3600

COMPUTER SYSTEMS
COMPASS

REFERENCE MANUAL

CONTROL DATA
CORPORATION

3600

COMPUTER SYSTEMS
COMPASS

REFERENCE MANUAL

CONTROL DATA
CORPORATION

60052500

REVISION RECORD

REVISION	NOTES
(11-67)	Original printing.
C	
(12-67)	Reprint with revision.

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

Pub. No. 60052500
September 1966

Address comments concerning this manual to:

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

or use Comment Sheet in the back of this manual.

CONTENTS

INTRODUCTION

COMPASS is the comprehensive assembly system for the Control Data® 3600 computer. Operating under control of the SCOPE supervisory system, COMPASS facilitates the writing of assembly language programs by providing a complete set of machine language mnemonics.

COMPASS LANGUAGE

The COMPASS language provides the following features for assembly language programmers:

Address arithmetic	Constants, symbolic addresses, literals, and arithmetic expressions may be used to represent the value of the address field.
Preloaded data	Data areas may be specified and loaded with values at the same time the program is loaded.
Common assignments	Common areas may be designated to provide for communication among COMPASS subprograms and subprograms written in other source languages.
Data definitions	Integer, floating-point, BCD, and typewriter constants may be programmed using the familiar notation.
Listing control	The format of the assembly listing may be controlled with COMPASS pseudo instructions.
Diagnostics	Diagnostics for source program errors are included in the output listing.
Variable Field Definitions	Information can be catenated in core storage without regard to word length.

MACRO INSTRUCTIONS

COMPASS provides three types of macro instructions:

<u>Programmer-defined macros</u> allow the programmer to specify a sequence of instructions as a macro definition; whenever the associated macro name appears subsequently, the sequence of instructions will be inserted by the COMPASS compiler.

Library macros supply sequences of instructions and can be referenced by programmers without the necessity of writing their own macros. The COMPASS macro library is flexible and can be easily expanded and modified to include installation oriented and specialized sets of instructions.

System macros provide a full set of instructions to perform various input/output operations, tape handling routines, interrupt controls, and library requests.

COSY

The COMPASS assembly option, COSY, may be selected to reproduce the source program compressed as much as 19:1, compared to the normal BCD source deck. The compressed source program makes faster reassembly and modification possible and simplifies maintenance of source programs on magnetic tape.

ASSEMBLER

The COMPASS assembly program converts programs written in COMPASS source language into binary machine language for execution under control of the SCOPE supervisory system. Source programs may consist of punched cards or BCD card images on magnetic tape. The assembly output provides an assembly listing and a relocatable binary object program on punched cards or on magnetic tape, and an optional compressed source language deck of COSY card images.

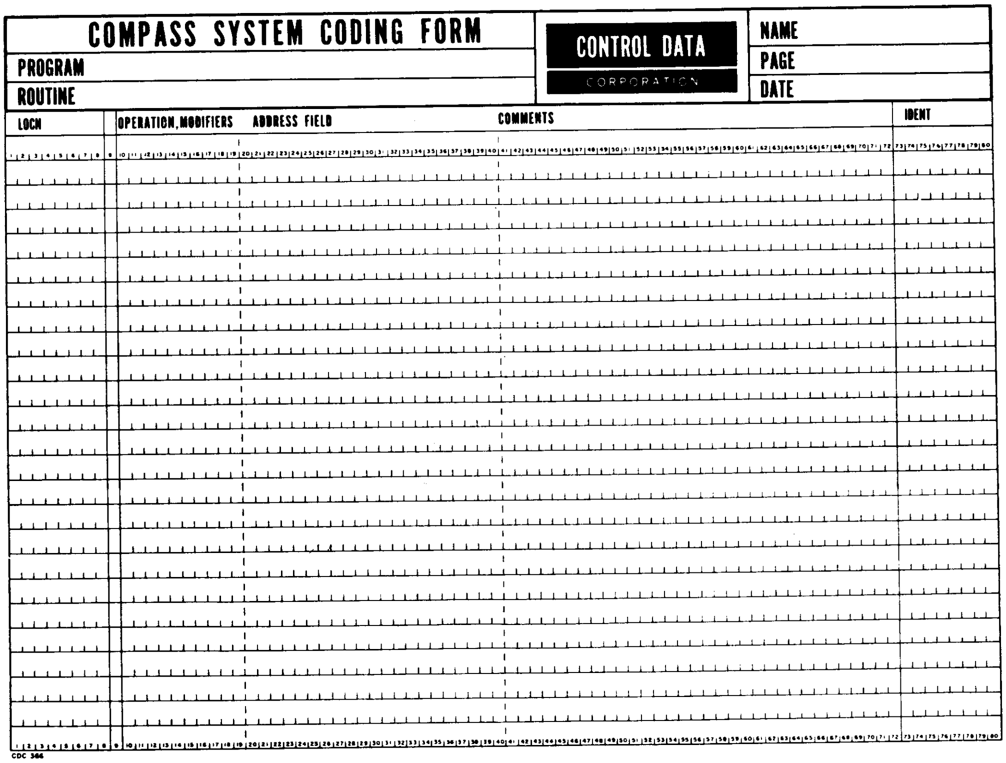
EQUIPMENT CONFIGURATION

The COMPASS assembler uses the SCOPE standard input and output units. A load-and-go unit makes it possible to execute a program immediately after assembly. Need for additional units depends on the demands of the COMPASS programs.

ASSEMBLY LANGUAGE CODING 1

1.1 CODING FORM

In the coding form for COMPASS 3600, the 80 columns correspond to the columns of the source card image. Each card image is represented by one line of coding. The form is further divided into five fields; location, operation, address, comments and identification.



COMPASS SYSTEM CODING FORM

CONTROL DATA CORPORATION

NAME

PAGE

DATE

PROGRAM

ROUTINE

LOCN | OPERATION, MODIFIERS | ADDRESS FIELD | COMMENTS | IDENT

CDC 366

LOCATION

The location field occupies columns 1-8. It may be blank or contain one of the following symbol types:

> An alphanumeric symbol of 1 to 8 characters composed of letters, numbers or special characters. The symbol may not begin with a numeric character. Allowable special characters are the internal BCD codes 14_8, 32_8, 52_8, and 33_8.

A plus or minus sign placed anywhere in the field, with the remaining columns blank.

An 8-character symbol consisting of any combination of numbers and blanks.

An alphanumeric location symbol labels a word for reference by instructions elsewhere in the program. Imbedded blanks are ignored; A B is equivalent to AB.

A plus or minus location symbol forces the instruction to be assembled into an upper or lower half-word, respectively. If a half-word is skipped by this action, it is filled with a no-operation (NOP) instruction.

A numeric location symbol identifies a numbered common block.

An asterisk in column 1 signifies that the entire card is a comment card. Columns 1 through 72 may contain any legal BCD characters. Such a line has no effect on the assembly, but columns 1-72 will be printed on the output listing.

OPERATION

The operation field begins in column 10 and ends with the first blank column. This field consists of one or more subfields separated by commas. The first subfield may contain:

One of the machine instruction mnemonics listed in Table 4

One of the pseudo instructions listed in Table 5

The name of a macro instruction

An octal number in the range, 0-77

Succeeding subfields contain operation modifiers from the set in Table 2. A blank in column 10 terminates the operation field and specifies an operation code of zero.

A three-letter machine mnemonic implies a 24-bit instruction; a four-letter machine mnemonic implies a 48-bit instruction. However, operation modifiers normally cause an instruction to be augmented to 48 bits.

ADDRESS

The address field may begin in any column following the operation field's blank terminator up to, and including, column 40. It is terminated by the first blank, but may not extend beyond column 72. The address field may have one or more subfields, separated by commas or parentheses. Machine instructions have implied subfields. If the address field of a machine instruction is blank, each of the implicit subfields assumes a zero value. A subfield may be skipped and assigned a zero value by giving only its trailing comma or; if it is the last subfield in the address field, by omitting both its value and the preceding comma.

For machine instructions, the address field may contain the following subfields:

Subfield	Contents
a	first bank designator †
i	second bank designator
m	first operand address
n	second operand address
y	operand
b	first index register
v	second index register †
e	equipment designator
g	bit designator
p	first source register
q	second source register
r	destination register
u	unit designator
w	operand
x	channel number

ADDRESS ELEMENTS

Address subfields are expressed in terms of one or more address elements. Address subfields may contain an alphanumeric symbol, a constant, an asterisk, a double asterisk, a dollar sign, a literal, data storage, a dollar sign plus a name, an expression, or mnemonics for 3600 operational registers.

ALPHANUMERIC SYMBOL

This symbol consists of 1 to 8 characters composed of letters, numbers or the special characters; the first character may not be a digit.

an apostrophe (internal BCD 14_8)

a plus zero, +0, (internal BCD 32_8)

a minus zero, -0, (internal BCD 52_8)

a period (internal BCD 33_8)

† a and v are not considered as implicit subfields in 24-bit instructions.

CONSTANT A constant may be a decimal or octal integer; an octal integer is suffixed by the letter B. The size of a constant depends upon the size of the subfield or the kind of subfield in which it is placed.

ASTERISK For m, n or y subfields, the character, *, is interpreted as the current value of the location counter, For a and i subfields, it is a bank relocatable element, and implies the bank into which the subprogram will be loaded

DOUBLE ASTERISK The symbol, **, gives a one value to each bit of the subfield. This element normally indicates a subfield to be set during program execution.

DOLLAR SIGN Subfields a or i may contain the character, $, alone or followed by an alphanumeric symbol. If $ appears alone, it implies the bank associated with the symbol in the operand address subfield. If a symbol follows $ in the same subfield, it implies the bank associated with that symbol. In either case, if the symbol to which $ applies is non-relocatable, no entry will be made in the Bank Relocation Table.

DATA STORAGE Data storage, signified by an equal sign followed by an S, establishes a storage area at the end of the program following the literals and substitutes its starting address for the address in the instruction. The starting location of the storage area is assigned the name which follows the S. The name is an alphanumeric symbol, and may be used in any expression except those which require that the symbol be previously defined. The length of the storage area is defined by a dimension string, enclosed in parentheses, following the name. Each dimension may consist of an expression containing constants and previously defined symbols. If no dimensions are given, one word is assigned.

Example: LDA=SX(5)

X	word 1
X+1	word 2
X+2	word 3
X+3	word 4
X+4	word 5

If the same name appears in more than one data storage element, the same location will be assigned for each occurrence, but the length will be that of the largest array. The coding should not depend on the order of assignment of the data storage areas.

If the name is defined elsewhere by an element other than data storage, no space will be reserved and the address substituted will be that of the otherwise defined symbol.

LITERAL A literal is signified by an equal sign followed by a mode designator and value. The value is converted according to the specified mode and stored as one or two words at the end of the program. The assigned address is substituted in the address of the instruction. When literals of the same value occur, storage is not duplicated.

The mode designator may be one of the following.

D	Decimal constant	The value is written in the format specified by the DEC pseudo instruction and is terminated by the first blank character, or by a comma if another subfield follows.
O	Octal constant	The value is written in the format specified by the OCT pseudo instruction and is terminated by the first blank character, or by a comma if another subfield follows.
H	Hollerith codes	The first eight characters following H specify the BCD value; the ninth character must be a comma or blank.
T	Typewriter codes	The first eight typewriter characters following T specify the value. A typewriter character is represented by one or two BCD characters, as in the TYPE pseudo-instruction. A blank or a comma must follow the eighth typewriter character.
DD	Double Precision Decimal Constant	The value must be in floating point format as specified by the DECD pseudo instruction. The result occupies two words.
DO	Double Precision Octal Constant	The value is written in the format specified by the OCT pseudo instruction. The result occupies two words.

Examples:

	Coded	Assembled Value
Single Precision	=D2222	0000000000004256
	=O01234567	0000000001234567
	=HABCDABCD	2122232421222324
Double Precision	=DO12345670123456701234	0000000000001234 5670123456701234

EXPRESSION

An expression consists of alphanumeric symbols, constants, and asterisks joined by the operators: + addition, - subtraction, * multiplication, and / division. The position of an asterisk defines its meaning. For example, in the expression, **2, the inner * denotes multiplication; the outer * is a self-reference symbol.

An expression is evaluated from left to right; Multiplication and division are performed before addition and subtraction. Parentheses may not be used for grouping: 15*A+5/2*C-5 is evaluated as $(15.A) + ((\frac{5}{2}).C)-5$.

A remainder generated by a divide operation will be lost. Thus 5/2*2=4. If a divide by zero is attempted, an address error will be flagged unless the dividend is also zero.

An expression may be scanned for legality using the following rules:

1. For each relocatable element, substitute R_i, where i = p for program relocatable, i = c1 for relocation with respect to the first common block, i = c2 for relocation with respect to the second common block, etc.

2. Algebraically, sum all terms within the expression. The only permissible results are zero, a non-relocatable value, or $\pm R_i$.

   $R_p + R_{c1} - R_p - R_{c1}$ = non-relocatable

   $2*R_p - R_p$ = program relocatable

   $-2*R_p + R_p$ = negative program relocatable

3. A single term may not contain more than one relocatable symbol.

4. Within a single term, a relocatable symbol may not be an operand in a divide operation, and no slashes (/) may occur to the right of a relocatable symbol.

5. An external symbol may not be part of a multiply or divide operation. No more than one external symbol may appear in an expression. The end result must be of the form:

   ± < external > ± < non-relocatable value >

6. Every relocatable element must be defined elsewhere.

OPERATION REGISTER MNEMONICS

These special mnemonic codes, listed in Table 1, may be used in certain instructions to represent registers pertinent to the instruction.

COMMENTS

The comments field may begin after the blank column which terminates the address field and may extend through column 72. If the instruction has no address field, comments may begin following the operation field; if the instruction has an address field which is null, comments may begin in column 41. The comments field has no effect on the assembly, but its contents are printed on the output listing.

IDENTIFICATION

Columns 73 through 80 may be used for identification; they have no effect on assembly. If a COSY option is requested, this identification will be replaced on the output listing by COSY sequence numbers.

1.2 BANK RELOCATABILTY

Subprograms and common blocks may be assigned to any memory bank or combination of memory banks. If the programmer does not designate the bank assignments, the SCOPE loader assigns each subprogram or block to the memory bank into which it best fits. For bank relocatable fields, COMPASS assembles a zero in the object deck and makes an entry in the Bank Relocation Table to enable the SCOPE loader to make the proper bank assignments.

ASTERISK

The symbol, *, in an a or i subfield is interpreted as the bank into which the subprogram in which it appears will be loaded.

DOLLAR SIGN

The symbol, $, appearing alone in an a or i subfield, is interpreted as the bank associated with the symbol appearing in the operand address subfield. If an alphanumeric symbol follows the $, in the same subfield, the $ implies the bank associated with that symbol.

If the symbol associated with $ is non-relocatable, no entry will be made in the Bank Relocation Table.

BANK RELOCATION TABLE

COMPASS produces, as part of its relocatable binary output, a Bank Relocation Table for the SCOPE loader. (See SCOPE Reference Manual Pub. No. 60053300 for full description of BRT). The BRT cards direct the loader to insert the proper bank references in instructions containing relocatable bank terms; a or i subfields which contain fixed values do not require BRT entries. A BRT entry would be made, for example, for each of the following:

```
LDA      ($)NAME1
ENO      $NAME1
STA      (*)NAME1
BRTJ     NAME1,,$
```

Two entries would be made for each of the following:

```
          XMIT          ($)NAME1,($NAME2)NAME3
          BRTJ          ($)NAME1,,$
          UBJP          (*)NAME1,,*
```

No BRT entry would be made for the following since the operand address subfield is not relocatable or external:

```
          BRTJ          ($)**,,$
          LDA           ($)0
NAME1     EQU           1
          XMIT          ($NAME1)NAME2, ($)**
```

If a subprogram will be referencing locations outside itself, including common blocks, and if these locations may reside in a different bank, the programmer must be certain that the bank registers are set and reset accordingly.

NOTE: For instructions BEGR, BEGW, XMIT, IOSR, IOTR, IOSW, and IOTW, if no a (and/or i) subfield is specified by the programmer, $ will be assumed, and a BRT entry will be produced for each bank subfield associated with a relocatable or external symbol. In the following example, NAME2 and NAME3 are relocatable or external:

```
XMIT    NAME2, NAME3           (2 BRT entries)
BEGR    3,NAME2,NAME3          (1 entry)
IOTW    NAME2,20               (1 entry)
XMIT    **,**                  (no entry)
```

1.3 INSTRUCTION PAIRING

The instruction set for the 3600 computer contains both 24-bit (half-word) instructions and 48-bit (full-word) instructions. The assembler organizes instructions in memory subject to the following considerations.

NORMAL PAIRING

Normally, two 24-bit instructions are stored in one computer word. Starting with the first two machine instructions in a subprogram, pairs of instructions are assigned to consecutive locations. The first instruction is assigned to the upper half, the second instruction to the lower half of the word. This sequence is maintained until forcing upper or forcing lower occurs.

FORCING UPPER

The following conditions force an instruction to begin in the upper half of the next available location. The lower half of the current location, if unused, is filled with a NOP instruction:

An alphanumeric symbol or a plus sign in the location field

A full-word instruction, 48 bits

One of the 24-bit half-word instructions; CPJ, DRJ, EQS, ISK, MEQ, MPJ, MTH, SSH, SSK, or THS, unless specifically forced lower.

One of the pseudo instructions; BCD, BES, BSS, DEC, DECD, OCT, ORGR or TYPE

An instruction immediately following a BLOCK, EJECT or REM pseudo instruction

FORCING LOWER

A minus sign in the location field of a 24-bit instruction forces the instruction to be located in the next available lower half-word. The upper half-word, if unused, is filled with a NOP instruction. A minus sign in the location field of a 48-bit instruction will not force lower, and it will be flagged as a location error on the output listing.

AUGMENTS

The single precision augment will automatically be inserted before a 24-bit instruction, making it a 48-bit instruction, if a bank designator or second index designator appears in the address field, or if an operation code modifier appears in any instruction except ROP, RXT, RSW, AJP, QJP, ARJ or QRJ.

In the case of ISK, SSK, SSH, EQS, THS, MEQ and MTH, which are normally forced upper, augmenting will force them to the lower half-word which will affect the manner of exit from the instruction. The double precision augment is automatically inserted for instructions DFAD, DFDV, DFMU, DFSB, DLDA and DSTA.

MNEMONICS

Instructions with 3-letter mnemonic codes are half-word instructions except where the single precision augment is used. Instructions with 4-letter mnemonic codes are full-word instructions.

Table 4 lists the mnemonic codes for machine instructions with the allowable modifiers. Modifiers which are mutually exclusive are listed in a vertical column. Modifiers may appear in any order and may be omitted except as noted. In the address field, the subfields must appear in the order specified. The bank designators (a) and (i) are optional.

PSEUDO INSTRUCTIONS 2

Pseudo instructions direct COMPASS to perform specific assembly functions. They are used to define assembler control, listing control, conditional assembly, program organization, data definition and macro coding. The general format for pseudo instructions is the same as that for machine instructions.

2.1 ASSEMBLER CONTROL

These pseudo instructions define assembly mode and subprogram linkage. They control the operation of the assembler but, except for CALL, do not generate code in the object program.

COMPASS includes a provision for assembling instructions coded in CODAP-1 format. The CODAP-1 mode is initiated using the CODAP pseudo instruction. An instruction with a punch in column 9 is always recognized as a CODAP-1 instruction, regardless of mode.

IDENT

1	8	10
		IDENT $m,n_1,n_2,\ldots$

IDENT must be the first instruction of each subprogram; it will be flagged as an O error if it appears again before an END instruction. The m subfield must contain an alphanumeric symbol of 1 to 8 characters, which names the subprogram. The location field should be blank. The n_i subfields are optional; they may contain address field expressions which result in values punched in the IDC card of the object deck. A typical use of the n_i subfields is the declaration of system parameters in the coding of Drum SCOPE background programs.

END

1	8	10
		END m

END signals termination of a subprogram. If an alphanumeric symbol appears in the address field, it will be punched as the symbolic transfer address on the TRA card of the relocatable object deck. The transfer address must be defined elsewhere in the program as an entry point. The location field should be blank.

ENTRY

1	8	10
		ENTRY $m_1, m_2, \ldots m_n$

ENTRY declares alphanumeric symbols within the subprogram as entry points which may be referenced by other subprograms. An entry point must be defined by its appearance in the location field elsewhere in the subprogram. Non-relocatable and program relocatable symbols may be declared as entry points. Symbols in the address field, separated by commas, may extend through column 72; the first blank column terminates the string. The location field of ENTRY should be blank.

Example:

```
                 ENTRY          SYM1,SYM2,SYM3
                   :
      SYM1       LDA            CON1
                   :
      SYM2       ENI            77777B
                   :
      SYM3       EQU            19
```

SYM1, SYM2 and SYM3 may be referenced by other subprograms.

EXT

1	8	10
		EXT $m_1, m_2, \ldots m_n$

EXT defines symbols which are external to the subprogram in which the EXT instruction occurs. They are of the same form as symbols declared by ENTRY and represent entry point names in subprograms called by this subprogram. At load time, external symbols are assigned the value corresponding to the symbol in another subprogram. Symbols, separated by commas, may extend through column 72. The first blank column terminates the string. The location field of EXT should be blank.

```
Subprogram 1                     Subprogram 2

EXT   SYM1,SYM2                            ENTRY SYM1,SYM2
 :                                           :
SLJ   SYM1                       SYM1      LDA   AA
 :                                           :
RTJ   SYM2                       SYM2      SLJ   **
```

The two symbols declared as entry points in subprogram 2 are defined as external symbols and referenced in subprogram 1.

CALL

```
1      8  10
symbol    CALL  m
```

CALL defines the symbol appearing in the address field as an external symbol, and it assembles a bank return jump to that symbol into the program, as follows:

```
BRTJ      ($)m, , $
EXT       m
```

Location field may contain a symbol identifying BRTJ instruction.

EQU

```
1      8  10
symbol    EQU  m
```

The EQU statement must appear before a reference is made to the symbol which it defines.

A location field symbol may be defined by equating it to an address field expression. An alphanumeric symbol in the location field is assigned the value of the address field expression. This value is substituted for all program references to the symbol. Address field symbols must be previously defined (used as location symbols earlier in the subprogram).

Symbols may be equated to other symbols or values within the same subprogram. Symbols may also be equated to external symbols which must appear singly in the address field; the location symbol will not appear in the external symbol list.

SET

```
1      8  10
          SET   s,n
```

SET permits the definition and subsequent re-definition of a symbol during assembly. The s subfield contains a single alphanumeric symbol. The n subfield contains an address field expression which may be relocatable or non-relocatable. Any symbols appearing in the n subfield must have been previously defined. The symbol s will retain the value n until altered by the occurrence of another SET pseudo-instruction. A symbol defined by SET may not be defined by any other type of symbol definition. A symbol defined by SET within a macro will be global. A symbol may be set equal to an external symbol according to the same rules as in EQU.

Any entry in the location field is meaningless and will be ignored.

CODAP

1	8	10
		CODAP

CODAP changes the assembly mode to accept instructions in CODAP-1 format. All subsequent instructions are so interpreted until a COMPASS psuedo instruction is encountered.

A location symbol is meaningless, and will be ignored.

COMPASS

1	8	10
		COMPASS

COMPASS reestablishes the COMPASS mode if a CODAP pseudo instruction was encountered previously; otherwise, it is ignored. An instruction with a punch in column 9 is always interpreted as a CODAP-1 instruction, regardless of mode.

A location symbol is meaningless, and will be ignored. Everything beyond column 17 is treated as comments.

SCOPE

1	8	10
		SCOPE

SCOPE terminates the assembly process and causes COMPASS to return control to the SCOPE monitor. The SCOPE pseudo instruction should follow an END pseudo instruction. If SCOPE follows any instruction other than END, it is treated as an END and flagged as an O error. Return is made to the monitor after the subprogram is assembled.

A location symbol is meaningless, and will be ignored. Everything beyond column 15 will be treated as comments.

2.2 LISTING CONTROL

The COMPASS assembly listing may be controlled with the following pseudo instructions; they are not printed on the output listing. Symbols in the location field will be ignored.

COMPASS will allow EJECT, SPACE, NOLIST, LIST and REM to be placed before or between macro definitions. They may not, however, precede LIBM, if it is present. TITLE, BRIEF and DETAIL may occur anywhere in the deck, following macro definitions, if any.

EJECT

1 8	10
	EJECT

EJECT causes the printer to eject paper to the top of the next page. The next instruction will be printed following the title line on the next page. EJECT also forces upper.

SPACE

1 8	10
	SPACE n

This pseudo instruction spaces the output listing the number of lines specified in the address field. If the spacing would cause an overflow at the bottom of the page, the page is ejected to the top ,of the next page. If there is an address field error, SPACE is ignored; no error diagnostic is given.

NOLIST

1 8	10
	NOLIST

NOLIST suppresses the printing of assembly lines until a LIST pseudo instruction is encountered. However, lines with error flags will still be listed.

LIST

1 8		10
		LIST

LIST resumes output listing and is meaningful only if a NOLIST has been encountered previously.

The list option on the COMPASS control card takes precedence over the LIST pseudo instruction. If listing is not specified on the control card, LIST has no effect.

TITLE

1 8		10
		TITLE any

TITLE may appear anywhere in a subprogram after macro definitions and before the END instruction. Columns 16-72 of the first TITLE instruction, no matter where it appears, will be printed on the top of the first page of the subprogram listing and at the top of all subsequent pages until another TITLE instruction is encountered. Second and subsequent TITLE instructions will cause a page eject.

If TITLE is not used in a subprogram, the contents of the IDENT address field will be printed as the title.

REM

1 8		10
any		REM any

REM produces a printed line containing remarks only. All columns, except 10-13, are printed as remarks. REM forces the next instruction upper.

*

1 8		10
*		

A comment line is also produced if an * appears in column 1. Such a card does not force the next instruction upper, and all columns, 1-72, are printed.

BRIEF

1 8		10
		BRIEF

BRIEF suppresses the listing of the following:

Literals

All but the first full word generated by DEC, OCT and DECD

The second half word and all subsequent words generated by VFD, TYPE or BCD

Location digits of second and subsequent array names of a COMMON pseudo instruction.

BRIEF may occur at any point in a program after macro definitions. It remains in effect until a DETAIL pseudo instruction is encountered.

DETAIL

1 8		10
		DETAIL

DETAIL causes a return to the normal listing mode to resume printing of information suppressed by BRIEF.

2.3 CONDITIONALS

Conditional pseudo instructions determine whether a specified number of lines are to be assembled. These instructions define a condition and test to see whether the condition is satisfied. If the condition is satisfied, the lines specified are assembled. If the condition is not satisfied, the bypassed code is not listed unless the M option is specified on the COMPASS control card.

An entry in the location field serves to identify the conditional if it is to be terminated by an ENDIF pseudo instruction. The symbol is not program-defined and can be used elsewhere without ambiguity.

END and IDENT may be skipped as the result of a conditional. Care should be taken that instructions which follow the skipped lines will be legally positioned in the resultant subprogram.

IFZ

```
1       8   10
symbol      IFZ  m,p
```

If the value of m is zero, the next p lines will be assembled. IFZ may appear anywhere in a subprogram. Both m and p may be address field expressions which are evaluated modulo $2^{15}-1$. Symbols must have been previously defined. The resultant value, m, is tested for zero. The expression, p, specifies a number of coding lines to be processed if the value of m is zero. These lines immediately follow the IFZ instruction. If m is not zero, the assembler will bypass the number of lines specified and continue assembling at line p+1.

If p and the preceding comma are omitted, the range of the IFZ is determined by an ENDIF pseudo instruction.

Example:

```
D        EQU      1
E        EQU      0
          .
          .
         IFZ      D,1
A        ENA      0
         IFZ      E,1
B        ENQ      0
          .
          .
```

In the above example, the values of D and E are tested for a zero condition when the IFZ instructions are encountered. Since D is assigned a non-zero value, the ENA instruction which follows the IFZ instruction is not assembled. Because the converse is true for E, the ENQ instruction will be assembled. The IFZ pseudo instruction does not result in a card image or assembled line. It merely tests a condition at assembly time.

IFN

```
1       8   10
symbol      IFN  m,p
```

IFN differs from IFZ only in that the lines are assembled if the value of m in the address field is non-zero.

IF

```
1      8  10
symbol    IF,s  m,n,p
```

IF may appear anywhere in a subprogram. The next p coding lines will be assembled if the relationship specified by the instruction modifier, s, exists between m and n. The next p lines are not assembled if the specified relationship does not exist. The m, n and p subfields may contain arithmetic expressions evaluated modulo 2^{15} -1. All symbols must be previously defined. If the p subfield and the preceding comma are omitted, the range of the IF will be determined by an ENDIF pseudo instruction. The modifier, s, may be any of the following:

mnemonic	meaning
EQ	$m = n$
NE	$m \neq n$
LT	$m < n$
LE	$m \leq n$
GT	$m > n$
GE	$m \geq n$

IFU

```
1      8  10
symbol    IFU  p
```

If the lower half-word of the preceding instruction is occupied, the next p lines are assembled; otherwise, they are not assembled. If p is omitted, the range of the IFU will be determined by an ENDIF pseudo instruction.

IFL

```
1      8  10
symbol    IFL  p
```

If the lower half-word of the preceding instruction is vacant, the next p lines are assembled; otherwise they are not assembled. If p is omitted, the range of the IFL will be determined by an ENDIF pseudo instruction

IFT/IFF

IFT and IFF are restricted to the range of an ECHO or MACRO pseudo instruction. Their use is defined in Chapter 3.

ENDIF

```
1      8  10
          ENDIF   m
```

Conditional pseudo instructions, not containing a line count, may be terminated by an ENDIF pseudo instruction which defines the limit of the conditional range. If lines are skipped because of a conditional pseudo instruction, the associated ENDIF causes normal processing to resume. The associated conditional is defined by a name in the ENDIF address field which matches the name in the location field of the conditional. An ENDIF with a blank address field is associated with the last encountered unlabeled conditional.

An ENDIF occurring in the range of a conditional with which it is not associated has no effect on that conditional, but is counted as a coding line.

Address symbols used with ENDIF are not program defined; therefore, they may be assigned without regard for any prior use. An entry in the location field will be ignored.

Example:

```
              .
              .
              .
        STA     CACHE
A       IF, EQ  CACHE, STORE
        ENI     99, 1
B       IFU
AB      RAO     AA, 1     Skipped if the "ENI 99, 1"
        IJP     AB, 1     instruction occupies upper
        ENI     49, 2     half-word.
C       IFU                                              Skipped if
        ENDIF   B                                        the symbol
        ENI     49, 2     Skipped if the "ENI 99, 1"     CACHE
AB      RAO     AA, 1     instruction occupies lower     ≠ STORE
        IJP     AB, 1     half-word.
        ENDIF   C
BC      RSO     BB, 2
        IJP     BC, 2
        ENDIF   A
        LDA     NEXT
              .
              .
              .
```

2.4 PROGRAM ORGANIZATION

These pseudo instructions are concerned with the organization of the program in memory. They establish storage areas local to the subprogram and storage areas used in common by more than one subprogram. They provide for the allocation of instructions and data to specified storage areas and make bank assignments for subprograms and common blocks.

BSS

```
1      8  10
symbol    BSS  m
```

BSS reserves a block of consecutive 48-bit machine words. The value of the expression in the address field determines the number of words to be reserved. Symbols in the address field expression must be previously assigned. If the value is zero, no space is reserved and the next instruction is forced upper before the location symbol is assigned. A location symbol is optional; if present, it is assigned to the first word reserved.

BES

```
1       8  10
symbol     BES   m
```

BES is identical to BSS, except that the location symbol, if present, is assigned to the last word reserved.

INTRODUCTION TO BLOCK AND COMMON

BLOCK and COMMON pseudo instructions reserve storage areas that can be addressed by more than one subprogram.

If the location symbol in the BLOCK pseudo instruction is alphanumeric, the storage area is termed labeled common. If the location symbol is numeric or blank, the storage area is assigned a separate portion of core storage termed numbered common. Constants and instructions that are not bank relocatable and do not contain external symbols can be assembled into a labeled common area. A numbered common area may not be preset.

COMMON pseudo instructions describe data arrays within an area assigned by BLOCK. To address a block of common storage reserved in another subprogram, the location field symbol of the BLOCK pseudo instruction must be identical in each subprogram, and the length of the second and subsequent blocks must be less than or equal to the first. The last numbered common block defined in a bank may vary in length from one subprogram to another.

BLOCK

```
1       8  10
symbol     BLOCK   m
```

BLOCK defines a block of common. The name of the block must be given in the location field by an alphanumeric symbol which identifies the block as labeled common, or a numeric symbol or blank which identifies the block as numbered common. Each block must have a unique name. If two or more blocks have the same name, a D error will be indicated on the output listing in each line where the duplicate symbol occurs.

A symbol in the BLOCK location field serves only to name the common storage area. The symbol may not be referenced elsewhere in the subprogram except by a subfield in the BANK pseudo instruction.

BLOCK forces the next subprogram instruction upper.

An expression in the address field of BLOCK specifies block size or the total length of the common block. The block size must be greater than or equal to the length expressed in the COMMON instructions subsequent to BLOCK. If the address field is blank or zero, block size is determined by the sum of the array sizes of subsequent COMMON instructions.

For operation under Drum SCOPE, a name of the form . POOLxxx in the location field has a special meaning; it is generally used to assign buffers for the Drum SCOPE system. A user may not have a common block with a name of this form. The m need not be specified with this usage of BLOCK as the system assigns a labeled common block the length of the drum block. xxx are arbitrary characters.

COMMON

1 8	10
	COMMON $A_1(i_1,j_1,k_1,\ldots n_1),\ldots A_n(i_n,j_n,k_n,\ldots n_n)$

COMMON defines the arrays to be included in the common block defined by the last encountered BLOCK. An entry in the location field will be ignored. The address field consists of one or more subfields, each of which defines an array to be included in the block. A subfield is terminated by a comma; the field is terminated by a blank. An array length of zero is legal.

The general form of an address subfield is:

A(i, j, k, ... n)

A is a symbol which names the first element of the array and i, j, k, ... n are the dimensions of the array.

The general form of the address field is:

A(i, j, k), P(l, m, n), . . .

i, j, k, l, m, and n are integers or expressions which result in non-relocatable values. Symbols must be previously defined. A 2-dimension array has two subscripts. A 1-dimension array has only one subscript. For a single element, no parenthetical term need appear.

Example:

COMMON A(15, 15), B(3, 4, 5), C, D(15)

The assembler will sum the expressed sizes of the arrays for all the common in one block. This sum will be the total number of computer words reserved for the block if the BLOCK address field value is zero. If the address field of the BLOCK pseudo instruction gives a number larger than this sum, that number will be the number of words reserved for the block. The first element of the first array in the block will occupy the first word of the reserved area.

ORGR

```
1      8 10
         ORGR   m
```

ORGR may be used at any point in a subprogram to initiate a sequence of instructions or constants at a location different from the current program location. The ORGR address field contains an expression which must result in a program or labeled common relocatable value. Symbols in the address field must be previously assigned. Subsequent instructions or data words are assembled sequentially, beginning at the location specified by that value. This sequence continues until another ORGR card or the end of the subprogram is encountered.

The number of words assembled into a labeled common block must not exceed the length of the block. Instructions assembled into labeled common may not contain external symbols or bank relocatable terms (a or i fields). The instructions XMIT, IOSW, IOTW, IOSR, IOTR, IOJP, BEGR and BEGW generate bank terms and are, therefore, excluded unless the address fields result in a nonrelocatable value.

The address expression, m, may not represent a location in numbered common. An entry in the location field will be ignored. ORGR forces the next instruction to begin in the upper half of a machine word. If the preceding instruction occupies an upper half-word, the lower half-word will be filled with a NOP.

When the main program storage assignment sequence is interrupted by an ORGR, the program location counter is saved. An ORGR with an asterisk (*) in the address field causes storage assignment to resume at that location.

Example:

```
                            IDENT     ABC
     00000            A     BLOCK     0
     00000                  COMMON    C(100B)
     00000            P1    ENA       0
     00001            P2    BSS       100B
     00101                  ENQ       0
           C00005           ORGR      C+5
     00005                  OCT       0
     00006                  LDA       P1
     00007            +     STA       P3
           P00102           ORGR      *
     00102            P3    OCT       0
     00103            P4    BSS       5
           P00004           ORGR      P2+3
```

```
00004          OCT      0
00005          OCT      1
00006          OCT      2
     P00110    ORGR     *
00110          ENQ      P3
               ENA      P4
            .
            .
            .
```

BANK

1 8		10
		BANK (a_1),name$_{11}$,name$_{12}$,...(a_2),name$_{21}$,name$_{22}$,...

BANK defines the memory bank assignment for subprograms and common blocks. The address field contains one or more bank terms, each followed by one or more names which represent entry points or common blocks. The bank term is enclosed in parentheses and designates a bank in one of three ways:

A digit in the range, 0-7

An entry point in a subprogram for which the bank is assigned at load time or by another BANK declaration

A common block name, enclosed by slashes within the parentheses, for which the bank is assigned at load time or by another BANK declaration

The names following the bank term are separated by commas; they represent common blocks, and entry point names in subprograms to be assigned to the same bank as that represented in the bank term.· Common block names are enclosed in slashes.

Entry points and common block names must be defined when the program is loaded, but they need not be defined or referenced in the subprogram containing the BANK pseudo instruction. COMPASS does not check the validity of such symbols, or proper address field formatting; the programmer must ensure that they are correct. The location field of BANK should be blank.

Example:

BANK (1), ENTRA, /BLKA/, (/BLKB/), ENTRB

The above bank declaration assigns the common block, BLKA, and the subprogram containing entry point, ENTRA, to bank 1; it assigns the subprogram containing entry point, ENTRB, to the same bank as assigned to common block, BLKB.

2.5 DATA DEFINITION

Data definition pseudo instructions cause data to be assembled into the subprogram or into a common block.

OCT

1 8	10
symbol	OCT $m_1, m_2, ... m_n$

OCT stores octal constants in consecutive machine words. Each address subfield specifies one constant of 1 to 16 octal digits, optionally preceded by a + or -. A constant of less than 16 octal digits will be right justified in the word, with the sign extended. Each constant is assigned to a separate word. A location symbol is optional; if present, it is assigned to the first word.

Example:

```
OCT +1,-57,2040,-2
```

word 1	0000000000000001
word 2	7777777777777720
word 3	0000000000002040
word 4	7777777777777775

DEC

1 8	10
symbol	DEC $d_1, d_2, ..., d_n$

DEC converts signed or unsigned decimal constants to binary and stores them in consecutive machine words. Each constant occupies a full machine word. Each subfield contains a sign (optional) and a string of 1 to 28 decimal digits. The value may be followed by a decimal scaling factor consisting of a D and 1 to 3 signed or unsigned decimal digits and/or a binary scaling factor consisting of a B and 1 to 4 signed or unsigned decimal digits. If the value contains a decimal point, it is converted to floating point form; the magnitude must fall in the range 10^{+307} to 10^{-308} decimal or $2^{\pm 1022}$ binary. If no decimal point appears, the value is stored in fixed point form; the magnitude must be less than 2^{47}. A decimal constant of the form fDdBb is equivalent to the expression $f. 10^{d}. 2^{b}$.

A location symbol is optional; if present, it is assigned to the first word.

Examples:

-38	fixed point decimal
7.3D-2	floating point decimal, decimal scaled
200B-7	fixed point decimal, binary scaled
36D1B+2	fixed point decimal, decimal and binary scaled

DECD

1 8	10
symbol	DECD $d_1,d_2,...,d_n$

DECD converts double precision floating point decimal constants to binary and stores them in consecutive pairs of machine words. The format is identical to DEC except that each constant will occupy two machine words. A location symbol is optional; if present, it is assigned to the first word.

BCD

1 8	10
symbol	BCD n,8n characters

BCD stores internal BCD characters in consecutive machine words.

The address field consists of a word count, n, followed by a comma and 8n characters, including blanks, ending before column 73. The word count may be a digit or an expression as previously defined. This results in n computer words, each containing 8 BCD characters. Anything after 8n characters is treated as remarks. If the value of n exceeds the number of characters which may be punched in a single card, blanks are filled in for the excess. If n is zero, no characters are stored, and no space is assigned.

A location symbol is optional; if present, it is assigned to the first word.

Example:

BCD 3, BLUE PLATE SPECIAL-$1.25

word one	2243642560474321
word two	6325606247252331
word three	2143405301330205

TYPE

1 8		10
symbol		TYPE n,8n characters

TYPE converts BCD characters into typewriter code and stores them in consecutive machine words. The format is the same as for the BCD pseudo instruction with the following exceptions:

> Lower case typewriter characters and functions not represented on the keypunch require a special 2-character code as shown in Table 6.
>
> Upper case typewriter characters require that the upper case mode be established using the function code, *U. The required characters are then entered if represented on the keypunch; otherwise, their lower case equivalents are used. *L must be used to return to lower case mode.

In computing the word count, n, the special 2-character codes are counted as on character. A location symbol is optional; if present, it is assigned to the first word.

Example:

TYPE 4, BLUE PLATE SPECIAL-*U$*L1. 25

word 1	0141355160544112
word 2	7551604654511116
word 3	1241046447667732
word 4	3757606060606060

The equivalent typewriter message is BLUE PLATE SPECIAL-$1. 25.

VFD

1 8		10
symbol		VFD $m_1n_1/v_1,...m_pn_p/v_p$

VFD (variable field definition) converts octal constants, Hollerith characters, typewriter characters, and arithmetic expressions and stores them as contiguous strings of bits without regard for word boundaries. The address field consists of one or more subfields separated by commas. Each subfield is in the form, mn/v, where m specifies mode, n specifies bit length and v defines the value to be stored. The address field is terminated by the first blank not a part of a Hollerith or typewriter value. If partially filled, the remainder of the half-word is padded with zeros.

A location symbol is optional; if present, it is assigned to the first word.

Five modes (m) are allowed:

On/v Octal number. If v is preceded by a minus sign, the one's complement form will be stored. The number of bits (n) may not exceed 48.

Hn/v Hollerith character code. The n term must be a multiple of 6; n/6 defines the number of characters which appear in the v term. The (n/6 + 1)th character must be a blank or a comma. If used within a MACRO or ECHO, blanks and commas in the v term act as normal delimiters when parameters are substituted.

Tn/v Typewriter character code. Same rules apply as in the Hollerith mode.

Bn/v Bank term. The n term may be omitted; it is always assumed 3. The v term, when stored, must coincide with one of the five bank designator positions of a machine word (bits 41-39, 34-32, 26-24, 17-15, 10-8). The v term may be an expression evaluated modulo 8, or a bank relocatable * (bank of the subprogram in which VFD appears) or $name (bank associated with that name). If the v term is bank relocatable, the 3-bit value generated by the assembler will be zero.

An/v Arithmetic expression or constant. The v term is an expression formed according to the rules for address arithmetic, with the following exceptions:

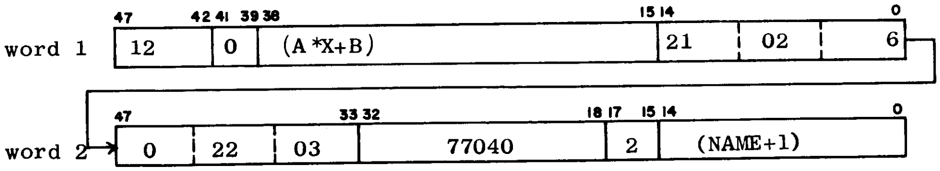
1. n must be ≤ 48.
2. A relocatable expression must be in the correct position to insure that it will be relocated by the loader. The result of a relocatable expression must fit, right justified, into bit position 24 or 0 of a machine word. A relocatable expression is evaluated modulo $2^{15}-1$. n must be ≥ 15.
3. If an expression results in a fixed value and field length, n, is not 15, it is evaluated 2^n. If field length is 15, and value is not 77777_8, modulus is $2^{15}-1$. If the field length exceeds the size required for a value, the value is right justified with the sign extended in the high order bits.

If the value of a non-relocatable A, O, or B subfield exceeds 2^n-1, it is truncated to fit the field; thus A4/20B produces zero.

Example:

```
VFD T6/A, B/*, A24/A*X+B, H30/A2 B3, O15/-737, B/2, A15/NAME+2
```

A, X and B are assumed non-relocatable symbols; NAME may be relocatable. The following two words are generated:



GENERATIVE CODING 3

Generative coding is a procedure whereby a single pseudo instruction may call for the assembly of a defined sequence of machine or pseudo instructions. Selected parameters in the instruction sequence may be specified by the calling instruction.

Pseudo instructions in this class define and call macros, or repeat coding sequences (ECHO). Other pseudo instructions described here are restricted to macro or ECHO coding. COMPASS recognizes three types of macros - system macros, library macros, and programmer macros.

System macros are always accessible to the subprogram through a macro call; they need not be declared or defined in the subprogram. System macros include calling sequences to SCOPE subroutines. They may be called at any point in the program after any LIBM declarations and programmer macro definitions.

Library macros are contained in an expandable macro library on the library tape. All library macros to be called in a subprogram must be declared in LIBM pseudo instructions immediately following the IDENT pseudo instruction.

Programmer macros are defined by the programmer for his own use. Programmer macro definitions may not precede LIBM pseudo instructions, if any, or the IDENT pseudo instruction. Programmer macro definitions must precede all other instructions except EJECT, SPACE, NOLIST, LIST and REM.

All macro definitions, regardless of type, consist of the following:

Macro heading	Names the macro and declares the formal parameters used in the prototype.
Prototype	Contains the instruction sequence with variable elements expressed as formal parameters.
Macro terminator	Defines the end of the macro definition.

The pseudo instruction which brings the prototype into the body of the program is referred to as a macro call. It consists of the macro name and a string of actual parameters to be substituted for the formal parameters in the prototype.

3.1 OPERATION SEARCH ORDER

In assigning formal parameter names and macro names, the programmer is not prohibited from using symbols which are identical to COMPASS mnemonics or names of macros of another type. When these symbols appear in an operation code field, their meaning is subject to the condition under which COMPASS encounters them. For example, if a formal parameter is named LDA, and LDA appears in an operation code field in the associated macro prototype, it is treated as any formal parameter; its meaning as a COMPASS mnemonic does not apply. Similarly, if a programmer macro is named READ, and a macro call to READ appears in the subprogram, the programmer macro is called, not the system macro of the same name.

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

1. Formal parameters
2. Programmer macro names
3. Library macro names
4. System macro names
5. COMPASS mnemonics

3.2 MACRO HEADING

```
1      8  10
name        MACRO  (formal parameters)
```

The macro heading consists of one or more lines of the pseudo instruction, MACRO. The location field contains the macro name which may be any alphanumeric symbol except IDENT, LIBM, MACRO, IFT, IFF, ENDIF, ENDM, END or SCOPE.

The address field contains a set of formal parameters. If the formal parameters extend beyond the length of a single code line, the mnemonic MACRO must be repeated in the next operation field; and the entire formal parameter list must be enclosed in parentheses. A blank character terminates the address field. If the list is enclosed in parentheses and a blank occurs before the final right parentheses, the following card should contain the MACRO pseudo instruction. If the list is not enclosed in parentheses, a blank will terminate the list. Parameters are expressed as alphanumeric symbols and are separated by commas. The parameter symbols are local to the macro and may be used elsewhere in a program without ambiguity. Individual parameters must be contained on a single line.

Examples:

```
DIVIDE      MACRO    P1,P2,P3,P4,P5
MULTIPLY    MACRO    (P1,P2,P3,P4,P5,
            MACRO    P6)
READ        MACRO    (P1,P2,P3,P4,P5,
            MACRO    P6,P7,P8,P9,P10,
            MACRO    P11)
```

As in the above examples, formal parameters should be short for efficient use of the 3600 macro facility. The formal parameters in the heading lines establish the order in which actual parameters must be declared in the macro call. The pseudo instruction, MACRO, does not produce a card image in the object deck.

3.3 PROTOTYPE

A set of instructions follows the heading line. Any operation code is acceptable in the prototype except SCOPE. It is up to the programmer to ensure that, when the macro is called, the resulting code will not contain illegalities. The prototype does not, itself, produce any code in the object deck.

Location, address, and operation fields for these instructions may be expressed as formal parameters. Any element in any field or subfield, except comments, may be a formal parameter. A location symbol in the prototype, which is not a formal parameter, is local to the macro and may be used elsewhere in the subprogram without ambiguity. COMPASS will substitute an internally generated symbol for the local location symbol and for all references to it within the macro. The location field of an asterisk comment card will not be substituted. For more efficient operation, local location symbols should be kept to a minimum.

3.4 MACRO TERMINATOR

```
1      8  10
          ENDM
```

This pseudo instruction terminates a macro definition: A location field symbol will be ignored. It does not produce a card image in the object deck.

Example:

```
ABC     MACRO    P3,P2,P1,P4       Macro heading
        ENI      P1,1
A       LDA      P2,1
        P3       B                 Prototype
        STA      P2
        IJP      A,1
        UBJP     SCRAM
B       DEC      P4
        ENDM                       Macro terminator
```

Formal parameter P1 represents an operand, P2 an address, P3 an operation code, and P4 a decimal constant. Location symbols A and B are local to the macro and may be used elsewhere without ambiguity. The formal parameters in the macro heading establish the order in which actual parameters must be specified in the macro call.

3.5 MACRO CALL

1 8	10
symbol	macro name (actual parameters)

The macro call names the macro to be inserted at this point in the program, and it assigns a set of actual parameters to be substituted for the formal parameters in the prototype. The actual parameters appear in the same order as the formal parameter list in the macro heading. The location field may contain blanks, an alphanumeric symbol, or a plus or minus symbol. A symbol in the location field will be assigned to the next available full word.

The macro name appears in the operation field and the address field contains the actual parameter string. The actual parameters are code elements or expressions separated by commas. If a single parameter contains commas, blanks or parentheses, the entire parameter must be enclosed in parentheses. If the actual parameter list extends beyond a single line, the macro name must be repeated; the location symbol should not be repeated.

The rules governing the use of parentheses are as follows:

> The entire actual parameter string must be enclosed in parentheses if it extends beyond a single line or if the first actual parameter is enclosed in parentheses.
>
> Groups of elements, separated by parentheses or commas, which form a single parameter, must be enclosed in parentheses.
>
> For each left parenthesis, there must be a matching right parenthesis.

Example:

A macro is defined as follows:

```
ABC     MACRO     P1,P2,P3
        LDA       P1
        STA       P2
        P3        P2
        BRTJ      ($)OUT
        ENDM
```

When the above macro is called by a macro call of the form

```
ABC (((*)M,1,2),(($)M2,1,2),
ABC (LDA,CM))
```

the actual parameter (*)M,1,2 is substituted for formal parameter P1
($)M2,1,2 is substituted for formal parameter P2
LDA,CM is substituted for formal parameter P3

and the following instruction sequence is produced:

```
LDA (*)M,1,2
STA ($)M2,1,2
LDA,CM ($)M2,1,2
BRTJ ($)OUT
```

The code generated by a macro call will not be listed unless the M option is specified on the COMPASS control card.

Consecutive commas in the actual parameter string define a null subfield; an explicit zero must be entered as a parameter.

Example:

Macro definition:

```
ABC           MACRO      P1,P2,P3,P4
              LDA        P1,P2
              STA        P3,P4
              ENDM
```

Macro call:

```
              ABC        ABLE,,BAKER,1
```

> In the above example, a null subfield, bounded by commas, corresponds to parameter P2. In this case, the second subfield in the first instruction is omitted; the trailing comma will appear.

If there are fewer actual parameters in a macro call than formal parameters for that macro, null subfields are assumed. Trailing commas need not appear for null subfields at the end of the list. If there are more actual parameters than formal parameters, extra actual parameters at the end of the list are ignored.

NESTING OF MACROS

A macro definition may, itself, contain macro calls to system, library, or programmer macros. These inner macro calls become effective at the time a call is made to the outer macro. A macro definition may also contain calls to itself; it is the programmer's responsibility to prevent infinite recursion through the use of conditionals.

Examples:

Macro definition:

```
     AAA         MACRO      P1,P2,P3
                 ENA        0
                 STA        P1
                 ENI        47,P3
     AA          QJP,P2     BB
     CC          QLS        1
                 IJP        AA,P3
                 SLJ        *+2
     BB          RAO        P1
                 SLJ        CC
                 ENDM
```

Macro definition:

```
     BBB         MACRO      P1,P2,P3
                 LDQ        P1
                 SSU        P2
                 RXT        A,Q
                 AAA        D,MI,P3
                 ADD        TOT
                 STA        TOT
                 ENDM
```

A macro call of the form

```
                 BBB        ONE,AA,3
```

will generate the following sequence of instructions:

```
                 LDQ        ONE
                 SSU        AA
                 RXT        A,Q
                 ENA        0
                 STA        D
                 ENI        47,3
     ↑↓000003    QJP,MI     ↑↓000001
     ↑↓000002    QLS        1
                 IJP        ↑↓000003,3
                 SLJ        *+2
```

```
    ↑↓000001          RAO        D
                      SLJ        ↑↓000002
                      ADD        TOT
                      STA        TOT
```

Macro definition:

```
    LOG               MACRO      NAME,VALUE
    A                 REM
                      IFZ        VALUE/2,1
    NAME              EQU        0
                      IFN        VALUE/2,2
                      LOG        A,VALUE/2
    NAME              EQU        A+1
                      ENDM
```

A macro call of the form

```
                      LOG        LOGNAM, 5
```

will generate the following sequence:

```
↑↓000001                                   (1)  Identifier of local symbol A.
              IFZ 5/2,1                    (2)  Condition not met.  Line 3 is skipped.
NAME          EQU 0                        (3)  Listed, but not assembled.
              IFN 5/2,2                    (4)  Condition met.  Lines 5 and 18 assembled.
              LOG ↑↓000001,5/2             (5)  Calls LOG with new parameters.
↑↓000002                                   (6)  2nd identifier of local symbol A.
              IFZ 5/2/2,1                  (7)  Condition not met.  Line 8 is skipped.
NAME          EQU 0                        (8)  Listed, but not assembled.
              IFN 5/2/2,2                  (9)  Condition met.  Lines 10 and 17 assembled.
              LOG ↑↓000002,5/2/2(10)            Calls LOG with new parameters.
↑↓000003                                   (11) 3rd identifier of local symbol A.
              IFZ 5/2/2/2, 1               (12) Condition met.  Line 13 assembled.
↑↓000002      EQU 0                        (13) Equates 2nd A identifier to zero.
              IFN 5/2/2/2,2                (14) Condition not met.  Lines 15 and 16 skipped.
              LOG A,VALUE/2                (15) Listed, but not assembled.
NAME          EQU A+1                      (16) Listed, but not assembled.
↑↓000001      EQU ↑↓000002+1               (17) Equates 1st A identifier to 2nd A
                                                identifier plus 1.
LOGNAM        EQU ↑↓000001+1               (18) Equates LOGNAM to 1st A identifier
                                                plus 1.
```

This example demonstrates a macro which produces no object code. It equates the symbol substituted for the formal parameter NAME with an integral logarithm of the number substituted for the formal parameter VALUE. The end result is represented by line 18 which is equivalent to LOGNAM EQU 2. In expanding the macro, COMPASS assigns a unique identifier for each occurrence of a local location symbol.

PARAMETER SUBSTITUTION

At macro call time, each line of the prototype is examined for substitutable elements. In a macro prototype, a single element is identified by the characters which bound it, and by the field in which it appears:

> A location field element is the group of non-blank characters contained in the location field. The location field may contain at most one element.
>
> An operation field element is a group of characters bounded by column 9, a blank, a comma, or a 0, 5, 8 punch.
>
> An address field element is a group of characters bounded by the following special characters:

blank
,
)
(
$
=
+
-
*
/
→ (punched as 0, 5, 8)

Location and address field elements are compared first with the local location symbol list and then with the formal parameter list. If the element is a local location symbol, an assembler-created symbol is substituted. If the element is a formal parameter, the corresponding actual parameter is substituted. An actual parameter which is identical to a local location symbol is not considered to be equivalent; such an actual parameter should be defined elsewhere. Operation field elements are compared only with the formal parameter list.

Special characters will appear in the generated line with the exception of → which the machine interprets as a signal to catenate. (See example).

If an actual parameter will not fit on a single card image, the action taken depends on the operation code. If the operation code is an ECHO or a macro call whose address field begins with a left parenthesis, COMPASS will generate a continuation card. Otherwise, substitution ceases when column 72 is encountered.

In the example, → in ST →P1, is interpreted as a symbol to catenate and ST → P1 becomes STA (where P1 = A). Similarly, B → P2 is catenated to B2 (P2 = 2), P3 → L becomes QL (P3 = Q), and P4 → P5 becomes LOC2 (P4 = LOC, P5 = 2).

Catenation is legal in the operation and address fields only and not in the location field.

```
                Location       Operation      Address
Example:

              ⎧ CATNATE        MACRO          P1,P2,P3,P4,P5
              ⎪                ST → P1        LOC1
Macro         ⎪                LIL            LOC1,P2
Definition:   ⎨                ROP,XOR        B → P2,MZ,P3 → L
              ⎪                ST → P3        P4 → P5
              ⎩                ENDM

                LOC1           BSS            1
                LOC2           BSS            1
                LOC3           BSS            1

Macro Call:                    CATNATE        A,2,Q,LOC,2
                             ⎧ STA      LOC1
Expansion:                   ⎨ LIL      LOC1,2
                             ⎪ ROP,XOR        B2,MZ,QL
                             ⎩ STQ      LOC2
```

3.6 SYSTEM MACROS

System macros are contained on the library tape but need not be declared with LIBM by the programmer. These macros include input/output control, tape handling, internal interrupt, and equipment status checks. They are accessible to the programmer through the macro calls listed below. These macros are described in SCOPE Reference Manual, Pub. No. 60053300 and Drum SCOPE Reference Manual, Pub. No. 60059200. The parameters indicated for these system macros are defined as follows:

Parameter	Meaning
a	starting or return adress †
ak	address key location
b	bank designator
bna	background program name address
c	disposition or assignment code
cwa	control word address †
d	density (HY, HI, LO or OP)
dr	direction of tape (RV or ND)
du	Duration in seconds or (seconds, milliseconds)
e	edition number
ea	exit address

† Associated actual parameters may have index subfields provided that the entire parameter is enclosed in parentheses.

Parameter	Meaning
ek	error key
f	format (BCD or BIN)
fn	family number
fwa	first word address of buffer area
hw	half-word
i	interrupt code (SHIFT, DIVIDE, EXOV, EXUN, OVER, ADDR, M1604, TRACE, INST, OPER or MANUAL)
ia	interrupt address †
in	index register number 1-6
k	stop key
la	label address †
lb	lower bound
ll	lower limit
lna	label name address †
M	master unit status
n	integer
os	overlay or segment
pv	priority value
r	reel number
ra	reject address †
rd	retention code or date written
rn	record or partition number
rna	record name address †
s	usage (RW, RO, WO, BY or RF)
sn	subroutine name
sq	sequence number
u	logical unit number or mnemonic †
U	unsave
ub	upper bound
uc	use code
ul	upper limit
w	word count

†Associated actual parameters may have index subfields provided that the entire parameter is enclosed in parentheses.

INPUT/OUTPUT

The following are tape and drum macros unless otherwise specified.

ADCOMP	(u, ra, ia)	a select interrupt on address compare
LOCATE	(u, cwa, ra, ia)	locate (Tape SCOPE only - parameters differ for Drum SCOPE)
LOCATEDR	(u, ra, ia)	locate (Tape SCOPE only)
MODE	(u, ra, s, f, d, dr)	declare mode on called logical unit (Tape SCOPE); declare mode on master logical unit (Drum SCOPE).
POSIT	(u, cwa, ra, ia)	position
RACT	(u, ra, ia)	read angular count
RDLABEL	(u, la, ra, ia)	read label
WRLABEL	(u, la, ra, ia)	write label
RELEASE	(u, ra, c)	release logical unit
READ	(u, cwa, ra, ia)	read
WRITE	(u, cwa, ra, ia)	write
REOT	(u, cwa, ra, ia)	read end of tape
WEOT	(u, cwa, ra, ia)	write end of tape
STATUS	(u, M)	request status. Format of reply to Status differs for Tape and Drum SCOPE.
WRCK	(u, cwa, ra, ia)	write check

TAPE CONTROL

BSPR	(u, ra, ia)	backspace one record
BSPF	(u, ra, ia)	backspace one file
UNLOAD	(u, ra, ia, c)	rewind and unload
SKIP	(u, ra, ia)	skip to end-of-file
ERASE	(u, ra, ia)	erase six inches of tape
MARKEF	(u, ra, ia)	mark end-of-file
LABEL	(u, lna, e, r, rd)	define label
SAVE	(u, U)	saves tapes at end-of-job; Tape SCOPE only; ignored by Drum SCOPE
UNSAVE	(u)	cancel previous save action

INTERNAL INTERRUPT

SELECT	(i, ia)	select interrupt	
REMOVE	(i)	remove interrupt	
BOUND	(lb, ub, ra, ia)	set program bounds	Stacked in Tape SCOPE; not stacked in Drum SCOPE.
UNBOUND		remove bounds	

CLOCK INTERRUPT	LIMIT	(du, ra, ia)	time limit specification
	FREE		free last imposed time limit
	TIME		return remaining time to Q, time of day to A
	DATE		date request to A in Tape SCOPE: to A and Julian date to Q in Drum SCOPE
OTHER SYSTEM MACROS	ATI	(in)	A to index register
	CALL	(sn)	BRTJ to designated subroutine
	CORE	(ll, ul)	set limits of available memory
	EXIT		return control to monitor
	HERESAQ		modify A and Q registers
	IAQ		register swap, A and Q
	LDCH	(a, in, in)	load byte instruction
	LIBRARY	(u, ra, rna, rn)	library requests (Tape SCOPE only)
	LOADER		loader requests
	MEMORY	(b, ll, ul)	set limits of available memory
	STCH	(a, in, in)	store byte instruction
DRUM/SCOPE ONLY	ABORT	(ek)	abort with diagnostic
	ASSIGN	(u, c)	assign hardware and logical unit[†]
	BINBCD		convert binary integers to BCD
	BUFFER	(fn, a_1, a_2)	declare family buffer
	BYNBY	(ia)	give control to interrupt address[†]
	CHECK	(u, ra)	check status
	CLBCD		convert column binary card images to BCD
	DISABLE		enter disable mode
	DISPOSE	(u, c, ra)	set unit disposition
	DYSTAT		status request but returns a dynamic status if unit is assigned
	ENABLE		enter enable mode
	ENTER	(u, ra)	enter data into system[†]
	EXIT		program complete
	FAMILY	(fn, u)	attach unit to family
	INFORM	(bna, fwa, w, ra)	send message to background program[†]
	INVOKE	(sn, ra)	invoke background subroutine[†]
	LIBRARY	(rna, ea)	library request

[†] Valid for background programs, only.

LOCATE	(u, ak, ra)	position specified drum unit
LOVER	(u, os, rn)	load partition
MEMBER	(u)	check family membership
PRISEQ	(u, pv, sq)	set priority and sequence †
READY	(u, ra, ia)	sense ready condition on unit
RETURN		return control to system
RETURNM	(hw, a, b,)	return control at address
SIWOH	(bna)	determine status of background program†
SYSIO	(uc, k, a, w)	system I/O
WAIT		return control to system temporarily †
WHERE		check last location executed
XFER	(u_1, bna, u_2, ra)	allow a background program use of same logical units as another background program †

The following names are reserved for system macro calls, and may not be used as programmer macro names.

Drum SCOPE Only

B
C
F
CALLIOC
EQUATE
FIELDS
FORBID
IOEX
IOEXW
IOG
IRUPT
LDRIV
LOAD

LOOKUP
OPTIONS
PERMIT
RESERVE
SENTRY
SHA
SHAQ
SHIFT
SHQ
STARTUP
SYSTEM
TABLER
Z

Tape SCOPE

CALL36
IOF
STAR

† Valid for background programs, only.

The following SCOPE mnemonics, used in the above system macro calls to indicate standard units, interrupts and equipment designation, may not be used in system macro calls for any other purpose. No ambiguity results, however, if the programmer defines and uses symbols of the same name elsewhere in the subprogram.

ABNORM
ACC
ADDR
BCD
BIN
BY
DIVIDE
EXOV
EXUN
FO
HY
ICM
INP
INST

LGO
LIB
LO
M1604
MANUAL
ND
OCM
OP
OPER
OUT
OVER
PUN
RO
RV

RW
SCR
SHIFT
S0
S1
S2
S3
S4
S5
S6
S7
S8
S9
TRACE
WO

3.7 LIBRARY MACROS

```
1      8  10
          LIBM  m1,m2,...mn
```

All library macros to be called in a subprogram must be declared in LIBM pseudo instructions immediately following the IDENT pseudo instruction. A library macro, once declared, is available for call as long as COMPASS remains in memory and the macro directory selected on the COMPASS control card remains the same. When several subprograms are assembled together, it is expedient to declare all the required library programs in the first subprogram. The names of the library macros to be called are entered in the address field, separated by commas. A location symbol will be ignored.

Library macros are contained in an expandable macro library on the system library tape. Macros may be added to or deleted from the library through the TAPE SCOPE PRELIB routine (SCOPE manual Pub. Number 60053300) or the Drum SCOPE LIBEDIT routine (Drum SCOPE manual Pub. Number 60059200).

3.8 ECHO

1 8	10
symbol	ECHO m,n,(p_1=a_1,a_2,...,a_n,p_2=b_1,b_2,
	ECHO ...,b_n,...,p_k=k_1,k_2,...,k_n)

ECHO causes the m instructions which follow it to be assembled n times. Parameters in the instruction sequence may be varied for each repetition but the parameter list may not contain blanks. The m and n subfields may contain expressions consisting of previously defined symbols. $p_1, p_2, \ldots, p_k$ are formal parameters; a, b,..., k are the actual parameters which are to be substituted for the formal parameters. In the first iteration, the actual parameters $a_1, b_1, \ldots, k_1$ are substituted for the formal parameters $p_1, p_2, \ldots, p_k$. In the second iteration, the actual parameters $a_2, b_2, \ldots, k_2$ are substituted for p_1, $p_2, \ldots, p_k$ and so on. If n is absent or zero, the m lines of code which follow are duplicated as many times as there are actual parameters associated with the first formal parameter. The n field must be defined by its trailing comma if n is omitted.

The expanded code generated by ECHO will not be listed unless the M option is specified on the COMPASS control card.

Any operation code is accepted in the range of an ECHO except END, ECHO and SCOPE. It is up to the programmer to ensure that code resulting from an ECHO statement will not contain illegalities. Comments within the range of ECHO will appear on the output listing only in the first repetition but the contents of cards with * in column 1 will be repeated each time.

The address field of ECHO may be terminated by a blank column following n if no parameters are required. If there is no parameter list for an ECHO within a macro, parameters within the range of ECHO will be associated with the macro parameter list.

Example 1:

```
ECHO        3,3,(T1=A,B,C,T2=D,E,F,T3=G,H,I)
LDA         T1
FAD         T2
STA         T3
```

expands to nine instructions:

```
LDA         A
FAD         D
STA         G
LDA         B
FAD         E
STA         H
LDA         C
FAD         F
STA         I
```

Example 2:

```
ECHO        2,3
OCT         1
OCT         0
```

expands to six assembled computer words, in the order:

```
OCT         1
OCT         0
OCT         1
OCT         0
OCT         1
OCT         0
```

This may be condensed to:

```
ECHO        1,3
OCT         1,0
```

which is an equivalent form producing the same six computer words.

If an ECHO with a parameter list appears within a macro, symbols are substituted as for any other macro. Therefore, an assembler-created symbol or a macro actual parameter may be substituted in the ECHO statement. Lines within an ECHO are local to the ECHO, and parameter substitution proceeds according to ECHO rules. However, parameter substitution is not allowed in the location field in the range of an ECHO within a macro because an ambiguity can result as to which processor (echo or macro) handles the label.

The formal parameter names are local to the range of the ECHO, and must be alphanumeric symbols. The actual parameters may be any expressions which legally may appear where the formal parameters occur. An actual parameter containing commas, blanks, or parentheses, must be enclosed in parentheses.

A location symbol within an ECHO range is assigned only in the first repetition and ignored in successive repetitions. + or - in a location field, however, is repeated in each iteration. A location symbol in an ECHO instruction is ignored.

The parameter list in ECHO may be continued on subsequent lines by repeating the ECHO pseudo instruction on each line. If the parameter list continues to a second line, the first line must not contain any blanks, even though this requires splitting a name between two cards. The rules governing the use of a parentheses in actual parameters are the same as for macro calls.

Example:

```
ECHO        3, 2, (P1=A, B, P2=

ECHO        C, D, P3=E, F)

LDA         P1
  .
  .
  .
```

The instructions within the range of an ECHO need not generate an integral number of machine words. Consider the two examples:

```
ECHO        1,3          ECHO        2,3
LDA         0            OCT         1
                         LDA         0
```

The first produces three 24-bit sequential machine instructions; the second produces five and a half computer words.

3.9 IFT

1 8	10
symbol	IFT,s m(i,j),n(i,j),p

IFT makes a comparison of character strings m and n to determine if coding lines which follow are to be assembled or skipped. IFT may be used only within the range of an ECHO or MACRO pseudo instruction. If it occurs elsewhere, it will be flagged as an O error. An operation modifier, s, is required. A number of lines, p, will be assembled if the condition m s n is satisfied. s may be any of the following:

	Assemble if:
EQ	$m = n$
NE	$m \neq n$
GT	$m > n$
GE	$m \geq n$
LT	$m < n$
LE	$m \leq n$
IN	m included in n; the character string n contains the characters in string m in sequence, but not necessarily consecutively.

The address field consists of two or three subfields. The m and n subfields must be present; either or both may be a single formal parameter as defined in the MACRO and ECHO discussions. Either may be a string of characters, enclosed in slashes, for literal comparison; such a string may not, itself, contain slashes. The slashes bounding such a string are not considered part of the string to be compared. The p subfield may contain an expression resulting in a non-relocatable value. Formal parameters may be included in the expression. If p and the preceding comma are omitted, the range of the IFT is determined by an ENDIF pseudo instruction.

The m and n subfields may include an optional modifier of the form (i, j) to define a portion of the actual parameter to be used in the comparison. This modifier may not contain formal parameters. The (i, j) modifier may be in one of four forms, in which x, y and z represent integers and k represents any non-blank BCD character except slash.

Form	Interpretation
(x, y)	y consecutive characters beginning with the xth character of the actual parameter
(z=k, y)	y consecutive characters following the zth occurrence of the character k
(x, z=k)	consecutive characters beginning with the xth character up to, but not including, the zth occurrence of the character k following the xth character
($z_1=k_1$, $z_2=k_2$)	consecutive characters following the z_1th occurrence of character k_1 up to, but not including, the z_2th occurrence of character k_2

If z, but not the equal sign, is omitted, z is assumed one.

Example:

Macro definition:

```
SAMPLE     MACRO     P1,P2
           LDA       P1
           ADD       P2
           IFT,EQ    P1,P2,1
           STA       A
           IFT,EQ    P2,/EED/,1
           STA       B
           IFT,EQ    P1(2,3),P2,1
           STA       C
           IFT,EQ    P1(2=E,2),/DL/
           STA       D
           ENDIF
           IFT,IN    P1(3,2=S),/EDXLEYS/,1
           STA       E
           IFT,LT    P1(1=H,3=E),/MEDL/,1
           STA       F
           ENDM
```

A macro call of the form

```
SAMPLE HEEDLESS,EED
```

will generate the following sequence of instructions:

	Explanation
LDA HEEDLESS	(STA A is skipped; HEEDLESS ≠ EED)
ADD EED	
STA B	EED = EED
STA C	2nd, 3rd and 4th characters of HEEDLESS = EED
STA D	The two characters following the second E in HEEDLESS = DL
STA E	The characters EDLES in HEEDLESS occur in EDXLEYS in the same order.
STA F	The characters EEDL in HEEDLESS < MEDL

A macro call of the form

SAMPLE HEEDLESS, HEEDLESS

will generate the following sequence of instructions:

	Explanation
LDA HEEDLESS	(STA B is skipped; HEEDLESS ≠ EED) (STA C is skipped; EED ≠ HEEDLESS)
ADD HEEDLESS	
STA A	HEEDLESS=HEEDLESS
STA D	Same as above
STA E	
STA F	

3.10 IFF

1 8	10
symbol	IFF,s m(i,j),n(i,j),p

IFF is identical to IFT, except that the lines are assembled if the condition tested for is false.

COMPRESSED SYMBOLIC (COSY) 4

In addition to listable and relocatable binary output, the user may elect to receive a compressed symbolic deck (COSY), in binary, as output from COMPASS. This results in a deck reduced in size by a maximum ratio of 19:1. The COSY deck may be used as input in subsequent COMPASS assemblies, thus realizing a significant saving in assembly time.

The COSY deck may be modified, using COMPASS symbolic language, by the COSY correction instructions, DELETE, REPLACE, and INSERT. An up-to-date COSY deck may be produced with each subsequent assembly. Since the COSY deck is a compressed image of the source deck, it will not contain expansions of ECHO's or macros; it will contain conditionals and all instructions in their range, whether or not the condition is satisfied.

A number of COSY decks, each comprising a subprogram, may be maintained contiguously on tape. The pseudo instruction, BYPASS, provides rapid access to a particular deck for processing.

The deck produced by specifying the COSY output option on the COMPASS card consists of COSY text card images and a COSY end card image. Each text instruction, beginning with IDENT, is assigned a sequence number which is printed at the right side of the output listing. These sequence numbers are used as reference points when modifying a COSY deck.

4.1 COSY INPUT

The input unit for COSY decks is specified on the COMPASS control card unless COSY input is from the standard input unit (INP). It may or may not be the same as the unit selected for Hollerith input. COSY correction decks containing COSY instructions and symbolic corrections to COSY decks are entered via the Hollerith input unit. If COSY input is to be used, the first card image following the COMPASS control card must be one of the COSY instructions, BYPASS, INSERT, REPLACE, DELETE or COSY.

BYPASS

1 8	10
	BYPASS n

Where a number of COSY decks are on the same tape, BYPASS provides a means of skipping n contiguous decks to arrive at a particular deck for processing. If BCD or COSY output is specified on the COMPASS control card, new Hollerith or COSY decks will be produced from the bypassed decks. BYPASS must follow the COMPASS card to skip from the beginning of a COSY tape; otherwise, it must follow the identifier, COSY.

COSY CORRECTIONS

Corrections are made to a COSY deck through the correction instructions INSERT, REPLACE and DELETE. Each correction instruction may be followed by a correction set consisting of machine and pseudo instructions coded in COMPASS symbolic form. A correction set is terminated by another correction instruction or by the instruction, COSY. Each inserted correction is printed on the preprocessor listing together with its COSY number.

```
1      8  10
          INSERT  m
```

The INSERT instruction causes the card images which follow it to be inserted after line m; m is a sequence number in the COSY deck. Card images are inserted until the next correction instruction or the instruction, COSY, is encountered.

```
1      8  10
          REPLACE  m,n
```

With REPLACE, lines m through n may be removed and replaced by the card images which follow; replacement need not be one for one. Any number of lines may be removed by a single REPLACE card. If a single line is to be replaced, only m need appear in the address field. Replaced lines are not logged in the correction listing. Sequence number m must be less than or equal to n.

```
1      8  10
          DELETE  m,n
```

With this card any number of lines m through n may be deleted. A single line may be deleted by specifying m only. Deleted lines are logged in the correction listing with their COSY numbers. If corrections follow DELETE, they replace the deleted lines as in the REPLACE pseudo instruction. Sequence number m must be less than or equal to n.

Thc correction deck is terminated by the occurrence of the identifier, COSY. It marks the separation of the correction deck and the COSY deck to which it applies.

COSY IDENTIFIER

```
1      8  10
          COSY
```

This instruction identifies the input which follows as a COSY deck to be read from the COSY input unit. Any corrections to the deck must precede the COSY identifier. If no corrections precede COSY, the assembly proceeds normally according to the COMPASS control card options.

Example:

The fourth deck on a COSY tape consists of the following subprogram:

```
           IDENT        PROG         1 (sequence numbers)
           ENTRY        A,C          2
A          SLJ          **           3
           ENI          19,1         4
B          LDA          C,1          5
           INA          50           6
           AJP,PL       D            7
           AJP,ZR       D            8
           ENA          0            9
           STA          C,1          10
D          IJP          B,1          11
           SLJ          A            12
C          BSS          20           13
           END                       14
```

The above deck is located on the tape and corrected by the following sequence:

```
           BYPASS       3
           REPLACE      4
           ENI          29,1
           DELETE       7,8
           AJP,PL       E
           INSERT       12
E          INA          -101
           AJP,MI       D
           SLJ          D-2
C          BSS          30
           DELETE       13
           COSY
```

Assuming that the COSY output option has been selected on the COMPASS control card, the first three COSY decks will be copied on the new COSY tape, and the corrected fourth deck written on tape as follows:

```
          IDENT       PROG          1
          ENTRY       A,C           2
A         SLJ         **            3
          ENI         29,1       ***4
B         LDA         C,1           5
          INA         50            6
          AJP,PL      E          ***7
          ENA         0             8
          STA         C,1           9
D         IJP         B,1          10
          SLJ         A            11
E         INA         -101      ***12
          AJP,MI      D         ***13
          SLJ         D-2       ***14
C         BSS         30        ***15
          END
```

4.2 COSY LISTING

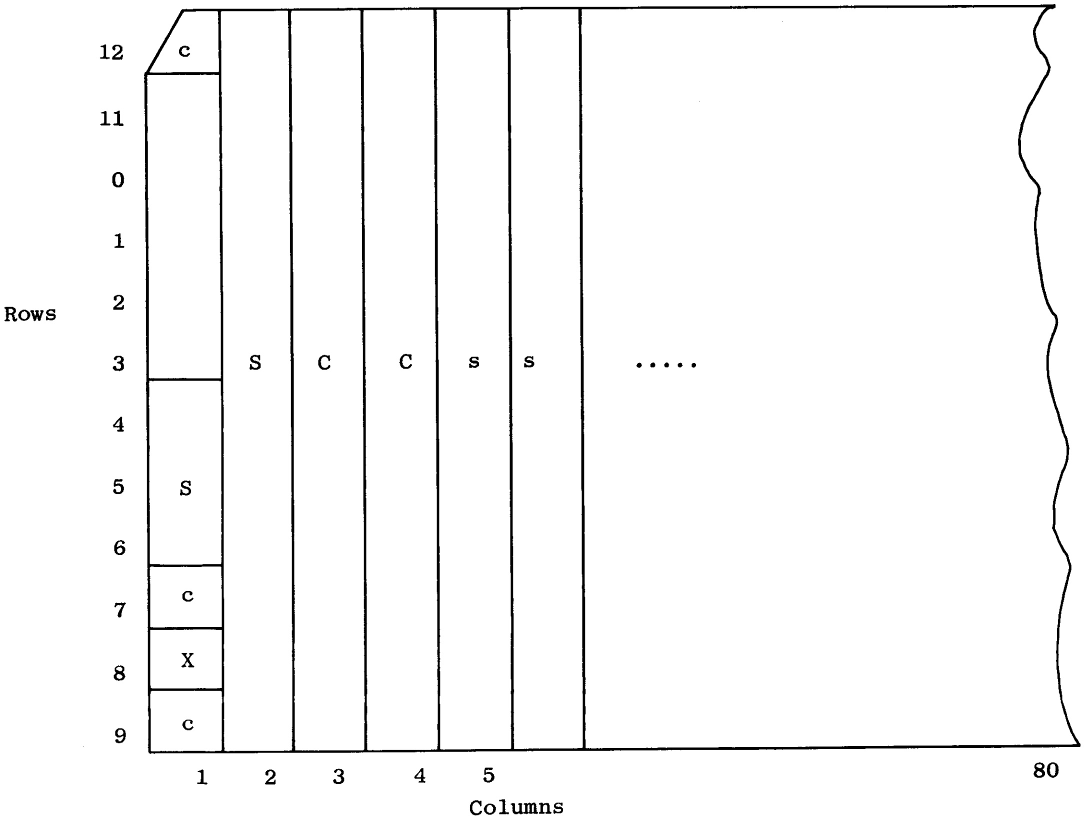
An assembly listing will be produced for a new COSY deck if the list option is specified on the COMPASS control card. The listing will begin with a preprocessor correction listing of all insertions and deletions, except that deletions resulting from REPLACE corrections will not be logged.

If the COSY output option is specified on the COMPASS control card, the listing will contain re-sequenced COSY sequence numbers. Numbers representing corrected lines will be preceded by a triple asterisk. If COSY output is not specified, sequence numbers are not changed and corrected lines are identified only by the symbol, ***.

4.3 COSY DECK FORMATS

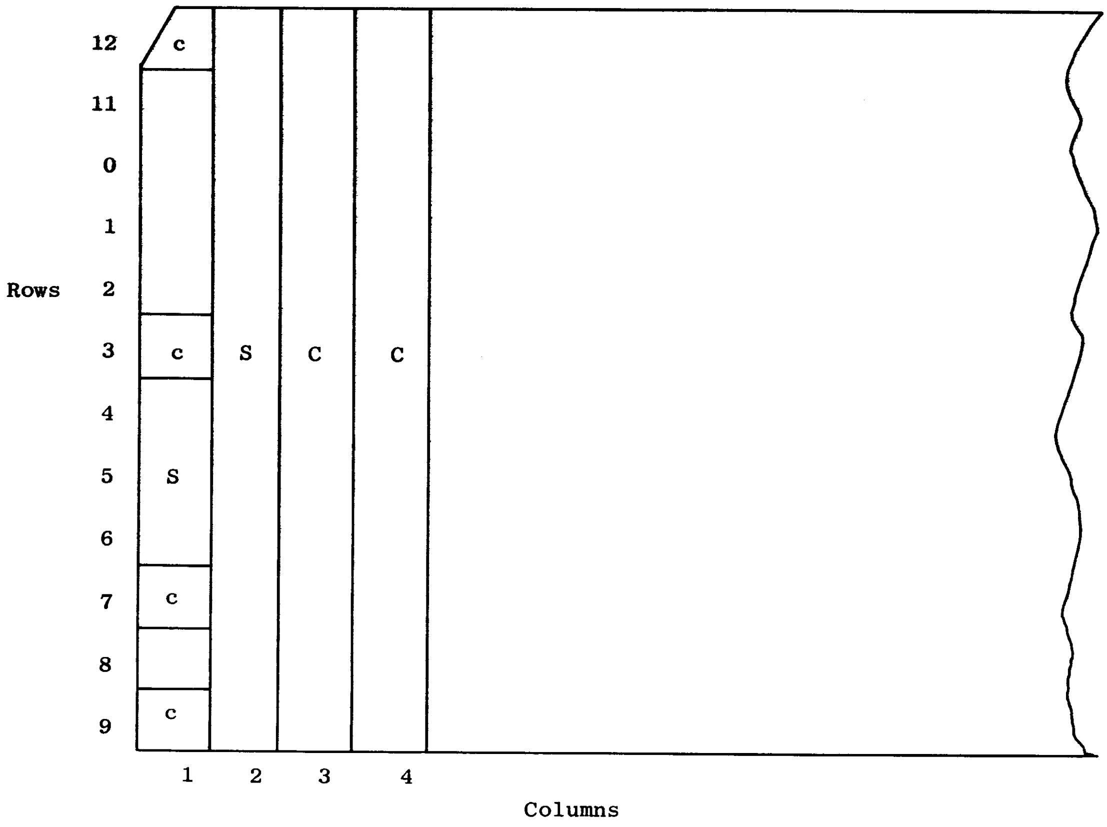
Compressed Symbolic Cards

Columns	Rows		Purpose
1	12,7,9	c	COSY deck Identification
	4,5,6	S	Sequence Number: the high order octal digit
	8	X	Ignore-checksum bit
2	12,11,0,1-9	S	Sequence Number: the low order four octal digits
3-4	12,11,0,1-9	C	Checksum: 24-bit
5		e	First card only: first compressed card image (col. 5 to a77$_8$) is COSY edition number, BCD right justified with leading blanks.
5-80	12,11,0,1-9	s	Compressed Symbolic Cards: 19 machine words



COSY End Card

Columns	Rows		Purpose
1	12, 3, 7, 9 4, 5, 6	c S	COSY End Card Identification Sequence Number: high order octal digit
2	12, 11, 0, 1-9	S	Sequence Number: low order four octal digits
3-4	12, 11, 0, 1-9	C	Checksum: 24-bit
5-80	12, 11, 0, 1-9		Zeros



4.4 COSY DIAGNOSTICS

Messages (Appear on listable output unit)	Assembler Action
CARDS NOT PROCESSED Sequence numbers specified in an INSERT, REPLACE or DELETE conflict with those on a previous instruction or are erroneous.	Correction set following erroneous correction instruction is bypassed.
FULL MEMORY-CORRECTIONS Compressed correction sets exceed available storage.	Remainder of correction deck and COSY deck are bypassed. Next subprogram is processed.

The programmer may assemble the subprogram in two passes. First, assemble the COSY deck with half of the correction deck, obtaining a new COSY deck. Second, after changing the sequence numbers specified in the correction instructions in the second half of the correction deck, reassemble with the new COSY deck.

SEQ. or CHKSUM ERROR XXXXXXXX Sequence or checksum error on COSY card. XXXXXXXX are columns 1 and 2 of the erroneous card.	Assembly continues if only the checksum is in error. A sequence error terminates assembly, preventing a load-and-go, the remainder of the COSY deck is bypassed and processing resumes with the next subprogram.
END CARD DELETED An END card has been deleted and there is no END card in the last correction set.	An END card with an illegal address is produced, which prevents a load-and-go.
EOF IN COSY DECK An end-of-file card has been encountered in COSY deck.	Terminates the job. The programmer may remove the card.

DECK STRUCTURE 5

The COMPASS assembler is called from the system library tape using a SCOPE control statement of the following form:

$^{7}_{9}$COMPASS, parameters

This card contains a 7-9 punch in column 1 followed in column 2 by COMPASS in Hollerith. The card is free field after column 2. The parameters are separated by commas and may appear in any order. The parameter field is terminated by a period or the end of the control card.

If no options are present, only lines with error flags and the basic assembler headings are printed. Options can be abbreviated to the first character. Most options may be followed by = n; n is the number of a logical unit to be used for that option. If = n is absent, COMPASS will make a standard assignment for the option. Unrecognized options and extraneous characters are ignored.

The assembler produces output in either of two passes. The first pass can produce COSY (C) or BCD(B). The second pass produces binary (P) or load-and-go (X). One logical unit may have only one output per pass assigned to it. Therefore C and B may not be assigned to the same unit, whereas C and X may be. Such a tape would have alternating decks of COSY (C) and load-and-go (X).

The optional parameters and their meanings are defined below:

Option	Abbreviation	Function	Value
INPUT(BCD)	I = n	BCD input is on logical unit n.	1 to 49 60 †
YINPUT (COSY input)	Y = n	Cosy input is on logical unit n.	1 to 49 60 †
PUNCH (binary output)	P = n	Punch relocatable binary deck on logical unit n.	1 to 49 62 †
COSY (COSY output)	C = n	Produce COSY output on logical unit n. If C is absent or zero, no COSY output is produced.	1 to 49 62 †

† Standard input or output device assigned if option is absent.

Option	Abbreviation	Function	Value
XECUTE (load-and-go tape)	X = n	Produce binary output for load-and-go on logical unit n. If X is absent or zero, no load-and-go tape is produced.	1 to 49 69 †
BCD Output	B = n	Produce Hollerith output on n. Logical unit n must be specified. If the C option is specified, the B option is ignored.	1 to 49 62
LIST	L = n	List assembled programs. If the option is absent or equal to zero, diagnostics will appear on the standard output list.	1 to 49 61 †
REFERENCE	R	List a cross reference symbol table. The list will appear on the unit requested by the LIST option or on standard unit if L is omitted. Undefined and doubly-defined symbols will appear whether or not R is specified.	none
MACRO LIST	M	List the ECHO expansion and macro calls, and list lines skipped following conditionals. If M is not specified, skipped lines are not listed and only the macro calls and ECHO prototypes are listed.	none
TAPE MACROS	T	Call tape system macros. This allows a programmer to assemble a program under a drum system using tape macro calls, and execute his program on a tape system. If T is not specified, macros of the current operating system are called.	
DRUM MACROS	D	Call drum system macros. This allows a programmer to assemble a program under a tape system using drum macro calls, and execute his program on a drum system. If D is not specified, macros of the current operating system are called.	

† Standard input or output device assigned if option is absent.

5.1 DECK SEQUENCE

COMPASS

The COMPASS card precedes subprogram decks to be assembled or the correction deck and COSY deck.

COSY ORDER

If the BCD and COSY input units are the same, each correction deck is terminated by the COSY identifier card and is followed by the associated COSY deck. The assembly process is terminated by a SCOPE card or end-of-file mark immediately following the last COSY deck.

If the BCD and COSY input units are not the same, the BCD input unit contains contiguous correction decks, eachterminated by a COSY identifier card. A BYPASS card is legal either as the first card or immediately following a COSY identifier card.

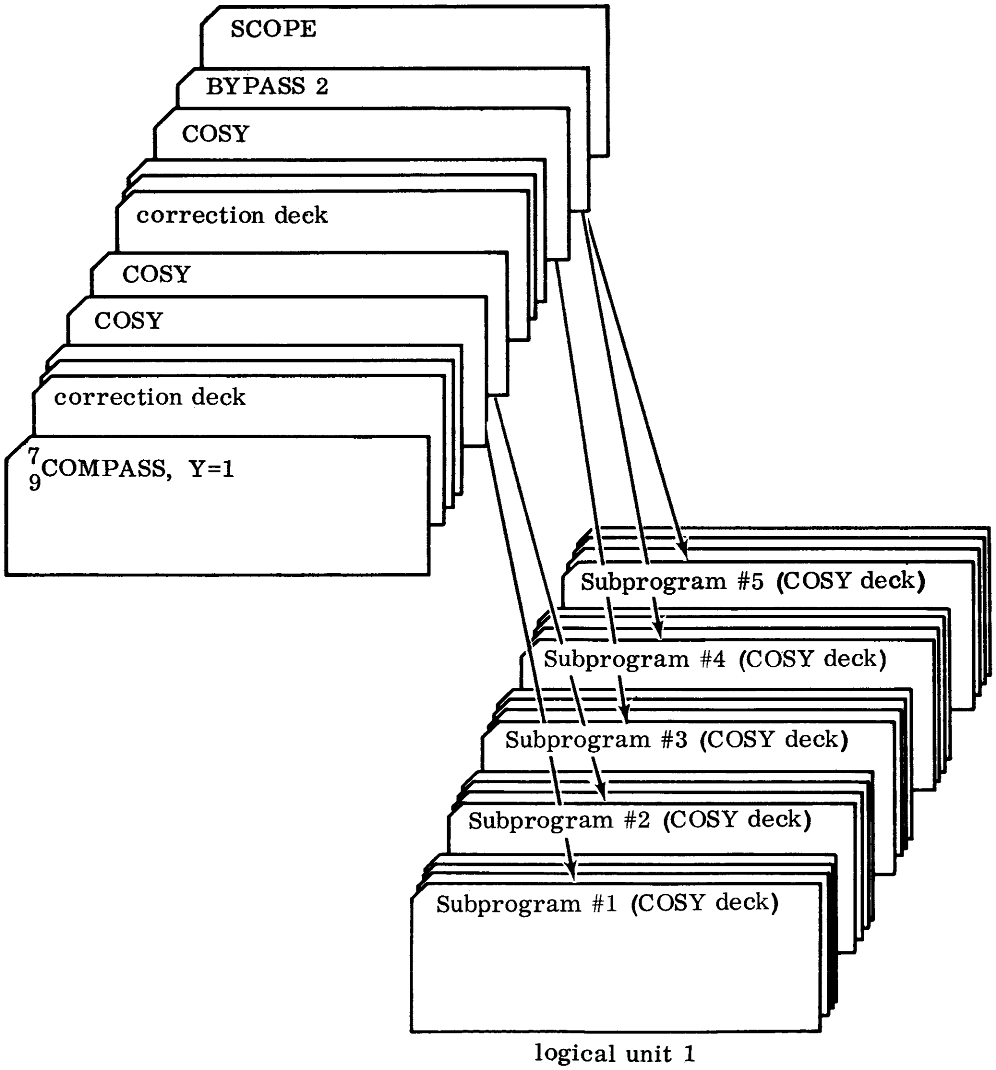
SUBPROGRAM ORDER

The first card of a subprogram deck must be an IDENT card. LIBM cards, naming library macros to be used, follow the IDENT card. Programmer macro definitions, each consisting of a MACRO heading card(s), prototype cards and an ENDM terminating card, appear next. If there are no LIBM's, the programmer macro definitions immediately follow the IDENT card, except that the pseudo instructions REM, LIST, NOLIST, SPACE or EJECT may precede MACRO. Next appear any of the machine instructions, macro instructions or pseudo instructions, in any order. The last card must be an END card to identify the end of the subprogram.

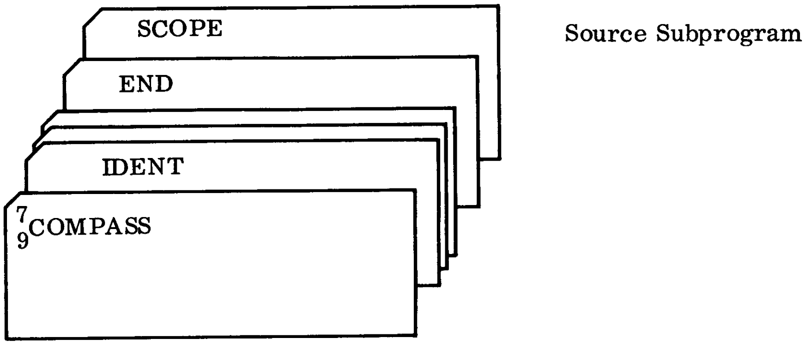
SCOPE

A SCOPE card terminates subprogram decks or the correction deck and previously assembled COSY deck.

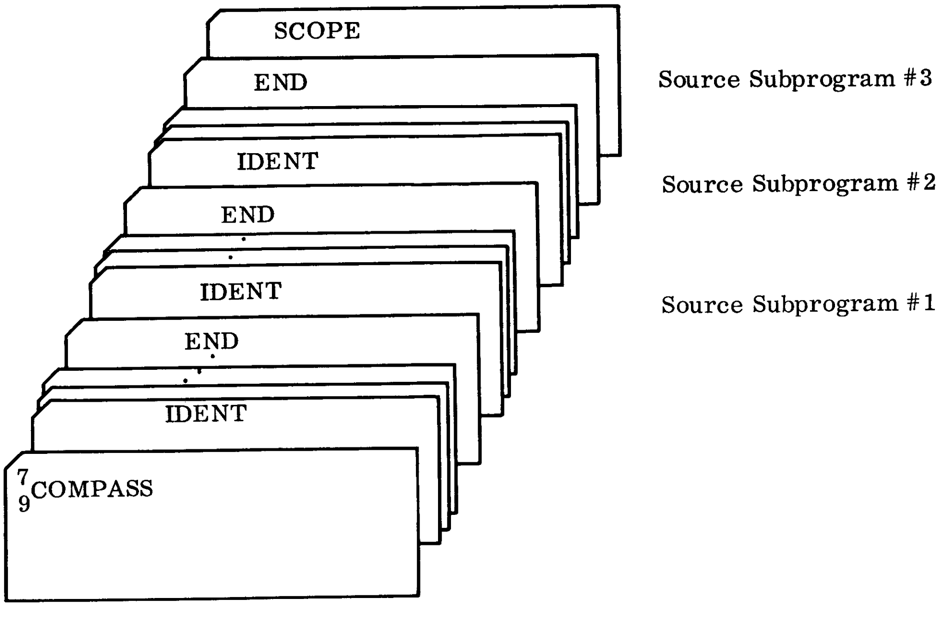
Maintaining a library of COSY decks.


SCOPE
BYPASS 2
COSY
correction deck
COSY
COSY
correction deck
$^{7}_{9}$COMPASS, Y=1
Subprogram #5 (COSY deck)
Subprogram #4 (COSY deck)
Subprogram #3 (COSY deck)
Subprogram #2 (COSY deck)
Subprogram #1 (COSY deck)
logical unit 1

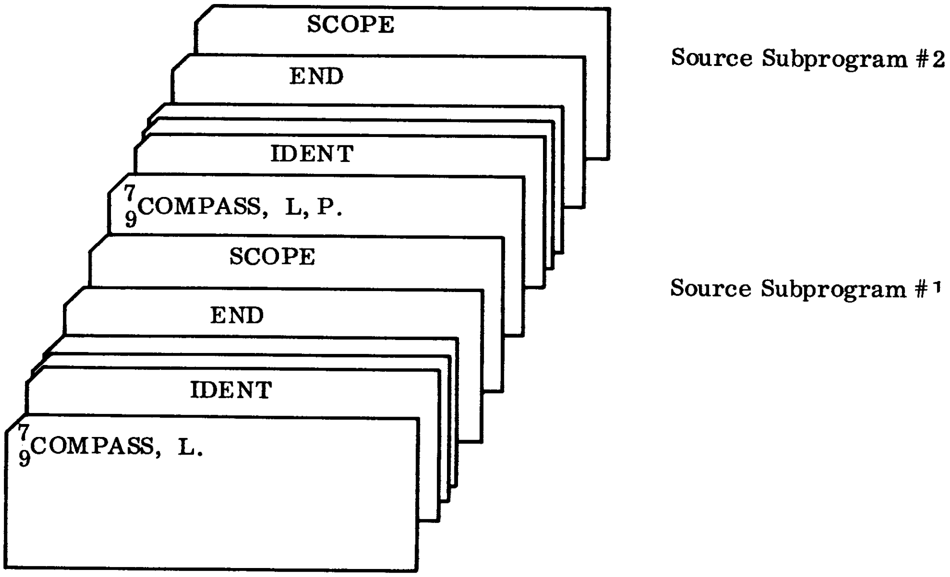
Placement of IDENT and END cards for assembly of a single subprogram:



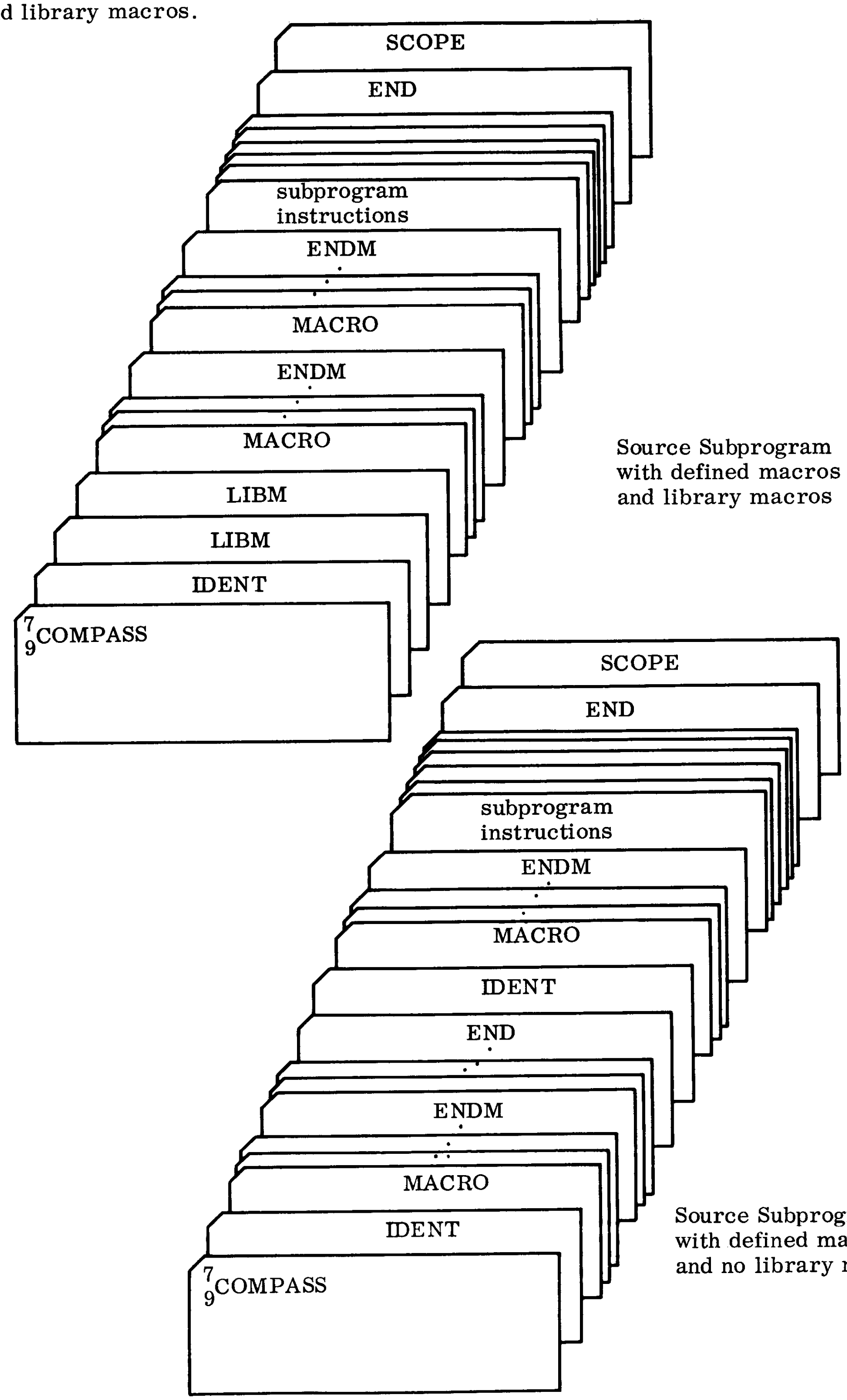
Placement of IDENT and END cards for assembly of multiple subprograms.



Placement of IDENT and END for assembly of several subprograms with differing COMPASS parameters.



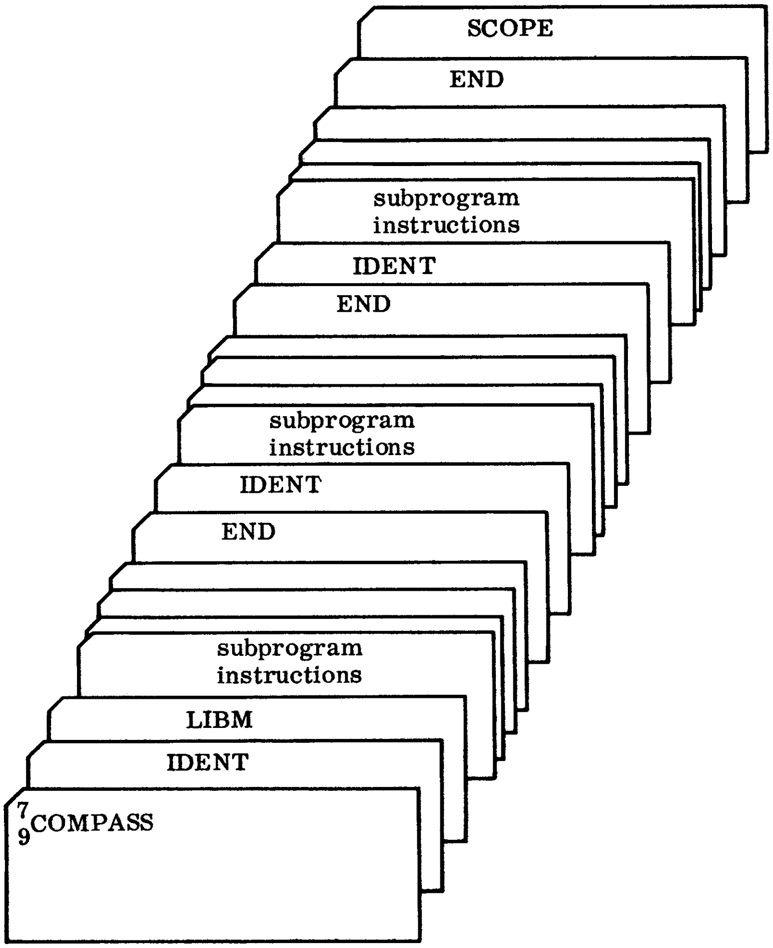
Placement of MACRO, ENDM and LIBMs for an assembly with defined macros and library macros.



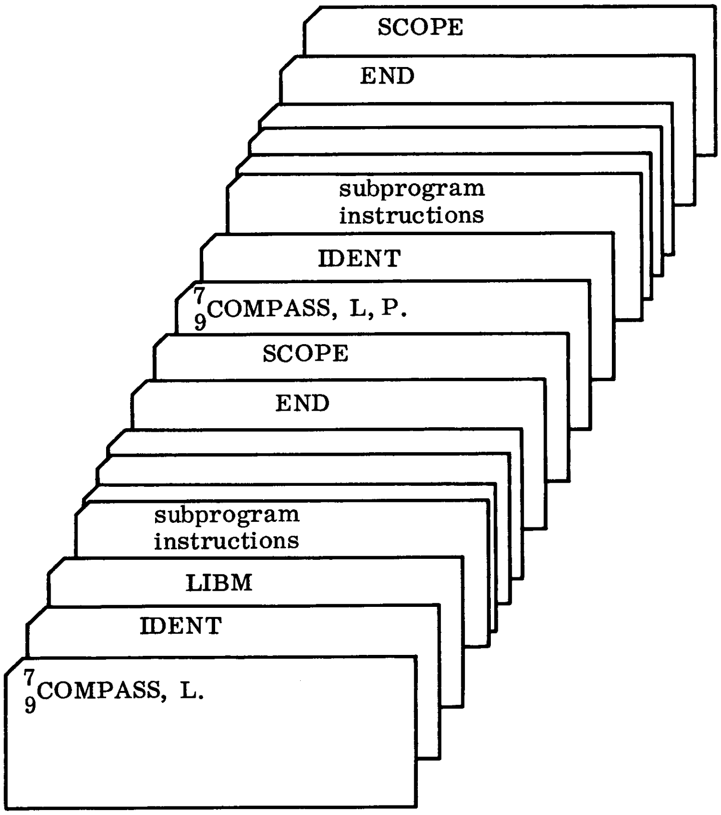
Source Subprogram with defined macros and library macros

Source Subprogram with defined macros and no library macros

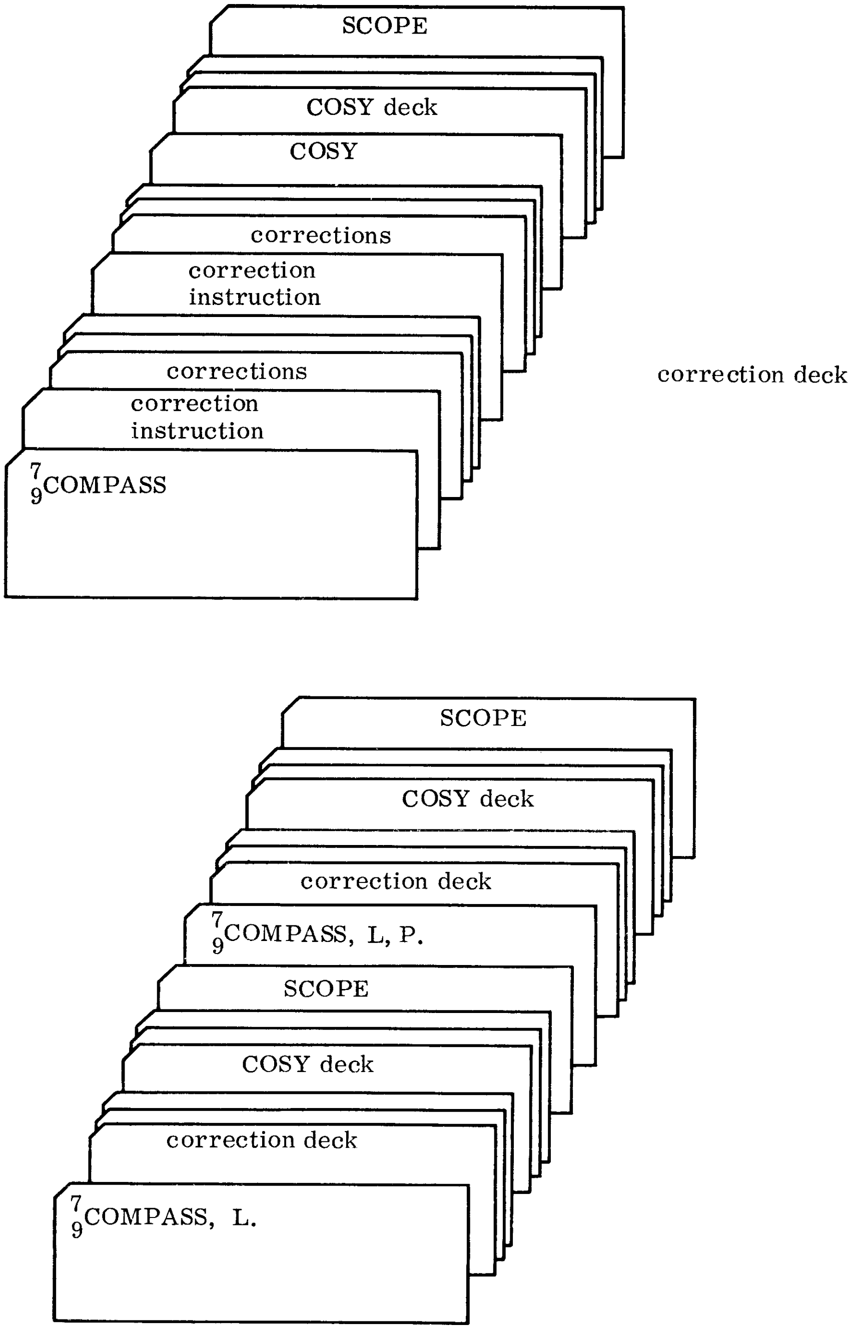
Placement of LIBMs for several source subprograms assembled tog ther which use library macros.



Library macros called once for all the subprograms



Placement of COSY deck and corrections.



ERROR DIAGNOSTICS AND OUTPUT LISTING 6

6.1 OUTPUT LISTING

An output listing is produced by the assembler if a ${}^{7}_{9}$COMPASS card specifies the list option. A line of print contains information as follows; ordered from left to right on the page:

error codes

location of the machine word

word contents

source card image

COSY line sequence number (if COSY input or output is used)

ERROR CODES

Listed error codes may include the following; The occurrence of any one of these errors causes a printed line even though the list option is not specified:

A error — An address field error occurred. Either too many subfields for a machine instruction appear on a code line, a subfield is terminated improperly, illegal elements appear, or a relocation error occurred.

C error — An attempt was made to preset a numbered common block or to store data beyond the last word of a labeled common block.

D error — A symbol is doubly defined. The assembler assigns the value for the first symbol encountered whenever the symbol is referenced.

F error — An assembler table is full. No assignment is made if a table entry would cause overflow of a COMPASS table.

L error — A location field error occurred. A location symbol is improperly formatted or the location field of an EQU instruction doesn't contain an alphanumeric symbol.

M error — An illegal or undefined modifier appears. A modifier appears where none are allowed or a required modifier is absent.

O error — An operation code is invalid or misplaced. For an invalid operation code or a misplaced IDENT, LIBM, MACRO, ENDM or ECHO, a half-word of zeros is substituted. A misplaced COMMON, IFT or IFF is listed but not processed. A misplaced SCOPE is interpreted by the assembler as an END followed by a SCOPE. An END occurring within the range of a macro or an ECHO terminates the assembly.

R error A range error code is signalled for an ENDM which appears within the range of an ECHO or within the range of a conditional that doesn't satisfy the assembly condition. The range of a conditional whose condition for assembly is not satisfied falls outside the range of an ECHO.

U error An undefined symbol appears in the address field; zeros are substituted.

MACHINE LOCATION

Five octal digits appear to the right of error codes signifying the location to which the machine word is assigned. The location digits appear only on upper half words.

FULL WORD CONTENTS

The assembled content of a machine half word appears next, consisting of three terms, a 2-digit operation, a 1-digit index designator, and a 5-digit address. Program addresses are preceded by a P, common addresses by a C. The first reference to an external address consists of all sevens preceded by an X. Subsequent references to that external address appear as the address of the previous reference to the external address and are preceded by an X.

SOURCE CARD IMAGE

The input card image appears to the right of the word content and is identical to the coded line.

COSY SEQUENCE NUMBER

The COSY sequence number is a five digit decimal number.

ADDITIONAL LINES

At the top of every page of the listing, a line is printed consisting of the COMPASS version number (enclosed in parentheses), the program title, the date, an edition number indicating the nth COSY deck, and a page number.

After the first title line, the following information is produced by the COMPASS list facility:

program length

block names and length

entry points and addresses

external symbols

At the end of the listing, 3600 COMPASS produces a list of the following:

undefined symbols

doubly defined symbols

an error count in octal

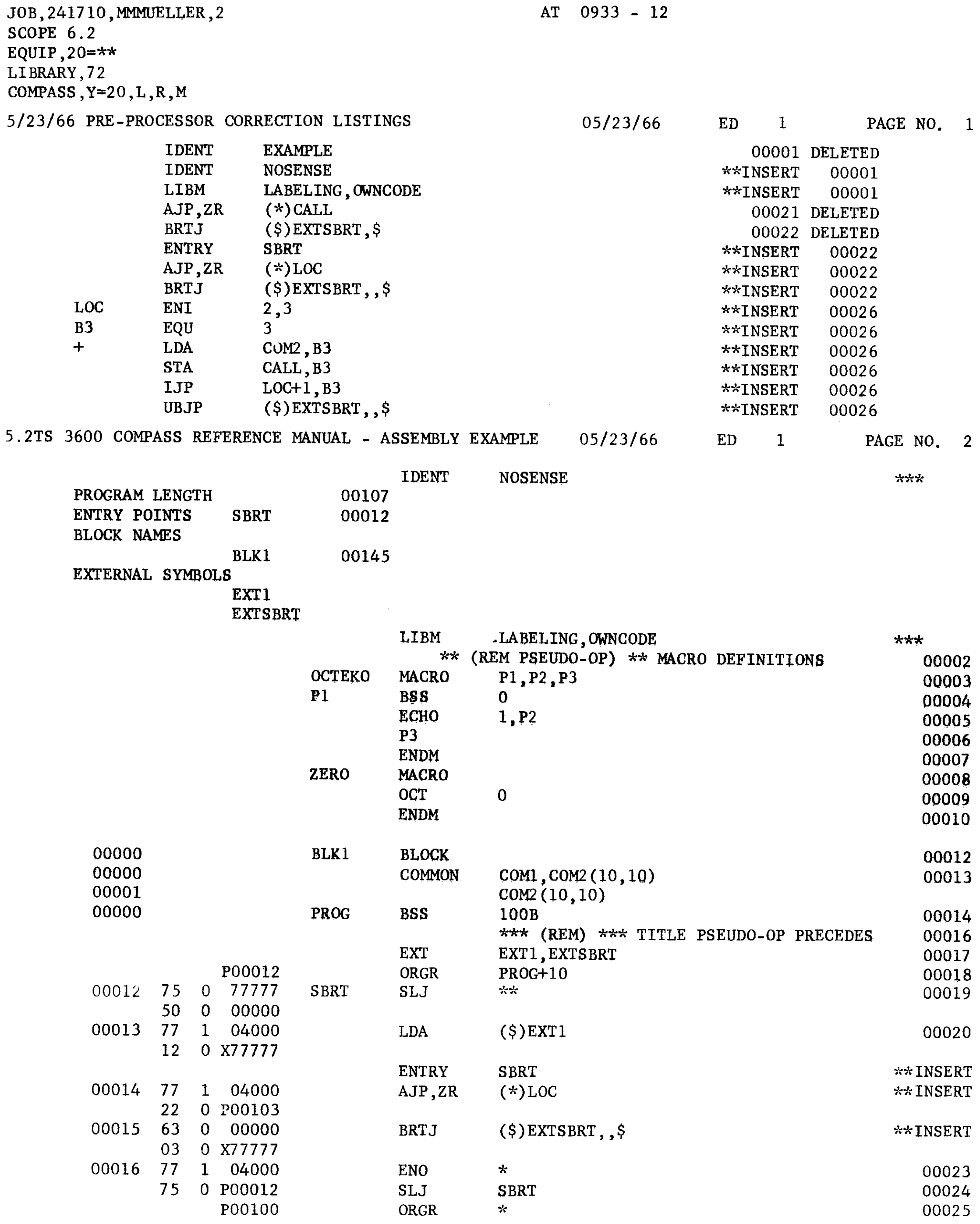
a cross referenced symbol table, if requested

Even though the list option is not requested, an IDENT line, program length, block names and lengths, entry point names and their program addresses, external symbols, undefined symbols, doubly defined symbols and any error diagnostics are printed.

6.2 ERROR MESSAGES

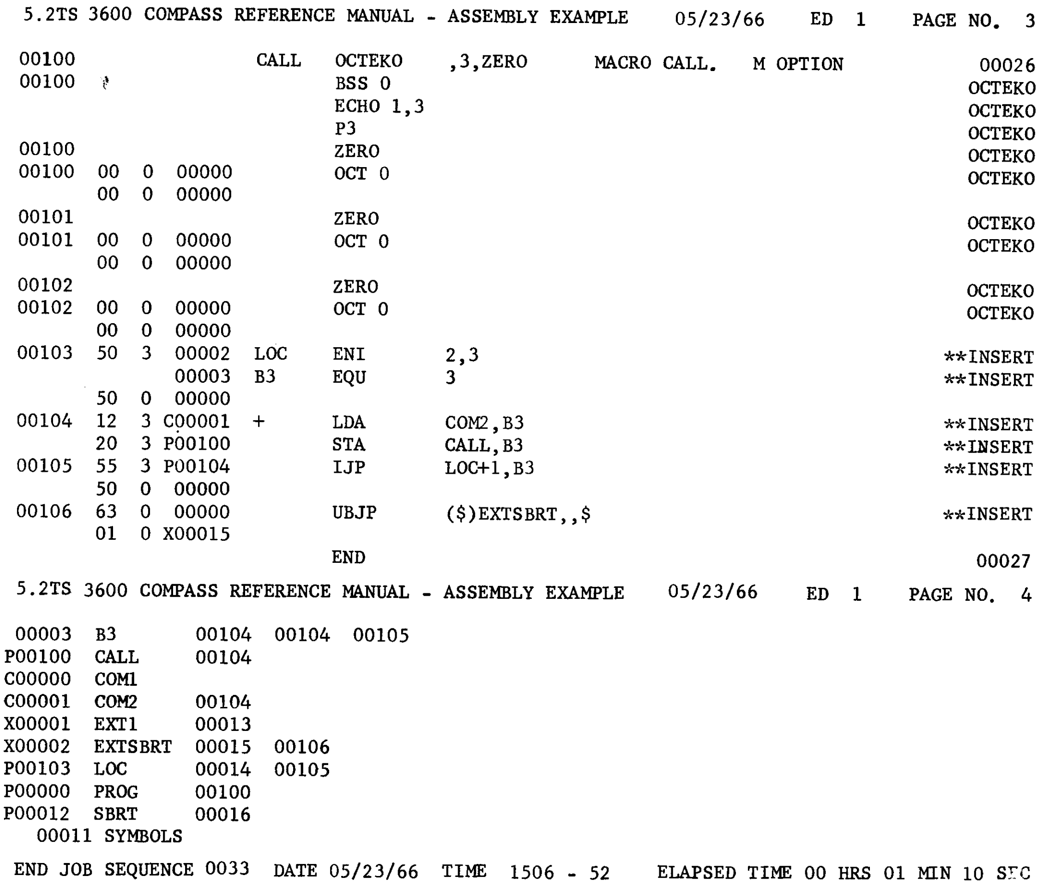
Error messages are placed on the standard output if certain conditions occur.

Message	Error Condition	Action
FULL MEMORY	Available memory is exceeded.	Processing is discontinued. Reduce number of programmer macros, size of correction deck, number of library macros requested, and/or number of system macros contained on library tape and reassemble.
NO IDENT CARD	The first card of a sub-program is not an IDENT, END, or SCOPE card.	Processing continues.
INVALID CHARACTERS ON FOLLOWING CARD ARE DENOTED AS ≤	A BCD character with an octal code of: 12,15,16,17,35, 36,37,55,56,57, 72,76,77 appears on the input card.	The assembler substitutes the character ≤ for the illegal input character. A load-and-go operation is not effected.
FAILURE XXXXX	A machine failure is suspected. XXXXX represents the absolute location at which the failure was detected.	Processing is discontinued.



```
JOB,241710,MMMUELLER,2                           AT  0933 - 12
SCOPE 6.2
EQUIP,20=**
LIBRARY,72
COMPASS,Y=20,L,R,M
5/23/66 PRE-PROCESSOR CORRECTION LISTINGS                  05/23/66     ED    1         PAGE NO.  1
                IDENT     EXAMPLE                                          00001 DELETED
                IDENT     NOSENSE                                        **INSERT   00001
                LIBM      LABELING,OWNCODE                               **INSERT   00001
                AJP,ZR    (*)CALL                                          00021 DELETED
                BRTJ      ($)EXTSBRT,$                                     00022 DELETED
                ENTRY     SBRT                                           **INSERT   00022
                AJP,ZR    (*)LOC                                         **INSERT   00022
                BRTJ      ($)EXTSBRT,,$                                  **INSERT   00022
       LOC      ENI       2,3                                            **INSERT   00026
       B3       EQU       3                                              **INSERT   00026
       +        LDA       COM2,B3                                        **INSERT   00026
                STA       CALL,B3                                        **INSERT   00026
                IJP       LOC+1,B3                                       **INSERT   00026
                UBJP      ($)EXTSBRT,,$                                  **INSERT   00026
5.2TS 3600 COMPASS REFERENCE MANUAL - ASSEMBLY EXAMPLE     05/23/66     ED    1         PAGE NO.  2
                                                  IDENT     NOSENSE                                ***
       PROGRAM LENGTH              00107
       ENTRY POINTS    SBRT        00012
       BLOCK NAMES
                       BLK1        00145
       EXTERNAL SYMBOLS
                       EXT1
                       EXTSBRT
                                                  LIBM      .LABELING,OWNCODE                      ***
                                                      ** (REM PSEUDO-OP) ** MACRO DEFINITIONS      00002
                                         OCTEKO   MACRO     P1,P2,P3                               00003
                                         P1       BSS       0                                      00004
                                                  ECHO      1,P2                                   00005
                                                  P3                                               00006
                                                  ENDM                                             00007
                                         ZERO     MACRO                                            00008
                                                  OCT       0                                      00009
                                                  ENDM                                             00010

         00000                           BLK1     BLOCK                                            00012
         00000                                    COMMON    COM1,COM2(10,10)                       00013
         00001                                              COM2(10,10)
         00000                           PROG     BSS       100B                                   00014
                                                            *** (REM) *** TITLE PSEUDO-OP PRECEDES 00016
                                                  EXT       EXT1,EXTSBRT                           00017
                         P00012                   ORGR      PROG+10                                00018
         00012  75  0  77777             SBRT     SLJ       **                                     00019
                50  0  00000
         00013  77  1  04000                      LDA       ($)EXT1                                00020
                12  0 X77777
                                                  ENTRY     SBRT                                 **INSERT
         00014  77  1  04000                      AJP,ZR    (*)LOC                               **INSERT
                22  0 P00103
         00015  63  0  00000                      BRTJ      ($)EXTSBRT,,$                        **INSERT
                03  0 X77777
         00016  77  1  04000                      ENO       *                                      00023
                75  0 P00012                      SLJ       SBRT                                   00024
                       P00100                     ORGR      *                                      00025
```



```
 5.2TS 3600 COMPASS REFERENCE MANUAL - ASSEMBLY EXAMPLE    05/23/66    ED  1    PAGE NO.  3

 00100                   CALL  OCTEKO    ,3,ZERO       MACRO CALL.  M OPTION              00026
 00100                         BSS 0                                                     OCTEKO
                               ECHO 1,3                                                  OCTEKO
                               P3                                                        OCTEKO
 00100                         ZERO                                                      OCTEKO
 00100  00  0  00000           OCT 0                                                     OCTEKO
        00  0  00000
 00101                         ZERO                                                      OCTEKO
 00101  00  0  00000           OCT 0                                                     OCTEKO
        00  0  00000
 00102                         ZERO                                                      OCTEKO
 00102  00  0  00000           OCT 0                                                     OCTEKO
        00  0  00000
 00103  50  3  00002  LOC      ENI       2,3                                           **INSERT
               00003  B3       EQU       3                                             **INSERT
        50  0  00000
 00104  12  3 C00001  +        LDA       COM2,B3                                       **INSERT
        20  3 P00100           STA       CALL,B3                                       **INSERT
 00105  55  3 P00104           IJP       LOC+1,B3                                      **INSERT
        50  0  00000
 00106  63  0  00000           UBJP      ($)EXTSBRT,,$                                 **INSERT
        01  0 X00015
                               END                                                        00027
 5.2TS 3600 COMPASS REFERENCE MANUAL - ASSEMBLY EXAMPLE    05/23/66    ED  1    PAGE NO.  4

 00003  B3       00104  00104  00105
P00100  CALL     00104
C00000  COM1
C00001  COM2     00104
X00001  EXT1     00013
X00002  EXTSBRT  00015  00106
P00103  LOC      00014  00105
P00000  PROG     00100
P00012  SBRT     00016
   00011 SYMBOLS

 END JOB SEQUENCE 0033  DATE 05/23/66  TIME  1506 - 52    ELAPSED TIME 00 HRS 01 MIN 10 SEC
```

TABLE SECTION

TABLE 1
MNEMONIC CODES FOR 3600 OPERATION REGISTERS

Source and Destination

Code	Register	Code	Register
LM	Limit Register	QL	Q - Lower Address
B1	B^1 (Index Register 1)	QU	Q - Upper Address
B2	B^2 (Index Register 2)	A	A - Full 48 bits
B3	B^3 (Index Register 3)	Q	Q - Full 48 bits
B4	B^4 (Index Register 4)	D	D Register
B5	B^5 (Index Register 5)	BR	Bounds Register
B6	B^6 (Index Register 6)	IM	Interrupt Mask Register
AL	A - Lower Address	OB	Operand Bank Register
AU	A - Upper Address		

Source Only

Code	Register	Code	Register
IR	Interrupt Register	NC	Normalization Count Register
PZ	Plus Zero (all zeros)	MS	Mode Selection Register
P1	Plus One	P	P Register
MZ	Minus Zero (all ones)	CK	Clock Register
IB	Instruction Bank Register		

NOTE: These mnemonic codes may be used only in ROP, RXT, RGJP, NBJP, ZBJP and RSW to define the p, q and r subfields. If identical symbols are used elsewhere, they must be program defined.

TABLE 2
MNEMONIC CODES FOR INSTRUCTION MODIFIERS

AND	Register and -- ROP instruction
Ao	Use A register in the LBYT or SBYT instruction; o is a one or two-digit decimal integer which specifies the rightmost bit of the byte in A.
AUG	Augment -- XMIT instruction
C	Chain to next control word -- I/O control words
CL	a) Clear source -- augmented instructions b) Clear unused portion of destination -- LBYT, SBYT instructions c) Clear bit g in register p after testing -- NBJP, ZBJP instructions
CM	a) Complement operand -- augmented instructions b) Complement bit g in register p after testing -- NBJP, ZBJP instructions c) Transmit complement -- XMIT instruction
CQ	Clear unused portion of q in RSW and RXT instructions
CR	Clear unused portion of r in RSW and RXT instructions
CW	Control Word to A -- COPY instruction
CWA	Control Word Address to Q -- COPY instruction
D	Conditional decrementing -- RGJP instruction
Ee	In the LBYT, SBYT, and SCAN instructions, e is a one or two-digit decimal integer which specifies the byte size in bits.
EO	End Off; shift is end off and no sign extension -- augmented instructions
EQ	Equal test -- RGJP, IFF, IFT instructions, register equivalence -- ROP instruction
GE	Greater or equal test -- RGJP, IFF, IFT instructions
GT	Greater test -- RGJP, IFF, IFT instructions
I	Indirect addressing -- SEQU, SMEQ, SEWL, SMWL instructions
IMP	Register implication -- ROP instruction

Γable 2 (Cont'd)

IN	Inclusion test -- IFF, IFT instructions
LE	Less or equal test -- RGJP, IFF, IFT instructions
LI	Left indexing -- LBYT, SBYT instructions
LT	Less test -- RGJP, IFF, IFT instructions
MG	Magnitude of operand -- augmented instructions
MI	Minus -- AJP, QJP, ARJ, QRJ instructions
MK	Transmit masked -- XMIT instructions
NE	Not equal test -- RGJP, IFF, IFT instructions
NZ	Non-zero -- AJP, QJP, ARJ, QRJ instructions
OR	Register or -- ROP instruction
PC	Transmit plus constant (in A) -- XMIT instruction
PL	Plus -- AJP, QJP, ARJ, QRJ instructions
Qo	Use Q register in the LBYT, SBYT, or SCAN instruction; o is a one or two-digit decimal integer which specifies right-most bit of the byte in Q.
RI	Right indexing -- LBYT, SBYT instructions
RP	Replace operation -- augmented instructions
SS	Signed shift -- (direction of shift determined by sign of shift count) -- augmented instructions
ST	Set to one -- NBJP, ZBJP instructions
TR	Truncated -- DVF instruction
UN	Un-normalize arithmetic -- augmented instructions
UR	Unrounded arithmetic -- augmented instructions
XOR	Register exclusive or -- ROP instruction
ZR	Zero -- AJP, QJP, ARJ, QRJ instructions
+	Register sum -- ROP instruction
-	Register difference -- ROP instruction

TABLE 3
ADDRESS SUBFIELDS

		Number of bits
Relocatable or fixed		
a	first bank designator	3
i	second bank designator	3
m	first operand address	15
n	second operand address	15
y	operand	15
Fixed only		
b	first index register	3
k	jump or stop key	3
e	equipment designator	3
g	bit designator	6
p	first source register	5
q	second source register	5
r	destination register	5
u	unit designator	9
v	second index register	3
w	operand	15
x	channel number	6

TABLE 4
MNEMONIC MACHINE INSTRUCTIONS

Operation Field	Address Field	Instruction
Inter-Register		
ROP, OR	p, q, r	Register operation
XOR		r = p op q
AND		
IMP		
EQ		
+		
-		
RSW, CQ, CR	q, r	Register swap
RXT, CQ, CR	q, r	Register transmit
Full Word Transmission		
LDA, CM, MG	(a) m, b, v	Load A
LAC, CM, MG	(a) m, b, v	Load A complement
LDQ, CM, MG	(a) m, b, v	Load Q
LQC, CM, MG	(a) m, b, v	Load Q complement
STA, CM, CL, MG	(a) m, b, v	Store A
STQ, CM, CL, MG	(a) m, b, v	Store Q
XMIT, CM, AUG	(a) m, (i) n	Transmit
MK		Note: If either bank term is missing, it is assumed ($).
PC		
Address Transmission		
LIU, CM, MG	(a) m, b, v	Load index upper
LIL, CM, MG	(a) m, b, v	Load index lower
SIU	(a) m, b, v	Store index upper
SIL	(a) m, b, v	Store index lower
SAU, CM, MG	(a) m, b, v	Substitute address upper
SAL, CM, MG	(a) m, b, v	Substitute address lower
ENI	(a) y, b, v	Enter index
ENA, CM	(a) y, b, v	Enter A
ENQ, CM	(a) y, b, v	Enter Q

Table 4 (Cont'd)

Operation Field	Address Field	Instruction
Fixed Point Arithmetic		
ADD, CM, MG	(a) m, b, v	Add
SUB, CM, MG	(a) m, b, v	Subtract
MUI, CM, MG	(a) m, b, v	Multiply integer
DVI, CM, MG	(a) m, b, v	Divide integer
MUF, CM, MG	(a) m, b, v	Multiply fractional
DVF, CM, MG, TR	(a) m, b, v	Divide fractional
Address Arithmetic		
INA, CM	(a) y, b, v	Increase A
INI	(a) y, b, v	Increase index
ISK	(a) y, b, v	Index skip
Single Precision Floating Point Arithmetic		
FAD, RP, CM, MG, UN, UR	(a) m, b, v	Floating add
FSB, RP, CM, MG, UN, UR	(a) m, b, v	Floating subtract
FMU, CM, MG, UN, UR	(a) m, b, v	Floating multiply
FDV, CM, MG, UN, UR	(a) m, b, v	Floating divide
ADX	w	Add to exponent
Double Precision Floating Point Arithmetic		
DLDA, CM, MG	(a) m, b, v	Load A
DSTA, CM, CL, MG	(a) m, b, v	Store A
DFAD, RP, CM, MG, UN, UR	(a) m, b, v	Floating add
DFSB, RP, CM, MG, UN, UR	(a) m, b, v	Floating subtract
DFMU, CM, MG, UN, UR	(a) m, b, v	Floating multiply
DFDV, CM, MG, UR	(a) m, b, v	Floating divide
Logical Operations		
SST, CM, MG	(a) m, b, v	Selective set
SCM, CM, MG	(a) m, b, v	Selective complement
SCL, CM, MG	(a) m, b, v	Selective clear
SSU, CM, MG	(a) m, b, v	Selective substitute

Table 4 (Cont'd)

Operation Field	Address Field	Instruction
Logical Operations (Cont'd)		
LDL	(a) m, b, v	Load logical
ADL, RP, CM, MG	(a) m, b, v	Add logical
SBL, RP, CM, MG	(a) m, b, v	Subtract logical
STL, CM, MG	(a) m, b, v	Store logical
Shifting Operations		
ARS, EO, SS	(a) y, b, v	A right shift
ALS, EO, SS	(a) y, b, v	A left shift
QRS, EO, SS	(a) y, b, v	Q right shift
QLS, EO, SS	(a) y, b, v	Q left shift
LRS, EO, SS	(a) y, b, v	Long right shift
LLS, EO, SS	(a) y, b, v	Long left shift
SCA	(a) y, b, v	Scale A
SCQ	(a) y, b, v	Scale AQ
Replace Operations		
RAD, CM, MG	(a) m, b, v	Replace add
RSB, CM, MG	(a) m, b, v	Replace subtract
RAO, CM, MG	(a) m, b, v	Replace add one
RSO, CM, MG	(a) m, b, v	Replace subtract one
Storage Test		
SSK	(a) m, b, v	Storage skip
SSH	(a) m, b, v	Storage shift
Search		
EQS	(a) m, b, v	Equality Search
THS	(a) m, b, v	Threshold search
MEQ	(a) m, b, v	Masked equality search
MTH	(a) m, b, v	Masked threshold search
SEQU, I	(a) m, n	Search for equality
SMEQ, I	(a) m, n	Search for masked equality
SEWL, I	(a) m, n	Search within limits

Table 4 (Cont'd)

Operation Field	Address Field	Instruction	
Search (Cont'd)			
SMWL, I	(a) m, n	Search magnitude within limits	
LSTU	b, v	Locate list element upper	
LSTL	b, v	Locate list element lower	
Jumps and Stops			
AJP, ZR	(a) m, v	A jump	A modifier is required, it does not cause insertion of the single precision augment instruction.
NZ			
PL			
MI			
QJP, ZR	(a) m, v	Q jump	
NZ			
PL			
MI			
ARJ, ZR	(a) m, v	A return jump	
NZ			
PL			
MI			
QRJ, ZR	(a) m,v	Q return jump	
NZ			
PL			
MI			
IJP	(a) m, b, v	Index jump	
SLJ	(a) m, k, v	Jump	
SJ1	(a) m, v	Selective jump key 1	
SJ2	(a) m, v	Selective jump key 2	
SJ3	(a) m, v	Selective jump key 3	
RTJ	(a) m, v	Return jump	
RJ1	(a) m, v	Selective return jump key 1	
RJ2	(a) m, v	Selective return jump key 2	
RJ3	(a) m, v	Selective return jump key 3	
SLS	(a) m, k, v	Stop	
SS1	(a) m, v	Selective stop jump key 1	
SS2	(a) m, v	Selective stop jump key 2	
SS3	(a) m, v	Selective stop jump key 3	

Table 4 (Cont'd)

Operation Field	Address Field	Instruction	
Jumps and Stops (Cont'd)			
SRJ	(a) m, v	Stop return jump	
SR1	(a) m, v	Selective stop return jump key 1	
SR2	(a) m, v	Selective stop return jump key 2	
SR3	(a) m, v	Selective stop return jump key 3	
EXEC	(a) m, b, v	Execute	
RGJP, EQ	p, y, m, b	Register jump	
GT		Note: Modifier is required.	
LT			
NE			
LE			
GE			
LT, D			
GE, D			
UBJP	(a) m, b, i	Unconditional bank jump	
BJPL	(a) m, b, i	Unconditional bank jump lower	
BRTJ	(a) m, b, i	Unconditional bank return jump	
BJSX	(a) m, b, i	Bank jump and set index	
NBJP, ST	p, g, m, b	Non zero bit jump	
CL			
CM			
ZBJP, ST	p, g, m, b	Zero bit jump	
CL			
CM			
MPJ		Main product register jump	
CPJ	x	Channel product register jump	
DRJ		D Register Jump	
Variable Data Field			
LBYT, Ao, Ee, LI, CL	m, b, v	Load byte	Modifiers Ao or Qo and Ee are required; if neither LI or RI appears, no indexing will be assumed.
Qo RI			
SBYT, Ao, Ee, LI, CL	m, b, v	Store byte	
Qo RI			

Table 4 (Cont'd)

Operation Field	Address Field	Instruction	
Variable Data Field (Cont'd)			
SCAN, Qo, Ee, EQ GT LT NE LE GE	m, b, v	Scan byte Note: Qo, Ee, and one of the comparison modifiers are required.	
Input Output			
CONN	x, e, u, n	Connect	
EXTF	x, w, n	External function	
BEGR	x, (a) m, n	Begin read	If the bank term is missing, it is assumed $.
BEGW	x, (a) m, n	Begin write	
COPY, CW, CWA	x, b	Copy status	
CLCH	x	Clear channel	
IPA		Input to A	
ALG	w	Perform algorithm	
Input Output Control Words			
IOSW, C	(a) m, w	Skip words (write zeros under word count control)	
IOTW, C	(a) m, w	Transmit data under word count control	
IOSR, C	(a) m, w	Skip words to end of record (and write end of record)	
IOTR, C	(a) m, w	Transmit data under word count or to end of record (and write end of record)	
IOJP	(a) m	Jump to (a) m for next control word	

In any of these instructions, if the bank term is missing, it is assumed ($).

Table 4 (Cont'd)

Operation Field	Address Field	Instruction
Others		
INF	w	Internal function
NOP	m	No operation
ENO	a	Enter operand bank register (single precision augment instruction in upper or lower half-word)
00	m, b	Octal instruction from 00-77
.	or	
.		
77	(a) m	
ZRO	(a) m, b, v	Operation code of 00

TABLE 5
PSEUDO INSTRUCTIONS

Mnemonic	Use	Page
BANK	Declare subprogram and common block banks	2-15
BCD	Insert BCD characters	2-17
BES	Reserve block of storage	2-12
BLOCK	Specify block of common	2-12
BRIEF	List control	2-7
BSS	Reserve block of storage	2-11
BYPASS	Skip COSY decks	4-1
CALL	Call an external subroutine	2-3
CODAP	Change input to CODAP-1 format	2-4
COMMON	Declare array in common	2-13
COMPASS	Change input to COMPASS format	2-4
COSY	COSY identification	4-3
DEC	Insert single precision decimal constants	2-16
DECD	Insert double precision decimal constants	2-17
DELETE	Delete portions of program	4-2
DETAIL	List control	2-7
ECHO	Replicate a sequence	3-14
EJECT	Eject a page on the output listing	2-5
END	Specify the end of a subprogram	2-1
ENDIF	Control pseudo instruction	2-10
ENDM	Terminate a macro-definition	3-3
ENTRY	Define entry points in a subprogram	2-2
EQU	Equate an undefined symbol to a defined symbol	2-3
EXT	Define external symbols	2-2
IDENT	Identify the subprogram by name	2-1
IF	Control pseudo instruction	2-9
IFF	Control pseudo instruction	3-18
IFL	Control pseudo instruction	2-9
IFN	Control pseudo instruction	2-8
IFT	Control pseudo instruction	3-16
IFU	Control pseudo instruction	2-9
IFZ	Control pseudo instruction	2-8

Table 5 (Cont'd)

Mnemonic	Use	Page
INSERT	Insert changes in a program	4-2
LIBM	Declare library macros	3-13
LIST	Resume output listing	2-6
MACRO	Define a macro	3-2
< macro name >	Call a macro	3-4
NO LIST	Suppress output listing	2-5
OCT	Insert octal constants	2-16
ORGR	Set location counter	2-14
REM	Insert remarks on the output listing	2-6
REPLACE	Replace portions of a program	4-2
SCOPE	Terminates assembly process	2-4
SET	Symbol definition	2-3
SPACE	Insert spaces in the output listing	2-5
TITLE	Title pages with program name	2-6
TYPE	Insert typewriter codes	2-18
VFD	Assign data in variable byte sizes	2-18
*	An asterisk in column 1 produces remarks on program listing	2-6

TABLE 6
SPECIAL CODES FOR TYPE ENTRIES

BCD Characters	Type Equivalent
*R	Carriage Return
*U	Shift to Upper Case
*L	Shift to Lower Case
*B	Backspace
*T	Tab
*X	
*A	
*S	

TABLE 7
CHARACTER REPRESENTATION AND USAGE

Internal BCD Code†	Character (501 Printer)	External BCD Code	Hollerith Card Punches	Usage Code††
00	0	12	0	A
01-11	1-9	01-11	1-9	A
12	:	00	8,2	B
13	=	13	8,3	C
14	≠	14	8,4 (apostrophe)	D
15	≤	15	8,5	E
16	%	16	8,6	B
17	[	17	8,7	B
20	+	60	12	F
21	A	61	12,1	G
22	B	62	12,2	G
23	C	63	12,3	G
24	D	64	12,4	G
25	E	65	12,5	G
26	F	66	12,6	G
27	G	67	12,7	G
30	H	70	12,8	G
31	I	71	12,9	G
32	<	72	12,0	D
33	.	73	12,8,3	H
34	)	74	12,8,4	I
35	≥	75	12,8,5	B
36	⌉	76	12,8,6	B
37	;	77	12,8,7	B
40	- (minus)	40	11	F
41	J	41	11,1	G
42	K	42	11,2	G
43	L	43	11,3	G
44	M	44	11,4	G
45	N	45	11,5	G
46	O	46	11,6	G
47	P	47	11,7	G
50	Q	50	11,8	G
51	R	51	11,9	G
52	∨	52	11,0	D
53	$	53	11,8,3	J
54	*	54	11,8,4	F
55	↑	55	11,8,5	B
56	↓	56	11,8,6	B
57	>	57	11,8,7	B

† Collating sequence, ascending order, for symbol table sort and for IFT/IFF comparisons.

†† Code is defined following the table.

Table 7 (Cont'd)

Internal BCD Code†	Character (501 Printer)	External BCD Code	Hollerith Card Punches	Usage Code††
60	blank	20	blank	K
61	/	21	0,1	F
62	S	22	0,2	G
63	T	23	0,3	G
64	U	24	0,4	G
65	V	25	0,5	G
66	W	26	0,6	G
67	X	27	0,7	G
70	Y	30	0,8	G
71	Z	31	0,9	G
72	]	32	0,8,2	L
73	,	33	0,8,3	M
74	(	34	0,8,4	N
75	→	35	0,8,5	O
76	≡	36	0,8,6	B
77	∧	37	0,8,7	B

Code Character Usage

A Numerics; legal in constants and as second or subsequent character in symbol.

B COSY compression characters. If one of these characters appears on a BCD input card, it will be replaced with the general illegal character and an error message will be printed.

C Delimiter; signals a literal or, in ECHO, signals an actual parameter list.

D Handled as alphabetic characters and therefore may be assigned to alphabetic characters which do not exist in our alphabet.

E General illegal character. 15_8 will be handled as though it were an alphabetic character.

F Delimiters representing addition, subtraction, multiplication and division. * may also represent:

1) this location
2) subprogram bank
3) as part of **, a subfield of one bits
4) in column 1, a comments card

/ may also represent:

1) in BANK, encloses a common block
2) in IFT/IFF, encloses a character string

G Alphabetic characters; legal in an alphanumeric symbol.

H Handled as alphabetic character, or, in DEC/DECD, indicates floating point format.

I Delimiter; used to close bank subfields or parameter lists.

J Delimiter; used to indicate "bank of . . . ".

K Delimiter; in general, used to terminate operation and address fields.

† Collating sequence, ascending order, for symbol table sort and for IFT/IFF comparisons.

†† Code is defined following the table.

Table 7 (Cont'd)

L Record mark; if 72_8 appears on an input BCD card, it will be replaced with the general illegal character, and an error message will be printed.

M Delimiter; used to separate subfields.

N Delimiter; used to open bank subfields, array, and parameter lists.

O Delimiter for use in macro definitions only; used to catenate two elements; illegal as part of a symbol; not a delimiter outside a macro.

INDEX

CONTROL DATA
CORPORATION

COMMENT AND EVALUATION SHEET

3600 Computer Systems
COMPASS Reference Manual

Pub. No. 60052500, C September, 1966

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

CUT ALONG LINE

PRINTED IN U.S.A.

FROM NAME : ______________________________

BUSINESS
ADDRESS : ______________________________

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

FOLD ON DOTTED LINES AND STAPLE

STAPLE STAPLE

FOLD FOLD

FIRST CLASS
PERMIT NO. 8241

MINNEAPOLIS, MINN.

BUSINESS REPLY MAIL

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

POSTAGE WILL BE PAID BY

CONTROL DATA CORPORATION
Software Documentation
4201 North Lexington Avenue
St. Paul, Minnesota 55112

CUT ALONG LINE

D248

FOLD FOLD

STAPLE STAPLE

CONTROL DATA

1/2
3/4
1
1-1/4

►►CUT OUT FOR USE AS LOOSE-LEAF BINDER TITLE TAB

3600 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. 60052500

Litho in U.S.A.