calls. If a macro definition appears within another macro definition, it is not defined by COMPASS until the outer macro is called; therefore it may not be called before the outer macro is called. It may be called from within the macro in which it is defined according to general macro rules.

Example:

```
          :
          :
NAME1 MACRO  A
      SB1    A
NAME2 MACRO  A
      SB4    A
NAME2 ENDM
      NAME2  ALPHA   This call of NAME2 is valid since it is
       .             not recognized as a macro call until
       :             NAME1 is called and expanded.
```

```
        :
        :
NAME1 ENDM
        .                       NAME2 may not be called in this
        :                       part of the subprogram.
        NAME1   X
        NAME2   X               This call to NAME2 is valid since
        :                       NAME1 has already been called.
        .
```

Since the characters = . $) (act as delimiters in the macro body for formal parameters, the programmer must be careful if he uses these characters in symbols. For example, given the macro definition:

```
ABC     MACRO   Z,VAL
Z       SET     VAL
        SA7     Z.ALPHA
          :
          :
        ENDM
```

and the macro call:

```
ABC     IOTA,1
```

the reference in the SA7 instruction will not be to the symbol Z.ALPHA but to IOTA.ALPHA, which is illegal since the symbol name is too long. The entire expansion will be:

```
IOTA    SET     1
        SA7     IOTA.ALPHA
```

4.1.3 MACRO TERMINATOR

An ENDM terminates a macro definition.

Location	Operation	Variable
blank or macro name	ENDM	ignored

To be recognized as a macro definition terminator, the ENDM location field must be blank or contain the name of a macro being defined. An ENDM with a blank location field terminates any and all macros being defined; a named ENDM terminates a macro with the same name and its inner macros. An ENDM which terminates a definition always terminates any inner macro definitions for which a matching ENDM was not found.

4.2 MACRO CALL

A macro name in an operation field constitutes a macro call; it may contain a symbol in the location field, and a parameter list in the variable field. The parameter list of the macro call is scanned to identify and extract the character strings to be substituted for parameters of the macro definition. The parameter list has the following form, where p is a character string denoting an actual parameter.

```
p,p,p,...,p
```

Parameters of the macro call are listed in the same order as the formal parameters in the macro definition parameter list. Missing actual parameters are empty, or null, and extra actual parameters are discarded. An explicit zero, if desired, must be entered as a parameter. A blank terminates the parameter list unless the blank is contained within parentheses.

When the left parenthesis is the first character of any parameter, all characters between it and the matching right parenthesis are considered part of that parameter. The outer pair of parentheses is removed when the parameter is substituted in a line. Parenthesized items may be embedded provided parentheses are well paired. Parenthesized items may contain blanks and commas.

Example: If the macro XAM is defined:

```
XAM   MACRO   A,B
      LDM     A
      LJM     B
      ENDM
```

and a call is issued:

```
      XAM     (SUM,10B),(SAM,IND3)
```

COMPASS will expand the call as:

```
      LDM     SUM,10B
      LJM     SAM,IND3
```

Using the same macro XAM but with a call:

```
      XAM     SUM,SAM
```

COMPASS will expand the call as:

```
      LDM     SUM
      LJM     SAM
```

Processing of a location symbol on the macro call is dependent on the way the macro was defined:

(1) Standard Form Macro Definition (macro name appeared in the MACRO location field): a location symbol on the macro call line will cause a force upper and the symbol will be defined as the value of the location counter. For example, if the macro XAM is defined:

```
XAM   MACRO   A,B,C
      SB1     A
      SB2     B
      ENDM
```

and a call is issued:

```
LOC   XAM     X,Y
```

COMPASS will expand the call as if it were:

```
LOC   BSS     0
      SB1     X
      SB2     Y
```

If, however, there is no location symbol on the call, no force upper will occur and the SB1 operation will fall into the first available space.

(2) Alternate Form Macro Definition (macro name appeared as the first variable field subfield): the location symbol of the macro call is passed as the actual parameter to be substituted for the first formal parameter (the location argument) in the definition. Forcing upper is determined by the first instruction of the expansion. If there is no location field symbol on the macro call, the first argument is null or blank.

For example, if macro XAM is defined:

```
      MACRO   XAM,A,B,C
A     SB1     B
      SB2     C
      ENDM
```

and a call is issued:

```
LOC   XAM     X,Y
```

the expansion will appear as:

```
LOC   SB1     X
      SB2     Y
```

A force upper occurs because of the location field entry in the first line. If however, macro XAM is defined:

```
        MACRO   XAM,A,B,C
        SB1     B
A       SB2     C
        ENDM
```

and a call is issued:

```
LOC     XAM     X,Y
```

the expansion will appear as:

```
        SB1     X
LOC     SB2     Y
```

No force upper will occur for the SB1 operation but it will occur for the SB2.

Also if the macro XAM is defined:

```
        MACRO   XAM,A,B,C
A       SB1     B
        SB2     C
        ENDM
```

and a call is issued:

```
        XAM     X,Y
```

then the expansion will appear as:

```
        SB1     X
        SB2     Y
```

No force upper will occur since the parameter A is null.

4.3 OPDEF

OPDEF is a special macro form provided for the definition or redefinition of instructions in the format of central processor machine instructions whereby the macro call is written in the same format as the central processor operations for COMPASS. OPDEF provides more extensive control than the standard macro form.

4.3.1 OPDEF DEFINITION

The pseudo instruction OPDEF is used in place of MACRO. Following the OPDEF line is LOCAL (if needed), the macro definition, and ENDM — specified in the same manner as described in the section on MACRO.

The OPDEF heading line indicates the mnemonic name and variable field format which will be recognized as an OPDEF call, and lists the substitutable parameters as follows:

Location	Operation	Variable
Description of operation field and variable field of the OPDEF call	OPDEF	parameter list

Location Field of the OPDEF Line

This field contains an abbreviated description of the entire instruction to be recognized as an OPDEF call. This includes operation code, registers and/or address expressions which constitute the variable field, and subfield separators of the variable field in the macro call.

The first part of the OPDEF location field entry describes the operation field of the OPDEF call; it consists of two letters. The first may be any letter; the second may be a register designator: A, B, or X. In this case, the operation field of the OPDEF call is defined to be ℓAn, ℓXn, or ℓBn, where ℓ is any first letter.

The n of the OPDEF call may be any valid register value as described in the Register Name section of Chapter 2. If the second letter is not A, B, or X, the operation field of the OPDEF call is defined as a two-letter mnemonic, such as EQ.

The second part of the OPDEF location field entry describes the variable field of the OPDEF call. This includes all registers and/or address expressions which constitute the variable field as well as all subfield separators. This part of the OPDEF name may contain none, one, two, or three of the following 22 subfield descriptors, each descriptor separated by a comma; r represents a register letter, A, B, or X; Q represents an address expression.

void	Q
r	rQ
-r	-rQ
r+r	r+rQ
-r+r	-r+rQ
r*r	r*rQ
-r*r	-r*rQ
r/r	r/rQ
-r/r	-r/rQ
r-r	r-rQ
-r-r	-r-rQ

For example, -r+r could describe -X3*X0; rQ could describe B2+ALPHA.

The two parts of the OPDEF location field — op code description and variable field descriptors — are not separated by any special character unless this character is the operator to the first descriptor. Examples of the OPDEF name field (location field of the OPDEF line) and the macro call they describe are as follows:

Name Field	Call Described
JPQ	JP address expression
single descriptor, of the form Q	
JPBQ	JP Bn±address expression
single descriptor of the form rQ	
JPB+BQ	JP Bn±Bn±address expression
single descriptor of the form r+rQ	
NEB,B,Q	NE Bn, Bn, address expression
three descriptors of the form r,r, and Q	
LJB-B,A-X,Q	LJ Bn-Bn, An-Xn, address expression
three descriptors of the forms r-r, r-r, and Q	
BX-X*X	BXn -Xn*Xn
one descriptor of the form -r*r	
SBX+B	SBn Xn+Bn
single descriptor of the form r+r	
LXB,X	LXn Bn,Xn
two descriptors of the forms r and r	

In the OPDEF call, an address expression must be preceded by a plus or minus unless the Q in the descriptor is not combined with register letters.

Examples:

OPDEF Name Field	Call
JPQ	JP address expression
JPBQ	JP Bn ± address expression
JPB,Q	JP Bn, address expression
JPX/XQ	JP Xn/Xn ± address expression

In the following examples of OPDEF location field entries, all instructions described happen to look like legal COMPASS machine mnemonics.

To identify the JP instruction with a single address expression	JPQ
To identify JP Bj+K	JPB+Q
To identify NE Bj,Bk,K	NEB,B,Q
To identify BXi -Xk*Xj	BX-X*X
To identify SBi Xj+Bk	SBX + B
To identify SBi Bj+Xk	SBB+X

Operation Field of OPDEF Line

OPDEF

Variable Field of OPDEF Line

parameter list

The number of formal parameters listed in the OPDEF instruction variable field must match the total number of register and expression designators (A, B, X, and Q) in the format description. They are in the same order as the registers and expressions in the format description.

Examples of Complete OPDEF Definitions

(1) To redefine the single-address long jump, JP, as the fast jump, EQ;

```
JPQ   OPDEF   P1
      EQ      P1
      ENDM
```

All JP instructions subsequently encountered which match the format described by the OPDEF location field will be expanded as EQ. JP instructions not of that format, such as JP B3+ALPHA, would not be affected.

(2) For debugging, trap all floating double precision subtraction instructions (DXi Xj-Xk) and jump to an error-check routine: I, J, and K are substitutable parameters presumably used within the definition prototype.

```
DXX-X OPDEF   I,J,K
        .
        .
        .
      RJ      CKOUT
      ENDM
```

(3) Define a new instruction as a set of code which performs a complete integer divide each time it is called and expanded:

```
IXX/X OPDEF   P1,P2,P3
        .
        .
        .
      integer divide code
        .
        .
        .
      ENDM
```

Each time an instruction of the format IXn Xn/Xn is used, the macro will be expanded.

(4) Define RXi k to be the same as AXi k

```
RXQ   OPDEF   P1,P2
      AX.P1   P2
      ENDM
```

The instructions RXi Xj$\begin{bmatrix} + \\ - \\ * \\ / \end{bmatrix}$ Xk will not be affected.

4.3.2 OPDEF CALLS

The registers and/or address expressions used in the macro call must match exactly the number and order of registers and/or expressions indicated in the OPDEF location field description or the line will not be considered an OPDEF macro call. For example, given the definition header:

```
SXX+B   OPDEF   I,J,K
```

The following lines will not cause an expansion of the macro:

```
        SX5     X4
        SX5     X4-B3
        SX5     B3+X4
        SX5     B3
```

Only a line of this format will cause an expansion: SXn Xn+Bn

Unlike MACRO-defined macros, only the register value given in the call of an OPDEF-defined macro is used in the substitution of parameters. For example, using the IXX/X macro illustrated above, the following code might be included in its definition:

```
IXX/X   OPDEF   P1,P2,P3
        PX.P2   X.P2
        PX.P3   X.P3
        NX.P2   X.P2,B4
          .
          .
          .
        ENDM
```

The instruction which calls the IXX/X macro might be:

```
        IX3     X4/X.DIV
```

The parameters passed along to the macro body are 3, 4, and DIV, not X3, X4, and X.DIV.

A second difference between MACRO-defined macros and OPDEF-defined macros is that actual parameters of an OPDEF call are separated by + - * / or comma according to the definition of the OPDEF macro. Parameters of a MACRO-defined macro call may be separated only by commas.

A location field entry on the macro call means the same as a location field entry on a normal MACRO-defined macro call.

OPDEF definitions may appear anywhere in a subprogram. OPDEF calls are recognized at any place after the definition.

4.4 SYSTEM MACROS

Macros of such general usefulness that they should be available to any program without each program having to define them may be defined as system macros; or they may be defined as a result of the XTEXT definitions which exist on a separate file accessible to COMPASS.

System macros are a set of SCOPE-defined macros for communication with the operating system. They include such system functions as opening and closing files, reading, writing, specifying parameters for a file environment table, etc. The definitions of these macros exist on a system-maintained file, and they are available to COMPASS for every assembly. The programmer simply writes a macro call whenever a system macro is needed. The use of the system macros is detailed in SCOPE reference documents.† The file of systems text may contain any kind of legal macro definition, including OPDEF. The system macro definitions will not be included in the subprogram listing. The expansion of a system macro call may be obtained by using the S option on the LIST pseudo instruction. System macros may not redefine COMPASS mnemonics.

4.5 OPERATION CODE RECOGNITION ORDER

COMPASS interprets an operation code according to the following order of precedence:

(1) Programmer macro (highest)

(2) System macro

(3) COMPASS machine or pseudo instruction (lowest)

If two or more macros have the same name, whichever was last defined will be the macro expanded.

† 6400/6500/6600 SCOPE, Publication number 60189400.

The COMPASS micro capability enables the programmer to symbolically reference a defined character string. At assembly time, the character string will be substituted at any point in the line where that name appears before any other interpretation of the statement is attempted.

5.1 MICRO SUBSTITUTION

At any place in a statement a micro mark (≠) may appear followed by a string of characters and another micro mark. The enclosed characters constitute a micro name, which signals a micro substitution to be made at that point. Before the assembler attempts to interpret the instruction, the micro identified by the bracketed micro name will be substituted.

Example: The micro NAME might be defined as the characters

```
LOC SA1 ADDRESS+
```

Then, a symbolic instruction introduced as follows, in column 2

```
≠NAME≠4
```

would be changed by COMPASS into

```
LOC SA1 ADDRESS+4
```

where LOC begins in column 2.

If the second micro mark does not appear or if the micro name is unknown, a non-fatal assembly error results and no substitution is made. Micro marks are not processed if they appear in comments lines (* in column 1), but they will be processed if they are written in the comments field of an instruction line.

If, as a result of micro substitution, column 72 of the last card read is exceeded, the assembler creates continuation cards up to a maximum of 9. Anything exceeding 9 continuation lines will be discarded without comment.

5.2 MICRO DEFINITION

The MICRO pseudo instruction is used to define a character string and to assign a name to that string. Such a string is called a micro.

Location	Operation	Variable
micro name	MICRO	3 subfields separated by commas

The variable field subfields are, in order:

Absolute address expression n_1

Absolute address expression n_2

Delimited character string, dccc...ccd. The delimiter d is any character, and ccc...cc is a string of any characters not including character d.

Counting the first character after d as character 1, the string is formed by extracting n_2 characters starting with character n_1. For example:

```
NAME    MICRO   1,19,*ALPHANUMERIC STRING*
```

If the second delimiter is encountered before count n_2 is exhausted, the string is terminated at that point. If n_1 is non-zero, and n_2 is zero or absent, length of the character string is considered to be all the characters between the n_1^{th} character and the closing delimiter. The following example is therefore equivalent to the above.

```
NAME    MICRO   1,,*ALPHANUMERIC STRING*
```

If n_1 is zero or absent, the character string is empty, and there is no substitution when this micro name is given in an instruction line. n_2 and the character string are ignored.

Previously defined micros may appear as part of a micro definition. Thus, one micro may be defined as a substring of another. For example, assuming the micro

```
NAME1   MICRO   1,25,*MAJOR ALPHANUMERIC STRING*
```

has been defined in the program, an equivalent micro to the examples above can be achieved by the micro:

```
NAME    MICRO   7,,*≠NAME1≠*
```

Also a micro may be defined as a combination of multiple, previously defined micros. The following series would result in another equivalent to the previous examples:

```
NAME1   MICRO   1,12,*ALPHANUMERIC*
NAME2   MICRO   1,7,*ΔSTRING*
NAME    MICRO   1,,*≠NAME1≠≠NAME2≠*
```

The delimiter (* in the example) may not appear in either of the character strings substituted for NAME1 or NAME2. If the delimiter is encountered before the count n_2 is satisfied, the string will be ended.

A micro may be redefined; NAME may be originally defined as one character string and later again defined, but with a different character string. After the redefinition, the original character string is no longer known to the assembler. The original micro may also be used as part of the redefinition.

Example:

```
NAME    MICRO   1,6,*STRING*
          .
          .
          .
        a series of statements (A)
          .
          .
          .
NAME    MICRO   1,19,.*ALPHANUMERIC≠NAME≠*
          .
          .
          .
        a series of statements (B)
          .
          .
```

During statement series A the first definition of NAME prevails. During statement series B the redefinition of NAME prevails and the original string no longer exists.

Micros of different names but with identical character strings may co-exist at any time. Varied manipulation of character strings — testing for a particular character, counting characters, catenating strings, etc. — is possible in COMPASS with the use of MICRO in conjunction with IFC, DUP, STOPDUP, and SET pseudo instructions.

ASSEMBLER INPUT/OUTPUT 6

6.1 COMPASS CONTROL CARD

The files COMPASS uses are specified on the control card as follows:

```
COMPASS(L=fname, I=fname, B=fname, S=rname)
```

The specifications may be in any order; the characters = , (may be used interchangeably as separators; . and) are card terminators.

Each option is specified as follows:

L option:	absent	Full listings on OUTPUT
	L	Full listings on OUTPUT
	L=0	Brief listings on OUTPUT
	L=fname	Full listings on file fname
I option:	absent	Input from INPUT
	I	Input from INPUT
	I=fname	Input from file fname
B option:	absent	Binary on LGO
	B	Binary on LGO
	B=fname	Binary on file fname
S option:	absent	Systems text from SYSTEXT
	S	Systems text from SYSTEXT
	S=rname	Systems text from library overlay named rname

6.2 INPUT AND OUTPUT FILES

COMPASS assembles all statements beginning at the current position of the file specified as input until an end-of-record or end-of-file. If the input file is positioned at an end-of-file mark (the input file is empty), COMPASS will produce an error. Other input is from the system text record and XTEXT files. All input cards may be 90 columns; a card longer than 90 columns will be truncated. All input files are coded files. The assembly output consists of one logical record of listable output (for 136-column printers), and several logical records of binary output.

The binary output from an assembly consists of all loader control cards (LCC) written as individual records. Then an identification table of 14 words is written (77-table) with the name of the ensuing deck in the next word, followed by the deck. If errors occur in assembly, no binary output, except the 77 table and any LCC records, will appear.

For absolute programs, following the 77 table is another control word followed by the absolute program. This control word contains:

CP Programs: 5000 $L_1 L_2$ ffff fftt tttt

- L_1, L_2 = 00 for first overlay
  = 01 for subsequent overlays
- ffffff = origin -1 as specified on the IDENT line
- tttttt = entry point address as specified on IDENT line

PP Programs: nnnn nn00 ffff 0000 cccc

- nnnnnn = program name
- ffff = origin -5 as specified on the IDENT line
- cccc = program length (including this control word) in central memory: (program length+9)/5

Normally, systems text is derived from the library overlay named SYSTEXT, although this may be changed through the S option. Systems text overlays on the library look like loader overlays with the following control word:

5000 0101 0000 0000 0000

Data consists of coded lines. A word of minus zero follows the last coded line.

Scratch File

For large assemblies, a magnetic tape scratch file may be desirable to eliminate disk conflicts. This will have a negligible effect upon the CP time, but it will improve throughput time considerably. This may be accomplished by assigning a file named CMPSCR to tape.

6.3 FIELD LENGTH REQUIREMENTS

Since all COMPASS tables are variable, an exact field length is impossible to specify. A field length of 34000_8 should be sufficient for most assemblies. As part of the listable output, COMPASS gives the amount of storage not needed for the assembly. This can be used to lower the field length for subsequent runs.

When COMPASS does not have enough storage to complete processing, part or all of the reference table will be discarded. If this fails to release enough storage, assembly will terminate with a dayfile message.

6.4 LISTABLE OUTPUT

COMPASS list output contains, as a minimum header information: program name and length, block names and length, external symbol names, entry points. In addition, any lines which cause an error flag to appear are unconditionally listed. At the end of assembly, an error directory and assembler statistics appear.

6.4.1 HEADER INFORMATION

At the beginning of the listing, all blocks are listed as shown below (all programmer-defined blocks, even if length is zero, are listed).

Origin	Length	Name	Type
nnnnnn	nnnnnn	ABSOLUTE*	local
nnnnnn	nnnnnn	PROGRAM*	local
nnnnnn	nnnnnn	LITERALS*	local
nnnnnn	nnnnnn	$NAME_1$	local or common
.	.	.	.
.	.	(Programmer-declared blocks)	
.	.	.	.
nnnnnn	nnnnnn	$NAME_n$	local or common

6.4.2 ASSEMBLED CODE

The LIST pseudo instruction specifies the contents of the listing; but the COMPASS control card provides an external list control which overrides any LIST directives. If the external option is "no list" (L=0) only header information and error diagnostics are listed, regardless of LIST specifications. If the external option is "list," listing control is directed by the internal LIST options.

Each line of the listing will contain the following items after the header information:

Error flags, if any

LOC flag (an L if location counter is different from origin counter)

Location counter value

Octal value of code

Address relocation indicator

Card image (columns 1-72)

Columns 73-90 of the source line, or an indication of source if generated line

6.4.3 DIAGNOSTICS, REFERENCE TABLE, AND STATISTICS

Two types of errors are detected by COMPASS--fatal and non-fatal. Any fatal error will suppress binary output as well as terminate the job when assembly is finished. Non-fatal errors are merely warnings. Errors flagged with an alphabetic character are fatal; non-fatal warning flags are numeric. All lines with errors are listed. A one-character indication of each error on the line appears to the left. At the end of the assembly an error directory is listed. For each kind of error, the pages on which it occurred are noted, and a brief description of the error is given. The error flags are listed below:

L Location field bad. This can occur only on instructions which require a location field entry. Illegal entries in other location fields produce a non-fatal error flag since the illegality might not affect the rest of the assembly.

O Operation field bad. This can be a result of:

An unrecognized entry in the operation field.

The operation and address fields do not describe a valid CP instruction.

An unrecognized modifier in IF or IFC.

Misplaced operation, such as ABS or PERIPH.

A Address field bad. A general flag to indicate an illegality in the variable field. It can occur on almost any operation.

D Doubly defined symbol. This appears on all operations which attempt to define a symbol with a value different than its previous value. It also occurs if an attempt if made to define a redefinable symbol or redefine a normal symbol.

R Data origins outside block. This appears when data is loaded outside the block ranges, or into blank common.

F Number of entries exceeds permissible amount. This appears when:

The total number of words required for any one literal, data item, or the entire address field of a LIT operation exceeds 100.

More than 63 parameters appear in a macro definition.

The assembler symbol table overflows. This limit is between 4096 and 4350 depending upon the symbols used.

U An undefined symbol is referenced.

V Invalid bit count on a VFD instruction. It must be an absolute value between 0 and 60.

P Produced by an ERR instruction.

NON-FATAL ERROR FLAGS

1 Bad location field entry. The symbol the programmer is attempting to define will not be defined.

2 Bad address element on a symbol definition instruction. The location symbol will not be defined.

3 Macro redefines a previously known operation.

4 Bad parameter name is ignored.

5 OPDEF is incorrectly specified.

6 Location field is meaningless.

7 Address value exceeds field size; the result is truncated.

8 Address subfield is missing, or there are too many subfields.

9 Micro substitution error; no substitution will be made.

Assembler statistics listed next include:

Decimal count of statements processed by COMPASS, including all generated lines

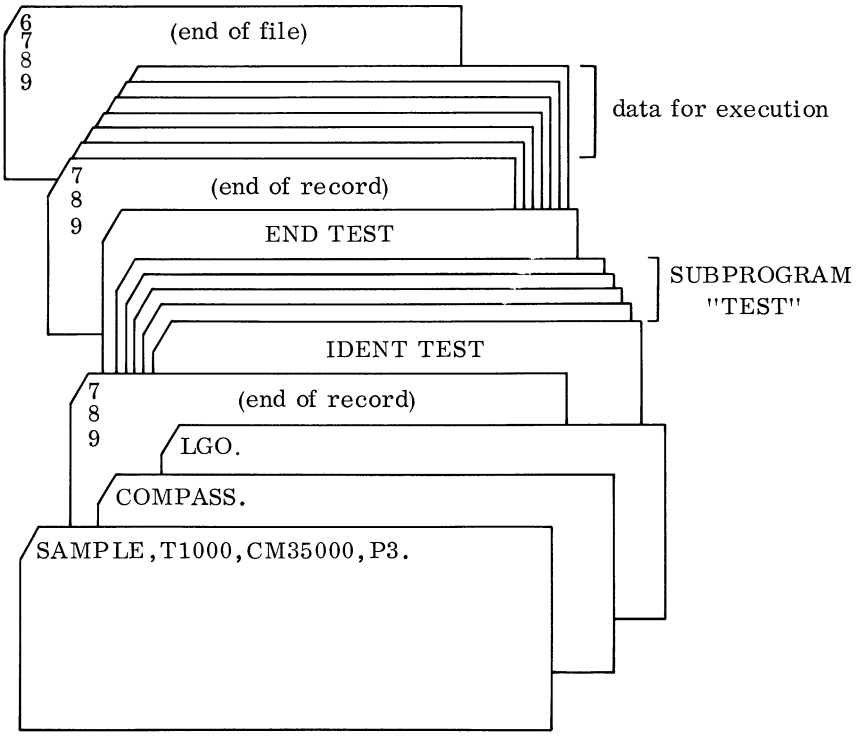
Indication of storage unused by the assembler so that field length might be adjusted in subsequent assemblies

Decimal count of reference table entries discarded because of restricted storage.

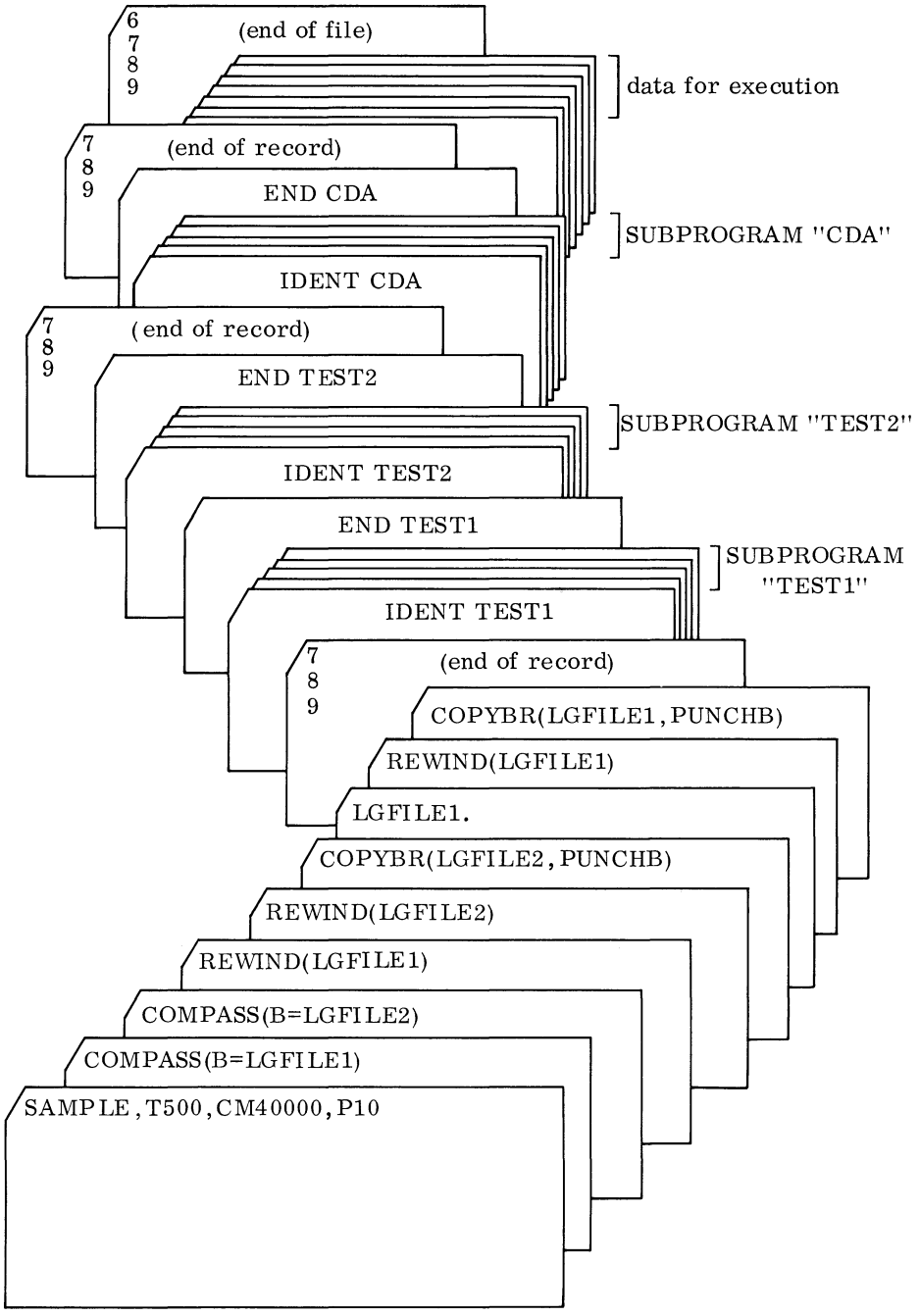
If a symbol reference table is requested, it is listed next. The reference table contains all symbols in alphabetical order (sorted according to the collating sequence in Appendix D), with their relocation, value, and all reference locations. Undefined symbols also appear, with a U error.

6.5 EXAMPLES OF JOBS

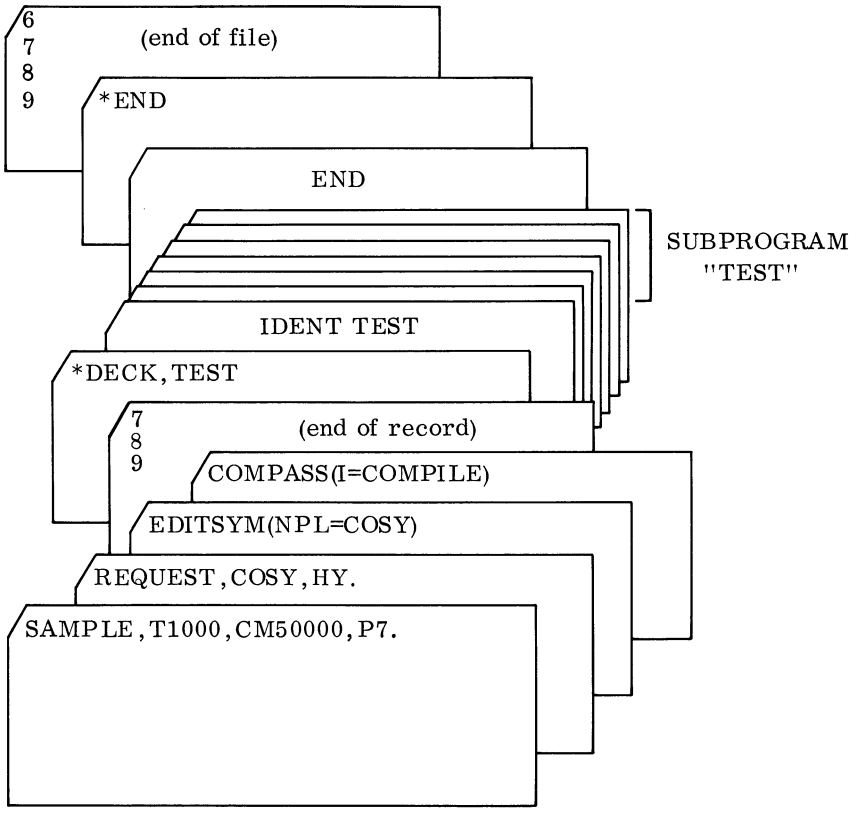
(1) Assemble with listing and binary output; subprogram execution with data input. Source logical record is on file INPUT, listing on file OUTPUT, binary on file LGO, execution data on file INPUT.



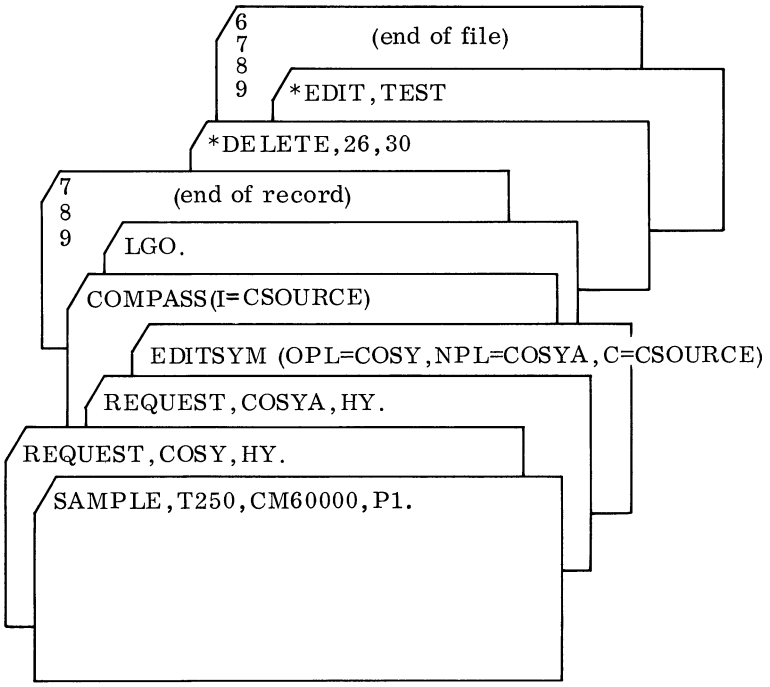
(2) Batch assemble with listing and binary output; punch the binary output and execute the first program.



(3) Create a compressed symbolic deck (via EDITSYM) of a subprogram. Assemble with listing.



(4) Update the compressed symbolic record COSY created in the previous record; write corrected compressed record on file COSYA and a corrected source record on file CSOURCE. Assemble the file CSOURCE and execute.



APPENDIX SECTION

CENTRAL PROCESSOR MNEMONICS A

Each operation is defined by listing the mnemonic, each subfield, the octal representation of the operation, and the instruction length in bits. Instructions are listed in order of octal operation value. An entry is given for each permissible variable field format. In the operation field and variable field subfield notations, the following symbology is used:

Xi,Xj,Xk	X register symbols. The number of the register is placed in the i,j, or k portion.
Ai,Aj,Ak	A register symbols
Bi,Bj,Bk	B register symbols
K	address expression (18 bits)
n	absolute address (6 bits)

Length	Mnemonic	Variable Field	Octal
30	PS		0000 000000
30	RJ	K	0100 K
30	RE	Bj,K	011j K
30	WE	Bj,K	012j K
60	XJ		0130 000000 46000 46000
30	JP	K	0200 K
30	JP	Bj+K	020j K
30	ZR	Xj,K	030j K
30	NZ	Xj,K	031j K
30	PL	Xj,K	032j K
30	NG	Xj,K	033j K
30	IR	Xj,K	034j K
30	OR	Xj,K	035j K
30	DF	Xj,K	036j K
30	ID	Xj,K	037j K
30	ZR	K	0400 K
30	EQ	K	0400 K
30	EQ	Bi,K	04i0 K
30	ZR	Bi,K	04i0 K
30	EQ	Bi,Bj,K	04ij K
30	NZ	Bi,K	05i0K
30	NE	Bi,K	05i0 K

Length	Mnemonic	Variable Field	Octal
30	NE	Bi,Bj,K	05ij K
30	PL	Bi,K	06i0 K
30	GE	Bi,K	06i0 K
30	GE	Bi,Bj,K	06ij K
30	LE	Bj,K	060j K
30	LE	Bj,Bi,K	06ij K
30	NG	Bi,K	07i0 K
30	LT	Bi,K	07i0 K
30	LT	Bi,Bj,K	07ij K
30	GT	Bj,K	070j K
30	GT	Bj,Bi,K	07ij K
15	BXi	Xj	10ijj
15	BXi	Xj*Xk	11ijk
15	BXi	Xj+Xk	12ijk
15	BXi	Xj-Xk	13ijk
15	BXi	-Xk	20ikk
15	BXi	-Xk*Xj	15ijk
15	BXi	-Xk+Xj	16ijk
15	BXi	-Xk-Xj	17ijk
15	LXi	n	20i n
15	AXi	n	20i n
15	LXi	Bj,Xk, or Xk,Bj	22ijk
15	AXi	Bj,Xk or Xk,Bj	23ijk
15	NXi	Xk	24i0k
15	NXi	Bj,Xk or Xk,Bj	24ijk
15	ZXi	Xk	25i0k
15	ZXi	Bj,Xk or Xk,Bj	25ijk
15	UXi	Xk	26i0k
15	UXi	Bj,Xk or Xk,Bj	26ijk
15	PXi	Bj,Xk or Xk,Bj	27ijk
15	FXi	Xj+Xk	30ijk
15	FXi	Xk-Xk	31ijk

Length	Mnemonic	Variable Field	Octal
15	DXi	Xj+Xk	32ijk
15	DXi	Xj-Xk	33ijk
15	RXi	xj+Xk	34ijk
15	RXi	Xj-Xk	35ijk
15	IXi	Xj+Xk	36ijk
15	IXi	Xj-Xk	37ijk
15	FXi	Xj*Xk	40ijk
15	RXI	Xj*Xk	41ijk
15	DXi	Xk*Xk	42ijk
15	MXi	n	43i n
15	FXi	Xj/Xk	44ijk
15	RXi	Xj/Xk	45ijk
15	NO		46000
15	CXi	Xk	47ikk
30	SAi	Aj+K	50ij K
30	SAi	K	51i0 K
30	SAi	Bj+K	51ij K
30	SAi	xj+K	52ij K
15	SAi	xj	53ij0
15	SAi	Xj+Bk or Bk+Xj	53ijk
15	SAi	Aj	54ij0
15	SAi	Aj+Bk or Bk+Aj	54ijk
15	SAi	Aj-Bk or -Bk+Aj	55ijk
15	SAi	Bj	56ij0
15	SAi	Bj+Bk	56ijk
15	SAi	-Bk	57i0k
15	SAi	Bj-Bk or -Bk+Bj	57ijk
30	SBi	Aj+K	60ij K
30	SBi	K	61i0 K
30	SBi	Bj+K	61ij K
30	SBi	Xj+K	62ij K
15	SBi	Xj	63ij0

Length	Mnemonic	Variable Field	Octal
15	SBi	Xj+Bk or Bk+Xj	63ijk
15	SBi	Aj	64ij0
15	SBi	Aj+Bk or Bk+Aj	64ijk
15	SBi	Aj-Bk or -Bk+Aj	65ijk
15	SBi	Bj	66ij0
15	SBi	Bj+Bk	66ijk
15	SBi	-Bk	67i0k
15	SBi	Bj-Bk or -Bk+Bj	67ijk
30	SXi	Aj+K	70ij K
30	SXi	K	71i0 K
30	SXi	Bj+K	71ij K
30	SXi	Xj+K	72ij K
15	SXi	Xj	73ij0
15	SXi	Xj+Bk or Bk+Xj	73ijk
15	SXi	Aj	74ij0
15	SXi	Aj+Bk or Bk+Aj	74ijk
15	SXi	Aj-Bk or -Bk+Aj	75ijk
15	SXi	Bj	76ij0
15	SXi	Bj+ Bk	76ijk
15	SXi	-Bk	76i0k
15	SXi	Bj-Bk or -Bk+Bj	77ijk

PERIPHERAL PROCESSOR MNEMONICS† B

Machine Instruction		Octal Value	
PSN		0000	
LJM	M,d	01dd	MMMM
RJM	M,d	02dd	MMMM
UJN	r	03rr	
ZJN	r	04rr	
NJN	r	05rr	
PJN	r	06rr	
MJN	r	07rr	
SHN	d	10dd	
LMN	d	11dd	
LPN	d	12dd	
SCN	d	13dd	
LDN	d	14dd	
LCN	d	15dd	
ADN	d	16dd	
SBN	d	17dd	
LDC	C	20CC	CCCC
ADC	C	21CC	CCCC
LPC	C	22CC	CCCC
LMC	C	23CC	CCCC
EXN	d	260d	
MXN	d	260d	
RPN	d	260d	
LDD	d	30dd	
ADD	d	31dd	
SBD	d	32dd	
LMD	d	33dd	
STD	d	34dd	
RAD	d	35dd	
AOD	d	36dd	
SOD	d	37dd	

† Notations: M = 12-bit address value, C = 18-bit address value, d = 6-bit index value, r = value between -31 and +31.

Machine Instruction		Octal Value	
LDI	d	40dd	
ADI	d	41dd	
SBI	d	42dd	
LMI	d	43dd	
STI	d	44dd	
RAI	d	45dd	
AOI	d	46dd	
SOI	d	47dd	
LDM	M,d	50dd	MMMM
ADM	M,d	51dd	MMMM
SBM	M,d	52dd	MMMM
LMM	M,d	53dd	MMMM
STM	M,d	54dd	MMMM
RAM	M,d	55dd	MMMM
AOM	M,d	56dd	MMMM
SOM	M,d	57dd	MMMM
CRD	d	60dd	
††CRM	M,d	61dd	MMMM
CWD	d	62dd	
††CWM	M,d	63dd	MMMM
††AJM	M,d	64dd	MMMM
††IJM	M,d	65dd	MMMM
††FJM	M,d	66dd	MMMM
††EJM	M,d	67dd	MMMM
IAN	d	70dd	
††IAM	M,d	71dd	MMMM
OAN	d	72dd	
††OAM	M,d	73dd	MMMM
ACN	d	74dd	
DCN	d	75dd	
FAN	d	76dd	
††FNC	M,d	77dd	MMMM

†† A warning flag will be given if d is absent.

PSEUDO INSTRUCTIONS C

ABS	Declares absolute assembly
BASE	Declares mode of integers - octal or decimal
BSS	Allocates a block of storage
BSSZ	Allocates a zero-filled block of storage
DATA	Defines absolute data items
DIS	Defines display code data
DUP	Duplicates a sequence of code
EJECT	Ejects a page
END	Ends a subprogram
ENDD	Ends a DUP range
ENDIF	Ends a conditional range
ENDM	Ends a macro definition
ENTRY	Declares subprogram entry points
EQU	Equates a symbol to a value
ERR	Produces a fatal error flag
EXT	Defines symbols external to the subprogram
HERE	Calls for remote coding to be assembled
IDENT	Identifies beginning of a subprogram
IFEQ, IFNE, . . .	Compares two values for EQ, NE, GT, GE, LT, LE, and conditionally assembles a code sequence.
IF	Tests a symbol for the attributes, absolute, relocatable, common, external, local SET, register, defined, and conditionally assembles a code sequence.
IFPP, IFCP	Tests assembly environment (PP or CP)
IFC	Compares two character strings
LIST	Declares assembly listing control parameters
LIT	Declares literals
LOC	Resets location counter
LOCAL	Declares symbols local to a macro
MACRO	Introduces a macro definition
macro name	Calls a macro

MICRO	Defines a micro (character string)
OPDEF	Defines a macro
ORG	Resets origin counter
PERIPH	Declares a peripheral processor subprogram
REP	Declares loader-controlled code duplication
RMT	Introduces a sequence of remote code
SET	Equates a redefinable symbol to a value
SPACE	Spaces output listing
STOPDUP	Stops a DUP process
TITLE	Defines listing title or subtitle
USE	Names a block for subsequent code to be placed into
VFD	Assigns data in variable byte sizes
XTEXT	Calls for text from an external source

CHARACTER CODES COLLATING SEQUENCE D

Character	Display Code	External BCD	Hollerith Punch Positions
A	01	61	12-1
B	02	62	12-2
C	03	63	12-3
D	04	64	12-4
E	05	65	12-5
F	06	66	12-6
G	07	67	12-7
H	10	70	12-8
I	11	71	12-9
J	12	41	11-1
K	13	42	11-2
L	14	43	11-3
M	15	44	11-4
N	16	45	11-5
O	17	46	11-6
P	20	47	11-7
Q	21	50	11-8
R	22	51	11-9
S	23	22	0-2
T	24	23	0-3
U	25	24	0-4
V	26	25	0-5
W	27	26	0-6
X	30	27	0-7
Y	31	30	0-8
Z	32	31	0-9
0	33	12	0
1	34	01	1
2	35	02	2
3	36	03	3
4	37	04	4

Character	Display Code	External BCD	Hollerith Punch Positions
5	40	05	5
6	41	06	6
7	42	07	7
8	43	10	8
9	44	11	9
+	45	60	12
-	46	40	11
*	47	54	11-8-4
/	50	21	0-1
(	51	34	0-8-4
)	52	74	12-8-4
$	53	53	11-8-3
=	54	13	8-3
blank	55	20	space
,	56	33	0-8-3
.	57	73	12-8-3
≡	60	36	0-8-6
[	61	17	8-7
]	62	32	0-8-2
:	63	00	8-2
≠	64	14	8-4
→	65	35	0-8-5
∨	66	52	11-0
∧	67	37	0-8-7
↑	70	55	11-8-5
↓	71	56	11-8-6
<	72	72	12-0
>	73	57	11-8-7
≤	74	15	8-5
≥	75	75	12-8-5
¬	76	76	12-8-6

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET
6400/6500/6600 COMPASS
Reference Manual

Pub. No. 60190900 April, 1967

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

FROM NAME: _____

BUSINESS ADDRESS: _____

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY

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

FOLD FOLD

STAPLE STAPLE

6400/6500/6600 COMPASS Reference Manual

CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60190900

Litho in U.S.A.