CONTROL DATA

1604/1604-A COMPUTER

1604/1604-A

CODAP-1/REFERENCE MANUAL

CODAP-1/REFERENCE MANUAL

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

CONTENTS

		Page
Chapter 1	INTRODUCTION	1
Chapter 2	MACHINE INSTRUCTION FORMAT	3
	Octal Format	3
	CODAP-1 Format	3
	Location Field Symbols	4
	Operation Code Field	4
	B Field	5
	M-Term Field	5
	Remarks Field	6
Chapter 3	PSEUDO-INSTRUCTIONS	7
	Constants	7
	Data Storage Assignment	10
	Symbol Assignments	15
	Format Control	16
	Monitor Control	17
Chapter 4	PROGRAM PREPARATION	19
	Instruction Pairing	19
	Library Subroutines	22
	Input-Output	22
	Interrupt	24
	Subprogram Format	24
Chapter 5	ASSEMBLY AND EXECUTION	27
	Monitor Input	27
	MCS Control Card	28
	CCS Control Card No. 1	28
	CCS Control Card No. 2	29
	Assembled Listing Format	30
	Error Codes	31
	Binary Object Deck Format	34

LIST OF TABLES

		Page
Table 1	1604 Machine Instructions	35
Table 2	Equivalent Mnemonic Instruction Codes	37
Table 3	Special Codes for FLX or TEL Pseudo-Instructions	38

INTRODUCTION 1

The CODAP-1 Assembly Program converts programs written in CODAP-1 source language into a form suitable for computer processing under the 1604 CO-OP Monitor System. As a part of the Monitor System, the assembly program is stored on the Monitor library tape and operates as a subroutine within the framework of the system.

Source program input to the assembler may be punched cards or card images on magnetic tape. The outputs from the assembler are an assembled listing, and a binary object program on punched cards or on magnetic tape (load-and-go tape). One or more of these outputs may be suppressed as indicated by a control card placed at the beginning of a CODAP-1 source program deck.

Source programs may be subdivided into subprograms, with each subprogram being assembled as a separate entity. Addresses assigned to instructions within a subprogram are relative to the beginning of the subprogram. Absolute addresses are assigned by the relocatable loader of the CO-OP Monitor System when the object program is loaded. Under the CO-OP Monitor System, the programmer has the option of either assembling and executing a program in one Monitor operation (the load-and-go option) or assembling and executing in two independent operations.

CODAP-1 functions as a two pass assembler. During pass one it saves each line of input in memory in a compressed form. If a program exceeds available memory, the remainder of the compressed form of pass one input is output on magnetic tape. After pass one is completed, pass two begins processing the results from pass one which are stored in core. Finally, any information stored on the intermediate tape is read in and processed to complete the assembly.

MACHINE INSTRUCTION FORMAT 2

The format for CODAP-1 symbolic instructions is similar to that for octal machine instructions described in the 1604 Computer Programming Manual. The major difference between the two formats is that symbolic notation may be used in CODAP-1 instructions. Also, two additional fields, a location field and a remarks field, are provided in CODAP-1 instructions. Because punched cards are the standard source medium, the symbolic fields are discussed in terms of columns on a punched card. A list of all 1604 mnemonic instructions is given in Table 1 at the back of this manual.

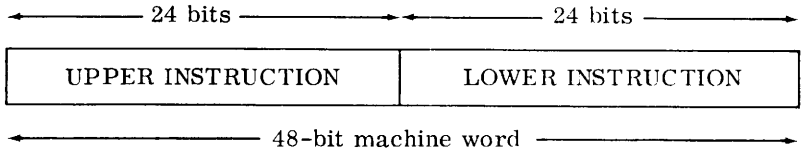
OCTAL FORMAT

A 1604 machine instruction consists of eight octal digits (24 bits) in the following format:

OP-CODE	B-FIELD	M-FIELD
(2 digits)	(1 digit)	(5 digits)

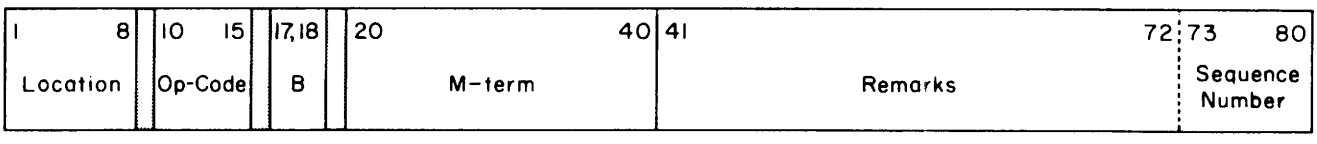
The OP-code is a two-digit octal number that specifies the operation to be performed. The B-field is a one-digit number that specifies either an index register, a stop key, jump key, or a condition to be looked for by the instruction. The M-field is a five-digit octal number that specifies a machine address (direct, indirect or relative), a shift constant or an octal constant to be used as an operand. These fields are described in the 1604 Computer Programming Manual.

Two octal instructions are stored as a pair in one 48-bit 1604 machine word as follows:



CODAP-1 FORMAT

All CODAP-1 instructions, whether symbolic, octal or pseudo-instructions, are written in the following format:



The OP-code, B-field and M-field are the same for both symbolic and octal instructions. The location field identifies (in symbolic notation) the address assigned to a pair of CODAP-1 instructions, or to the first or last location of a block of data. The remarks field is used by the programmer for identification and description. Remarks are not part of the assembled program that is executed; however, they appear in the listing of the assembled program.

Since two instructions are stored in one 1604 machine word, the programmer should know in which half of the machine word a given instruction will be assigned. Detailed rules for instruction pairing are given in Chapter 4.

LOCATION FIELD SYMBOLS

A symbol in the location field (LOCN) is placed left justified, in columns 1-8. Symbols identify the address of a pair of instructions or the first or last address of a block of data. (Data blocks are assigned by pseudo-instructions described in Chapter 3.)

Location field symbols consist of a maximum of eight alphabetic, numeric or special characters such as comma, $ and imbedded blanks. Any combination of the three types may be used as a symbol, however the first character must be non-numeric. Plus and minus signs should not be used as symbols or as special characters within symbols. The * or ** are not to be used as location symbols by themselves.

Acceptable location symbol examples:

A
A1
ABCDEFGH
A1234567
8765
(12)

OPERATION CODE FIELD

The operation code (OPN) field consists of any of the 1604 mnemonic or octal instruction codes†, or any of the pseudo-instructions. The code is placed, left justified, in columns 10-15.

---

†Since the Monitor normally controls all input-output and interrupt operations, the EXF and SEN instructions should not be used in a CODAP-1 program that is to be executed under Monitor control.

The alternate mnemonic codes provided in CODAP-1 for some of the 1604 mnemonic instructions are given in Table 2. For example, the code RTJ (return jump) may be used instead of the code SLJ for certain operations.

B FIELD

The B-field (Columns 17 and 18) contains an octal digit that specifies either an index register modifying the M-term, a stop or jump key, indirect addressing or a condition to be sensed by an instruction. Specific uses of the B-term are explained in the 1604 Programming Manual. Selective stop and jump switches should not be referenced by CODAP-1 programs as the CO-OP Monitor System makes use of these switches during program assembly and execution.

For certain instructions, the B-field may consist of an alphanumeric character instead of an octal digit (see Table 2). Also, the prefix B may be used to identify an octal digit used as an index reference.

M-TERM FIELD

The M-term consists of either a numeric or symbolic term or a combination of the two; it is used as either a direct, relative or indirect storage reference, or as an operand. The M-term is placed in columns 20 through 40. M-term rules for pseudo-instructions are given in Chapter 3.

Address arithmetic is permitted in the M-term in that a numeric value may be added to or subtracted from a symbolic term. Symbols cannot be added to or subtracted from other symbols.

Symbols used as M-terms must appear in the location field of a machine or pseudo-instruction, or in the M-term of an EXT (external symbol) pseudo-instruction. Symbols defined as external symbols cannot be modified by address arithmetic during assembly.

Numeric constants are written and interpreted either as signed decimal or octal numbers. A number is assumed to be decimal unless followed by a B, which indicates an octal number. Decimal numbers may be followed by a D for identification. Decimal numbers cannot exceed ±32,767; octal numbers cannot exceed ±77777.

A single asterisk (*) in the M-term indicates that the M-term is to be assigned the address of the instruction word containing the M-term. A double asterisk (**) in the M-term indicates an M-term of -0 (77777). For example, if the instruction pair

```
IJP 1 *        ENA 0 **
```

is stored in location 1000, the M-term of the IJP instruction is assembled as 01000, and the M-term of the ENA instruction is assembled as 77777.

Relative addressing may be used in a CODAP-1 instruction by placing an asterisk (*) plus or minus a numeric constant in the M-term of the instruction. The M-term *+5 indicates that the address referenced by the instruction is five locations ahead of the address containing the *+5 term.

REMARKS FIELD

The remarks field extends from column 40 through column 80. Remarks may consist of any of the standard key punch characters. When sequence numbers are used, they appear in columns 73 through 80.

PSEUDO-INSTRUCTIONS 3

The following pseudo-instructions provide for conversion of constants, data storage area assignments, input-output control, symbol identification, CO-OP Monitor communication and insertion of remarks. The general format for pseudo-instructions is the same as that for machine instructions given in Chapter 2, except the B-term is blank. Some pseudo-instructions cannot have location symbols, others allow multiple symbols in the M-term. The mnemonic codes placed in the OP-code field define the pseudo-instruction.

CONSTANTS

Octal, decimal, and binary-coded decimal constants may be inserted in a CODAP-1 program by using the pseudo-instructions below. Additional pseudo-instructions allow conversion of constants to Flexowriter and Teletype codes. The M-term may extend beyond column 40 if necessary.

BCD

The BCD pseudo-instruction converts a maximum of 56 characters to standard BCD code and stores them as consecutive 48-bit words within the assembled program. The M-term consists of an octal control digit, n ($1 \leq n \leq 7$), followed by a string of alphanumeric characters, which may be any of the characters on a standard key punch, including spaces. The octal digit, n, specifies the number of consecutive words into which the characters will be stored. A maximum of seven words can be filled by one BCD pseudo-instruction.

If less than 8n characters are written in the M-term, the remaining portion of the n words is filled with blanks (20_8). If more than 8n characters appear in the M-term, only the first 8n characters immediately following the control digit will be converted and stored. The remainder will appear only in the assembled listing.

A symbol may be placed in the location field, which identifies the first word of the n-word block generated by the BCD instruction.

Example

L-term	Op-code	B-term	M-term
CONST	BCD		2ABCD1234EFGH5678

The constant in the M-term is stored in two consecutive words. The first word contains ABCD1234, and the second word contains EFGH5678. The first word of this two-word block is identified by the symbol CONST. If n had been 3 or greater, the remaining words would contain blanks.

DEC

The DEC pseudo-instruction converts decimal constants to equivalent octal values. Each constant may be converted in either fixed or floating point format.

The decimal numbers to be converted are written in the M-term of the DEC instruction in one of the following ways.

1. A sign followed by a maximum of 14 decimal digits and a decimal point. (Plus signs may be omitted.) This results in a floating point constant.

2. A decimal number as described above, written without a decimal point. This results in a fixed point integer constant.

3. A decimal number as described in (1) or (2), followed by a decimal scaling factor, D ± N. D is the decimal identifier and N is the number representing the scale factor. Both the decimal number and the scaling factor are converted to octal before the number is scaled. Only the integral portion of fixed point numbers is retained after scaling. Loss of significance due to scaling is not detected.

4. A decimal number as described in (1), (2), or (3) followed by a binary scaling factor B ± N. N is a decimal number indicating the number of binary places (left or right) the number is to be shifted after being converted to octal. Loss of significance due to scaling is not detected.

The constant must be written, left-justified, starting in column 20. A constant may be scaled by both decimal and binary factors. No spaces may occur within a number, including its associated scale factors, as a space indicates the end of the constant. Plus signs may be omitted.

Examples

L-term	Op Code	B-Term	M-Term	Comments
CONST A	DEC		-12345.	FLOATING PT CONST
CONST B	DEC		+12345	FIXED PT CONST
CONST C	DEC		-12345.D+5	FLOATING PT CONST, DECSCALE
CONST D	DEC		12345D-3	FIXED PT CONST, DECSCALE
CONST E	DEC		+12345B+8	FIXED PT CONST, BINSCALE
CONST F	DEC		+12345.D12B-18	FLOATING PT CONST, DECBIN SCALE

The constants in the example above are converted to 16-digit octal numbers as shown:

Name	Decimal	Octal
CONST A	-12345	57611761
		57777777
CONST B	+12345	00000000
		00030071
CONST C	-12345.D+5	57403315
		30145777
CONST D	-12345.D-3	00000000
		00000014
CONST E	+12345.B+8	00000000
		14034400
CONST F	+12345.D12B-18	20455366
		73311360

More than one constant may be written in the M-term using commas to separate each constant.

Example (3 constants)

```
MULCON    DEC          +1D5,06D+8,115.D-6B9
```

OCT

The OCT pseudo-instruction stores one or more octal constants, consecutively, as written in the M-term. Each constant consists of a sign followed by a maximum of 16 octal digits. Plus signs may be omitted. Constants are set off from each other by commas, and may be written in the M-term beyond column 40. A blank character signals the end of the line of constants.

Example (6 constants)

```
OCTCON    OCT          +1,-57,20,2040,1776,-2
```

FLX

The FLX pseudo-instruction is similar to the BCD pseudo-instruction except that Flexowriter codes are generated for each character in the M-term constant. The special codes for carriage return, tab, backspace, and others, are given in Table 3. The format for the constant in the M-term is an octal control digit punched in column 20 followed by a string of alphanumeric characters. A space signals the end of the string of characters.

If more than 8n characters are contained in the M-term, the excess characters are ignored. If less than 8n characters are contained in the M-term, the constant is left justified within the assigned block in storage and the remainder of the block is filled with space codes.

Up to seven consecutive words will be filled with the Flexowriter constant, as specified by the octal control digit.

The control characters in Table 3 must be preceded and followed by a zero with a minus overpunch.

Example Using carriage return and tab

```
FLEXCON    FLX          1ONE0̄T0̄TWO0̄R0̄
```

One word will be generated containing the Flexowriter codes for the words ONE and TWO. ONE is followed by the tab code 51, and TWO is followed by the carriage return code 45. The four zeros with minus overpunches are not counted as part of the 8-character field.

TEL

TEL pseudo-instruction is the same as FLX pseudo-instruction except that Teletype codes are generated instead of Flexowriter codes. Table 3 contains the special control characters such as carriage return and tab.

DATA STORAGE ASSIGNMENT

The following pseudo-instructions reserve storage areas for blocks of data. BSS and BES reserve storage blocks within the subprogram with which they are assembled. If these storage areas are to be referenced by other subprograms, the name assigned to the block is declared as an entry point in the program containing the block, and as an external symbol in the program referencing the block. The ENTRY and EXT pseudo-instructions are discussed under Symbol Assignments. BLOCK and COMMON reserve storage areas for data arrays that are to be referenced by more than one subprogram. EXT and ENTRY are not needed to identify data names specified as COMMON.

BSS

The BSS pseudo-instruction reserves a specific storage area within a subprogram. The M-term contains a positive numeric value, either octal or decimal, that specifies the length of the reserved area. The address of the first word of this reserved area is identified by a symbol in the location field. Other words in the storage area are referenced by address arithmetic, using the location field symbol as the first term, or by indexing.

Example

A block of 64 words is to be reserved within a subprogram. The pseudo-instruction would be written as

```
STORE    BSS          64
```

If the instruction immediately preceding STORE was assigned to location 01477_8, the 64-word storage area STORE would be assigned starting at location 01500_8. The next instruction pair following the BSS would be assigned to location 01600_8. If the preceding instruction was an upper instruction, the lower instruction of that word would be set to ENI 0.

BES

The BES pseudo-instruction is the same as BSS, except that the symbol in the location field identifies the last word of the block, rather than the first.

BLOCK AND COMMON

A general method of reserving storage areas that can be referenced by more than one subprogram is by use of the BLOCK and COMMON pseudo-instructions. BLOCK reserves a common storage area, as specified in the M-term. This storage area may be at the beginning of the area assigned to the subprogram containing BLOCK, or it may be assigned to another part of the computer core storage. The type of location symbol in the BLOCK pseudo-instruction determines which of these two areas is to be used. If the location symbol is alphanumeric, the storage area is reserved at the beginning of the subprogram. This area is termed "labeled common". If the location symbol is a numeric value or blank, the storage area is assigned a separate portion of core storage termed "numbered common".

The main difference between the two types of common storage is that constants can be assembled into a labeled common area, but not into a numbered common area. The location symbol of BLOCK pseudo-instructions is never referenced by any instruction other than another BLOCK. To describe data arrays within an area assigned by BLOCK, the COMMON pseudo-instructions are used. These two pseudo-instructions are always placed at the beginning of a subprogram. To reference a block of common storage reserved in another subprogram, the location field symbol of the BLOCK pseudo-instruction in each subprogram must be identical. Also, the length of these two blocks must be the same.

BLOCK

The BLOCK pseudo-instruction reserves areas of labeled or numbered common. If the location symbol is alphanumeric, starting with a non-numeric character in column 1, the reserved area will be in labeled common. If the symbol starting in column 1 is numeric or blank, the reserved area will be in numbered common. Numeric symbols can contain only digits with the first digit in column 1.

Labeled common symbols

ABCDE

A1234

Numbered common symbol

12345

Illegal symbol

123A5

The M-term of BLOCK can be either blank or a numeric value. A blank M-term indicates that the size of the reserved area is equal to the sum of the sizes of the data arrays specified by the COMMON pseudo-instructions immediately following BLOCK. A numeric value in the M-term specifies the length of the reserved area which should be equal to or greater than the sum of the associated data arrays.

COMMON

The COMMON pseudo-instruction is used to identify one or more arrays of data in labeled or numbered common. The M-term contains the names and sizes of the arrays. Each array may have from one to three dimensions. Each array name may be any legitimate M-term symbol as described for the symbolic machine instructions. The dimensions of each array are specified within parentheses immediately following the array name. Each dimension is written as a decimal integer; dimension integers are set off from each other by commas. If more than one array is specified in the M-term, the arrays are separated by placing a comma after the parentheses enclosing the dimensions of each array. The array names and dimensions may extend to column 72, however no blanks may occur within the string or at the beginning, as the first blank column indicates the end of the M-term. A location symbol in a COMMON pseudo-instruction is illegal. An example using three arrays of one, two and three dimensions follows:

```
        COMMON      A(5),B(5,10),C(5,10,20)
```

Array A is assigned a 5-word block, array B is assigned a 50-word block and array C is assigned a 1000-word block.

The use of BLOCK and COMMON is illustrated by examples which demonstrate ways of assigning data arrays in numbered and labeled common storage areas.

Example 1

Assign three arrays, ALPHA, BETA, and GAMMA, to labeled common storage.

```
ARRAYS      BLOCK       300
            COMMON      ALPHA(100)
            COMMON      BETA(10,10)
            COMMON      GAMMA(10,5,2)
```

A 300-word block ARRAYS will be reserved at the beginning of the subprogram containing the BLOCK and COMMON instructions. The block size would have been established as the sum of the array sizes if the M-term 300 had been omitted. Each of the three arrays is 100 words long, and stored sequentially starting with ALPHA. CODAP-1 does not provide for subscripted variables, therefore the programmer should use indexing to select a word within an array.

Example 2

Reference the array GAMMA of Example 1 in the next subprogram. The coding for BLOCK and COMMON should appear in the second subprogram as follows:

```
ARRAYS      BLOCK
            COMMON      A(100)
            COMMON      B(100)
            COMMON      GAMMA(100)
```

Within the original block ARRAYS in Example 1, GAMMA is assigned locations 201-300.

To reference array GAMMA in the subprogram of this example, a BLOCK pseudo-instruction with the same name as the original block is placed at the beginning of the second subprogram. (The M-term is blank, but could be 300.) Following BLOCK are one or more COMMON instructions that indicate the position of the desired array within the block. Array names are of no consequence in communication between subprograms: only array lengths are important. In this example the COMMON statement for array GAMMA is preceded by two COMMON statements, the lengths of which indicate that GAMMA starts at word 201 within ARRAYS. These two statements could be replaced by one COMMON instruction having an array length of 200. Since array names are not communicated between subprograms, any name could have been used in the second subprogram in place of GAMMA. The coding could have been written as follows:

```
ARRAYS      BLOCK
            COMMON      DELTA(200)
            COMMON      ZETA(100)
```

Now array ZETA occupies the same storage area as array GAMMA in the first subprogram block ARRAYS. Arrays ALPHA and BETA comprise the larger array DELTA of the second subprogram.

To demonstrate that more than one array can be written in the M-term of COMMON, these statements can be rewritten as follows:

```
ARRAYS      BLOCK
            COMMON      DELTA(200),ZETA(100)
```

The element that identifies each group of arrays as belonging to the same group is the name in the location field of BLOCK. Unless this name is repeated each time, separate array blocks will be reserved. In these two examples, the name indicates labeled common storage. If numbered common had been specified (by a numeric or blank location symbol) the same rules would apply.

Example 3

Insert constants in a labeled common area by use of the assembly program. The constants can be any of those allowed by CODAP-1.

```
CONST     BLOCK     500
          COMMON    A(6)
            .
            .
            .
            .       (program instructions)
            .
            .
            .
            .
          ORGR      A
          DEC       1234
          DEC       5678
          BCD       1ABCD
          BCD       1EFGH
          OCT       4321
          OCT       7654
          ORGR      *
            .
            .       (program instructions)
            .
```

The block CONST will be assigned 500 words of storage in labeled common. The first six locations in this block are occupied by the array A. By use of the ORGR pseudo-instruction, the six constants following ORGR A are inserted (preset) in array A when the program is loaded. The ORGR * returns the CODAP-1 program counter to the next sequential address in the program instruction sequence, as if the six constants had not intervened.

The block must be designated as labeled common, and the name of the array to be preset is placed in the M-term of the first ORGR pseudo-instruction. The constants to be inserted in the array immediately follow the first ORGR. The last constant is immediately followed by ORGR *.

The following rules apply to BLOCK and COMMON.

1. Symbols are left-justified in the location field of BLOCK. Blank location fields for BLOCK should be avoided in CODAP-1 programs.

2. The M-term of BLOCK is blank or it contains a number that is equal to or greater than the sum of the length of all arrays within the block.

3. All COMMON pseudo-instructions associated with a BLOCK immediately follow the BLOCK pseudo-instruction.

4. All BLOCK and COMMON pseudo-instructions for a subprogram must appear at the beginning of the program after the IDENT instruction and before the first program instruction.

SYMBOL ASSIGNMENTS

The three pseudo-instructions, EQU, ENTRY, and EXT, define symbols as equal to other symbols, or identify symbols that are defined in other subprograms. Linkage between the same symbol in separate subprograms is provided by the CO-OP Monitor System.

EQU

The EQU pseudo-instruction assigns the address of the symbol in the location field of EQU to the address of the symbol in the M-term, or to a numeric constant in the M-term. An M-term symbol must be used as a location field symbol elsewhere in the same program or subprogram.

Example

```
OUT             EQU             JUMP
```

If JUMP is assembled to address 00100, OUT will also be assigned address 00100.

Numeric constants must follow the rules for symbolic instructions. Only one symbol or constant is allowed in the M-term; however, address arithmetic is permitted (Symbol ± constant).

ENTRY

Symbols to be referenced by other subprograms must be declared as entry points within the subprogram that defines these symbols. (A symbol is defined when it appears in the location field of a machine or pseudo-instruction.) Symbols are declared as entry points by placing them in the M-term of one or more ENTRY pseudo-instructions. Two or more symbols in the M-term are separated by commas. No spaces (blanks) can appear within a string

of symbols, as a space indicates the end of the string. The M-term of the ENTRY pseudo-instruction may be extended out to column 72. The location field must be blank.

Example

```
    ENTRY          SYM1,SYM2,SYM3
```

SYM1, SYM2, and SYM3 can now be referenced by other subprograms.

EXT

Symbols used by a subprogram which are defined in another subprogram are declared external symbols by placing them in the M-term of one or more EXT pseudo-instructions contained in the user subprogram. For example, to use the symbols SYM1, SYM2 and SYM3 declared as entry symbols in another subprogram, the pseudo-instruction would be written as:

```
    EXT            SYM1,SYM2,SYM3
```

The M-term may be extended to column 72; symbols are separated from each other by commas. No spaces (blanks) can appear in a string of symbols, as a space indicates the end of the string. The location field of an EXT must be blank. If a symbol in an EXT M-term is the name of a CO-OP library subroutine, the subroutine will be loaded with the object program containing the EXT. Address arithmetic cannot be performed on external symbols.

LIB

The LIB pseudo-instruction acts as an EXT instruction, except that only one symbol may be placed in the M-term. The location field of LIB may contain a symbol, thereby allowing renaming of an external symbol. Symbols defined by LIB cannot be modified by address arithmetic.

FORMAT CONTROL

The pseudo-instructions which provide format control for assembled listings are shown below. The pseudo-instruction codes do not appear on the assembled listing.

SPACES

The line spacing on an assembled listing can be controlled by the SPACES pseudo-instruction. A decimal constant in the M-term designates the number of spaces to be skipped before printing the next line. If the number of spaces to be skipped is greater than the number of lines remaining to be printed on a page, the line printer skips to the top of the next page. The next machine instruction following the SPACES pseudo-instruction is assembled as an upper instruction. A symbol in the location field is ignored.

EJECT

The EJECT pseudo-instruction causes the line printer to skip to the top of the next page when the assembled program is listed. The next machine instruction after EJECT is assembled as an upper instruction. A symbol in the location field is ignored.

REM

The REM pseudo-instruction is used to insert program comments in an assembled listing. The M-term can be extended to column 72. Any standard key punch character can be used in the comments. If the comments are to be written on more than one line, successive REM pseudo-instructions may be used as necessary. The next machine instruction following REM is assembled as an upper instruction. A symbol in the location field is ignored.

MONITOR CONTROL

The following pseudo-instructions provide communication between CODAP-1 programs and subprograms and the Monitor. Some are required in every program and subprogram; others are optional.

IDENT

An IDENT pseudo-instruction appears at the beginning of every CODAP-1 program and subprogram. The M-term contains the name of the program or subprogram, which can be a maximum of eight alphanumeric characters, the first being alphabetic. Spaces within a name are ignored and are not counted as characters. A symbol in the location field is illegal.

I/O

The I/O pseudo-instruction is used to check that the logical input-output unit numbers in the M-term are defined by the MCS control card required by the Monitor. (See the CO-OP Monitor Programmer's Guide.) The unit numbers in the M-term are separated by commas, and the string of numbers is terminated by a blank. The M-term may be extended to column 72. A location field symbol is illegal.

ORG

Location of a program or subprogram may start at an absolute machine address by using the ORG pseudo-instruction. The M-term contains either a decimal or octal machine address. An octal address is followed by a B. ORG should be used only for special applications. Programs containing ORG pseudo-instructions cannot be loaded by the Monitor.

ORGR

This pseudo-instruction signals the assembler that the instructions following ORGR are to be assembled as relocatable, starting at the address (decimal or octal) given in the M-term. Since programs are normally assembled as relocatable, the principal use of ORGR is to assemble constants into a block of labeled common storage. The labeled common storage area must be defined by a COMMON pseudo-instruction at the beginning of the subprogram. The set of BCD, OCT, DEC, FLX or TEL constants are immediately preceded by an ORGR pseudo-instruction containing the name (without dimension) of the array in the labeled common area to be filled. This sets the assembly program counter to the first address of the labeled common area. Another ORGR is placed immediately after the last constant to be stored in labeled common. The second ORGR is written with an * in the M-term, which causes the assembly program counter to be set back to the address it contained before the first ORGR was encountered. The first machine instruction following an ORGR * is assembled as an upper instruction. See the COMMON pseudo-instruction for details of labeled common.

END

The end of a program or a subprogram is indicated by an END pseudo-instruction. The M-term identifies the starting address of the program when one or more subprograms are loaded together as one program. One of the subprogram END cards contains the symbol appearing in the location term of the first instruction to be executed. Only one END card can contain an M-term. A location term for an END pseudo-instruction is ignored.

The Monitor requires that a second END card follow the END card of the last subprogram in a program deck. If this card is missing, the program will not execute properly.

FINIS

The FINIS pseudo-instruction terminates an assembly operation. It is a signal to the Monitor that no more programs are to be assembled. The FINIS card is placed after the second END card of the last program in a symbolic input card deck. See Chapter 5 for symbolic card deck composition.

PROGRAM PREPARATION 4

This chapter is to be used as a guide in coding CODAP-1 programs. Rules are given for instruction pairing, library subroutine references, input-output and the program format required by CODAP-1.

INSTRUCTION PAIRING

As specified in the 1604 programming manual, certain 1604 instructions are required to be upper instructions; other instructions must be lower instructions. The rules listed below govern instruction pairing in CODAP-1.

1. The normal sequence of instruction pairing by the assembly program is as follows. Starting with the first two machine instructions in a subprogram, each pair is assigned as one machine word in 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 one of the conditions described in paragraphs 2 through 6 is encountered. Pseudo-instructions (except as described in rule 4) do not affect the sequence of instruction pairing.

The following instructions:			are assembled as:	
IDENT	PROCESS		IDENT	PROCESS
ENTRY	START		ENTRY	START
EXT	OPEN		EXT	OPEN
REM	PARTA		REM	PARTA
SLJ	**	L	SLJ	**
ENA	POINTER		ENA	POINTER
SAL	TRANSFER	L+1	SAL	TRANSFER
RTJ	TRANSFER		RTJ	TRANSFER
NOP		L+2	NOP	
ENA	PROCA		ENA	PROCA
REM	PARTB		REM	PARTB
SAL	TRANSFER	L+3	SAL	TRANSFER
RTJ	TRANSFER		RTJ	TRANSFER

The first four lines are pseudo-instructions and are not assigned to machine locations. The instruction SLJ ** is assigned as the upper instruction in location L, and ENA POINTER is assigned as the lower instruction in location L. The remaining machine instructions are assembled in sequence (upper, lower, upper, lower). The pseudo-instruction REM after ENA PROCA does not affect the assembly sequence. The NOP instruction is equivalent to the machine instruction ENI 0.

2. A symbolic instruction that contains a symbol in the location field is assembled as an upper instruction. If the previous instruction was also assembled as an upper instruction, a pass (ENI 0) instruction is generated by the assembly program and placed in the lower half of the previous word.

   The following consecutive instructions:

```
        A    LDA    1    ALPHA
        B    STA    2    BETA
        C    LDQ    3    GAMMA
             STQ    4    DELTA
```

   are assembled as:

```
L       A    LDA    1    ALPHA    (upper)
             ENI    0             (lower)
L+1     B    STA    2    BETA     (upper)
             ENI    0             (lower)
L+2     C    LDQ    3    GAMMA    (upper)
             STQ    4    DELTA    (lower)
```

   Instruction A is assembled as the upper instruction in address L, instruction B is assembled as the upper instruction in address L+1, and the instruction C is assembled as the upper instruction address L+2. The ENI 0 instructions are generated by CODAP-1.

3. A + character in column 1 indicates that the associated symbolic instruction is to be assembled as an upper instruction. ENI 0 instructions are generated as explained in rule 1.

   The following instructions:

```
        +    LDA    1    ALPHA
        +    STA    2    BETA
        +    LDQ    3    GAMMA
```

   are assembled the same as those in rule 1 but without location field symbols.

4. A - (minus) character in column 1 forces an instruction to the lower half of a word.

   The following instructions:

```
+    ADD    0    ECHO
     SUB    4    FOXTROT
-    MUI    5    HOTEL
-    DVI    6    KILO
```

   are assembled as:

```
L    ADD    0    ECHO
     SUB    4    FOXTROT
L+1  ENI    0
     MUI    5    HOTEL
L+2  ENI    0
     DVI    6    KILO
```

   If a symbol is placed after a minus sign in column 1, the minus sign takes precedence and the symbol is ignored.

5. The following mnemonic instructions are normally assembled as upper instructions, unless preceded by a minus sign in column 1: EQS, THS, MTH, MEQ, ISK, SSH, SSK, EXF7 and SEN† .

   The following instructions:

```
     EQS    0    FIVE
     THS    0    SIX
     MEQ    1    SEVEN
```

   are assembled as:

```
L    EQS    0    FIVE
     ENI    0
L+1  THS    0    SIX
     ENI    0
L+2  MEQ    1    SEVEN
```

†EXF7 and SEN should not be used in CODAP-1 programs, as the Monitor provides these functions.

The next instruction in sequence would be assembled as a lower instruction unless otherwise indicated.

6. The pseudo-instructions that reserve data storage areas within a subprogram, (DEC, OCT, BCD, FLX, TEL, BSS, BES) force the next instruction to be an upper instruction of the next word in sequence. If the preceding instruction had been an upper instruction, an ENI 0 would be inserted in the lower half of that word.

   The following instructions:

```
        LOAD    LDA     0    DATA+5
        DATA    BSS          10
                STA     0    TEMP
```

   are assembled as:

```
L       LOAD    LDA     0    DATA+5
                ENI     0
L+1     DATA    BSS          10
L+11            STA     0    TEMP
```

   The LDA instruction has a symbol in the location field, therefore it is assembled as an upper instruction. Since the next instruction is a BSS, the assembly program inserts an ENI 0 in the lower half of L. The BSS reserves ten locations starting at L+1, and the STA is the upper instruction at L+11.

LIBRARY SUBROUTINES

The library subroutines are described in the CO-OP Monitor Library Subroutine Manual. To load a subroutine with a CODAP-1 object program, the name of the subroutine is placed in the M-term of an EXT or LIB pseudo-instruction. This same method calls routines that are part of the CO-OP Monitor itself.

INPUT-OUTPUT

The 1604 machine instructions for input-output are not to be used in CODAP-1 programs, as all input-output is controlled by two subroutines in the CO-OP Monitor. READ* and WRITE* may be used for programmed input-output on

any of the 1604 on-line peripheral equipment. The calling sequence format is the same for both subroutines, as follows:

```
        EXT         READ*(or WRITE*)
        ENQ         Y
L       RTJ   0     READ*(or WRITE*)
        rm    0     n (logical unit number)
L+1     fc    0     A (FWA)
        i     0     B (LWA+1)
L+2     SLJ   0     C (alternate return)
                    e (error flag)
L+3     SLJ   0     D (normal return)
```

The definitions of the codes are given below:

rm An octal digit designating the recording mode (binary, coded, BCD, Hollerith).†

fc An octal number designating the function to be performed by the specified peripheral device.†

i A single digit, 0 or 1, indicating whether or not to recognize an interrupt condition such as the termination of a buffer operation or sensing equipment ready. If the digit is 0, the interrupt condition is ignored. If the digit is 1, the interrupt is recognized and an exit is made to the starting address of the interrupt subroutine supplied by the programmer. This address is stored in the Q register.

Y Symbolic address of the start of the interrupt subroutine supplied by the programmer.

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

A Symbolic first word address of input or output area.

B Last word address + 1 of input or output area.

C Symbolic address of alternate return to be taken when an error is sensed during a read or write operation.

e Error flag set by the READ* or WRITE* subroutine to indicate an error or abnormal condition (such as parity error or end-of-file) that occurred during operation.†

D Symbolic address of normal return to be taken by READ* or WRITE*.

†See CO-OP Monitor Programmer's Guide for numeric codes and error codes.

INTERRUPT

The internal and external interrupts for the 1604 are selected, sensed and removed by the CO-OP Monitor subroutines SELECT*, DETECT* and REMOVE*. Calling sequences and subroutine descriptions are given in the CO-OP Monitor Programmer's Guide.

SUBPROGRAM FORMAT

The positioning of certain of the pseudo-instructions within a subprogram is extremely important. The first instruction in a subprogram is the IDENT pseudo-instruction, which has the name of the subprogram in the M-term. Following IDENT are all BLOCK and COMMON pseudo-instructions. After the last of these is placed the remainder of the coding for the subprogram.

The last instruction in any subprogram, the END pseudo-instruction, may have a symbolic transfer address in the M-term identifying the starting address of the main program of which the subprogram is a part. This symbol must appear in the location field of the first machine instruction in the program to be executed, and it must be declared as an entry point by placing the symbol in the M-term of an ENTRY pseudo-instruction. If a single subprogram is to be executed as a main program, the END pseudo-instruction for this program must have the symbolic starting address in the M-term. For programs consisting of more than one subprogram, the symbolic transfer address may appear in any one of the END M-terms. This symbol is declared an entry point symbol in the subprogram containing the starting address.

Because the Monitor loader program executes a return jump to the location declared as the starting address, when the program is loaded and executed, the upper instruction of this address should be an SLJ 0 **. The return jump operation will replace the ** with the return address to the Monitor loader. This allows the programmer to exit back to the Monitor by a jump to the starting address. This jump would normally be the last executable instruction in the program. When the jump at the end of the program is executed, processing of this program is terminated and control is returned to the Monitor system.

Saving of Index Registers

The contents of all index registers used in CODAP-1 programs should be saved at the beginning of the program and restored at the end of the program. This allows the CODAP-1 program to be used as a subroutine in another program without destroying index register settings in the other program.

Coding Example

The following example illustrates the order of instructions in a CODAP-1 subprogram.

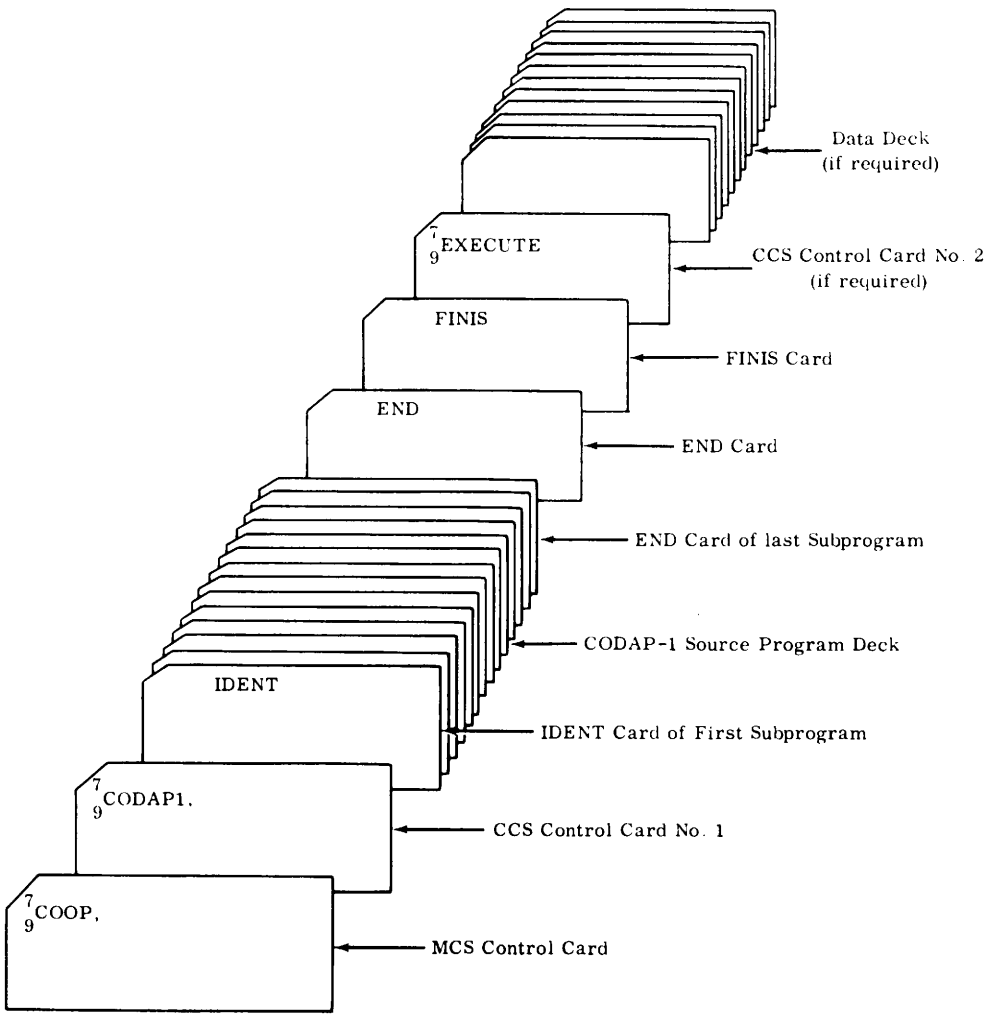
SAMPLE PROGRAM

```
LOCN      OPN       B    OPERAND OR M-TERM                    REMARKS
          10        17   20                                   41

          IDENT          SQUARES
ARRAY     BLØCK
          CØMMØN         INTEGER(101),SQUARE(100),BUFF(1500)
          ENTRY          START
          EXT            BINBCD,PRINT

START     SLJ            **
          SIU       1    REG1                                 SAVE INDEX
          ENI       1    0
          ENA            1
          STA            INTEGER
MULT      LDA       1    INTEGER
          MUI       1    INTEGER
          STA       1    SQUARE
+         LDA       1    INTEGER
          INA            1
          STA       1    INTEGER +1
+         ISK       1    99
          SLJ            MULT
          RTJ            BINBCD
+         RTJ            PRINT
REG1      ENI       1    **                                   RESTØRE INDEX
          SLJ            START
          END

          IDENT          PRINT
ARRAY     BLØCK
          CØMMØN         A(201),BUFF(1500)
          ENTRY          PRINT
          EXT            WRITE*,START
PRINT     SLJ            **
          SIU       5    REG1                                 SAVE INDEX
          SIL       4    REG1
          ENI       5    0
          ENI       4    0
NEXT      ENA       4    BUFF
          SAU            TAG
          ENA       4    BUFF+15
          SAL            TAG
          ENQ            **
+         RTJ            WRITE*
          1              51
TAG       3              **
          0              **
          RTJ            20B
          ZRØ            0
          INI       4    15
          ISK       5    99
          SLJ            NEXT
REG1      ENI       5    **                                   RESTØRE INDEX
          ENI       4    **
          SLJ            PRINT
          END            START
          END
          FINIS
```

ASSEMBLY AND EXECUTION 5

MONITOR INPUT

To assemble and execute a CODAP-1 program under control of the CO-OP Monitor, the control cards shown below must be placed at the beginning and end of each CODAP-1 program deck.



The formats for the MCS and CCS control cards are described in detail in the CO-OP Monitor Programmer's Guide. A brief description is given below.

MCS CONTROL CARD

The MCS Control Card is placed at the beginning of a CODAP-1 program deck to be assembled. The general form of this card is as follows, starting in column 1 from left to right:

$^{7}_{9}$COOP, Accounting Number, Programmer's Name or Initials, Input-Output Assignments, Time Limit, Printer Line Count, Recovery Key, Comments.

A comma terminates each field except the last, which is terminated by a period.

Example

$^{7}_{9}$COOP,31003-00,ABC,I/1/O/2/S/3,10,1000,4,ASSEMBLE AND EXECUTE.

The $^{7}_{9}$COOP identifies the control system to be used, 31003-00 is the accounting number and ABC are the programmer's initials. I/1 assigns logical unit 1 as an input device, O/2 assigns logical unit 2 as an output device, and S/3 assigns logical unit 3 as a scratch unit. (A scratch unit may be used as either an input or output device.) The 10 following S/3 indicates a time limit of ten minutes, the number 1000 indicates the maximum number of lines to be printed, and 4 indicates that recovery key 4 is to be used during execution. The time limit is the maximum time allowed for assembling and executing the program. The remainder of the card may contain program comments. Omitted fields are indicated by one or more commas. For example, to omit the input-output assignment, the example above would be written as follows:

$^{7}_{9}$COOP,31003-00,ABC,,10,1000,4,ASSEMBLE AND EXECUTE.

The $^{7}_{9}$COOP, the accounting number and the programmer's name or initials must be on the MCS card. The line may be terminated by a period at the end of the name or any subsequent field. The example below contains the minimum number of fields.

$^{7}_{9}$COOP,31003-00,ABC.

CCS CONTROL CARD NO. 1

The first CCS control card identifies the subordinate control system to be used in processing the program deck. For CODAP-1 assemblies the subordinate control system is CODAP-1. The format for this card is as follows, starting in column 1:

$^{7}_{9}$CODAP1, Listing Key, Binary Object Card Key, Load-and-Go Tape Unit Number.

$^{7}_{9}$CODAP1 indicates that the CODAP-1 assembly program is to be loaded and executed. The listing key is 1 if an assembled listing is to be printed, otherwise it is 0 or blank. The binary card key is 1 if binary cards are to be punched, 0 or blank if no cards are to be punched. The last field is the logical unit number of the tape unit to be used to write a load-and-go tape.† Each field except the last is terminated by a comma; the last field is terminated by a period.

To write a load-and-go tape on logical unit 3, and to obtain binary cards and an assembled listing, the control card would be punched as follows:

$^{7}_{9}$CODAP1,1,1,3.

To obtain only an assembled listing, the card would be punched:

$^{7}_{9}$CODAP1,1.

To obtain an assembled listing and a load-and-go tape on logical unit 3, the card would be punched:

$^{7}_{9}$CODAP1,1,0,3.

The combination $^{7}_{9}$CODAP1,0,1. would result in a binary deck only.

CCS CONTROL CARD NO. 2

If, after the program has been assembled, it is to be immediately loaded and executed from a load-and-go tape, a second CCS control card is placed after the FINIS card. The format of this card is as follows, starting in column 1:

$^{7}_{9}$EXECUTE,Time Limit, Logical Unit Number of Load-and-Go Tape,
Memory Map Key.

The value in the time limit field indicates the maximum number of minutes the Monitor will allow for execution time. For load-and-go assembly operations, the time limit may be omitted, as the time allowed for execution is determined by the Monitor. The time taken by the actual assembly is subtracted from the time limit on the MCS card to determine the execution time. The logical unit field contains the number of the logical tape unit specified as the load-and-go unit on the first CCS control card.

The memory map key indicates whether or not the Monitor will list the absolute address of all external symbols and program names referenced by the program; 0 indicates the listing is to be produced, 1 indicates no listing.

---

†A load-and-go tape contains the binary card images of the object program generated by the assembler. This allows loading and executing of the object program from the load-and-go tape under Monitor control.

Example

$^{7}_{9}$EXECUTE,5,3,1.

will cause the Monitor to load and execute a program from logical tape unit 3, setting a time limit of 5 minutes. No memory map will be produced.

$^{7}_{9}$EXECUTE,,5.

will execute the program from logical tape unit 5. The time allowed will be the time specified on the MCS Control Record card less the assembly time. A memory map will be produced.

If data is to be loaded from the standard input unit,† the data deck should follow the EXECUTE card as shown. If a program is to be assembled but not executed, the EXECUTE card is omitted from the program deck.

ASSEMBLED LISTING FORMAT

An assembled listing of the program in Chapter 4 is shown at the end of this Chapter. The addresses assigned to each subprogram are relative addresses only. Absolute addresses are assigned when the program is loaded by the Monitor load routine. All COMMON blocks are assigned consecutively, starting at relative location 00000. The first machine instruction is assigned to the first word following the last COMMON block. The range of locations assigned to the machine instructions (first word address and last word address plus one) are given at the beginning of each subprogram. Following this is a list of all entry points and external symbols, and the address assignments for all BLOCK and COMMON pseudo-instructions. (The second and following symbols in a string of M-term symbols are repeated individually on separate lines.)

External symbols are assigned identification numbers by the assembler. The Monitor loader assigns the proper absolute addresses.

The address of each instruction word is the leftmost field for each instruction in the assembled listing. (Error codes would appear to the left of this field.) The + after an address indicates that the address is relative and will be modified by the Monitor loader. The next three fields, from left to right, are the octal operation code, B-term and M-term. Relative addresses are indicated by a + to the right of the octal address. External M-term symbols are indicated by an X immediately to the left of the octal M-term. This value corresponds to a number in the list of external symbols at the beginning of the subprogram. The remaining fields correspond to those in the symbolic source program.

At the end of each subprogram is a list of symbols that may have been used incorrectly. Doubly defined symbols are those that appear two or more times in a location field or in the M-term of a COMMON, EXT or LIB pseudo-instruction. Undefined symbols are those in the M-term of a machine instruction

†The standard input unit is that which is normally used by the Monitor for loading all jobs.

which are not defined in any location field or in a COMMON, EXT or LIB pseudo-instruction. Nulls are those symbols that are defined but not used in any machine instruction M-term.

ERROR CODES

The following error codes appear as the leftmost field on an assembled listing.

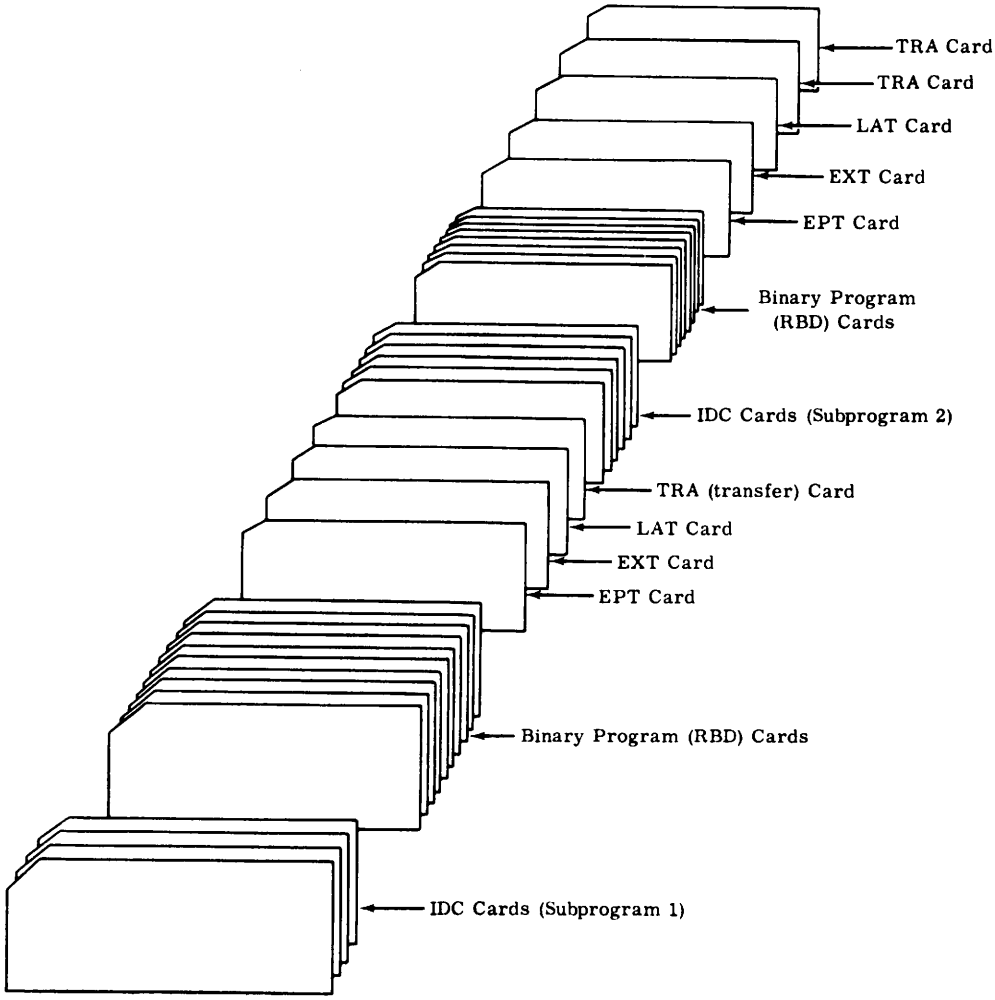
Code	Explanation
B	Illegal character in the B-term. B-term follows a NOP instruction which is not blank or zero. B-term symbol appears where none is required. Substitute a zero.
C	Attempt to assemble too much data in a labeled common block. Assembler continues processing and listing source input but no binary output is produced until the C error flag is cleared. (The C error flag is cleared when the assembler encounters the next ORGR or ORGR * instruction.)
D	Same symbol used in more than one location field term. Only the first symbol is recognized; the remainder are ignored. A list of doubly defined symbols appears on the assembled listing.
F	Symbol table is full. No more location field symbols will be recognized.
I	IDENT pseudo-instruction occurs in position other than as the first card in a subprogram deck. Misplaced IDENT cards are ignored.
L	LOCN term is missing from an EQU pseudo-instruction. LOCN term is present where none is permitted. A + or - character for forcing upper or lower appears in the LOCN term of an OCT, DEC, BCD, FLX, TEL, BSS, or BES pseudo-instruction. No location symbol assignment is made in such cases. Location field symbol of a BLOCK pseudo-instruction is not left-justified. Leading character of location field symbol for BLOCK is numeric, followed by non-numeric characters.
O	Illegal operation code. Zeros are substituted for the operation code
U	Undefined symbol. The assembler assigns the symbol to a region following the last program entry. A list of undefined symbols will appear on the output listing.

```
BEGIN JOB 091 05 08 63
COOP,31003≠00,ABC,1000,5.
CODAP1,1.

                                        IDENT     SQUARES
RANGE                FWA   -   LWA+1
                    03245       03260
ENTRY POINTS
                    03245       START
EXTERNAL SYMBOLS
                    00001      BINBCD
                    00002       PRINT
  00000+                      ARRAY     BLOCK
  00000+                                COMMON    INTEGER(101),SQUARE(100),BUFF(1500)
  00145+                                          SQUARE(100)
  00311+                                          BUFF(1500)
                    03245+                ENTRY     START
  00001                                 EXT       BINBCD,PRINT
  00002                                           PRINT
  03245+   75  0  77777  START          SLJ       **
           56  1  03257+               SIU     1 REG1                  SAVE INDEX
  03246+   50  1  00000                 ENI     1 0
           10  0  00001                 ENA       1
  03247+   20  0  00000+                STA       INTEGER
           50  0  00000
  03250+   12  1  00000+ MULT           LDA     1 INTEGER
           24  1  00000+               MUI     1 INTEGER
  03251+   20  1  00145+               STA     1 SQUARE
           50  0  00000
  03252+   12  1  00000+ +             LDA     1 INTEGER
           11  0  00001                INA       1
  03253+   20  1  00001+               STA     1 INTEGER+1
           50  0  00000
  03254+   54  1  00143  +             ISK     1 99
           75  0  03250+               SLJ       MULT
  03255+   75  4 X00001                RTJ       BINBCD
           50  0  00000
  03256+   75  4 X00002  +             RTJ       PRINT
           50  0  00000
  03257+   50  1  77777  REG1          ENI     1 **                    RESTORE INDEX
           75  0  03245+               SLJ       START
                                       END

     NO DOUBLY DEFINED
     NO UNDEFINED SYMBOLS
     NO ERRORS
        NULLS                             BUFF
```

```
                                                IDENT     PRINT
RANGE                    FWA   -   LWA+1
                        03245       03262
ENTRY POINTS
                        03245       PRINT
EXTERNAL SYMBOLS
                        00001     WRITE*
  00000+                        ARRAY     BLOCK
  00000+                                  COMMON    A(201),BUFF(1500)
  00311+                                            BUFF(1500)
                        03245+              ENTRY     PRINT
  00001                                     EXT       WRITE*
  03245+   75   0   77777   PRINT          SLJ       **
           56   5   03260+                 SIU     5 REG1                  SAVE INDEX
  03246+   57   4   03260+                 SIL     4 REG1
           50   5   00000                 ENI     5 0
  03247+   50   4   00000                 ENI     4 0
           50   0   00000
  03250+   10   4   00311+  NEXT           ENA     4 BUFF
           60   0   03254+                 SAU       TAG
  03251+   10   4   00330+                 ENA     4 BUFF+15
           61   0   03254+                 SAL       TAG
  03252+   04   0   77777                 ENQ       **
           50   0   00000
  03253+   75   4  X00001   +              RTJ       WRITE*
           01   0   00063                 1         51
  03254+   03   0   77777   TAG            3         **
           00   0   77777                 0         **
  03255+   75   4   00020                 RTJ       20B
           00   0   00000                 ZRO       0
  03256+   51   4   00017                 INI     4 15
           50   0   00000
  03257+   54   5   00143   +              ISK     5 99
           75   0   03250+                 SLJ       NEXT
  03260+   50   5   77777   REG1           ENI     5 **                    RESTORE INDEX
           50   4   77777                 ENI     4 **
  03261+   75   0   03245+                 SLJ       PRINT
           50   0   00000
                    00000                 END       START
     NO DOUBLY DEFINED
     NO UNDEFINED SYMBOLS
     NO ERRORS
        NULLS
                                                A
                                          END


     1 MINUTES, 5 SECONDS.
END JOB 091.
```

BINARY OBJECT DECK FORMAT

The format of the binary object program deck that is produced for a CODAP-1 program consisting of two subprograms is shown below. The format for each type of card is described in the CO-OP Monitor Programmer's Guide.



The binary deck for each subprogram contains one or more IDC cards, the RBD (program) cards, and one or more EPT, EXT, LAT, TRA cards. The last subprogram is followed by a second TRA card. The complete program deck (consisting of all subprograms) is loaded and executed under control of the CO-OP Monitor as described in the CO-OP Monitor Programmer's Guide.

TABLE 1 1604 MACHINE INSTRUCTIONS

Octal	Mnemonic	Description
00	ZRO	(not used)
01	ARS	A Right Shift
02	QRS	Q Right Shift
03	LRS	AQ Right Shift
04	ENQ	Enter Q
05	ALS	A Left Shift
06	QLS	Q Left Shift
07	LLS	AQ Left Shift
10	ENA	Enter A
11	INA	Increase A
12	LDA	Load A
13	LAC	Load A, Complement
14	ADD	Add
15	SUB	Subtract
16	LDQ	Load Q
17	LQC	Load Q, Complement
20	STA	Store A
21	STQ	Store Q
22	AJP	A Jump
23	QJP	Q Jump
24	MUI	Multiply Integer
25	DVI	Divide Integer
26	MUF	Multiply Fractional
27	DVF	Divide Fractional
30	FAD	Floating Add
31	FSB	Floating Subtract
32	FMU	Floating Multiply
33	FDV	Floating Divide
34	SCA	Scale A
35	SCQ	Scale AQ
36	SSK	Storage Skip
37	SSH	Storage Shift
40	SST	Selective Set
41	SCL	Selective Clear
42	SCM	Selective Complement
43	SSU	Selective Substitute
44	LDL	Load Logical
45	ADL	Add Logical
46	SBL	Subtract Logical
47	STL	Store Logical
50	ENI	Enter Index
51	INI	Increase Index

TABLE 1 1604 MACHINE INSTRUCTIONS (CONT)

Octal	Mnemonic	Description	Octal	Mnemonic	Description
52	LIU	Load Index, U	65	THS	Threshold Search
53	LIL	Load Index, L	66	MEQ	Masked Equality
54	ISK	Index Skip	67	MTH	Masked Threshold
55	IJP	Index Jump	70	RAD	Replace Add
56	SIU	Store Index, U	71	RSB	Replace Subtract
57	SIL	Store Index, L	72	RAO	Replace Add One
60	SAU	Substitute Address, U	73	RSO	Replace Subtract One
61	SAL	Substitute Address, L	74	EXF	External Function
62	INT	Input Transfer	75	SLJ	Selective Jump
63	OUT	Output Transfer	76	SLS	Selective Stop
64	EQS	Equality Search	77	SEV	(not used)

TABLE 2

EQUIVALENT MNEMONIC INSTRUCTION CODES†

1. A-JUMP AND A-RETURN-JUMP

Operation		Equivalent	
AJP	Z	AJP	0
AJP	N	AJP	1
AJP	P	AJP	2
AJP	M	AJP	3
ARJ	Z	AJP	4
ARJ	N	AJP	5
ARJ	P	AJP	6
ARJ	M	AJP	7

2. Q-JUMP AND Q-RETURN-JUMP

QJP	Z	QJP	0
QJP	N	QJP	1
QJP	P	QJP	2
QJP	M	QJP	3
QRJ	Z	QJP	4
QRJ	N	QJP	5
QRJ	P	QJP	6
QRJ	M	QJP	7

3. RETURN JUMP

RTJ	0	SLJ	4
RTJ	1	SLJ	5
RTJ	2	SLJ	6
RTJ	3	SLJ	7

4. STOP AND RETURN JUMP

SRJ	0	SLS	4
SRJ	1	SLS	5
SRJ	2	SLS	6
SRJ	3	SLS	7

5. NO OPERATION

NOP		ENI	0

6. EXTERNAL FUNCTIONS

SEN	0	EXF	7	
SEL	0	EXF	0	
ACT	b	EXF	b	(b = 1-6)

† Z = Zero, N = Non-zero, P = Plus, M = Minus

TABLE 3

SPECIAL CODES FOR FLX OR TEL PSEUDO-INSTRUCTIONS

These alphabetic codes are used with the FLX and TEL pseudo-instructions to indicate the specified control functions.

FLX ENTRIES

Code		FLX Octal Equivalent
R	carriage return	45
U	shift to upper case	47
L	shift to lower case	57
B	backspace	61
C	color shift	02
T	tab	51
S	stop	43
D	delete	77
F	tape feed	00

TEL ENTRIES

Code		TEL Octal Equivalent
R	carriage return	10
U	shift to figures	33
L	shift to letters	37
F	line feed	02
I	ignore	00

INDEX

	Page No.		Page No.
Assembled listing format	30	Interrupt	24
B-Field	3, 5	Job deck structure	27
BCD	7	Labeled common	11
BES	11	LIB	16
Binary object deck format	34	Location field	3
Binary scaling factor	8	Location field symbols	4
BLOCK	11	Lower instruction	3
BSS	10	M-Field, term	3, 5
Calling CO-OP Monitor Library Subroutines	23	MCS control card	28
		Minus sign in column 1	21
CCS control card No. 1	28	Numbered common	11
CCS control card No. 2	29	OCT	9
COMMON	12	OP-code	3, 4
Common storage areas	11	ORG	17
DEC	8	ORGR	18
Decimal scaling factor	8	Plus sign in column 1	20
EJECT	17	Presetting common	14, 18
END	18	READ*	22, 23
ENTRY	15	Relative Addressing	6
EQU	15	REM	17
Error Codes	31	Remarks	3, 6
EXT	16	Rules for BLOCK and COMMON	15
FINIS	18	Sample Assembled Listing	32
Floating point constant	8	Sample Program	25
FLX	9	Sequence number	3
IDENT	17	SPACES	16
I/O	17	Subprograms	24
Input-Output	22	TEL	10
Instruction pairing	19	Upper instruction	3
Integer constant	8	WRITE*	22, 23

Other publications concerning programming for the CONTROL DATA 1604 and 1604-A Computers are:

3-Phase Automonitor	#131
PERT for 1604 Computer	#133
1604 Programming Manual	#167b
Fortran Autotester	#186a
1604-A Reference Manual	#245
Fortran-62	#506a
CO-OP Monitor/Programmer's Guide	#508
CO-OP Monitor/Operator's Guide	#509
CODAP-1 Reference Manual	#510
CDM2 Linear Programming System	#511
CO-OP Monitor/Library Subroutines	#516a
Introduction to COBOL	#520
COBOL/Reference Manual	#521
Fortran 63 Volume 1	#527
Fortran 63 Volume 2	#528

CONTROL DATA
CORPORATION

PROGRAMMING SYSTEMS

CODAP-1 System Revisions

The CODAP-1 Assembly program has been revised to include numeric, Hollerith, Teletype and Flexowriter literals, special symbolic tags, page title cards, and special pseudo instructions. Also, a symbol cross reference table, unless deleted, is printed at the end of the assembled listing of each subprogram.

CONTENTS

Literals	1
Special Symbolic Tags	3
Pseudo Instructions	3
LITDEC	3
LITOCT	3
UNLIST	4
LIST	4
TITLE	4
DETAIL	4
REF	4
SYSTEM	4
CALL	4
REMARK	4
Page Titles	5
Symbol Cross Reference Table	5
Control Card Format	6

CONTROL DATA CORPORATION
Documentation & Evaluation
3330 Hillview Avenue
Palo Alto, California

December, 1963 PSB-AE-10

LITERALS

The literals are placed in the M-terms of machine instructions in the following manner, starting in column 20: =cL

The = sign identifies the M-term as a literal value, c is an alphabetic character or a period that identifies the type of literal, and L is the literal. The value of each literal is computed and stored as one word in a table at the end of the subprogram. The address assigned to the word containing the literal is inserted in the octal equivalent of the symbolic machine instruction.

Literals may be of any of the types defined in the table below:

Code (c)	Type
D	Decimal literal. One or two decimal constants as specified by the DEC pseudo instruction may be used as decimal literals. Two constants, separated by a comma, result in a two-word double-precision literal. Decimal constants are assumed to be integers unless a decimal point or an E scale factor is encountered in the constant field. The form =.n is always assumed to be a decimal floating point constant.
O	Octal literal. One or two octal constants of up to 16 digits can be used as in an M-term as an octal literal. Two constants, separated by a comma, result in a two-word double-precision literal.
H	Hollerith literal. Up to eight Hollerith characters in the M-term are converted to 6-bit BCD codes. If more than eight characters are placed in the M-term, only the first eight are converted; less than eight characters are left justified within the word and the remainder of the word is filled with blanks (20_8).

Code (c)	Type
F	Flexowriter literal. Up to eight characters in the M-term are converted to 6-bit Flexowriter codes. Spaces are added to fields that are shorter than eight characters. Special characters such as carriage return and tab are indicated as specified for the FLX pseudo instruction.
T	Teletype literal. Up to eight characters in the M-term are converted to 6-bit Teletype codes. Spaces are added to fields that are shorter than eight characters. Special characters such as carriage return and tab are indicated as specified for the TEL pseudo instruction.

Examples:

Octal Literals

```
=012345
=0-56767
=0+54321
=0+7,-7    (double precision)
```

Decimal Literals

```
=D7
=D187.91E+14 (floating point)
=D-13
=D-4,+4     (double precision)
```

Hollerith Literal

```
=HABCDEFGH
```

Flexowriter Literal

```
=FMNOPQ
```

Teletype Literal

```
=TRYRYRYRY
```

Numeric literals may be written without the identifying code (D or O). In any subprogram, numeric literals are assumed to be decimal unless the instruction containing the literal is preceded by a LITOCT pseudo instruction. All numeric literals not identified by a D or O following a LITOCT pseudo instruction are assumed to be octal. The decimal mode may be reestablished at any point in a subprogram by a LITDEC pseudo instruction.

SPECIAL SYMBOLIC TAGS

Instead of reserving one word of temporary storage with a BSS or BES pseudo instruction, a special symbolic tag can be used in the M-term of any machine instruction. This tag has the form: =S symbol. The =S identifies the symbol as being a special tag, and the symbol is any legal CODAP-1 symbol that does not contain a plus or minus sign. Therefore, in the instruction: STORE STA =SFIVE the contents of the A register will be stored in location FIVE. The assembler will reserve a word of temporary storage at the end of the subprogram with the symbolic tag FIVE. If this tag has been used in a location field elsewhere in the subprogram, that location assignment would override the special symbol tag and FIVE will be assigned to the location with that tag in the location field.

STORAGE OF LITERALS AND SPECIAL SYMBOL TAGS

Symbol tags are assigned storage locations immediately following the last location reserved by explicit program instructions.

Literals are assigned storage locations immediately following the area reserved for symbol tags. Up to 500 words of storage may be used for literals. Any two literals with the same internal octal representations will be assigned to the same storage area. Unless a TITLE pseudo instruction is used, all numeric literals will appear at the end of the program listing.

PSEUDO-INSTRUCTIONS

The following pseudo instructions are added to those currently in CODAP-1, and are used to set the literal mode. If neither of these has appeared, the literal mode is decimal.

LITDEC indicates that all subsequent numeric literals of the form =n, = + n, = -n are to be treated as decimal.

LITOCT indicates that all subsequent numeric literals of the form = n, = + n, = - n are to be treated as octal.

UNLIST indicates that no assembled listing is to be produced for the subsequent instruction. This pseudo instruction applies only if the list option is selected on the CCS control card.

LIST reinitiates listing of the subprogram after an UNLIST pseudo instruction has suppressed it.

TITLE In a normal assembly listing for CODAP-1, all M-term strings are listed on the same line with the pseudo instruction identifying the string, such as EXT, ENTRY, COMMON, BCD, FLX. Each element, starting with the second, is also listed on a separate line following the pseudo instruction.
The TITLE pseudo instruction eliminates these single entries; therefore they appear only once in the M-term of the pseudo instruction. For example, the pseudo instruction

```
ENTRY       START, ONE, TWO THREE
```

is normally listed as

```
ENTRY       START, ONE, TWO, THREE
            ONE
            TWO
            THREE
```

The TITLE pseudc instruction would result in only the line containing the card image being printed.

DETAIL cancels the TITLE format for assembled listings and restores the normal format.

REF deletes the symbol cross reference from the assembled listing of a subprogram.

SYSTEM generates library directory cards for the associated subprogram. SYSTEM is used in a subprogram that is to be added to a CO-OP Monitor subroutine library using the Monitor routine LIBEDIT.

CALL is equivalent to two instructions: an EXT pseudo instruction and an RTJ machine instruction. CALL is used when entering other subprograms such as library subroutines. The M-term contains the name of the subroutine to be entered; this name is inserted in the M-term of the RTJ instruction and in the External Symbol Table. For example, the pseudc instruction

```
CALL    SQRT
```

generates one line of coding:

```
RTJ     SQRT
```

and defines SQRT as an external symbol, just as if an EXT SQRT had been used.

REMARK is identical to the REM pseudo instruction except that REMARK does not force the next machine instruction to be an upper instruction.

PAGE TITLES

An assembled listing is produced with page titles by including a page title card before the IDENT card of the first subprogram to be assembled. The page title is continued throughout the assembly unless another page title card is encountered.

The format for page title cards is as follows:

Column 1	* (asterisk)
Columns 2-8	Blank
Columns 9-72	Page title

A maximum of 64 characters can be used for a page title.

The current date and the page number are always printed on each page of the assembled listing. Page title cards may be placed anywhere in a source program deck. Each card causes a page eject for the assembled listing and sets the page numbering sequence to 1.

SYMBOL CROSS-REFERENCE TABLE

Each symbol in a subprogram is printed in a table at the end of that subprogram assembly listing. The table lists the address of the instruction which contains the symbol in the location field, and also the addresses of all instructions that use the symbol in the M-term.

CODAP1 Card

7
9CODAP1, options

Field 1: 7-9 punch in column 1 followed immediately by CODAP1.

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

7
9CODAP1, L,E,B.

7
9CODAP1, LIST =1, E=10.

Any option may be followed by =n, where n is the number of a logical unit which is to be used for that option.

Options:		$n \neq 0$
LIST	List assembled programs	List source language program
PUNCH	Punch relocatable binary deck on logical unit 52	Punch binary on unit n.
EXECUTE	Write load-and-go tape 56	Write load-and-go tape n
INPUT	Input source from 50. Same if option is not present	Input source from n
SYMBOLS	Allot 2048 words to assembler symbol table; if option is omitted, allot 1024 words.	Allot max.(M,1024) words to the assembler symbol table
REFERENCES	Suppress assembler symbol reference table; if option is omitted, print table	Suppress table
NULLS	Suppress null listing; if option is omitted, print null listing.	Suppress null listing

(If n is 0, the option is interpreted as if it were not present).

COSY

AN ASSEMBLY PROGRAM FOR THE 1604

COSY

COSY is a 1604 assembly program which offers in addition to the options of CODAP1, the following new options:

Outputs compressed symbolic deck (COSY deck).

Outputs a CODAP1 symbolic deck sequenced with eight characters: the first three characters of the IDENT name and five digits.

Accepts a COSY deck as input for assembly or a correction deck for updating and assembly.

COSY is called by the COSY control card described on page 4. A COSY deck consists of a COSY identification card, followed by the compressed symbolic cards. The COSY identification card contains the following:

Columns	10-13	40-80	
	COSY	DATE	(date is optional)

The first word of a compressed symbolic card is: (in column binary)

3XSS
6XSS
XXSS
5XSS

The first word of the last card is:

3X3X
7X7X
XXXX
5X5X

XXXXX is a sequence number and SSSSSSSS is a checksum. The remainder of the card is empty.

Two pseudo instructions, DELETE and INSERT, may be combined with CODAP1 cards to form a Correction Deck for updating a COSY deck.

Columns	10-15	20-31
	DELETE	m
or	DELETE	m,n

The CODAP1 cards to replace the deleted instructions follow this card. This card deletes symbolic input line m or lines m through n; m and n are sequence numbers ($m < n$). All subsequent CODAP1 symbolic cards replace the deleted cards until a DELETE, INSERT or COSY identification card is encountered.

Columns	10-15	20-25
	INSERT	m

The CODAP1 symbolic cards following this card up to a Codap1 END card are inserted between line m and m+1, until an INSERT, DELETE or COSY identification card is encountered. (Any Codap cards following this END card are ignored.) An Inserted END card will cause the loss of all COSY information following that point.

All DELETE and INSERT cards must be arranged in the Correction Deck so that COSY sequence numbers are referenced in ascending order. If the first card of the deck is not an INSERT, DELETE or COSY identification card, it is assumed to be a CODAP1 symbolic deck. A listing of the correction deck appears before the assembly listing detailing changes. If a DELETE or INSERT card appears out of order, it is flagged as an error and its CODAP1 correction cards are inserted under the control of previous INSERT or DELETE. A sequence error in the COSY deck causes an error diagnostic and the assembly is terminated. A checksum error is flagged, but assembly continues.

In the assembly listing, inserted cards are flagged with *** unless a CODAP symbolic deck is requested. A new sequence number will appear if a COSY deck or a CODAP symbolic deck is requested. Otherwise the sequence number is blank.

SAMPLE LISTING

PRE-PROCESSOR CORRECTION LISTINGS

```
DELETE     153,156                                          ***
BCD        7                                                   00153 DELETED
BCD        7`                                                  00154 DELETED

BCD        7                                                   00155 DELETED
BCD        7                                                   00156 DELETED


BCD        6                                                ***      INSERTED
BCD        6                                                ***      INSERTED
BCD        6                                                ***      INSERTED
BCD        6                                                ***      INSERTED
BCD        4                                                ***      INSERTED
DELETE     878                                              ***
STA        AR+2                                                00878 DELETED
ALS        6                                                ***      INSERTED
STA        AR                                               ***      INSERTED
DELETE     1194                                             ***
ENA    5   0              EXT NUMBER                           01194 DELETED
LDA        EXTEQUIV                                         ***      INSERTED
DELETE     1221,1222                                        ***
ENA    5   0              EXT NUMBER                           01221 DELETED
SLJ        ENTRY7A                                             01222 DELETED
ENI    4   AR                                               ***      INSERTED
LDA        EXTEQUIV       EXT NUMBER                        ***      INSERTED
RTJ        PKLD           PACK LOCATION DIGITS              ***      INSERTED
SLJ        ENTRY7A                                          ***      INSERTED
DELETE     1240                                             ***
ENA    5   0              YES                                  01240 DELETED
```

SAMPLE LISTING
(cont'd.)

```
       LDA       EXTEQUIV      YES                                ***      INSERTED
       DELETE    1254                                             ***
LIB4   LDQ       WS2           ENTER L-TERM INTO SYMBOL TABLE       01254  DELETED
LIB4   LDQ       EXTEQUIV      ENTER L-TERM INTO SYMBOL TABLE     ***      INSERTED
       DELETE    1466                                             ***
       SLJ       BSS8                        (PASS2), GO TO       01466  DELETED
       SLJ       BSS1B                                            ***      INSERTED
       DELETE    1471                                             ***
       SLJ       BSS8                                               01471  DELETED
BSSIB  LDQ       LOCCTR                                           ***      INSERTED
       SLJ       BSS9                                             ***      INSERTED
       INSERT    1490                                             ***
       LDQ       LARELBIT                                         ***      INSERTED
       STQ       RELBITS                                          ***      INSERTED
       INSERT    1498                                             ***
       LDQ       LARELBIT                                         ***      INSERTED
       STQ       RELBITS                                          ***      INSERTED
       DELETE    1511                                             ***
       SLJ       BSS8          AND GO TO DUMPBIN                    01511  DELETED
       SLJ       BSS1B                                            ***      INSERTED
       INSERT    1533                                             ***
       LDA       LARELBIT                                         ***      INSERTED
       STA       RELBITS                                          ***      INSERTED
```

INPUT TO COSY

COSY will accept three basic input decks:

A Standard CODAP1 (BCD) deck.
The first card is an IDENT or TITLE card. No address fields may extend beyond column 72. (columns 73-80 are ignored by COSY.)

A COSY deck.
The first card has COSY punched in columns 10-13.

A COSY deck preceded by a correction deck.
The first card of the correction deck contains either INSERT or DELETE in columns 10-15.

COSY OUTPUT

1. When the COSY option appears on the control card, a new COSY deck is output on the punch unit for each subprogram assembled. The card sequence numbers correspond to the numbers on the assembled listing. Preceding each COSY deck is a COSY identification card with the current date. If two END cards appear in sequence, the second one is punched in BCD, and a FINIS card is punched following the last COSY deck or the second of two sequential END cards.

2. When the BCD option appears on the COSY control card, a CODAP1 symbolic deck is output on the punch unit for each subprogram assembled. Each deck is sequenced in columns 76-80, with numbers from 1 to 99999. Columns 73-80 of all cards except the IDENT and any title card preceding it, contain the first three letters of the IDENT name. The second of two END cards and the FINIS cards are not punched. If the COOP standard punch unit (52) is not used, two EOFs are written after the last symbolic deck. The unit is then backspaced one record if possible.

CODAP1 Card

$^{7}_{9}$COSY, OPTIONS

Field 1 7-9 punch in column 1 followed immediately by COSY.

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

$^{7}_{9}$COSY, L,E,B.

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

Any option may be followed by =n. (If n is 0, the option is interpreted as if it were not present).

Options		n≠ 0
LIST	List assembled programs	List source language program
PUNCH	Punch relocatable binary deck on logical unit 52	Punch binary on unit n.
EXECUTE	Write load-and-go tape 56	Write load-and-go tape n
INPUT	Input source from 50. Same if option is not present.	Input source from n
SYMBOLS	Allot 2048 words to assembler symbol table; if option is omitted, allot 1024 words.	Allot max. (M, 1024) assembler symbol table
REFERENCES	Suppress assembler symbol reference table; if option is omitted, print table	Suppress table

NULLS	Suppress null listing; if option is omitted, print null listing.	Suppress null listing
COSY	Punch compressed symbolic deck on unit 52	Punch on unit n
BCD	Punch BCD deck on unit 52	Punch on unit n

If both COSY and BCD are requested on the same unit, the BCD request will be ignored.

COSY COMPRESSION CODES

Cosy compresses multiple blanks into one or two BCD characters according to the following table:

1 blank = 20
2 blanks = 15
3 blanks = 16
4 blanks = 17
5 blanks = 35
6 blanks = 36
7 blanks = 37
8 blanks = 55
9 blanks = 56
10 blanks = 57
11 blanks = 75

Twelve or more blanks are represented by 76nn ($00 \leq nn \leq 74_8$); nn is the number of blanks in excess of 12. The result is packed into column binary.

Example,

L	OP	M
	NOP	

would result in:

5 4 0

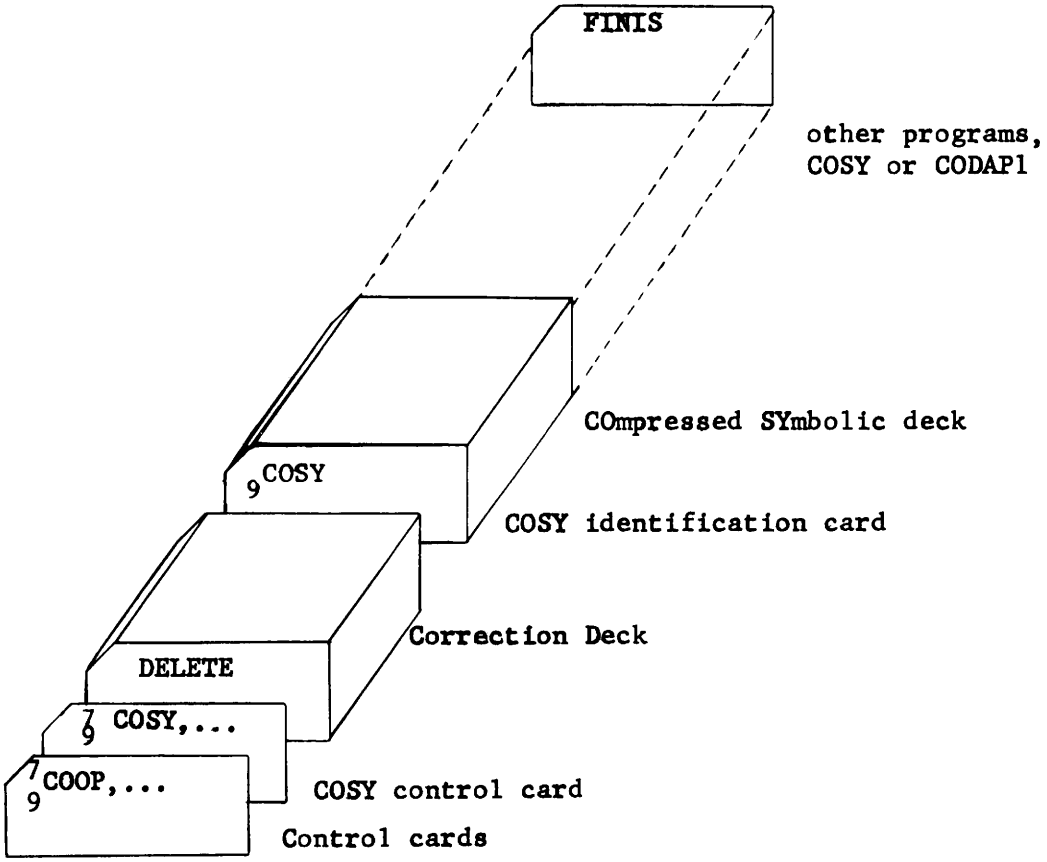
6 6 0

4 4

5 7

The special code 77 or 00 (when not preceded by 76) indicates the end of a CODAP1 card.

U P D A T E D E C K


FINIS
other programs,
COSY or CODAP1
COmpressed SYmbolic deck
9 COSY
COSY identification card
Correction Deck
DELETE
7 9 COSY,...
7 9 COOP,...
COSY control card
Control cards

CODAP | SAG NR. | INIT | SIDE | AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
NDREEL	SLJ		**		
	SiU		UDHØp2		
	SiL		UDHØp2		
	STA		ASTØRE		
	~~STQ~~		~~ASTØRE~~		
	LDA		ENDREEL		
	ARS		24		
	SAU		UDHØp2+1		

CODAP | SAG NR. | INIT | SIDE | AF | DATO | ⅘ REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
QB1	ENA		RCETL		
	iNA		-1		
	SAU		OKUD		
	SAK		*+1		
	SAL		MARK		
	ENi	1	**	702	
	LDA	1	0		
	SAL		BIT45iQ+1	676	
	SAU		OKUD+1	676	
	SA				
ARK	ENi	2	Bi745iQ		
	SiL	2	**	702	
	SLJ		UDHOP1		

CODAP | SAG NR. | INIT | SIDE | AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
;T45;Q	SiU	1	UDHOp2		
	SiL	2	UDHOp2		
	STA		ASTORE		
	ENi	2	**	676	
~~	~~~		~~~		
~~	~~~		~~~		
	~~				

CODAP | SAG NR. | INIT | SIDE | AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
HECKPS	LiU	1	ØKUD		
	iNi	1	-3		
	LDA	1	0		
	SCL		MASKE		
	SUB		iPSNUMRE+5		
	AJP	0	Øk		
	SLJ		ØKUD+1		
	NOP				
k	LDA		ASTØRE		
	LLS		48		
	SST		BiT45		
	LLS		48		
kUD	SiL	2	**		

CODAP | SAG NR. | INIT / SIDE / AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
	NOP				
	EMA		**	676	
	SAU		UDHOP2+1		
	SLJ		UDHOP1		

CODAP | SAG NR. | INIT | SIDE | AF | DATO | REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
DHOp	LDQ		QSTORE		
	NOP				
DHOp1	LDA		ASTORE		
	NOp				
DHOp2	ENi	1	**		
	ENi	2	**		
	SLJ		**		

CODAP | SAG NR. | INIT | SIDE 1 AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
	ENA		RCETK		
	INA		-1		
	SAU		ØKUD		
AB1	SAQ		*+1		
	SAL		MARK		
	ENI	1	** 702	ADRESSE paa (udhop)	
	LDA	1	0		
	SAL		MARK,Q		
	SAU		ØKUD+1		
	SAK		UDhop		
ARK	ENI	2	MARK,Q		
	SIL	2	** 702		
	SLJ		SLUT		

CODAP | SAG NR. | INIT | SIDE 2 AF | DATO | % REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
ARKiQ	SiL		2 OKUD		
	ENi		2 **	6/6	
	SLJ		CHECKPS		
	Nop				
k UO	SiL		2 **	7/2	
	ENi		2 **		
	SLJ		**	6/6	

CODAP | SAG NR. | INIT | SIDE 3 AF | DATO | ⅛ REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
	- -				
				oprindelig adr indsættes	
	'				
	'				
HECKPS	LIU	1	OKUD		
	INI	1	-3		
	'				
	STA		ASTORE		
	LDA	1	0		
	SAL		*+1		
	NOP				
	ENA		** PSNO		
	SUB		IPSNUMRE+5		
	AJP	0	OK		

CODAP | SAG NR. | INIT | SIDE 5 AF | DATO | ⅘ REGNECENTRALEN

LOCATION	OP-CODE	B	M-TERM	REMARK	SEQUENCE
	LDA		ASTAXE		
dhop	~~SLJ~~		** 676		
	SLJ		**		
	~~dhop~~				
k	LDA		ASTORE		
	LLS		48		
	SST		B;T45		
	LLS		48		
	SLJ		Okud		

10