14061305 A

CONTROL DATA CORPORATION

MP–32 COMPASS REFERENCE MANUAL

CONTROL DATA®
MP-32
COMPUTER SYSTEMS

REVISION RECORD	
REVISION	DESCRIPTION
A	Original release.
(01/03/83)	
Document No. 14061305	

Revision letters I, O, Q, and X are not used.

Please address comments concerning this manual to:

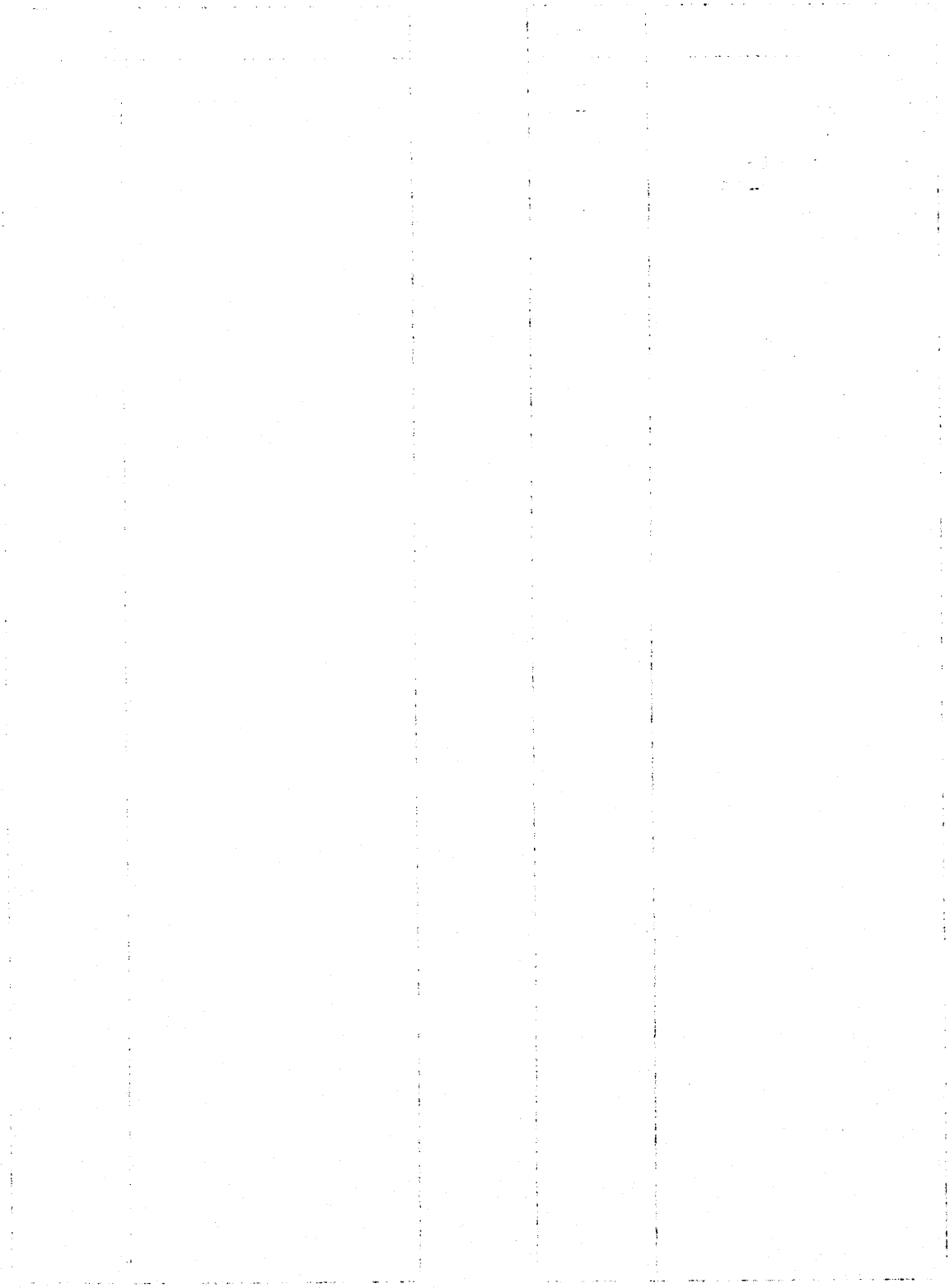
CONTROL DATA CORPORATION
SYSTEMS TECHNOLOGY DIVISION
215 Moffett Park Drive
Sunnyvale, California 94086

Or use the Comment Sheet in the back of this manual.

LIST OF EFFECTIVE PAGES

Page	Revision
Cover	-
Title Page	-
ii through x	A
1-1 through 1-4	A
2-1 through 2-10	A
3-1 through 3-22	A
4-1 through 4-6	A
A-1 through A-19	A
B-1 through B-2	A
C-1 through C-2	A
D-1 through D-2	A
I-1 through I-4	A
Comment Sheet	A

Page	Revision



PREFACE

This publication is a reference manual for the programmer using the MP-60 COMPrehensive ASsembly System (COMPASS) language. The manual defines the external characteristics of the MP-60 COMPASS, that is, those features which are observable or under the active control of the user.

Knowledge of the MP-60 computer operation is assumed for this manual. The operation of the MP-60 machine instructions is described in the MP-60 Computer System Reference Manual (Control Data publication No. 14306500).

CONTENTS

Section		Page
1	INTRODUCTION	1-1
	MP-60 COMPASS Language	1-1
	COMPASS Subprograms	1-2
2	COMPASS PROGRAMS	2-1
	Statement Format	2-1
	Location Field	2-3
	Operation Field	2-4
	Address Field	2-4
	Symbols	2-5
	Constants	2-6
	Expressions	2-7
	Literals	2-8
	Comments Field	2-9
3	PSEUDO INSTRUCTIONS	3-1
	Program Sectioning and Linking	3-1
	IDENT	3-1
	END	3-2
	ENTRY	3-2
	EXT	3-2
	SCOM	3-3
	DCOM	3-4
	COMPOOL	3-4
	Program Control	3-5
	ORG	3-5
	ORGR	3-6
	CONDITIONALS	3-6
	IF	3-6a

CONTENTS (CONT.)

Section		Page
	IFZ	3-6a
	IFG	3-6b
	IFGE	3-6b
	IFGT	3-6b
	IFL	3-6c
	IFLE	3-6c
	IFLT	3-6c
	IFN	3-6d
	ENDIF	3-7
	FINIS	3-8
	Symbol Definitions	3-9
	EQU	3-9
	EQU,H	3-10
	EQU,C	3-10
	EQU,W	3-11
	SET	3-11
	Storage Definitions	3-11
	BSS	3-12
	BSS,H	3-13
	BSS,C	3-14
	BSS,B	3-14
	BSS,D	3-15
	Data Definitions	3-15
	GEN	3-15
	VFD	3-17
	TEXT	3-18
	TEXTC, TEXTH	3-18
	Listing Control	3-19
	TITLE	3-19
	EJECT	3-19
	SPACE	3-19
	LISTF	3-20
	LIST	3-20

CONTENTS (CONT.)

Section		Page
	LISTCC	3-21
	LISTIF	3-21
	LISTMC	3-21
	BOX	3-22
	EBOX	3-22
4	MACROS	4-1
	Macro Heading	4-1
	Prototype	4-3
	Macro Terminator	4-3
	Macro Calls	4-3
	Nesting of Macros	4-6
	Library Macros	4-6

APPENDIXES

A Machine Language Instructions

B COMPASS Control Card

C COMPASS Output

D COMPOOL/Library Macros

INDEX

ILLUSTRATIONS

Figure		Page
2-1	Typical COMPASS Coding Form.	2-2

INTRODUCTION 1

MP-60 COMPASS is the COMPrehensive ASsembly System for the CONTROL DATA® MP-60 Computer System. MP-60 COMPASS translates the programmer's symbolic assembly language into machine language instructions, assigns storage locations, and produces a printed assembly listing and a loadable output.

MP-60 COMPASS LANGUAGE

The MP-60 COMPASS assembly language is designed to enable efficient use of all the computer resources while giving the programmer maximum flexibility in program construction. The language allows all hardware functions of the MP-60 computer system to be expressed symbolically. The user writes machine language programs in mnemonic instructions and symbolic addresses. In addition to individual computer instructions and data items, MP-60 COMPASS accepts programmer-defined macros.

Programmer-defined macros allow definition of a sequence of instructions as a macro. Once the macro is defined, each time the associated macro name appears, the sequence of instructions will be inserted by the MP-60 COMPASS assembler.

In order to simplify program coding and debugging, MP-60 COMPASS provides a number of pseudo instructions which control the assembly process. A number of the features of MP-60 COMPASS are summarized in the following listing.

- Address Arithmetic and Validation

  Constants, symbolic addresses, and arithmetic expressions may be used to represent the value of the address field.

- Preloaded Data

  Data storage areas may be specified and loaded with values at the same time the program is loaded. These areas may be used for subprogram communications.

- Common Assignment

  Common areas may be designated to provide for communication among MP-60 COMPASS subprograms.

- Data Definition

  Integer, floating point, ASCII, and hexadecimal constants may be coded using familiar notation.

- Variable Field Definition

  Structured information can be linked as arbitrary fields within core storage with little regard for word boundaries.

- Listing Control

  The format of assembly listings may be controlled with pseudo instructions.

- Diagnostics

  Diagnostics (error flags) for source program listings are included in the output listing.

- Macro

  The capability to define, retrieve, and expand macros is provided. Macros may be nested.

- IF Statements

  The conditional assembly of statements is provided by IF and ENDIF statements.

- Relocation

  Subprograms may be defined at absolute or relocatable origins. Multiple use of the ORG or ORGR statements is allowed, within specified limits.

- Subprogram Linkage

  Statements that define the entry points and external linkage of a subprogram are provided.

- Assembled Listing

  A listing containing source program statements, the resulting object code, program size, core storage assignments, and a cross-reference of symbol definitions and address field use is given.

COMPASS SUBPROGRAMS

To execute a program, the operating system often loads and links together several routines that have been assembled separately. For this reason, it is customary to refer to individual COMPASS routines as subprograms.

COMPASS accepts assembly language subprogram inputs on cards, magnetic tape, or disk in source language. It translates these source language statements into machine language instructions, and data, which are referred to as object code. Object code is prepared in a loadable format that can be loaded into memory when the program is to be executed. The assembler produces the following outputs:

- A printer listing of the assembled subprogram with hexadecimal object code and symbolic source instructions side by side. Error diagnostics automatically accompany this listing.

- Loadable binary output for subsequent loading and execution of the assembled subprogram.

Each subprogram consists of an IDENT pseudo instruction, subsequent lines of coding, and an END pseudo instruction. The last subprogram must be immediately followed by a FINIS pseudo instruction. Subprograms communicate with each other by use of entry points and external symbols (ENTRY and EXT pseudo instructions).

Three main regions provide storage for assembled subprograms.

- Data common region

- Scratch common region

- Subprogram region

The three types of regions are defined at assembly time. When more than one subprogram is to be loaded for execution at run time, the first specification of a data or common region must be the maximum length region for all programs loaded.

COMPASS object code may contain relocatable addresses, which are modified by a relocation factor during loading to obtain the actual address in the computer memory. When assembling subprograms, COMPASS assumes that the initial location in each of the three types of regions (data, subprogram, common) has a relocatable address of zero. Locations are then assigned sequentially from zero unless the pseudo instruction ORGR is encountered. ORGR instructs COMPASS to assign the value in the address field of ORGR as the relocatable address of the following instruction and assign storage sequentially from that relocatable address. The address counter affected by ORGR is the counter in use when ORGR is processed. All counters are initialized to zero before assembling a new subprogram.

COMPASS PROGRAMS 2

An MP-60 COMPASS subprogram consists of a sequence of symbolic source statements. These statements are made up of language elements combined in accordance with a set of definable rules.

Statements fall into four categories:

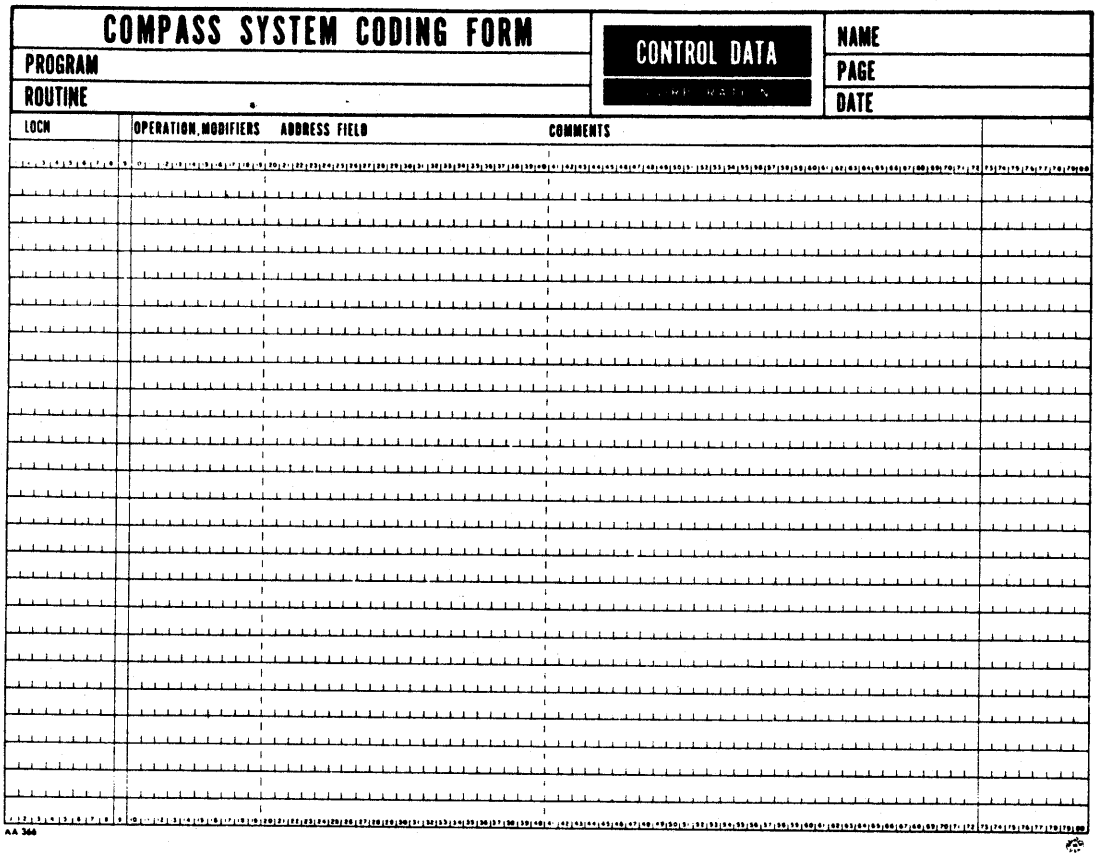
- A normal statement, which is assembled and which may produce loadable output
- A control statement, which affects the operation of the assembler
- A statement that is part of a definition (those lines contained between a macro and its ENDM)
- A comment statement, which appears in the listing but has no effect upon the assembly

Elements are classified as follows:

- Operation codes
- Pseudo operation codes
- Symbols
- Constants
- Literals
- Operators
- *(asterisk)
- Macro names

STATEMENT FORMAT

COMPASS statements are written on coding forms (refer to Figure 2-1) and subsequently punched into cards or prepared on other media for input to COMPASS. Each line on the coding sheet is normally punched into a single card. The correspondence between columns on coding sheets and cards is one to one.



NOTE: The coding form shown is presented as an illustration of a typical form. Field boundaries shown are not to be construed as limitations of COMPASS.

Figure 2-1. Typical COMPASS Coding Form

Each line of code is field free; all statements are defined in terms of the contents of the following fields.

	Field	Columns
1)	Location	The location field begins in column 1 and is terminated by the first blank column.
2)	Operation	The operation field contains the operation code. The field begins after the first blank column and is terminated by a blank.
3)	Address	The address field begins after the blank terminating the operation field and terminates with a blank or column 80.
4)	Comments	If an address field is used, the comments field starts after the blank ending the address field. If an address field is not required, the comments field is separated from the operation field by at least one blank. If an address field is required but is blank, comments must start after column 40. Comments may extend through column 80.

LOCATION FIELD

The location field begins in column 1 and is terminated by the first blank. It may not extend for more than eight characters. The location field may contain a symbol or an asterisk (*), or it may be left blank.

The location field symbol is used to identify a position within a subprogram or may be equated to a value. This symbol can be referenced by an instruction or pseudo instruction throughout the program. A location field symbol consists of from one to eight characters. The first character of a symbol must be alphabetic and must be placed in column 1.

Embedded blanks are illegal. An illegal symbol is flagged as an L error on the assembly listing.

When an asterisk appears in column 1, columns 2 through 80 are treated as a comment.

Examples of location field symbols:

Acceptable	Unacceptable
A123456	12345678
H3	.2345678
ABCDEFGH	TOOMANYCHARACTERS

OPERATION FIELD

The second field on the COMPASS coding form is defined as the operation field. The operation field may contain instruction mnemonics with specific related modifiers, pseudo instruction mnemonics, or macro instruction names.

An operation code modifier may occur after the mnemonic to further define the desired instruction. Modifiers are separated from operation codes by commas.

Illegal operation codes and modifiers are flagged as diagnostic errors on the assembly listing.

Examples of acceptable and unacceptable operation fields are as follows:

Acceptable	Unacceptable	Reason
STC, F	STC,	Illegal terminator
LDH, F	LDHF	Missing comma
BSS, D	R	No modifier

ADDRESS FIELD

The third field coded on the COMPASS coding form is the address field. The contents of the address field are dictated by the operation code. The address field begins after the blank terminating the operation field and terminates with a blank or column 80.

The following terms can be used in the address subfields:

- Symbols (location or value)
- Constants

- Register designators [constant or symbol (value)]
- Expressions consisting of symbols and constants joined by operators
- Asterisk
- Double Asterisk

SYMBOLS

A symbol is a string of one to eight characters representing a value or an address. A symbol conforms to the rules described for the location field. An address field symbol may constitute the entire field or it may be one of several elements in the field. Any symbol used in the address field must be defined by the appearance in the location field of another statement in the subprogram, or it must be declared external (EXT). A symbol may be nonrelocatable or relocatable.

A nonrelocatable symbol references an absolute address. The value assigned to the nonrelocatable symbol will not be modified during loading. A symbol can be defined as nonrelocatable through the use of an EQU pseudo instruction.

A relocatable symbol represents a symbolic address. Relocatable addresses are values related to a predefined memory area. These values are incremented or decremented by the loader prior to loading of the instruction in which the address occurs. Relocatable symbols may be:

- Subprogram relocatable
- External symbols
- Data relocatable
- Common relocatable

A symbol may represent a 16-bit (word), 17-bit (halfword), 18-bit (character), or 21-bit (bit) address, and the usage of symbols is interchangeable. If, for example, a word address defined symbol is used in a byte addressing instruction the word address is effectively shifted left two places with zero fill.

If a symbol is used in a smaller subfield (such as 18 bit to 16 bit) the address is shifted right; if a one bit is lost by the shift, a T error occurs.

The special symbol, *, may be placed in the address field and used as any symbol. The * is interpreted as the current value of the COMPASS counter in effect when the * is encountered (program or data). If the machine instruction occupies two words, * is the address of the first word.

The special symbol, **, may be used as the only entry in a field or subfield. The ** yields a subfield containing a one in each bit position. Normally, the field represented by the ** will be modified during the execution of the program.

CONSTANTS

The address field may contain signed or unsigned decimal, hexadecimal, or ASCII integers. If the sign is not present, the integer is assumed to be positive.

The allowable formats in the address field of an instruction are:

- Decimal

  Any numeric value is processed as decimal. The decimal value may be modified by a decimal power of 10 (P), a binary scale factor (S), or both.

  Examples:

Address field Constant	Hexadecimal Result (16-bit address)
1	0001
-1	FFFF
1P1	000A
1P1S-1	0005

- Hexadecimal

  Data preceded by a $ specifies a hexadecimal constant. The specified value may be modified by a binary scale factor.

  Examples:

Address Field Constant	Hexadecimal Result (16-bit address)
$A0	00A0
$A0S1	0140
-$1	FFFF

- ASCII

  Data enclosed within apostrophes specifies 8-bit ASCII code characters. Characters are right justified with zero fill. Blank is a significant character.

  Examples:

Address field Constant	Hexadecimal Result (16-bit address)
'B'	0042
' B'	2042
-'AA'	BEBF

EXPRESSIONS

In an address field or subfield, symbols, the special symbol, *, and constants may be combined with the operators, plus or minus, to form an address expression. The value of the expression is calculated by substituting the numeric value of the symbol and performing 16-, 17-, 18-, or 21-bit arithmetic with the designated operators. External symbols, the double asterisk, and literals may not appear in an address expression.

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

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

D_i = data common relocatable addresses

P_i = subprogram relocatable addresses

C_i = scratch common relocatable addresses

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

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

LITERALS

If the address field or subfield of an instruction refers to an operand which may be a single or double precision value, the entry may be a literal expressed as an equal sign followed by a mode designator and a value (=mv). The absence of a mode designator implies decimal mode.

The equal sign denotes that the field contains a literal; m indicates the mode of the literal; v is the value of the literal. The precision of the literal is derived from the precision of the operation performed (LD, LDD, etc.).

The mode of a literal may be decimal, hexadecimal, ASCII, or symbolic.

Decimal Literals: =v

The value of the decimal literal is expressed in the same manner as the GEN pseudo instruction; they may be signed, cannot be more than 10 digits (20 for double precision), and may be followed by a scaling factor. A blank terminates the field. If a decimal point (.) is encountered in the literal, the value is converted to floating point format.

Hexadecimal Literals: =$v

The value of the hexadecimal literal is written in the same manner as a GEN pseudo instruction; it may be signed, cannot be more than eight digits (16 for double precision), and may be followed by a scaling factor. A blank terminates the field. A nonhexadecimal character is illegal.

ASCII Literals: ='v'

The ASCII literal is expressed as a string of up to four or eight characters enclosed in quote (') marks. Blanks are significant in an ASCII literal.

Symbolic Literals: =Sv

The symbolic literal is expressed as a legal COMPASS symbol. The symbolic literal causes storage to be reserved for the symbol.

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

COMMENTS FIELD

Comments may be included with any instruction. The comments field may begin after the blank column that terminates the address field and may extend through column 80. If the instruction has no address field, comments can begin in column 41. The comments field has no effect on the assembly but its contents are printed on the output listing. If an asterisk is placed in column 1, the assembler interprets the entire line as comments.

REGISTER EQUATES

The COMPASS assembler will preset the following equates which are used as a coding convention in defining register equates.

```
X0    EQU    0          R0    EQU    16
X1    EQU    1          R1    EQU    17
X2    EQU    2          R2    EQU    18
X3    EQU    3          R3    EQU    19
X4    EQU    4          R4    EQU    20
X5    EQU    5          R5    EQU    21
X6    EQU    6          R6    EQU    22
X7    EQU    7          R7    EQU    23
H0    EQU    8          R8    EQU    24
H1    EQU    9          R9    EQU    25
H2    EQU    10         RA    EQU    26
H3    EQU    11         RB    EQU    27
H4    EQU    12         RC    EQU    28
H5    EQU    13         RD    EQU    29
H6    EQU    14         RE    EQU    30
H7    EQU    15         RF    EQU    31
```

PSEUDO INSTRUCTIONS 3

Pseudo instructions are nonmachine language instructions used to prepare a subprogram for COMPASS assembly. Some of the pseudo instructions provide the COMPASS assembler with the required information. Others are programmer aids for defining parts of a program, allocating storage, declaring symbols and constants, and generally controlling the assembly and listing processes. The pseudo instructions associated entirely with macros are discussed in Section 4.

PROGRAM SECTIONING AND LINKING

This group of pseudo instructions defines the name, beginning, and end of a subprogram and the linkage between subprograms during execution.

IDENT

Location	Operation	Address	Comments
1 8	10	20	41
	IDENT	m	

IDENT defines the name and beginning of a subprogram. The address field contains the subprogram name. It may include from one to eight characters. The subprogram name is printed as the title on the output listing unless a TITLE listing control pseudo instruction intervenes. The location field should be blank; it is ignored by COMPASS. IDENT must be the first statement of a subprogram. Instructions following the IDENT statement are assembled using the subprogram address counter.

END

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	END	m	

END terminates a subprogram. The symbol (m) in the address field is used as the symbolic transfer address. This is the starting address for execution. The symbolic transfer address, if used, must also be defined as an entry point in the subprogram. The location field may be used to define the last location of the subprogram.

ENTRY

Location	Operation	Address	Comments
1 8	10	20	41
	ENTRY	m_1, m_2,.....	.,m_n

ENTRY defines one or more locations which may be referenced by another subprogram if the same location is defined as external in the other subprogram. The address field contains one or more location names separated by commas; it may not contain blanks. The location names must be defined in the subprogram; otherwise, an error occurs.

EXT

Location	Operation	Address	Comments
1 8	10	20	41
	EXT	m_1,m_2,.....	, m_n

EXT declares the symbols in the address field to be external to the subprogram. The address field contains one or more names separated by commas; it may not contain blanks. The symbol may not be defined within the subprogram which declares it as external.

SCOM

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	SCOM	m	

The SCOM statement defines storage in common. The location field defines the name of the (scratch) common block. The address field may specify the block length, or if the address field is blank, the assembler determines the block length. An attempt to allocate more memory than declared as the block length is an error. The location name and address field are used to establish subprogram linkage. The block is used by the subprogram in common with all other subprograms declaring, by location field symbol, the same block.

A subprogram may contain up to 30 common blocks (scratch and data). A program, consisting of many subprograms, may contain as many common blocks as available memory allows. The programmer may label, reserve, or otherwise organize scratch common, but information may not be assembled in scratch common.

Example:

```
BLOCK1      SCOM
ARRAY       BSS       10
BUFFER      BSS,C     100
BLOCK2      SCOM
SCRATCH     BSS       5
TEMP        BSS       1
TEMP2       EQU       TEMP
```

DCOM

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	DCOM	m	

The DCOM statement defines data common. The location field defines the name of the common block. The address field may specify the block length or, if the address field is blank, the assembler determines the block length. An attempt to allocate more memory than declared as the block length is an error. Data common differs from scratch common in that data may be assembled in the data block.

Example:

```
DATA1     DCOM    3
LETTERS   GEN     $41,$2400,$43
DATA2     DCOM    11
DIGITS    GEN     0,1,2,3,4,5,6,7,8,9,$F
```

COMPOOL

Location	Operation	Address	Comments
1 8	10	20	41
	COMPOOL	m	

The COMPOOL statement causes COMPASS to search the library file for the named common source code string. The address field specifies the name of the common structure to be extracted from the library file (refer to Figure D-1). A COMPOOL entry on the library must conform to the rules of the SCOM and DCOM statements. COMPOOL is intended to aid the programmer in managing common blocks between a number of subprograms. An alternate library (other than the system library) file may be specified on the COMPASS control card (refer to B-1). Appendix D contains a description of the library file.

PROGRAM CONTROL

This group of pseudo instructions defines the beginning and end of an assembly of one or more subprograms. This group also includes pseudo instructions which control the conditional assembly of source statements.

ORG

Location	Operation	Address	Comments
1 8	10	20	41
	ORG	m	

ORG defines the subprogram to be absolute. The ORG pseudo must precede any instruction or data generation statements. The address field (m) term may contain a nonrelocatable symbol, value, or *. If the address field is not blank, the address counter is set to the value specified. The subprogram address counter may be reset to the program address (not common) it had prior to the last ORG occurrence by entering an asterisk (*) in the address field of the ORG statement. Either an ORG or an ORGR statement, but not both, may be used an unrestricted number of times in a subprogram.

Example:

```
                                    IDENT     ROUTINE7
                                    ORG       $100
            0100                    LD,4      WORK
                .
                .
                .
            01FF                    ST,1      BUFF
Subprogram  0200                    ST,2      BUFF+1
Address     0201                    ST,3      BUFF+2
Counter                             ORG       $400

            0400    P1              GEN       3.14
            0401    DTORAD          GEN       0.0175
                                    ORG       *

            0202                    LDC,3     ='X'
                .
                .
```

ORGR

Location	Operation	Address	Comments
1 8	10	20	41
	ORGR	m	

The address field (m) term must contain a relocatable symbol or an asterisk. If the address field is not *, the address counter is set to the value specified. The subprogram address counter may be reset to the program address (not common) it had before the last ORGR statement by entering an asterisk (*) in the address field of the ORGR statement. Either an ORG or an ORGR statement, but not both, may be used an unrestricted number of times in a subprogram.

CONDITIONALS

The source of subprogram assembly may be conditional as stated by the pseudo instructions listed below. COMPASS tests for the condition and includes or omits subsequent lines of code, depending on the outcome of the test.

IF	Zero
IFZ	Zero
IFG	Greater than or equal to zero
IFGE	Greater than or equal to zero
IFGT	Greater than zero
IFL	Less than or equal to zero
IFLE	Less than or equal to zero
IFLT	Less than zero
IFN	Not equal to zero

IF

Location	Operation	Address	Comments
1 8	10	20	41
	IF	m_1, m_2	

If the value of m_1 is zero, begin assembly with the next statement. If the value of m_1 is not zero, input cards are skipped depending on the value of m_2.

m_1 An expression, the value of which is computed as any address expression and evaluated module $2^{16}-1$ with sign extended.

m_2 If the term is a symbol, cards are processed according to the conditional until an ENDIF card with the same symbol in the location field is encountered. If m_2 is blank, cards are processed according to the conditional until an ENDIF card with a blank location field is encountered.

The range of the conditional will be terminated by a FINIS or END statement, and if the conditional started within a macro definition, its range cannot extend beyond the macro definition.

IFZ

Location	Operation	Address	Comments
1 8	10	20	41
	IFZ	m_1, m_2	

IFZ is the same as IF.

IFG

Location	Operation	Address	Comments
1 8	10	20	41
	IFG	m_1, m_2	

IFG is the same as IF, except the test is for the value of m_1 greater than or equal to zero.

IFGE

Location	Operation	Address	Comments
1 8	10	20	41
	IFGE	m_1, m_2	

IFGE is the same as IFG.

IFGT

Location	Operation	Address	Comments
1 8	10	20	41
	IFGT	m_1, m_2	

IFGT is the same as IF, except the test is for the value of m_1 greater than zero.

IFL

Location	Operation	Address	Comments
1 8	10	20	41
	IFL	m_1, m_2	

IFL is the same as IF, except the test is for the value of m_1 less than or equal to zero.

IFLE

Location	Operation	Address	Comments
1 8	10	20	41
	IFLE	m_1, m_2	

IFLE is the same as IFL.

IFLT

Location	Operation	Address	Comments
1 8	10	20	41
	IFLT	m_1, m_2	

IFLT is the same as IF, except the test is for the value of m_1 less than zero.

IFN

Location	Operation	Address	Comments
1 8	10	20	41
	IFN	m_1, m_2	

IFN is the same as IF, except the test is for a value of m_1 not equal to zero.

ENDIF

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	ENDIF		

The ENDIF statement is used with the IF statement. An ENDIF ends the skipping of source statements initiated by an IF statement. If the symbol in the location field of an ENDIF statement does not match the symbol in the address field of an IF statement, the ENDIF statement is ignored and the skipping of input cards continues.

FINIS

Location	Operation	Address	Comments
1 8	10	20	41
	FINIS		

FINIS is the final instruction of an assembly run and indicates that all subprograms have been submitted for assembly. Location and address fields are ignored. FINIS should immediately follow an END pseudo instruction.

COMPASS recognizes FINIS at any point. The occurrence causes the assembler to return to the operating system.

Example:

```
IDENT          TEST
  .
  .
  .
END
IDENT          TEST 1
  .
  .
  .
END
FINIS
```

SYMBOL DEFINITIONS

A symbol in the location field may be defined by equating it to the value of another symbol, a constant, an asterisk, or an expression of the address field. It may be defined as an absolute value or a relocatable address (word, half word, byte or bit). If a symbol is declared an entry point, it must not be equated to a symbol declared as external. When symbols are equated, they are identical and interchangeable.

All symbols in the address field must have been previously defined by appearance in the location field of a preceding instruction or pseudo instruction.

EQU

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	EQU	m	

The location symbol is equated to the symbol, value, or expression in the address field. The symbol in the location field takes on the attributes of a relocatable symbol in the address field (word, half word, byte or bit). If the location field does not contain a symbol, an error occurs.

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

- An integer modulo $2^{16}-1$.
- A symbol defined by appearance in the location field of a preceding instruction. If the symbol in the address field is relocatable to a given area (program, SCOM, DCOM), the symbol in the location field is also relocatable to that area.
- An address expression containing symbols defined as above and conforming to the rules for address subfields.

EQU,H

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	EQU, H	h	

The symbol is equated to a 17-bit address, 17-bit constant, or another symbol. If the location field does not contain a symbol, an error occurs.

The address field determines the definition of the symbol in the location field, as for the EQU pseudo. However, the resultant value must be consistent with modulo $2^{17}-1$ arithmetic and half-word address subfields.

EQU,C

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	EQU, C	c	

The symbol is equated to an 18-bit address, 18-bit constant, or another symbol. If the location field does not contain a symbol, an error occurs.

The address field determines the definition of the symbol in the location field, as for the EQU pseudo. However, the resultant value must be consistent with modulo $2^{18}-1$ arithmetic and byte address subfields.

EQU,B

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	EQU, B	b	

The symbol is equated to a 21-bit address, 21-bit constant, or another symbol. If the location field does not contain a symbol, an error occurs.

The address field determines the definition of the symbol in the location field, as for the EQU pseudo. However, the resultant value must be consistent with modulo $2^{21}-1$ arithmetic and bit address subfields.

EQU,W

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	EQU,W	m	

The EQU,W pseudo instruction differs from the EQU pseudo in that it forces the symbol in the location field to be defined as a word address symbol.

The address field determines the definition of the symbol in the location field and consists of a 16-bit constant, symbol or expression consistent with word address subfields.

SET

Location	Operation	Address	Comments
1 8	10	20	41
Symbol	SET	m	

SET functions in almost the same manner as an EQU pseudo instruction. The only difference is that SET may be used to repeatedly redefine a location field symbol. If a location field symbol is redefined with the EQU instruction, an error occurs. The same modifiers apply to SET as to EQU with the results being identical.

STORAGE DEFINITIONS

Storage definition statements are used to reserve storage for specific symbols. This group of statements usually has a location field symbol which allows other source statements to reference the defined block of reserved storage.

BSS

Location	Operation, Modifier	Address	Comments
1 8	10	20	41
Symbol or blank	BSS	m	

BSS reserves and labels a block of words in any area. The location field may be blank or contain a symbol defined as the first word in the reserved block. The address field specifies the number of words to be reserved and must contain a constant, a symbol, or an address expression which results in a nonrelocatable value. The double asterisk in the address field is illegal; symbols in the address field must be defined in the location field of a preceding instruction.

If the address field is in error or is zero, no storage is reserved but a symbol in the location field is defined. If the preceding instruction assigned memory in other than word mode, a BSS forces the next instruction, which consumes space to a new word.

Examples:

```
ALPHA    TEXTC    3,ABC     ALPHA      | A | B | C |
         BSS      0
         TEXTC    3,GHI     ALPHA+1    | G | H | I |

ALPHA    TEXTC    3,ABC     ALPHA      | A | B | C | G |
         TEXTC    3,GHI     ALPHA+1    | H | I |
```

BSS,H

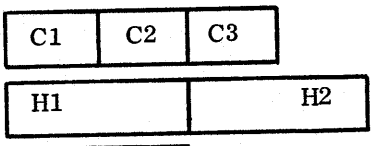
Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	BSS, H	m	

BSS, H reserves and labels a block of half-word positions. The location field may be blank or contain a symbol which is defined as a 17-bit address of the first half word in the block to be reserved. The address field specifies the number of half words to be reserved. It must contain a constant, a symbol, or an address expression which will result in a nonrelocatable value.

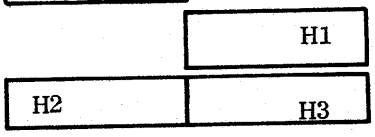
A zero address field does not reserve space, but the location field symbol is defined. BSS, H will assign the first half-word position at a half-word boundry consistent with preceding memory usage.

Examples:

```
ALPHA   BSS, C   3        ALPHA
BETA    BSS, H   3        BETA
```




```
ALPHA   BSS, C   2        ALPHA
BETA    BSS, H   3        BETA
```



BSS,C

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	BSS, C	m	

BSS, C reserves and labels a block of byte or character positions. The location field may be blank or contain a symbol which is defined as an 18-bit address of the first byte in the block to be reserved. The address field specifies the number of bytes to be reserved. It must contain a constant, a symbol, or an address expression which will result in a non-relocatable value.

A zero address field does not reserve space, but the location field symbol is defined. BSS, C will assign the first byte position at a byte boundary and consistent with preceding memory usage, as shown for BSS, H.

BSS,B

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	BSS, B	m	

BSS, B reserves and labels a block of bit positions. The location field may be blank or contain a symbol which is defined as a 21-bit address of the first bit in the block to be reserved. The address field specifies the number of bits to be reserved. It must contain a constant, a symbol, or an address expression which will result in a nonrelocatable value.

A zero address field does not reserve space, but the location field symbol is defined. BSS, B will assign the first bit position following the last entity boundry assigned by the preceding instruction (word, half word, byte, or bit).

BSS,D

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	BSS, D	m	

BSS, D reserves and labels a block of double-word positions. The location field may be blank or contain a symbol which is defined as a 16-bit address of the first word pair in the block to be reserved. The address field specifies the number of double words to be reserved. It must contain a constant, a symbol or an address expression which will result in a nonrelocatable value.

A zero address field does not reserve space, but the location field symbol is defined. BSS, D will assign the first position of the block at a word boundry.

DATA DEFINITIONS

Data may be stated as hexadecimal, decimal, or ASCII character strings in the source language. They may occupy fixed or variable positions in memory, consistent with the various address modes of COMPASS.

GEN

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	GEN	$m_1, m_2, \ldots\ldots$	$., m_n$
	GEN, H		
	GEN, C		
	GEN, B		
	GEN, D		

The GEN pseudo instruction causes numeric constants to be assembled in storage. The data is organized as single entities of the size specified by the modifier (GEN defines word data items, GEN, H half words, GEN, C bytes, GEN, B bits, and GEN, D double word data items). The location field may be blank or contain a symbol which specifies the address of the first constant in the address field.

The address field may express as many constants as can be written through column 80; they are separated by commas. Terms cannot be combined into expressions with the use of operators. Negative constants are specified by a minus (-) sign; a plus (+) sign is optional. Constants are right justified in storage if the definition does not fill the data field. A constant may not exceed the size of the specified data field (word, half word, byte, bit, or double-word).

Examples of address field constants are:

10	decimal integer
-2	decimal integer
+38	decimal integer
5P1	decimal integer, decimal scale factor
5S-1	decimal integer, binary scale factor
5P1S-1	decimal integer, both scale factors
1.5	floating point
1.5P3	floating point, decimal scale factor
$FFF	hexadecimal constant
-$A	hexadecimal constant
'ABC'	ASCII characters

VFD

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	VFD	$n/m_1,n/m_2,\ldots$	, n/m_n

Variable field definition (VFD) enters data into variable length fields assigned as continuous strings of specified length. Entries in the address field define the list of data stored. Each entry consists of a decimal value (n), a slash, and a term (m) which may be either a symbol or a constant. The decimal value specifies the length in bits of the term. Symbols evaluated as addresses are assembled right-justified in the address field of a word. The address field represents the lower 16 bits (full word), lower 17 bits (half word), lower 18 bits (character), or lower 21 bits (bit). The decimal bit count determines which type of address is assembled. The data is inserted without regard to full word, half word, character, or bit boundaries. The symbol in the location field (optional) is assigned the address of the first n/m definition in the address field.

Examples:

```
DATA7    VFD     16/$EF00, 8/44, 8/'Z'
         VFD     16/0, 16/DATA7. . . . word address
         VFD     11/0, 21/WORK . . . . bit address
         VFD     32/Symbol ..... not legal
         VFD     16/A, 16/B ..... not legal if symbol A is
                                  relocatable
```

TEXT

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	TEXT	$n, C_1\ C_2 \cdots\cdots C_n$	

TEXT causes characters to be assembled into words. A character count (n) is specified in the address field, followed by the characters to be assembled in the text form. The characters are stored as 8-bit ASCII codes in byte format. The character count in the address field is expressed in decimal. If n is greater than the number of characters that can be expressed on the card through column 80, the additional characters are stored as blanks. If a symbol is entered in the location field, the symbol is assigned the address of the first full word of text data. If n is not specified, the assembler counts the number of characters specified and reserves (through the last non-blank character in or before column 80) enough words to contain the characters; then the characters are stored in successive character positions. Unused character positions in the last word are filled with blanks.

TEXTC, TEXTH

Location	Operation	Address	Comments
1 8	10	20	41
Symbol or blank	TEXTC	$n, C_1 C_2 \cdots\cdots, C_n$	
Symbol or blank	TEXTH	$n, C_1 C_2 \cdots\cdots, C_n$	

TEXTC is similar to the TEXT statement. The TEXTC statement causes the symbol appearing in the location field to reference the first character address of the block of text data stored in memory.

TEXTH is similar to the TEXT statement. The TEXTH statement causes the symbol appearing in the location field to reference the first half-word address of the block of text data stored in memory.

LISTING CONTROL

TITLE

Location	Operation	Address	Comments
1 8	10	20	41
	TITLE	Heading	

TITLE causes a heading to be printed at the top of each page of an assembly listing. The heading is described in the address field of the statement. The pseudo instruction TITLE does not appear on the output listing. Up to 56 characters may be included in the heading.

EJECT

Location	Operation	Address	Comments
1 8	10	20	41
	EJECT		

EJECT causes the next line of the assembly listing to appear at the top of a new page.

SPACE

Location	Operation	Address	Comments
1 8	10	20	41
	SPACE	m	

SPACE causes the number of lines (m) specified in the address field to be skipped on the printed output listing.

LISTF

Location	Operation	Address	Comments
1 8	10	20	41
	LISTF	m	

LISTF is used to inhibit listing large quantities of data associated with certain statements like GEN. The address field can contain either ON or OFF. If a LISTF statement is used with OFF specified, the pseudo instruction operation code (GEN, for example) is printed but multiple lines of associated data are inhibited. Normal listing of statements is restored with another LISTF statement, which specifies ON in the address field. Normally this switch is ON.

LIST

Location	Operation	Address	Comments
1 8	10	20	41
	LIST	m	

LIST is used to suppress the listing of all or parts of the source program. When the address field specifies OFF, statements are inhibited until another LIST statement occurs with an address field specifying ON. Normally this switch is ON.

LISTCC

Location	Operation	Address	Comments
1 8	10	20	41
	LISTCC	m	

LISTCC causes COMPASS listing control cards (SPACE, EJECT, LISTF, LIST, etc.) to be printed; these are normally not listed. When the address field specifies ON, control statements are printed until another LISTCC statement appears with OFF in the address field. Normally this switch is OFF.

LISTIF

Location	Operation	Address	Comments
1 8	10	20	41
	LISTIF	m	

LISTIF allows listing control over cards to be skipped within an IF range. If m is an alphabetic OFF, listing of skipped cards will be suppressed. If m is an alphabetic ON, skipped cards will be listed. Normally this switch is OFF.

LISTMC

Location	Operation	Address	Comments
1 8	10	20	41
	LISTMC	m	

LISTMC allows listing control over the listing of macro calls. If m is an alphabetic OFF, listing of macro call lines is suppressed. If m is an alphabetic ON, macro call lines are listed. Normally this switch is ON.

BOX

Location	Operation	Address	Comments
1 8	10	20	41
	BOX		

BOX provides a convenient way to offset a collection of comments from machine instructions. When the BOX pseudo is encountered, its position on the assembly listing is replaced by a string of alternate column asterisks in the card image portion of the COMPASS output. This results in the BOX pseudo being treated as a comment card. COMPASS then inserts asterisks in columns 1 and 80 of all source statements following the BOX pseudo until and EBOX pseudo instruction is encountered. This action converts the source statements into comments.

EBOX

Location	Operation	Address	Comments
1 8	10	20	41
	EBOX		

EBOX terminates a BOX pseudo instruction. The EBOX is replaced on the assembly listing by a string of alternate column asterisks.

MACROS 4

A macro is a sequence of instructions that may be assembled (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 sequence of instructions being assembled at that point in the program. The use of a macro requires two steps, defining the sequence of instructions and calling the macro. The macro definition is to precede the first call to the macro in the subprogram.

All macro definitions are composed of the following:

- Macro headings: Names the macro and declares parameters, if any, used in the prototype.
- Prototype: Contains the sequence of instructions with variable elements expressed as parameters.
- Macro terminator: Defines the end of the macro definition.

MACRO HEADING

Location		Operation	Address	Comments
1	8	10	20	41
		MACRO		

The MACRO pseudo instruction marks the beginning of a macro definition. It consists of the MACRO statement followed by a dummy parameter line. The dummy parameter line specifies the name of the macro and the parameters associated with the macro. The name of the macro is specified in the operation field of the dummy parameter line. The name must be at least one character in length and is terminated by the occurrence of a comma or blank.

The parameters may be specified in the location field, operation field, or address field. The parameters must begin with an alphabetic character and consist of alphanumeric characters. If more than one parameter is specified in one field, the parameters must be separated by commas.

Examples of macro headings:

	Location	Operation	Address	
MACRO HEADING	BETA	MACRO TABLE, A	DATA	A macro named TABLE has parameters in the prototype called BETA, A, and DATA which are in the location, modifier, and address field, respectively.
PROTOTYPE	BETA	EQU GEN, A GEN, A	DATA DATA DATA	
TERMINATOR		ENDM		
		MACRO ABORT		A macro named ABORT has no parameters within the prototype.
		RTJ	ABORT	
		ENDM		
		MACRO TIMES	A, B	A macro named TIMES has parameters in the prototype called A and B which are in the address field.
		LD MP	1, A 1, B	
		ENDM		

PROTOTYPE

A set of instructions called the prototype follows the macro heading. These instructions may be machine instructions or pseudo instructions. They include use of the parameters specified in the macro heading.

Reference may be made within the prototype to symbols external to the subprogram if they are declared by EXT pseudo instructions within either the macro or the subprogram. An EXT declaration within the macro remains in force for the entire subprogram.

Within a macro, references can be made to symbols defined elsewhere within the subprogram.

MACRO TERMINATOR

Location	Operation	Address	Comments
1 8	10	20	41
NAME or blank	ENDM		

ENDM terminates a macro definition. The name of the macro may be specified in the location field but is optional except for macros within macros.

MACRO CALLS

Location	Operation	Address	Comments
1 8	10	20	41
	NAME	$P_1, P_2, \ldots, P_n$	

The macro call names the macro to be inserted at that point in the subprogram and assigns a set of actual parameters to be substituted for the dummy parameters in the prototype. The actual parameters must appear in the same order as the parameter list in the macro heading.

The operation field may contain any macro name defined for the subprogram by the MACRO pseudo instruction, if the macro is defined for the subprogram. COMPASS assembles and inserts the macro code at the point at which the macro name appears in the operation field.

The address field of the macro call contains the list of actual parameters, separated by commas. Single parameters may also be enclosed in parentheses within the list. Single parameters may not include blanks or commas unless the entire parameter is enclosed in parentheses.

Example A shows how the macro call may specify symbolic name in the location field as well as parameters in the operation and address fields.

Example A:

	Location	Operation	Address
MACRO DEFINITION		MACRO MNAME LD,2 AD,2 ST,2 P5 ENDM	 P3,P4,P5 P3 P4 P4
MACRO CALL	MROUT3	MNAME	BUF,WORK,(UJP A3)
ASSEMBLED STATEMENTS	MROUT3	. . . EQU LD,2 AD,2 ST,2 UJP . . .	 * BUF WORK WORK A3

When a macro call initiates the assembly of a macro definition, actual parameters are matched with dummy parameters by position. Character strings are extracted from the call statement and assigned to dummy parameter positions until each parameter is defined. The insertion of parameters within a field is terminated when a blank is encountered where a parameter separator should be. Dummy parameters that appear in the location field and are not specified in the macro call are defined with "created" symbols to avoid doubly defined symbols. Created symbols have the form SN, where

S = special character

N = four-digit decimal sequence number

Example B shows the result of created symbol insertion during assembly. When the operation field or operand field is terminated before all parameters in the prototype are defined, zeros are substituted for the undefined parameters.

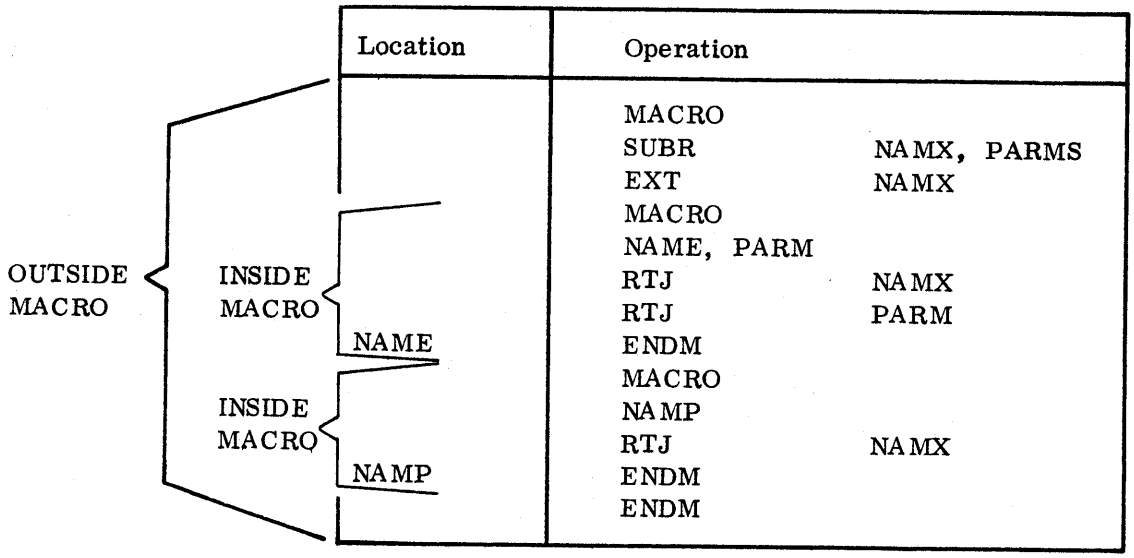
Example B:

	Location	Operation	Address
MACRO DEFINITION	LPARM	MACRO MACIO	 EXIT, FIRST, SECOND
		LD,1 AD,1 ST,1 UJP	FIRST SECOND LPARM EXIT
	LPARM	BSS ENDM	10
MACRO CALL 1		MACIO	ABORT, SUM1, SUM2
ASSEMBLED INSTRUCTIONS		LD,1 AD,1 ST,1 UJP	SUM1 SUM2 <0001 ABORT
	<0001	BSS	10
MACRO CALL 2		MACIO	ABORT, SUM1 SUM2
ASSEMBLED INSTRUCTIONS		LD,1 AD,1 ST,1 UJP	SUM1 SUM2 <0002 ABORT
	<0002	BSS	10

NESTING OF MACROS

The nesting of macros is a technique which allows the programmer to define a macro within a macro. The inside macro must have its name specified in the location field of its ENDM statement. The name of the outside macro is not required with its associated ENDM statement. Any number of macros may be defined within an outside macro.

Example:



LIBRARY MACROS

Location	Operation	Address	Comments
1 8	10	20	41
	LIBM	m_1, m_2	m_n

LIBM instructs COMPASS to obtain the named library macros from the system library or as specified by the M = parameter on the COMPASS control card. The symbols in the address field name the library macros.

MACHINE LANGUAGE INSTRUCTIONS A

All the instructions of the MP-60 computer may be coded in the COMPASS language using mnemonic codes and symbolic programming techniques. This appendix describes how machine language instructions are expressed in COMPASS, how COMPASS assembles them, and how they appear in the object program.

INSTRUCTION SUBFIELDS

Instruction fields may be optional or mandatory. An optional field may be expressed or not, as the programmer requires; a mandatory field must be present and must contain only specific parameters.

ADDRESS SUBFIELDS

m, h, c, b, n and y	The operand portion of a machine instruction may be represented by a symbol, literal, constant, external symbol, expression, or the special characters * and **. The m, h, c, and b nomenclature represents operand addresses of type word, half word, byte or character, and bit, respectively; n and y represent an immediate operand.
r and s	Machine language instructions requiring byte buffer addressing contain r and s subfields. The fields may be represented by a symbol, literal, constant, external symbol, expression, or the special characters * and **. The r subfield represents the buffer first word address; the s subfield represents the buffer last address plus one.
X	The X subfield may be represented by a constant 1 through 31, symbols equated to the value 1 through 31, or an expression with nonrelocatable value 1 through 31. The X subfield designates an index register usage of the register file. The value must be consistent with the address type of the instruction. The specification of an index is optional unless specified otherwise.
A, B, and C	Register operation class instructions require one, two or three register designators. The subfields A, B, and C may be represented by constants, equated symbols, or nonrelocatable expressions in the range 0 through 31.

BIT	The bit subfield may be represented by a constant, equated symbol, or a nonrelocatable expression in the range 0 through 31.
D	The relative displacement subfield of the test instructions may be represented by a signed constant, symbol, expression, or the special character*.

INSTRUCTION LIST FORMAT

The machine language instruction list table format is as follows:

Mnemonic	Code	Address field	Operation
AD, F	14	m, x	F = (F) + (M)
Instruction mnemonic code in COMPASS and required modifier	6-bit hexadecimal operation code or 6-bit hexadecimal operation code and 5-bit suboperation code; expressed as n.m. The code is not in packed machine representation address subfields.	The address mode of the instruction, for the operand, is indicated by the symbol used.	Representation of the instruction operation.

SYMBOL DEFINITIONS

The following designators are used throughout the instruction list:

A	=	file register A, specifying one of 32 source operands
b	=	unmodified bit address
B	=	effective bit address, after indexing; file register B, specifying one of 32 source operands

BIT	=	bit number 0 through 31
BR	=	bit register
c	=	unmodified byte or character address
C	=	effective byte or character address, after indexing; file register C, specifying one of 32 operand destinations
D	=	relative displacement
F	=	one of 32 file registers; specifying the operand register
h	=	unmodified half-word address
H	=	effective half-word address, after indexing
IMR	=	interrupt mask register
k	=	unmodified shift count
K	=	effective shift count, after indexing
m	=	unmodified word address
M	=	effective word address, after indexing
n	=	I/O controller number, page register number, console CRT select code
P	=	program address register
r	=	buffer first byte address
s	=	buffer last byte address plus one
S	=	machine state
y	=	unmodified immediate operand
Y	=	effective immediate operand, after indexing
X	=	index designation; specifying one of 32 file registers

INDEXING AND ADDRESS MODIFICATION

M = m + (X); X = registers 0 through 31	16-bit result
H = h + (X); X = registers 0 through 15	17-bit result
C = c + (X); X = registers 0 through 7	18-bit result
B = b + (X); X = registers 0 through 31	21-bit result
K = k + (X); X = registers 0 through 31	6-bit result
Y = y + (X); X = registers 0 through 31	16-, 17-, 18-, or 21-bit result

D = m - ((P) + 1); D can be plus or minus

INSTRUCTION LIST

Mnemonic	Code	Address Field	Operation
AABL	37.05	A,B,C	(C) = (A) * 32 + (B)
AABR	37.06	A,B,C	(B) = (A) .AND. $1F_{16}$ (C) = (A) / 32
AAL	31	A,B,C	(B) = (A) * 2 (C) = (A) * 4
AD,F	14	m,X	F = (F) + (M)
ADD,F	15	m,X	F, F + 1 = (F, F + 1) + (M, M + 1)
ADI,F	13	y,X	F = (F) + Y; y sign extended
AND	26	b,X	BR = (BR) .AND. (B)
BJPF	2F.0B	m,X	P = M if (BR) = 0; else P = (P) + 1
BJPT	2F.0A	m,X	P = M if (BR) = 1; else P = (P) + 1
BSK	2E.10	A,BIT,m	P = (P) + D if BIT of A = 1; else P = (P) + 1

Mnemonic	Code	Address Field	Operation
BSK,C	2E.12	A,BIT,m	P = (P) + D if BIT of A = 1, clear BIT; else P = (P) + 1, clear BIT
BSK,S	2E.11	A,BIT,m	P = (P) + D if BIT of A = 1, set BIT; else P = (P) + 1, set BIT
BSK,T	2E.13	A,BIT,m	P = (P) + D if BIT of A = 1, toggle BIT; else P = (P) + 1, toggle BIT
BSK,Z	2E.14	A,BIT,m	P = (P) + D if BIT of A = 0; else P = (P) + 1
BSK,ZC	2E.16	A,BIT,m	P = (P) + D if BIT of A = 0, clear BIT; else P = (P) + 1, clear BIT
BSK,ZS	2E.15	A,BIT,m	P = (P) + D if BIT of A = 0, set BIT; else P = (P) + 1, set BIT
BSK,ZT	2E.17	A,BIT,m	P = (P) + D if BIT of A = 0, toggle BIT; else P = (P) + 1, toggle BIT
CBIT,F	37.02	bit,X	bit + (X) = number of bits to clear in the F register
CBR	30.0C		BR = 0
CONT,F	3F.03		CXPA = (NXPA)
DINT	3F.0F		Disable interrupts
DLD,F	32	m,X	F = (M), M = FFFFFFFF

Mnemonic	Code	Address Field	Operation
DST,F	3F.13		Deadstart emulator n; n indicated by bits 29-31 of F
DV,F	20	m,X	F - (F, F + 1)/(M), F + 1 = remainder
DVS,F	36	m,X	F = (F)/(M), no remainder
DXJP	2F.1E	m,X	P = P + 1 if X ≠ 0. Then P = M, X = X - 1 P = P + 2 if X = 0
EINT	3F.05		Enable interrupts; one more instruction executed before recognition
EXM,F	3F.03		P = (F_{16-31}), BR = (F_{11}), S = (F_{12-14}), IMR_{0-7} = (F_{0-7}); the interrupt system is enabled and internal faults are cleared
F,AB	31.09	A,C	C = abs((A))(floating point)
F,ABD	31.0A	A,C	C,C+1 = abs((C,C+1)) floating point
F,ABI	31.0B	A,C	(C) = abs((A))
F,CS	31.01	A,C	C = cos(A)
F,DTS	31.08	A,C	C = (A, A + 1); double floating point ⟶ single floating point
F,F	31.03	A,C	C = (A); integer to floating format
F,SN	31.00	A,C	C = sin (A)

Mnemonic	Code	Address Field	Operation
F,SQ	31.02	A,C	C = $\sqrt{(A)}$
F,STD	31.07	A,C	C, C + 1 = (A); single floating point ⟶ double floating point
F,UF	31.04	A,C	C = (A); floating point to integer
FAD,F	16	m,X	F = (F) + (M), floating point
FADD,F	17	m,X	F, F + 1 = (F, F + 1) + (M, M + 1), floating point
FD,F	31.05	A,C	C, C + 1 = (A, A + 1); integer to floating point
FD,UF	31.06	A,C	C, C + 1 = (A, A + 1); floating point to integer
FDV,F	21	m,X	F = (F)/(M), floating point
FDVD,F	22	m,X	F, F + 1 = (F, F + 1)/(M, M + 1), floating point
FILL	37.12	A,B,C	C = (A), C + 1 = (A)...C + (B) -1 = (A) character mode if bit zero of C not set, word mode if zero of C set
FMP,F	1E	m,X	F = (F) * (M), floating point
FMPD,F	1F	m,X	F, F + 1 = (F, F + 1) * (M, M + 1), floating point
FSB,F	1A	m,X	F = (F) - (M), floating point
FSBD,F	1B	m,X	F, F + 1 = (F, F + 1) - (M, M + 1), floating point

Mnemonic	Code	Address Field	Operation
FSK,EQ	2F.02	y,X	P = (P) + 2 if (X) = y; else P = (P) + 1
FSK,GE	2F.00	y,X	P = (P) + 2 if (X) $\geq$ y; else P = (P) + 1
FSK,GT	2F.04	y,X	P = (P) + 2 if (X) > y; else P = (P) + 1
FSK,LE	2F.01	y,X	P = (P) + 2 if (X) $\leq$ y; else P = (P) + 1
FSK,LT	2F.05	y,X	P = (P) + 2 if (X) < y; else P = (P) + 1
FSK,NE	2F.03	y,X	P = (P) + 2 if (X) $\neq$ y; else P = (P) + 1
HLT	2F.0C	m,X	Halt Program execution; on restart P = M
IN	3A.00	A,C	Perform input operation from I/O TTY card with address register A and data to register C. On internal reject, P = (P) + 1; external reject, P = (P) + 2; normal return P = (P) + 3
JSX	2F.09	m,X	X = (P) + 1, P = m, X must be specified
LCPN,F	2F.1F		F = CPU number
LD,F	09	m,X	F = (M)
LDA,F	08	m,X	F = M, zero extended

Mnemonic	Code	Address Field	Operation
LDB	01	b,X	BR = (B)
LDBA,F	07	b	F = b, zero extended
LDC,F	02	c,X	F_{24-31} = (C), zero extended
LDCA,F	06	c,X	F = C, zero extended
LDD,F	0A	m,X	F, F + 1 = (M, M + 1)
LDF	37.03	A,B,C	Load Register A with the number of bits specified in Register C beginning with the bit address Register B
LDH,F	03	h,X	F_{16-31} = (h), zero extended
LDHA,F	05	h,X	F = H, zero extended
LDI,F	04	y,X	F = Y; y sign extended
LDM,F	2F.11	n,D	F = (P) + 1 + D, F + 1 = (P) + 2 + D, ..., F + n = (P) + n + 1 + D
LDP,F	33	m,X	F = (M); M is relocated by contents of relocation register
LMM	3F.15	A,C	Micro address C = (A)
LOPR	3F.12	A,C	Operand registers, state (A_{20-22}) = (C, C + 1, ..., C + 30)
LOR,F	2A	y	F = (F) .OR. y; y is 21 bits zero extended

Mnemonic	Code	Address Field	Operation
LP,F	29	y	F = (F) .AND. y; y is 21 bits zero extended
LPIR	3F.14	A,C	Page index register for state (A_{20-22}), PIR - 0 = (C_{0-15}); PIR - 1 = (C_{16-31}); ....; PIR - 15 = (C + 7_{16-31})
LXPA,F	3F.04		F = CXPA
LXR,F	2B	y	F = (F) .XOR. y; y is 21 bits zero extended
MON,F	2F,0E	m	(F) through (F + 3) to state zero registers 28 through 31; m_{16-31}= (P), m_{12-14}=(S), m_{11}= BR, M_{0-7}= (IMR_{0-7}), P = m + 1; the interrupt system is disabled
MOVA	37.14	A,B,C	Move and align data (A) = word address - from (B) = bit offset in first from word (B + 1) = number of words to transfer (B + 2) = number of bits in last transfer (C) = word address - to
MOVC	37.11	A,B,C	C = A, C + 1 = A + 1 ... C + (B) - 1 = A + (B) - 1; byte transfer
MOVE	37.10	A,B,C	C = A, C + 1 = A + 1 ... C + (B) - 1 = A + (B) - 1; word transfer
MOVN	37.17	A,B,C	Move and reformat 6-bit data bytes into 8-bit data bytes (A) = byte address - from (B) = number of 6-bit bytes to unpack (C) = byte address - to

Mnemonic	Code	Address Field	Operation
MOVP	37.16	A,B,C	Move and reformat 8-bit data bytes into 6-bit data bytes (A) = byte address - from (B) = number of 8-bit bytes to pack (C) = byte address - to
MOVT	37.13	A,B,C	Move and transliterate (A) = byte address - from (B) = number of 8-bit bytes to move (B + 1) = byte address - transliteration table (C) = byte address - to
MOVU	37.15	A,B,C	Move and unalign data (A) = word address - from (B) = bit offset in first "to" word (B + 1) = number of words to transfer (B + 2) = number of bits in last transfer (C) = word address - to
MP,F	ID	m,X	F, F + 1 = (F) * (M)
MPI,F	IC	y,X	F = (X) * y; 16-bit result
MPS,F	35	m,X	F = (F) * (M); 32-bit result
NBR	30.0A		BR = (BR)
NIO	3A.02	A,C	Set/sample data channel with address/control register A and data register C
NOP,F	00	m,X	No operation
OR	27	b,X	BR = (BR) .OR. (B)
OST	3F.1D	A,B,C	(A) = new entry address (B) = list offset and flag (C) = pointer to top of list

Mnemonic	Code	Address Field	Operation
OSU	1E.A	A,C	(A) = address of list entry to be removed (C) = list offset (16-bit, right justified)
OUT	3A.01	A,C	Perform output operation on I/O TTY card with address register A and data register C; P = (P) + 1 if internal reject/timeout; P = (P) + 2 if external reject; P = (P) + 3 if normal return
PAUS,F	3F.19		Pause for (F) microseconds
PTHD	37.1D	A,B,C	Add new entry to threaded list (A) = address of new entry (B) = placement flag and offset (C) = address of list pointer
R,AND	30.04	A,B,C	C = (A) .AND. (B)
R,NOT	30.08	A,B,C	C = $(\overline{B})$; B = $(\overline{A})$, one's complement
R,OR	30.05	A,B,C	C = (A) .OR. (B)
R,S*	30.0D	A,B,C	C = (A) * (B); 32-bit result
R,S/	30.0E	A,B,C	C = (A) / (B); 32-bit result
R,SCL	30.07	A,B,C	C = (A) .AND. $(\overline{B})$, one's complement
R,XFR	30.09	A,B,C	C = (B); B = (A)
R,XOR	30.04	A,B,C	C = (A) .XOR. (B)

Mnemonic	Code	Address Field	Operation
R,+	30.00	A,B,C	C = (A) + (B)
R,-	30.01	A,B,C	C = (A) - (B)
R,*	30.02	A,B,C	C, C + 1 = (A) * (B)
R,/	30.03	A,B,C	C = (A, A + 1) / (B); C + 1 = remainder
RAD, F	25	m,X	M = (F) + (M); C is unmodified
RD,XFR	30.1E	A,B,C	C, C + 1 = (B, B + 1); B, B + 1 = (A, A = 1)
RD,+	30.1C	A,B,C	C, C + 1 = (A, A + 1) + (B, B + 1)
RD,-	30.1D	A,B,C	C, C + 1 = (A, A + 1) - (B, B + 1)
RET	3F.12	m,X	Restore data stored by SAV
RF,+	30.14	A,B,C	C = (A) + (B), floating point
RF,-	30.15	A,B,C	C = (A) - (B), floating point
RF,*	30.16	A,B,C	C = (A) * (B), floating point
RF,/	30.17	A,B,C	C = (A) / (B), floating point
RFD,+	30.18	A,B,C	C, C + 1 = (A, A + 1) + (B, B + 1), floating point
RFD,-	30.19	A,B,C	C, C + 1 = (A, A + 1) - (B, B + 1), floating point

Mnemonic	Code	Address Field	Operation
RFD,*	30.1A	A,B,C	C, C + 1 = (A, A + 1) * (B, B + 1), floating point
RFD,/	30.1B	A,B,C	C, C + 1 = (A, A + 1) / (B, B + 1), floating point
RIM,F	3A.0D		F = (IMR)
RMIO	3A.05	A,C	C, C + 1 = (ADT register (A))
RMS	30.0F	C	C = time of day clock
RPF	3A.07	A,C	C = (page index file address (A))
RPG.F	3F.07	n	F = (page index register n)
RRM.F	3A.0E		F = (RTIM)
RSR	3A.0A	A,C	C = read from DMA channel number A
RTJ	2F.08	m,X	M = (P + 1); P = M + 1
SAV	3F.11	m,X	Store contents of file register and page index register for the state indicated by bits 20-22 of X
SB,F	18	m,X	F = (F) = (M)
SBD,F	19	m,X	F, F + 1 = (F, F + 1) - (M, M + 1)
SBR	30.0B		BR = 1
SBIT,F	37.01	bit,X	bit + (X) = number of bits to be set in the F register

Mnemonic	Code	Address Field	Operation
SCIM,F	3F.09		IMR = (IMR) .AND. ($\overline{F}$)
SCPN,F	3F.1A		CPU number = (F_{17-19})
SCRM,F	3A.0C		RTIM = (RTIM) .AND. ($\overline{F}$)
SF,F	23	k,X	Shift (F) by shift count K; +K specifies left end around; -K specifies right sign extended
SFD,F	24	k,X	Shift (F, F + 1) by shift count K; +K specifies left end around; -K specifies right sign extended
SIT,F	3F.0D		Transfer (F) to real-time clock interval timeout register
SMIO	3A.04	A,C	ADT register (A) = (C, C + 1)
SOPR	3F.11	A,C	C, C + 1, ..., C + 30 = operand registers, state (A_{20-22})
SPF	3A.08	A,C	Page index file address = (A); PIR -0 = ((C_{0-15})); PIR -1 = ((C_{16-31})); .... PIR -15 = (($C + 7_{16-31}$))
SPG,F	3F.06	n	Page index register n = (F)
SPS	3A.03	A,C	Sample position and status after ADT end of operation interrupt
SRTC,F	3F.0C		Transfer (F) to real-time clock register
SSIM,F	3F.08		IMR = (IMR) .OR. (F)

Mnemonic	Code	Address Field	Operation
SSRM,F	3A.0B		RTIM = (RTIM) .OR. (F)
SST,F	3F.10		Set relocation register to state defined by bits 20-22 of F
ST,F	11	m,X	M = (F)
STB	0B	b,X	B = (BR)
STBA,F	10	m,X	$M_{11-31} = (F)_{11-31}$
STC,F	0C	c,X	$C = (F)_{24-31}$
STCA,F0F	m,X		$M_{14-31} = (F)_{14-31}$
STD,F	12	m,X	M, M + 1 = (F, F + 1)
STF	37.04	A,B,C	(A) = Store field (B) = to bit address (C) = number of bits to store from (A), right justified
STH,F	0D	h,X	$H = (F)_{16-31}$
STHA,F	0E	m,X	$M_{15-31} = (F)_{15-31}$
STM,F	2F.10	n,D	(P) + 1 + D = (F), (P) + 2 + D = (F + 1), ..., (P) + n + 1 + D = (F + n)
STP,F	34	m,X	M = (F); M is relocated by contents of relocation register
TBIT,F	37.00	bit,X	Bit + (X) = number of bits to toggle in the F register

Mnemonic	Code	Address Field	Operation
TRC,F	3F.0E		Transfer contents of real-time clock register to F
TSK,S	3F.04	m	P = m, state = S; the interrupt system is enabled and internal faults are cleared
TST,EQ	2C.02	A,B,m	P = (P) + D if (A) = (B); else P = (P) + 1
TST,GE	2C.00	A,B,m	P = (P) + D if (A) $\geq$ (B); else P = (P) + 1
TST,GT	2C.04	A,B,m	P = (P) + D if (A) > (B); else P = (P) + 1
TST,LE	2C.01	A,B,m	P = (P) + D if (A) $\leq$ (B); else P = (P) + 1
TST,LT	2C.05	A,B,m	P = (P) + D if (A) < (B); else P = (P) + 1
TST,NE	2C.03	A,B,m	P = (P) + D if (A) $\neq$ (B); else P = (P) + 1
TSTD,EQ	2D.02	A,B,m	P = (P) + D if (A, A + 1) = (B, B + 1); else P = (P) + 1
TSTD,GE	2D.00	A,B,m	P = (P) + D if (A, A + 1) $\geq$ (B, B + 1); else P = (P) + 1
TSTD,GT	2D.04	A,B,m	P = (P) + D if (A, A + 1) > (B, B + 1); else P = (P) + 1
TSTD,LE	2D.01	A,B,m	P = (P) + D if (A, A + 1) $\leq$ (B, B + 1); else P = (P) + 1

Mnemonic	Code	Address Field	Operation
TSTD,LT	2D.05	A,B,m	P = (P) + D if (A, A + 1) < (B, B + 1); else P = (P) + 1
TSTD,NE	2D.03	A,B,m	P = (P) + D if (A, A + 1) ≠ (B, B + 1); else P = (P) + 1
TSTF,EQ	2C.0A	A,B,m	P = (P) + D if (A) = (B), floating format; else P = (P) + 1
TSTF,GE	2C.08	A,B,m	P = (P) + D if (A) ≥ (B), floating format; else P = (P) + 1
TSTF,GT	2C.0C	A,B,m	P = (P) + D if (A) > (B), floating format; else P = (P) + 1
TSTF,LE	2C.09	A,B,m	P = (P) + D if (A) ≤ (B), floating format; else P = (P) + 1
TSTF,LT	2C.0D	A,B,m	P = (P) + D if (A) < (B), floating format; else P = (P) + 1
TSTF,NE	2C.0B	A,B,m	P = (P) + D if (A) ≠ (B), floating format; else P = (P) + 1
TSTFD,EQ	2D.0A	A,B,m	P = (P) + D if (A, A + 1) = (B, B + 1), floating format; else P = (P) + 1
TSTFD,GE	2D.08	A,B,m	P = (P) + D if (A, A + 1) ≥ (B, B + 1), floating format; else P = (P) + 1
TSTFD,GT	2D.0C	A,B,m	P = (P) + D if (A, A + 1) > (B, B + 1), floating format; else P = (P) + 1
TSTFD,LE	2D.09	A,B,m	P = (P) + D if (A, A + 1) ≤ (B, B + 1), floating format; else P = (P) + 1

Mnemonic	Code	Address Field	Operation
TSTFD,LT	2D.0D	A,B,m	P = (P) + D if (A, A + 1) < (B, B + 1), floating format; else P = (P) + 1
TSTFD,NE	2D.0B	A,B,m	P = (P) + D if (A, A + 1) ≠ (B, B + 1), floating format; else P = (P) + 1
UJI	2F.07	m,X	P = (M)
UJP	2F.06	m,X	P = M
UTHD	37.1E	A,C	Remove an entry from a threaded list (A) = address of entry to be removed (C) = offset
WPF	3A.06	A,C	Page index file address (A) = (C)
WSR	3A.09	A,C	Write (C) to DMA state register (A)
XJP	2F.0D	m,X	P = m if (X) ≠ 0, X = (X) - 1; else P = (P) + 1
XOR	28	b,X	BR = (BR) .XOR. (B)
XSK	2F.0F	y,X	X = (X) + 1; P = (P) + 2 if (X) + 1 = y, zero extended; else P = (P) + 1

COMPASS CONTROL CARD B

The COMPASS control card causes the assembler to be loaded and executed under control of the MPX operating system. This appendix describes the parameter options available to the programmer.

CARD FORMAT

*CMP (I = u, L = u, P = u, X = u, M = u,N)

Parameter definitions:

I = u: Source input is from logical unit or file u. If = u is absent, standard input is assumed. Input must be specified.

L = u: Assembly listing is on logical unit or file u. If = u is absent, standard output is assumed. If L is absent, no listing is generated. However, the subprogram name and error lines will be output on standard output.

P = u: Binary output is on logical unit or file u. If = u is absent, standard punch is assumed. If P is absent, no binary output is generated.

X = u: Load and go binary output is on logical unit or file u. If = u is absent, standard load and go is assumed. If X is absent, no load and go is generated.

M = u: Library input is from logical unit or file u. If M is absent, the system library is assumed.

N: Suppresses the automatic register equate assignment.

COMPASS OUTPUT C

The output produced by COMPASS is an assembly listing consisting of a side-by-side list of assembly error codes, location, machine language, and source card image.

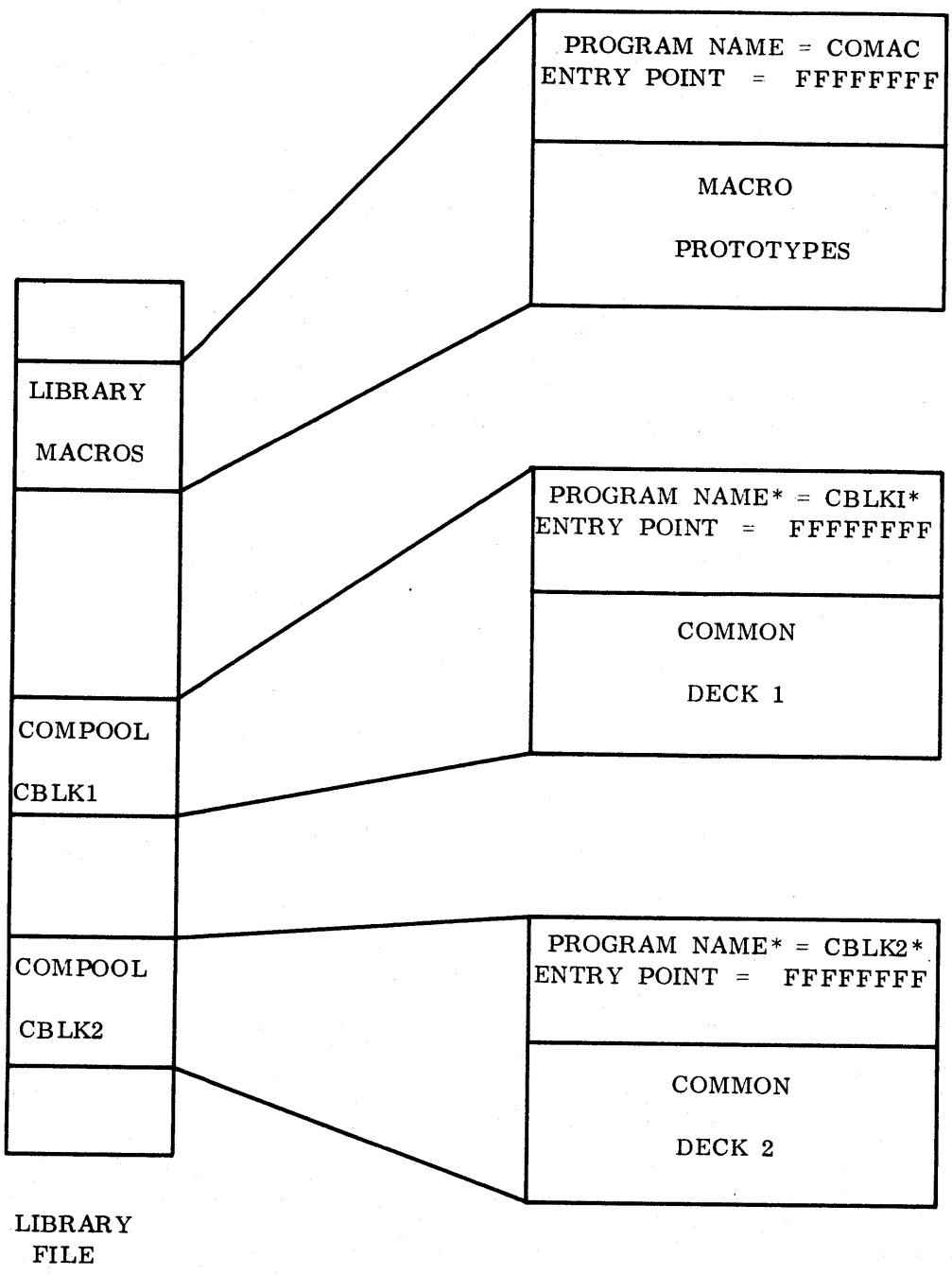
ASSEMBLY ERRORS

Error Code	Meaning
A	address field error a) too many address subfields b) subfields improperly terminated c) subfield contains illegal elements d) relocation error occur e) constant too large for data field
C	preset SCOM error
D	doubly defined symbols
F	COMPASS table full a) symbol table b) reference table c) macro table
L	location field error a) location is improperly formatted
M	modifier error a) improper or undefined modifier b) modifier where none allowed c) modifier missing
N	null symbol
O	op code error
P	program location counter error a) counter different on pass 2 than on pass 1
R	register usage
T	address truncation error
U	undefined symbol

COMPOOL/LIBRARY MACROS D

A COMPASS program may include source code which is not explicitly supplied by the programmer but is extracted from a library file as a named block of COMPASS statements. Common data structures (COMPOOL) and macro prototypes (LIBM) are examples of named source blocks. A library file (either the standard system library or an auxiliary library) containing code is generated using the PRELIB facility (Control Data publication No. 14062200). Such code blocks may have any name of eight or fewer characters, except for the reserved name COMAC for library macros. Briefly, a library file as illustrated in Figure D-1 would be generated by PRELIB using the following control statements:

```
*PRELIB
   .
   .
   .
* TEXT(ID=COMAC)
      Source code statements
*ENDT
*TEXT(ID=CBLK1)
      Source code statements
*ENDT
*TEXT(ID=CBLK2)
      Source code statements
*ENDT
*ENDPLIB
```



*Address field of COMPOOL statement

Figure D-1. Library Source Code Formats

INDEX

Item	Page
ASCII	
Literal	2-8
Constant	2-7
Text	3-18
Asterisk	
Comments	2-9
Double	2-6
ORG	3-5
ORGR	2-6
BOX	3-22
BSS	3-12
BSS, B	3-14
BSS, C	3-14
BSS, D	3-15
BSS, H	3-13
Common	
DCOM	3-4
SCOM	3-3
COMPOOL	3-4, Appendix D
Symbols	2-5
COMPOOL	3-4, Appendix D
Constants	2-6, 3-16
Control Card	Appendix B
DCOM	3-4

INDEX (CONT.)

Item	Page
EBOX	3-22
EJECT	3-19
END	3-2
ENDM	4-3, 4-6
ENDIF	3-7
ENTRY	3-2
EQU	3-9
EQU, B	3-10
EQU, C	3-10
EQU, H	3-10
EQU, W	3-11
Error Codes	Appendix C
Expressions	2-7
EXT	3-2
FINIS	3-8
GEN	3-15
IDENT	3-1
IF	3-6
Instruction Formats	Appendix A
Listing	
Control	3-1, 3-19 to 3-22
Format	Appendix C
Logical Unit	Appendix B
LIBM	4-6
LIST	3-20

INDEX (CONT.)

Item	Page
LISTCC	3-20
LISTF	3-20
LISTIF	3-21
LISTMC	3-21
Literals	2-8
Macros	4-1, 3-21, Appendix D
ORG	3-5
ORGR	3-6
SCOM	3-3
SET	3-11
SPACE	3-19
Statement Format	2-1
Symbols	2-5
TEXT	3-18
TEXTC	3-18
TEXTH	3-18
TITLE	3-19
VFD	3-17

COMMENT SHEET

TITLE: MP-60 COMPASS Reference Manual

PUBLICATION NUMBER: 14061305 REVISION: A

NAME:

COMPANY:

STREET ADDRESS:

CITY: STATE: ZIP CODE:

Control Data Corporation welcomes your evaluation of this manual. Please indicate any errors, suggested additions or deletions, or general comments below (please include page number references).

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

FOLD ON DOTTED LINES AND TAPE

TAPE

TAPE

FOLD

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 8241 MINNEAPOLIS, MINN.

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION

Systems Technology Division
215 Moffett Park Drive
Sunnyvale, California 94086

CUT ALONG LINE

FOLD

FOLD