SYSTEMS
ENGINEERING LABORATORIES

REFERENCE MANUAL
SYSTEMS 810A/810B Assembler

SYSTEMS
ENGINEERING LABORATORIES

REFERENCE MANUAL
SYSTEMS 810A/810B Assembler

December 1968

323-095052-002
Price: $10.00

6901 West Sunrise Blvd., Fort Lauderdale, Florida 33313
Area Code 305 587-2900

LIST OF EFFECTIVE PAGES

The total number of pages in this manual is 40, consisting of the following:

Page Number	Issue	Page Number	Issue
Title	Original		
A	Original		
i and ii	Original		
1-1 and 1-2	Original		
2-1 thru 2-6	Original		
3-1 thru 3-6	Original		
4-1 thru 4-4	Original		
5-1 thru 5-4	Original		
6-1 thru 6-4	Original		
A-1 and A-2	Original		
B-1 and B-2	Original		
C-1 and C-2	Original		
D-1 thru D-4	Original		

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont'd)

LIST OF ILLUSTRATIONS

LIST OF TABLES

SECTION I
THE SEL 810A/810B ASSEMBLER PROGRAM

1-1 GENERAL DESCRIPTION

1-2 This manual describes programming of the SEL 810A General Purpose Digital Computer using the Assembler Program. The various sections describe the language, operations, and programming techniques. The machine language programming computer characteristics, and computer operating instructions are found in the SEL 810A Reference Manual and 810A Operating Instructions Manual.

1-3 SCOPE AND PURPOSE

1-4 The Assembler is a programming aid that will translate a symbolic program into machine language code. It provides the following features:

a. Enables substitution of mnemonic codes (i. e., AMA, SMA, LAA) for their octal equivalents, while maintaining the characteristics, flexibility, speed, and conciseness of machine language.

b. Permits the programmer assign symbolic address to storage locations.

c. Provides pseudo-operations to supplement the standard instruction repertoires.

1-5 COMPUTER AND PERIPHERAL EQUIPMENT

1-6 The Assembler will operate with a basic SEL 810A computer configuration. This consists of:

a. A minimum 4096 words of memory.

b. An ASR-33 Keyboard Printer, Paper Tape Reader and Punch.

1-7 Assemblers are available for 810A computers with extended memory. The Assembler may also utilize optional peripheral devices: card reader, card punch, line printer, magnetic tape, paper tape reader, and paper tape punch.

SECTION II
THE ASSEMBLER

2-1 INTRODUCTION

2-2 The SEL 810A/810B Assembler Program is a two pass assembly program that accepts symbolic instruction input from such devices as the typewriter keyboard, a card reader, or a paper tape reader. Output consists of binary relocatable cards, paper tape or magnetic (object tape) ready for loading into the computer, and an optional symbolic listing with error messages and a side-by-side octal listing on the console typewriter or line printer.

2-3 INPUT LANGUAGE FORMAT

2-4 The general format of the SEL 810A Assembly symbolic instruction input (source Input) consists of five major fields. These fields are: Location, Operation, Address-Index, Comments, and Identification. A discussion of the contents of each field follows. The standard SEL coding form is shown in Figure 2-1.

2-5 LOCATION FIELD (CARD COLUMNS 1 - 4)

2-6 This field provides a method of symbolic identification of this location in the code string. The symbolic label consists of 1 to 4 characters with the first character being a letter and the remaining characters being either letters or digits. The first character must be alphabetic and must begin in Column 1. The Assembler assigns actual memory addresses to the symbolic locations when assembling the object program.

2-7 OPERATION FIELD (CARD COLUMNS 6 - 9)

2-8 The operation field consists of a mnemonic computer instruction or a pseudo instruction. A list and definition of mnemonic operation instructions is given in Appendix A. A list and definition of pseudo-operation instructions is provided in Section III. Mnemonic computer instructions consist of three letters. If the instruction's address is to be made indirect, the three-letter instruction is followed by the asterisk (*) character. Pseudo instructions consist of 3 or 4 letters and represent either instructions to the assembly program or data definitions.

2-9 ADDRESS, INDEX FIELD/VARIABLE FIELD (STARTS IN COLUMN 1)

2-10 Memory reference instructions use this field to define the operand address. The address followed by the characters "comma" and "one" (, 1) signifies indexing where the B-Accumulator is the index register. In the 810B Computer the optional index register may also be used for indexing. Instructions, such as shift, I/O machine operations, and some pseudo-operations, have special formats for the variable field. These are discussed later in this section.

2-11 COMMENTS FIELD (START AFTER FIRST SPACE IN ADDRESS FIELD)

2-12 The Comments Field may be used for any comments that the programmer cares to make. Contents of this field have no effect upon the assembler, but is printed on the symbolic listing. The comments field must not start before column 13. Any line that has an asterisk (*) in the first character position of that line is considered a line of comments.

2-13 IDENTIFICATION FIELD (CARD COLUMNS 73 - 80)

2-14 This field is not checked and is considered part of the comments, and is provided as a programmers aid. For example, it may be used to identify a card or cards in a card deck or for sequencing the card deck.

2-15 SYMBOLIC CODING USING ASSEMBLER

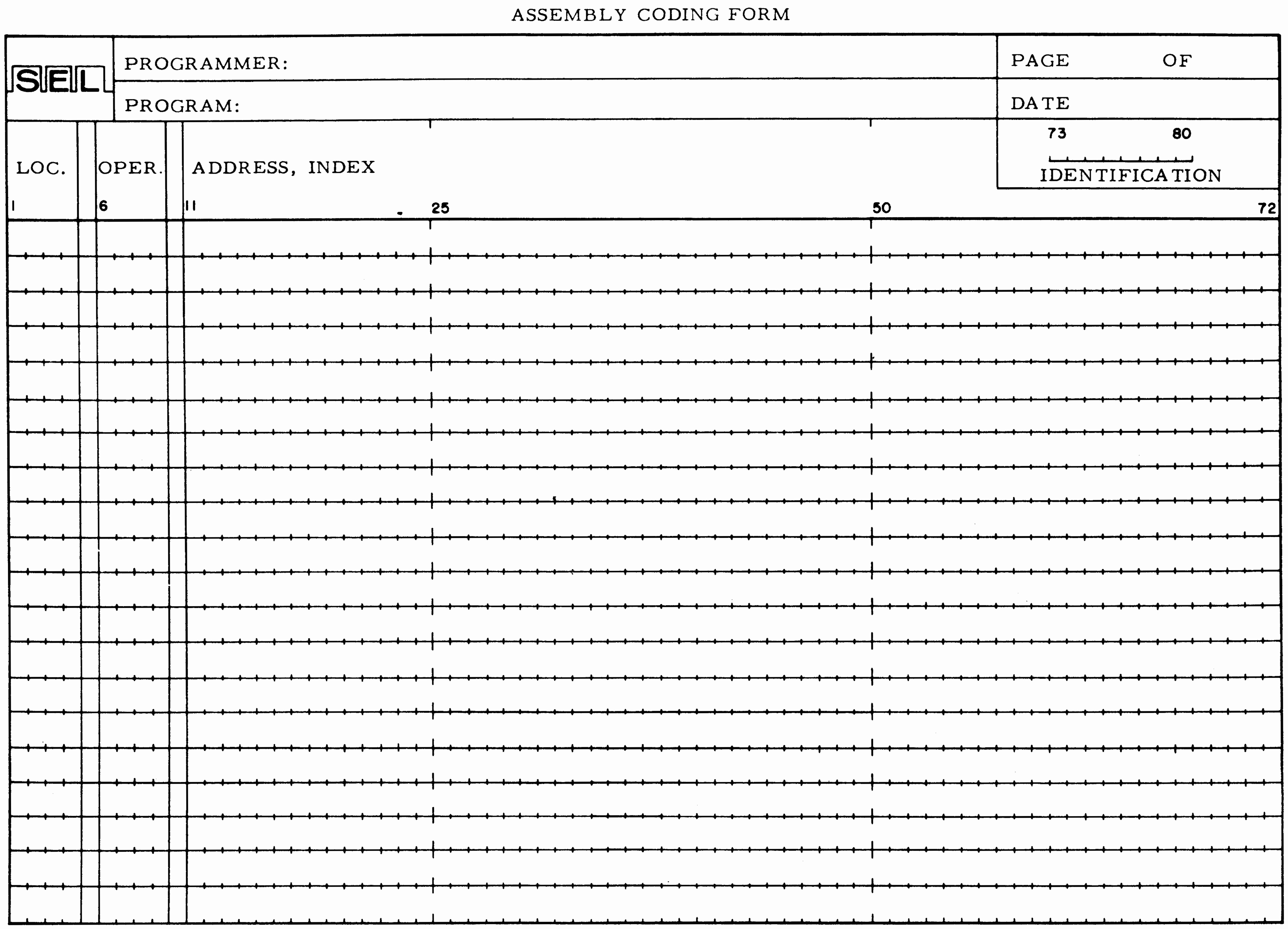
2-16 The number field title boxes of the coding form refer to card column locations. Regardless of input device the same form is used.

2-17 LOCATION FIELD

2-18 The Location Field may consist of a symbol for an address or control, or it may contain nothing.

2-19 OPERATION FIELD

2-20 The Operation Field is the same as the operation portion of the machine language instruction, and is coded as a mnemonic computer in-



ASSEMBLY CODING FORM

SEL

PROGRAMMER:

PROGRAM:

PAGE OF

DATE

73 80

IDENTIFICATION

LOC. OPER. ADDRESS, INDEX

1 6 11 25 50 72

Figure 2-1. SEL Assembler Coding Form

LOC.			
1	6	11	25 — 50
STRT			(ASSEMBLER ASSIGNS LØCATIØN
A001			STRT LABEL)
1STR			(ILLEGAL)

	OPER.		
1	6	11	25 — 50
	AMA		(MNEMØNIC INSTRUCTIØN)
	LBA		(MNEMØNIC INSTRUCTIØN)
	CMA*		(MNEMØNIC INSTRUCTIØN
*			WITH INDIRECT ADDRESS)
	NAME		(PSEUDØ ØPERATIØN
	EQU		(PSEUDØ ØPERATIØN

struction. The mnemonic instruction consists of three alpha characters and a pseudo-operation of three of four alpha characters.

2-21 ADDRESS AND INDEX FIELD (VARIABLE FIELD)

2-22 The Address, Index Field (operand address) is the address portion referred to by memory reference instructions, shift instructions, I/O machine operations, and some pseudo-operations. An operand address may have any of the following formats:

a. BLANK ADDRESS: When no address subfield is required.

b. SYMBOLIC ADDRESS: Consists of 1 to 4 characters starting with a letter.

	OPER.	ADDRESS, INDEX	
1	6	11	25 — 50
	SAZ		(NØ ADDRESS REQUIRED)
	BRU	0, 1	(BLANK ADDRESS INDEXED)

	OPER.	ADDRESS, INDEX	
1	6	11	25 — 50
	STA	ALPH	(SYMBØLIC ADDRESS)
	STA	B002, 1	(SYMBØLIC ADDRESS INDEXED)

c. EXTERNAL SYMBOLIC ADDRESS: An external symbolic address consists of a dollar sign ($) followed by 1 to 6 characters, the first of which must be a letter. This external address is presumed undefined within the program in which it is contained. It refers to a sub-routine or item located in a different sub-program or the library tape. A CALL* statement does not require a $XXXX Lable. No address arithmetic may be performed on External Symbolic Addresses.

1	OPER. 6	ADDRESS, INDEX 11	25 … 50
	IMS	$SQRT	(EXTERNAL SYMBØLIC ADDRESS)
	CALL	SUBRT	(EXTERNAL CALL USING CALL PSEUDØ-ØP)
	SPB	$SUBRT	(EXTERNAL CALL USING SPB INSTRUCTIØN)

1	OPER. 6	ADDRESS, INDEX 11	25 … 50
	LAA	548	(ABSØLUTE ADDRESS, DECIMAL)
	STA*	'1062	(ABSØLUTE ADDRESS, ØCTAL INDIRECT)

d. ABSOLUTE ADDRESS: An absolute address is used to reference a fixed memory location or to represent a count as in a shift instruction. The address is presumed to be decimal unless preceded by an apostrophe ('), in which case it is treated as an octal number. Absolute notations for the variable field are shown in Appendix D.

e. CURRENT LOCATION: The location of this instruction is used as the instruction's effective address if a single asterisk (*) appears in the address sub-field. This allows for reference to this or nearby relative location without assigning a symbolic name.

f. ADDRESS ARITHMETIC: Any current location (*), symbolic (NAME), or absolute ('1234) address may be joined with a constant, current locations (*), symbolic (NAME) or absolute (1234) address by an intervening plus (+) or minus (-) operator to define an effective address (NAME + 4). The above may be extended to more than two operands (A - B + 2).

1	OPER. 6	ADDRESS, INDEX 11	25 … 50
	LAA	*+2	(CURRENT LØCATIØN+2)
	AMA	*	(CURRENT LØCATIØN)
	LBA	*,1	(CURRENT LØCATIØN INDEXED)

1	OPER. 6	ADDRESS, INDEX 11	25 … 50
	LBA	*-2	(CURRENT LØCATIØN MINUS CØNSTANT)
	STB	NAME+4	(SYMBØLIC LØCATIØN PLUS CØNSTANT)
	SMA	CØMN-2,1	(SYMBØLIC LØCATIØN MINUS CØNSTANT INDEXED)
	STA	ALPH+B002,1	(SYMBØLIC ADDRESS ARITHMETIC INDEXED)
	LBA	*-NAME+9	(ADDRESS ARITHMETIC USING CURRENT
			LØCATIØN, SYMBØLIC AND A CØNSTANT

	OPER. 6	ADDRESS, INDEX 11	25	50
	AMB	=399	(LITERAL DECIMAL CØNSTANT)	
	LAA	='-2	(MINUS ØCTAL LITERAL	

	OPER. 6	ADDRESS, INDEX 11	25	50
	LAA	**	(ADDRESS TØ BE FILLED)	
	LAA	**,1	(ADDRESS TØ BE FILLED INDEXED)	

g. LITERAL ADDRESS: Literal addresses allow a constant to be defined, assigned to a memory cell and the location then used as the address for this instruction. All constants defined in literal addresses will optimize storage so that all identical constants (regardless of their format) will be assigned only once. A literal address consists of an equal sign (=) followed by the constant. Any decimal integer, octal number, single asterisk (current location) symbolic name or combination of these formats joined by a + or - may follow the equal sign in a literal address.

h. LOCATION TO BE FILLED: A double asterisk (**) indicates the address portion of the instruction is to be filled in by the object program at run time. This address is set during assembly to an absolute address of 00000.

LOC. 1	6	ADDRESS, INDEX 11	25	50
*			(THIS IS AN EXAMPLE ØF A CØMMENT)	
*			(FIRST CARD ØF THE PRØGRAM)	

NOTE

The assembly program presumes the computer has a 15-bit address and, therefore, does not attempt to reduce the argument address to a 9-bit address. When the resulting object tape is loaded by the loader into memory starting at a location determined by the operator, these 15-bit addresses are modified as follows:

1. If the argument address is located in MAP zero, the address is truncated to 9-bits and the MAP bit is set to zero.

2. If the argument address is located in the same MAP as the instruction in which it is contained, the address is truncated to 9-bits and the MAP bit is set to one.

3. Otherwise, truncate the 15-bit address to 14-bits and store the 14-bit address and its indirect and index bits automatically into a cell in MAP zero, set the 9-bit address of this cell in the instruction being loaded, and set the MAP bit to zero and the indirect bit to one.

2-23 COMMENTS FIELD

2-24 The comments field starts immediately after the first space in the variable address field, but never before column 13. This field has no effect on the assembler but is printed out on the symbolic listing if a listing is requested. Any line which has an asterisk (*) in the first character position of that line will be considered a line of comments.

2-25 Because of width limitations on the typewriter, comments appearing after column 50 cannot be printed. If a line printer is used for listing, comments after column 50 will be printed.

SECTION III

PSEUDO-OPERATIONS

3-1 GENERAL DESCRIPTION

3-2 In addition to symbolic instructions, the MNEMBLER Assembler will accept certain pseudo-operations for controlling the assembly process. Examples of the general format and use of the pseudo-operations are given in figure 3-1.

3-3 PSEUDO-OPERATIONS

3-4 The following describes the pseudo-operations used with the Assembler:

ABS Set the mode of the assembly program to ABSolute. When in this mode, all symbolic addresses will be assigned relative to location 00000 and output in a non-relocatable format.

REL Set the mode of the assembly program to RELative. When in this mode, all symbolic addresses will be assigned relative to the start load address (assigned when loading the program into memory) and output in a relocatable format compatible with the loader. The assembly program is initialized to the absolute mode and will remain in this mode until changed by an REL pseudo-op).

ORG The variable field specifies an address. When the assembly is in absolute mode, this address specifies the location of the next instruction. When the assembly is in relative mode, this address will be added to the start load address (assigned when loading the program into memory) in order to specify the location of the next instruction. In either case, all following instructions will be stored sequentially in memory until another ORG pseudo-op is given.

EQU The symbol in the location field will be assigned the address or value specified in the variable field. Constant values may not exceed 15 bits.

DAC This pseudo-op is used to generate Direct Address Constants used as argument addresses for subroutine calling sequences, or referred to by indirect instructions. The address in the variable field may be in any of the formats shown previously. The address will be truncated to 14 bits and will occupy bits 2 to 15 of the resulting word. The address may be indexed and the pseudo-op may be tagged indirect if required. (Setting bits 0 and 1.)

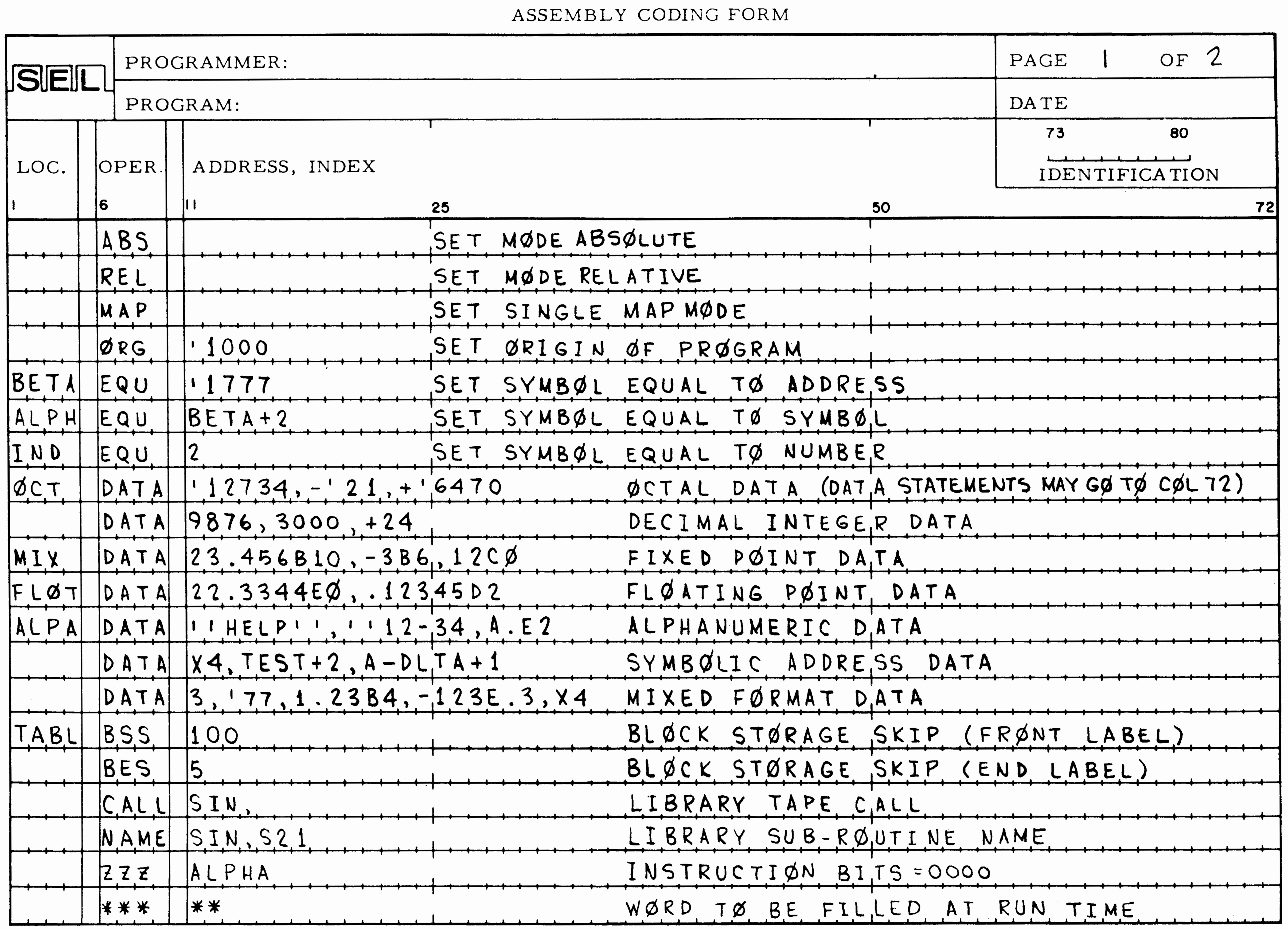
EAC This pseudo-op is used to generate 15-bit Extended Address Constants used as arguments of Long Branch instructions. Any of the Formats shown previously are acceptable in the variable field, except that the instruction may not be indexed nor made indirect.

DATA The variable field of this pseudo-op may contain any number and any mixture of the following data item formats. The variable field may extend to column 72. If the location field contains a symbol, it will be assigned the location of the first data item. If more than one data item is present (separated by commas), they will be assigned sequential storage locations.

a. Octal Data Item - Format: An optional sign (+ or -), followed by an apostrophe character ('), followed by 1 to 6 octal digits (0 through 7 $-/N_8/\leq$ '77777). If less than 6 digits are present, the number will be right justified with leading zeros added. If a minus sign is present, the number will be 2's complemented, a plus sign is ignored.

b. Decimal Integer - Format: An optional sign (+ or -) followed by 1 to 5 decimal digits (0 through 9 $-/N/\leq$ 32767). The number will be converted to binary and stored at a scale of B15. The number will be stored positively unless a minus sign is present. A minus sign will cause the 2's complement of the number to be stored.

c. Fixed-Point Single Precision Decimal Data - Format: An optional sign (+ or -), 1 to 5 decimal digits (0 through 9), mixed with an



ASSEMBLY CODING FORM

SEL

PROGRAMMER:

PROGRAM:

PAGE 1 OF 2

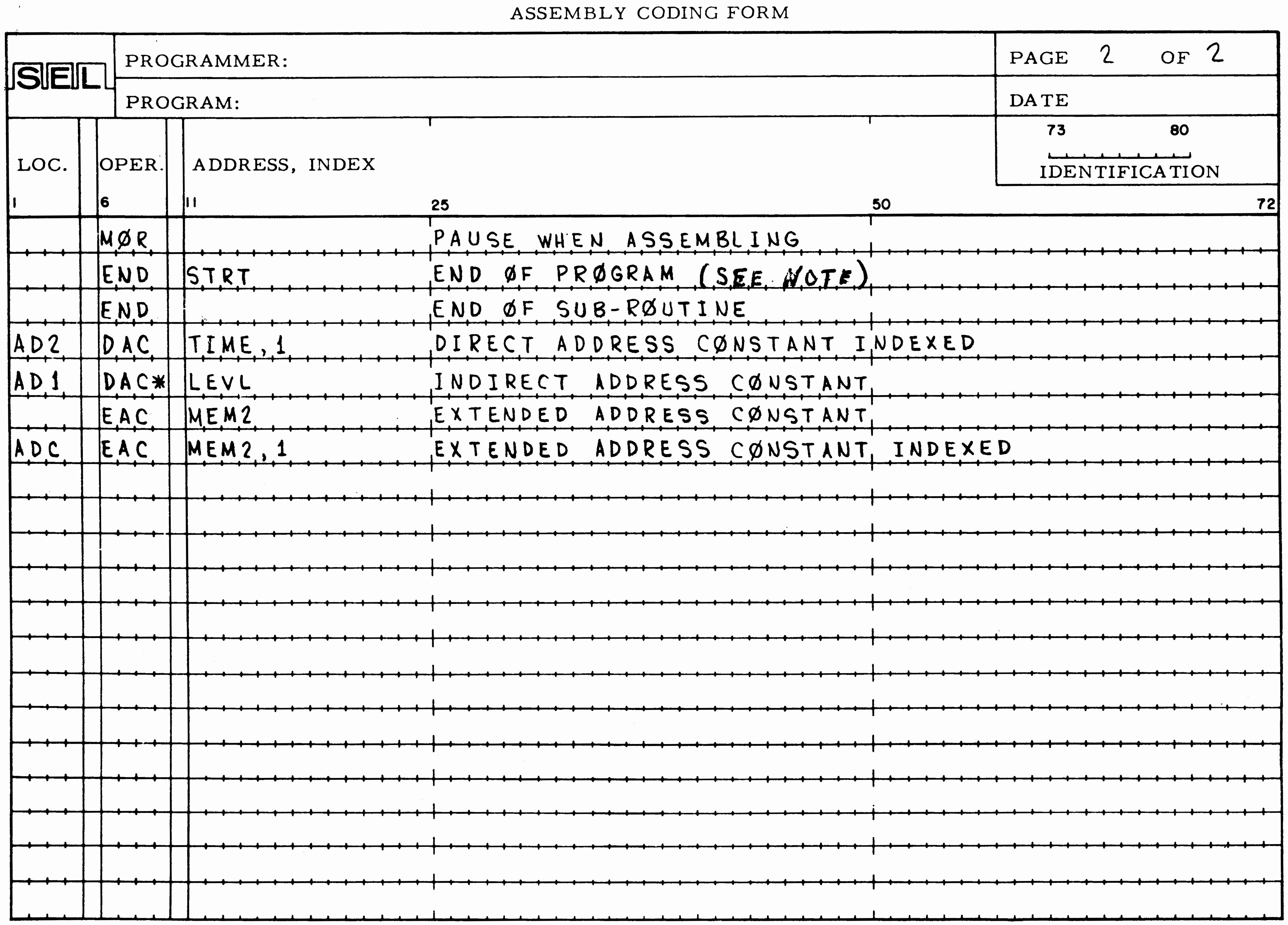
DATE

73 80 IDENTIFICATION

LOC. 1 | OPER. 6 | ADDRESS, INDEX 11 | 25 | 50 | 72

```
     ABS                 SET MØDE ABSØLUTE
     REL                 SET MØDE RELATIVE
     MAP                 SET SINGLE MAP MØDE
     ØRG  '1000          SET ØRIGIN ØF PRØGRAM
BETA EQU  '1777          SET SYMBØL EQUAL TØ ADDRESS
ALPH EQU  BETA+2         SET SYMBØL EQUAL TØ SYMBØL
IND  EQU  2              SET SYMBØL EQUAL TØ NUMBER
ØCT  DATA '12734,-'21,+'6470          ØCTAL DATA (DATA STATEMENTS MAY GØ TØ CØL 72)
     DATA 9876,3000,+24               DECIMAL INTEGER DATA
MIX  DATA 23.456B10,-3B6,12CØ         FIXED PØINT DATA
FLØT DATA 22.3344EØ,.12345D2          FLØATING PØINT DATA
ALPA DATA ''HELP'',''12-34,A.E2       ALPHANUMERIC DATA
     DATA X4,TEST+2,A-DLTA+1          SYMBØLIC ADDRESS DATA
     DATA 3,'77,1.23B4,-123E.3,X4     MIXED FØRMAT DATA
TABL BSS  100                         BLØCK STØRAGE SKIP (FRØNT LABEL)
     BES  5                           BLØCK STØRAGE SKIP (END LABEL)
     CALL SIN,                        LIBRARY TAPE CALL
     NAME SIN,S21                     LIBRARY SUB-RØUTINE NAME
     ZZZ  ALPHA                       INSTRUCTIØN BITS=0000
     ***  **                          WØRD TØ BE FILLED AT RUN TIME
```

Figure 3-1. Pseudo Operation Code Summary (Sheet 1 of 2)



ASSEMBLY CODING FORM

SEL

PROGRAMMER:

PROGRAM:

PAGE 2 OF 2

DATE

73 80 IDENTIFICATION

LOC. (1)	OPER. (6)	ADDRESS, INDEX (11)	(25)
	MØR		PAUSE WHEN ASSEMBLING
	END	STRT	END ØF PRØGRAM (SEE NOTE)
	END		END ØF SUB-RØUTINE
AD2	DAC	TIME,1	DIRECT ADDRESS CØNSTANT INDEXED
AD1	DAC*	LEVL	INDIRECT ADDRESS CØNSTANT
	EAC	MEM2	EXTENDED ADDRESS CØNSTANT
ADC	EAC	MEM2,1	EXTENDED ADDRESS CØNSTANT INDEXED

Note: STRT is address of first location to be executed.

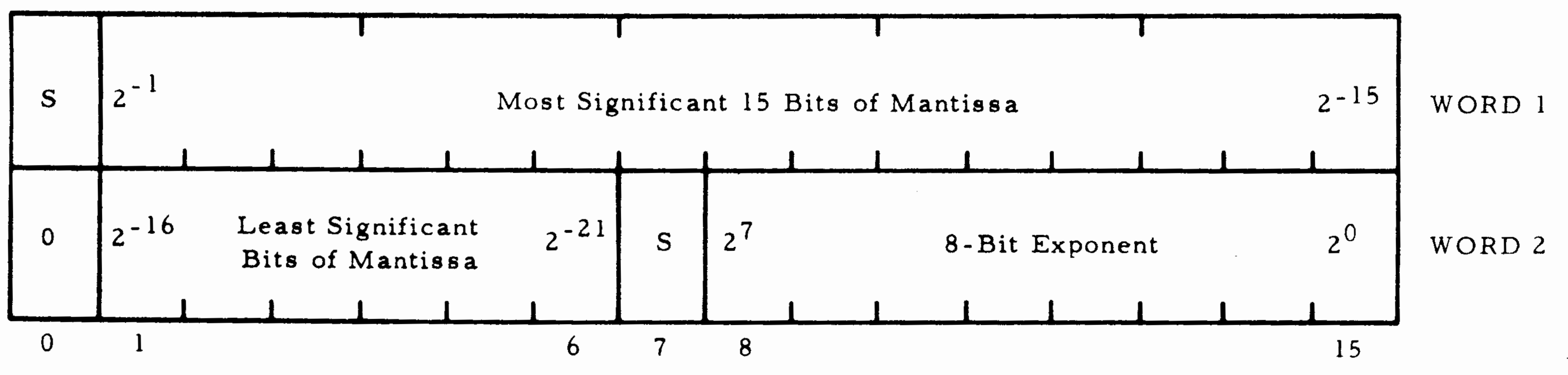
Figure 3-1. Pseudo Operation Code Summary (Sheet 2 of 2)

optional decimal point, the letter B, followed by a decimal number (SCALE FACTOR) between +15 and -15. Example: -3. 141B6. A minus sign will cause the 2's complement of the number to be stored. One word will be generated.

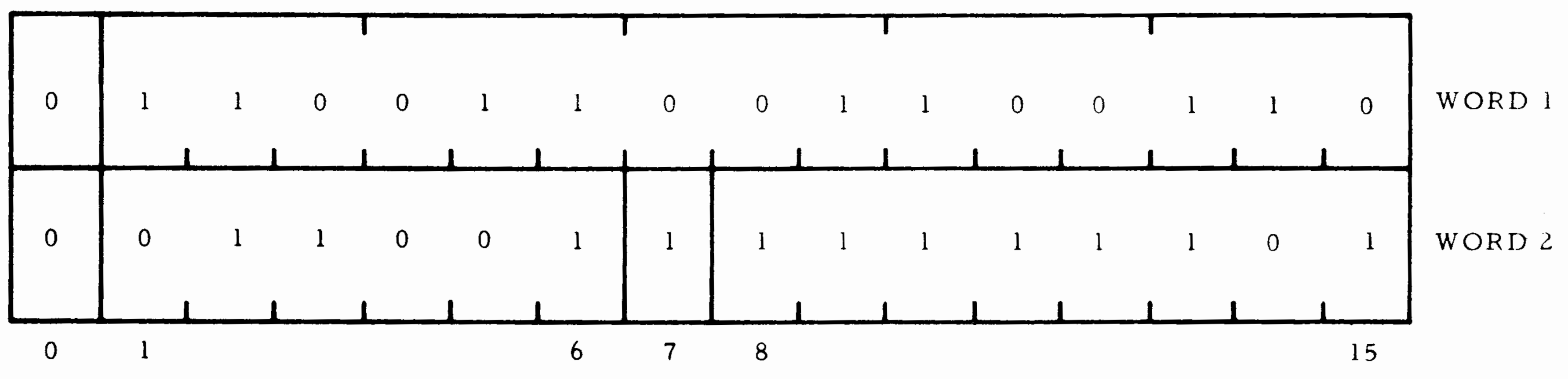
d. Fixed-Point Double Precision Data - Format: An optional sign (+ or -), 1 to 10 digits (/N/≤ 1073741823) mixed with an optional decimal point, the letter C, followed by a decimal number between +30 and -30. Example: 103. 637942C10. A minus sign will cause the 2's complement of the number to be stored. Two words will be generated.

e. Floating Point Data - Format: Written on the coding with optional sign (+ or -), 1 to 6 decimal digits (0 through 9), mixed with an optional decimal point, and optionally followed by a decimal exponent consisting of the letter E, preceding a decimal number between +75 and -75. (Either the decimal point, the letter E, or the sign of the exponent must be present. Two sequential memory cells are generated for each floating point data item using the following format.

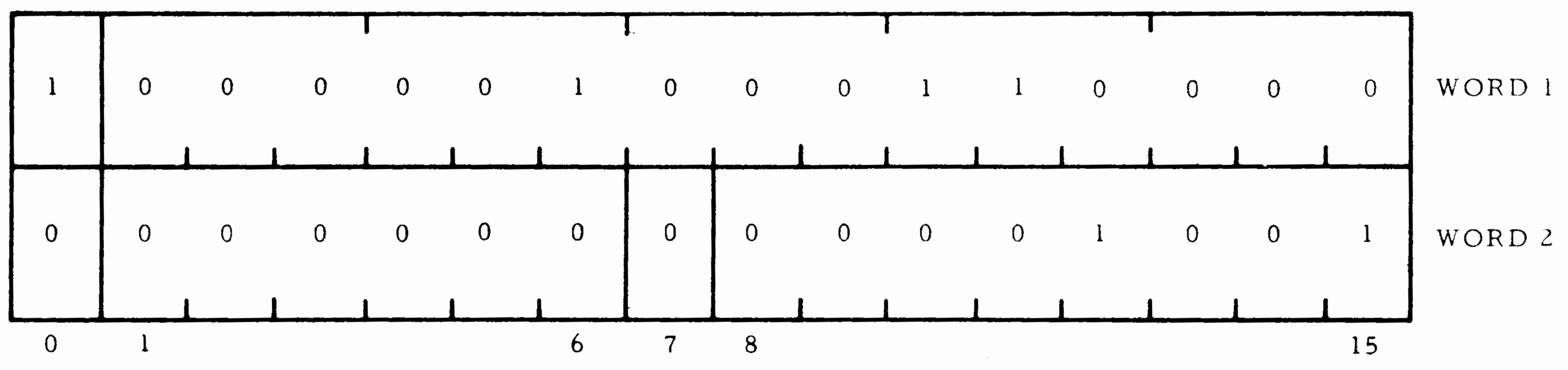
SINGLE-PRECISION FLOATING POINT DATA



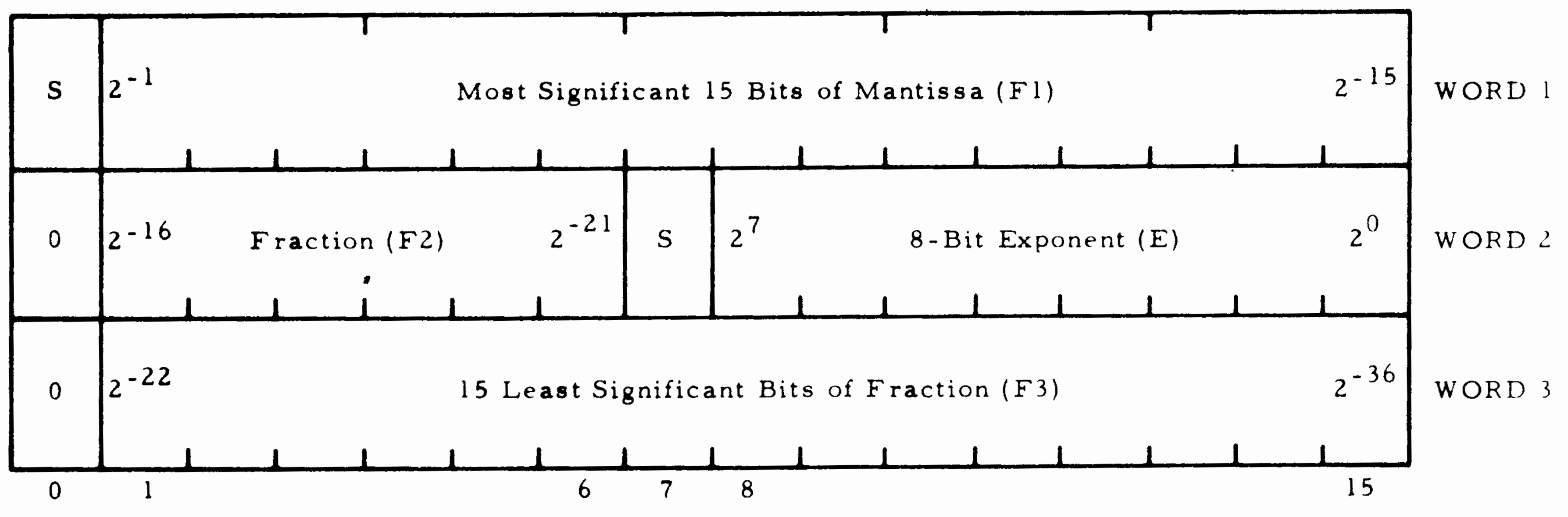
EXAMPLES:
0. 1 IS STORED AS:



-503. 25 IS STORED AS:



f. Floating Point Double Precision Data - Format: An optional sign (+ or -), 1 to 11 digits mixed with an optional decimal number between +75 and -75. Three sequential memory cells are generated for each double-precision floating point item using the following format:



E = Characteristic (2's complement if negative)
F1, F2, F3 = Double-Precision Fraction (2's complement if negative)

g. Alphanumeric Data - Format: Two apostrophe characters (' ') followed by any number of characters (including blanks) until another pair of apostrophes is read. The characters within the apostrophe pairs are stored 2 per word (last character left justified, if necessary).

EXAMPLE: "ALPHA TEST" is to be stored into memory starting at location 2000.

2000	1	100	000	111	001	100	AL
2001	1	101	000	011	001	000	PH
2002	1	100	000	110	100	000	Ab
2003	1	101	010	011	000	101	TE
2004	1	101	001	111	010	100	ST

b = space

3-5 The above example is in ASR-33 code (FULL ASCII code). This code will be used internally by the assembler to represent alphanumeric data. The I/O handling sub-routines will translate from external to internal code and vice versa when necessary depending upon the I/O device in use.

h. Symbolic Address Data - Format: Any symbolic address optionally followed by address arithmetic. The effective address will be stored in memory as a 15-bit address (similar to that generated by the EAC pseudo-op). The address may not be tagged as indexed or indirect.

FORM This pseudo-op is used to set up the format for the FDAT pseudo-op. There is no data generated and no memory locations aroused. This pseudo-op allows the programmer to define the bit assignments of 16-bit words generated by the FDAT statement. Up to 8 fields are allowed but the total number of bits must not exceed 16. All FDAT statements that follow a FORM will be in the same format until another FORM is encountered.

Example:

FORM 6, 4, 3, 1, 2

This assigns the FDAT bits as follows:

Field 1 - 6 bits (bits 0-5)
Field 2 - 4 bits (bits 6-9)
Field 3 - 3 bits (bits 10-12)
Field 4 - 1 bit (bit 13)
Field 5 - 2 bits (bits 14-15)

FDAT This pseudo-op is used to generate data in a format which has been previously defined by a FORM statement. The variable field for this instruction will accept decimal, octal, and alphanumeric data, but will mask off the most significant bits not defined by the previous FORM statement. Multiple FDAT statements may be placed on a card separated by slashes (/). If the location field contains a symbol, it will be assigned the location of the first data item. Example (using the FORM defined above):

FDAT "A", 8, 7, 0, 1/'75, '13, 4, 1, 3

This will generate the following two consecutive octal words:

'003071
'173347

BSS Block Start Symbol. Reserve a block of memory storage starting at the current location and extending for the number of words specified in the variable field. (If the variable field is symbolic, it must have been defined by a previous input line.) The location field is optional but if a symbol is inserted in this field, it will refer to the first word in the block.

BES Block End Symbol. Same as BSS except that if the location field is occupied, it refers to the last + 1 word in the block.

CALL CALL Library Tape. This pseudo-op will generate the necessary coding and actions to call in a sub-routine from a library tape into memory. The CALL pseudo-op is then replaced by a sub-routine transfer instruction (SPB) to this sub-routine. The variable field contains the sub-routine name. The location field, when occupied, refers to the resulting SPB instruction. Logic is contained within the loader to assure that only one copy of a sub-routine is called into memory from the library tape regardless of the number of CALL's for that sub-routine. The sub-routine's name must start with a letter and may contain from 1 to 6 characters. An equally good way to call external sub-routines would be with a leading dollar sign on the sub-routine's name.

EXAMPLES: SPB $SQRT

OR

CALL SQRT

NAME — NAME of Library Sub-routine. When writing sub-routines for inclusion into a library tape, the name by which the sub-routine must be called is specified by the NAME pseudo-op, followed by the specified name. The variable field consists of two symbolic names. The first is the name of the sub-routine and is 1 to 6 characters long (FORTRAN compatible). The second name is the symbolic entry location for the sub-routine and is 1 to 4 characters long, the first character being a letter. More than one NAME pseudo-op may be included in a sub-routine if alternate names for the sub-routine exist with either the same or different entry points. Also, external variables are defined by the NAME pseudo-op.

ZZZ — The instruction bits (0 to 3) are set to 0000. The rest of the instruction is determined by the variable field and the presence of an indirect indicator (*) following the pseudo-op (ZZZ*).

*** — Same as ZZZ but indicates the instruction will be filled at run time.

MOR — This pseudo-op causes a pause in the assembly process useful when the source program is on more than one tape, and a pause is needed to change tapes.

END — This pseudo-op must appear as the last instruction in any program or sub-routine being assembled and tells the assembly program that assembly is complete. If the variable field is not blank, it should specify the starting location of the program just assembled.

LIST — Set the mode of the symbolic output routine to list the output provided sense switch one is not ON. The assembler assumes the LIST mode until otherwise directed.

NOLS — Set the mode of the symbolic output routine to suppress the listing of output unless an error is detected. This pseudo operation remains in control until a LIST pseudo-operation is encountered.

NOTE

The LIST and NOLS statements are not printed as part of the listing output except when they appear within a MACRO prototype. The line count reflects the presence of the pseudo-ops even though they do not appear on the listing.

SECTION IV

MACRO SYSTEM

4-1 INTRODUCTION

4-2 The Macro System generates in-line coding according to the respective prototype and parameter list assigned to a given Macro call name. The general form of a Macro prototype is as follows:

Loc.	Oper.	Address, Index
NAME	MACR	A SET OF DETAIL STATEMENTS
	EMAC	

Columns 1-4	The call name of the Macro which can be any combination of legal characters, blanks included.
Column 5	Blank
Columns 6-9	To denote the beginning of a prototype code MAC or MACR. To denote the end of a prototype code EMA or EMAC.
Column 10	Blank
Columns 11-72	Can be used as comments.

NOTE

Do not use an END or a $ end of job statement in a Macro prototype.

4-3 MACRO PROTOTYPE

4-4 The prototype is a set of detail statements which can contain elements to be supplied either internally or from a list of parameters. Elements are of three basic types as follows.

- Internal to a given Macroprototype
- Parameter supplied by user
- Fixed element name

4-5 The internal assignment applies only to labels and must be of the form @X where the at sign (ASCII 300) must be the first character of the label. The X is a decimal value from one through 16 and can be assigned in any order (not necessarily monotonically) per Macro. Leading zeros are suppressed, @009 is the same as @9. Each call of the same Macro which contains internal labels will generate a unique respective set increased by the last assembler assigned label plus one. The assembler will not allow more than 999 internal labels to be generated. All assignments in excess of 999 will be flagged as an error.

4-6 Example of internal label:

Loc.	Oper.	Address, Index
WAIT	MACR	NAME AND BEGIN PROTOTYPE
@1	NOP	INTERNAL LABEL FIXED OP
	NOP	FIXED OPERATION CODE
	NOP	FIXED OPERATION CODE
	BRU @1	FIXED OP CODE, INTERNAL LABEL
	EMAC	END OF WAIT PROTOTYPE

4-7 Every call to WAIT will generate @1 into a unique label for each wait loop.

4-8 The user supplied parameter can apply to any field of a valid assembler statement. The form of a user parameter is #X where the number sign (ASCII 243) may appear anywhere in a label or value to be specialized and must be immediately followed by a decimal value from one through 16 representing the correct parameter number to be concatenated into the generated element. Leading zeros are suppressed on parameter number assignments. Parameters which are requested but omitted from the list are replaced by a single blank character. Parameter numbers in excess of 16 will not be processed and will be flagged as an error.

4-9 An example of user supplied parameters is:

LOC	OPER	VARIABLE
FILL	MACR	
	LAA	=#1 Character to fill area

```
        LBA      =-#2      Size of area

        STA      #3+#2, 1  Area plus size minus
                           index

        IBS                Increment index

        BRU      *-2       Loop to fill area

        EMAC
```

4-10 To use the Macro, enter an M in column 5 of the coding sheet, followed by the prototype name in columns 6-9 and the parameter list starting in column 11, e. g. , the general form of the call for FILL is

MFIL DATA, SIZE, AREA,

so we call FILL by

```
LOC     OPER     VARIABLE

X    M  FILL     '240, 80, TABL,
```

which will fill 80 locations of the buffer TABL with right justified ASCII spaces.

4-11 Parameter elements in the main program call list are separated by a single level of delimiter which can be a comma, left parenthesis, or right parenthesis. The parameter list may be terminated by on of the three delimiters. Extra sets of parenthesis can be added for clarity but each must be counted when assigning values to the elements of a detail entry. Elements can be assigned in any order or any set of digits provided a parameter exist for the desired elements.

Example: To use even numbered parameters only.

```
LOC     OPER     VARIABLE

PAR     MACR

        DATA     #2, #4, #6, #8, #10

        EMAC

          .

          .

          .

          .

     M  PAR      (13)(6)("A")(3. 14159B3)('377)

or   M  PAR      , 13, , 6, , "A", , 3. 14159B3, , '377
```

which is not the same as

```
M  PAR      13, 16; "A", 3. 14159B3, '377,
```

The second call requires a prototype like so:

```
LOC     OPER     VARIABLE

PAR     MACR

        DATA     #1, #2, #3, #4, #5

        EMAC
```

NOTE

The delimiters may not be used as data in a parameter list needed, supply them as octal data. If a comma and/or parentheses are needed, supply them as octal data.

4-12 The fixed element name is any field in which the detail statement supplies the value, operation code, operand or any portion of a statement.

Example:

```
LOC     OPER     VARIABLE

X       MACR

        CLA

        LBA      =1    Load the accumulators
                       with a double precision
                       constant (1).

        EMAC
```

To use this Macro we need only to write:

```
LOC     OPER     VARIABLE

    M   X
```

4-13 Comments may be entered in a prototype; however, only the asterisk and the next 24 positions will be retained when the prototype is specialized. If a detail line has comments as a continuation of a statement, they will not be processed at the time of specialization.

4-14 The MACRO storage area is normally 700_{10} words with a name table capacity of 30 names. The formula for computing the exact number of words needed to store a prototype is as follows: Sum of words for each statement +1.

4-15 The words for each statement = 1+ (number of characters in location field + number of characters in op code field + number of characters in variable field) ÷ 2 + 0 if the remainder is 1, 1 if the remainder is 0. Count internal and parameter supplied labels as 2, that is, #003 + #02 + 6 is counted as 7 characters. A general safe rule-of-thumb would be to multiply the number of detail lines by 5 to obtain the storage requirements for a Macro prototype.

SECTION V
ASSEMBLER OPERATING INFORMATION

5-1 INTRODUCTION

5-2 This section contains a description of the Assembler Symbolic Listing Format. Also included is a sample source program written on the coding form plus a description and output listing of the same program after it has been assembled. Procedures for source program preparation and assembly are referenced.

5-3 COMPUTER CONFIGURATION

5-4 The assembler program will operate within a minimum SEL 810A/810B Computer system: 4096 words of memory for the SEL 810A and 8192 for the SEL 810B.

5-5 MODE OF OPERATION

5-6 The SEL 810A/810B program operate in the two pass mode.

5-7 TWO-PASS MODE(PREFERRED)

5-8 In the two-pass mode of assembly, the source program is read twice. During the second reading, a symbolic listing is generated complete with octal equivalences and error messages. Also, an object output tape is produced during the second reading which represents the assembled program in a binary relocatable format acceptable to the Loader Program.

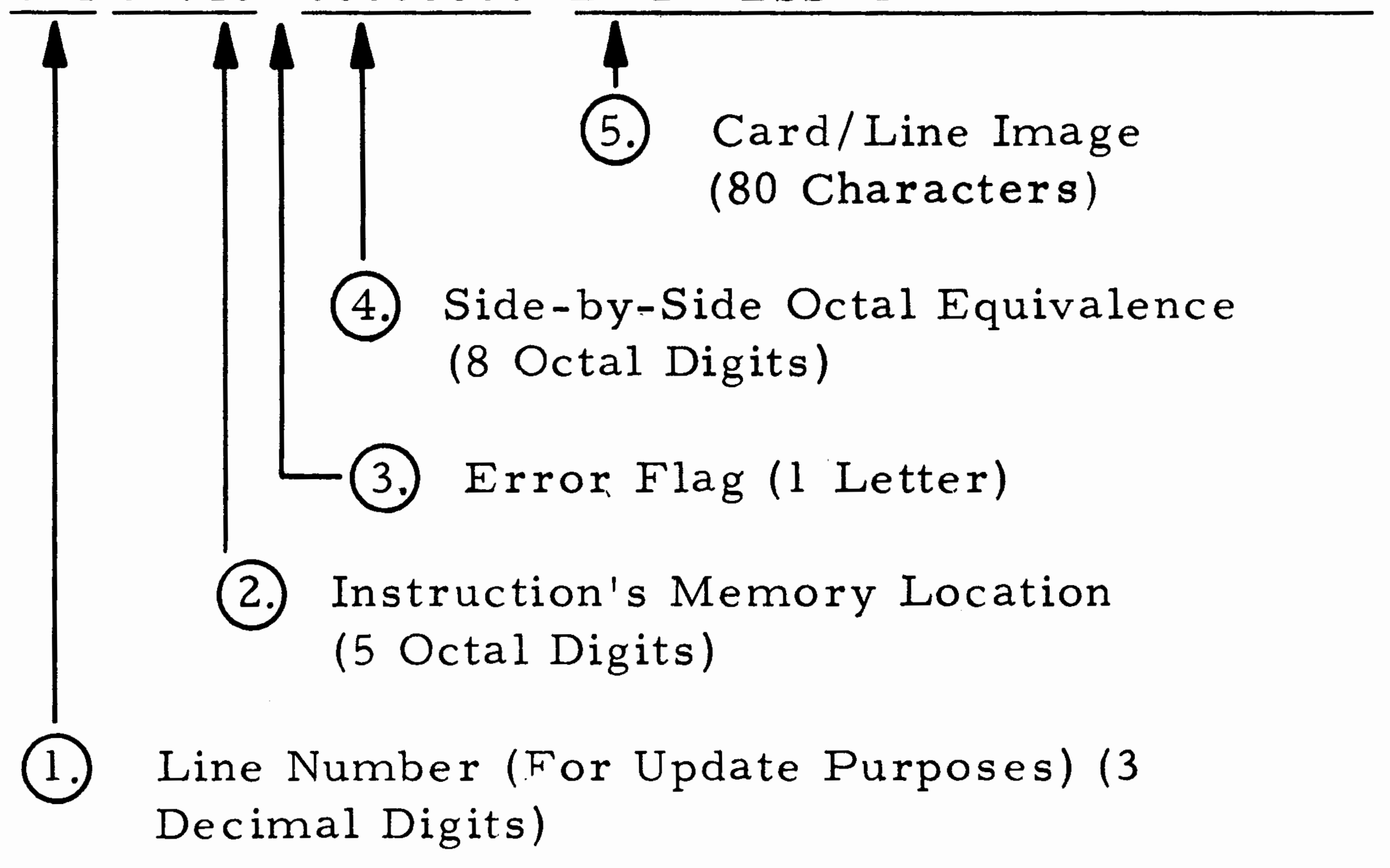
NOTE

The side-by-side octal listing is complete, all error messages are printed with the line they effect, and the object output tape is approximately 30 percent shorter than that produced by the one-pass mode of assembly (although the memory requirement for the program is the same). The source tape or deck must be read twice for each assembly (two-pass).

5-9 SYMBOLIC LISTING FORMAT

5-10 The symbolic listing is optional and when requested, is made on the console typewriter or line printer (device determined by sense switch setting). A typical line has the following format:

```
016 01733M01301745 ALPH LAA*DATA TEST FLAG
017 01734  00000610      RSA  6      NEXT TEST
018 01735  11101360      BRU  AL30   RESTART
019 01736  00000001      DATA 1, 2, 3SWITCHES
020 01737  00000002
021 01740  00000003
022 01741  00000004 LIST  BSS  4      LIST
023 01745  00101733 DATA  DAC  ALPH, 1 COUPLING
024 01746  00000000 DATA  BSS  1
```


5. Card/Line Image (80 Characters)
4. Side-by-Side Octal Equivalence (8 Octal Digits)
3. Error Flag (1 Letter)
2. Instruction's Memory Location (5 Octal Digits)
1. Line Number (For Update Purposes) (3 Decimal Digits)

(1.) LINE NUMBER - This decimal number can be referred to when updating a source tape using the Update Program. (See Appendix D-1).

(2.) MEMORY LOCATION - This octal number represents either the actual (or relative) memory location where the assembled instruction will be stored.

(3.) ERROR FLAG - (See Paragraph 4. 8 for interpretation). The letter represents a suspected error situation within the line.

(4.) OCTAL INSTRUCTIONS - The octal listing may have any of five following formats:

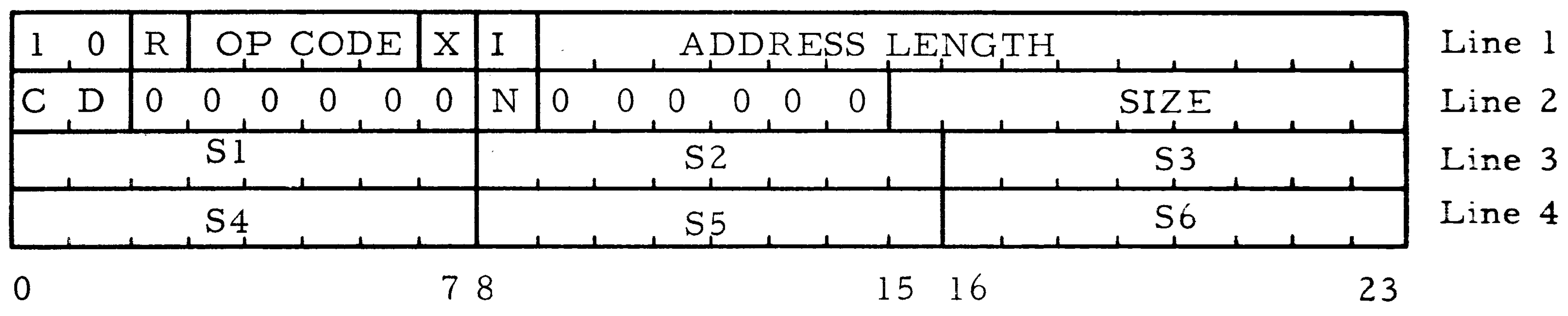
A. These four provide information to be loaded into memory.

oocaaaaa	Assembled instruction usually resulting from a DATA or EQU pseudo-operation instruction where "caaaaa" is any octal number with "c"

being one or zero. The "caaaaa" portion of the listing will be loaded into memory exactly as it appears.

abcddddd	Assembled instruction resulting from a DAC or EAC pseudo-operation, with "a" setting the mode (ABS or REL) and "b" indicating DAC or EAC. "c" indicates indexing and indirect addressing. The address is indicated by "ddddd".
ooobbxx	Assembled augment instruction where "bb" is up to 6 augmenting bits and "xx" is the operation code. The instruction will appear in memory as "oobbxx".
xxyzzzzz	Assembled memory referencing instruction where "xx" is the operation code, "y" is index, indirect and relocation bits. "zzzzz" is the relative address except where literal addressing is used. In that case, "zzzzz" is the octal equivalent of the literal address. The "xx" portion of the listed instruction is the only part which will always appear the same in memory. The other portions will be altered depending on how the loader must load them. (See Appendix D.)

B. The following two formats provide information to the loader only, with one exception, i. e., CALL, which will generate an SPB and load it into memory.

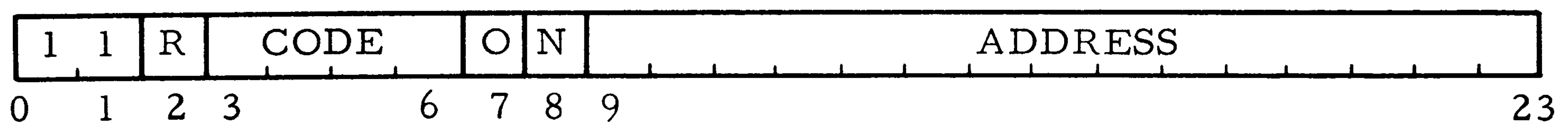


CD = 00: Subroutine definition (NAME)
address = relative entry point

CD = 10: Common definition
address = length

CD = 11: Common request
address = relative to block
N = negation flag

CD = 01: Subroutine call (CALL)
Address -0



Code = 00, Establish Load Point (ORG)
= 01, END Jump (END)
= 02, STRING
= 03, 9-bit ADD-TO (from one-pass assembly)
= 04, 14-bit ADD-TO (DAC) (from one-pass assembly)
= 05, 15-bit ADD-TO (EAC) (from one-pass assembly)
= 06, Turn on CHAIN flag
= 07, Turn on LOAD flag
= 10, END-OF-JOB ($)

(5) CARD/LINE IMAGE - The complete input line will be listed in its original format, if output is on the line printer, or the first 50 characters of the input line will be output if output is on the typewriter.

5-11 PAPER TAPE PREPARATION OF SOURCE PROGRAM

5-12 Paper tape source programs may be prepared using the Console Keyboard in the following manner:

Step 1. Place the Console Keyboard in the OFF-LINE mode.

Step 2. Depress the HERE IS key on the Console Keyboard (provides leader for the Paper Tape

Punch). Depress as many times as necessary to get the required length of leader.

Step 3. Type in the instructions and data words from an ASSEMBLER SOURCE PROGRAM coding form. Include comments if so desired. Blank spaces must be punched everywhere a blank is shown except after last character in comments field.

LOC. 1	OPER. 6	ADDRESS, INDEX 11
	END	
$		

NOTE

The last character on the paper tape should be a "FORM" character (\); this character is used by the UPDATE Program as TERMINATOR.

Punch the tape utilizing the standard Systems Engineering Laboratories, Incorporated coding form.

5-13 PUNCH CARD PREPARATION OF SOURCE PROGRAM*

5-14 Punch card source programs may be prepared for the SEL 810A computer using an off-line 80 column card keypunch. Punch the cards from the ASSEMBLER coding form utilizing the card columns as shown on the standard Systems Engineering Laboratories, Incorporated Assembler coding form. Use one card for each line of coding.

To assemble using a punched card source program the following steps are necessary:

Step 1. Load the Assembler Program. (See Appendix B.)

Step 2. Place the symbolic punched card deck in the optional Systems Engineering Laboratories, Incorporated Card Reader.

* Optional SEL Model 80-410 Card Reader Required.

Step 4. Depress the CARRIAGE RETURN (CR) and LINE FEED after each statement line. These must be punched or typed in this order.

Step 5. Complete preparation of the source program.

Step 6. The last two instructions on the ASSEMBLER coding form must be as follows:

FOLLOWED BY CARRIAGE RETURN AND LINE FEED

FOLLOWED BY CARRIAGE RETURN AND LINE FEED

Step 3. Set the proper Sense Switches. (See Appendix B.)

NOTE

Output will be a punched paper tape. The control cards are:

The last two cards of a punched card source program must be an END pseudo-op in columns 6-8 followed by a card with a dollar sign ($) punched in column 1.

5-15 DELETION OF ERRORS ON THE SOURCE TAPE OR SOURCE DECK

5-16 Errors encountered during punching of the original source input tape may be deleted if discovered before punching the LINE FEED and CARRIAGE RETURN for that statement line by depressing the "UP ARROW" (↑), followed by CARRIAGE RETURN and LINE FEED. This will cause the entire statement line, including the "UP ARROW", to be ignored by the ASSEMBLER. If the "UP ARROW" is not followed by a CARRIAGE RETURN and a LINE FEED, the entire line will be processed by the ASSEMBLER as an error. The correct line must be the next information punched on tape.

Errors in the Source Card Deck may be corrected by removing the error card and replacing with the correct card.

After assembly, errors in the source tape may be corrected using the UPDATE Program. (See UPDATE Program, Appendix D.)

5-17 ERROR MESSAGES

5-18 The error flag letters given in the symbolic listing have the following meaning:

U - -	Undefined Symbol
M - -	Multiple defined symbol
A - -	Address field missing when required
S - -	Use of MAP in one-pass mode or with program over 512 words
Q - -	Undefined operation name
D - -	Data conversion has exceed limits
T - -	Assignment table full
E - -	Any other type of detected error

Errors will cause the affected fields to be set to zero. More than one error may be detected for each line but only the last error is flagged in the listing.

NOTE

See Appendix C for examples of Error messages.

5-19 ASSIGNMENT TABLE SIZE

5-20 The assignment table contains all symbols defined in the program being assembled plus all literal constants, subroutine names, and modifier instructions. For a 4096 word memory the number of entries is approximated by:

$$\underline{400 = S + L + C + M}$$

S = No. of symbolic addresses

L = No. of unique literal constants defined

C = No. of unique subroutine names called

M = No. of times an "undefined" symbol appears in the Variable field of an instruction combined with constants by a + or -.

If the computer configuration is greater than 4K, the assignment table can be easily expanded and the following equation holds:

$$400 + 1365 (N-1) = S + L + C + M$$

where

N = Number of 4096 word memories

SECTION VI
SAMPLE ASSEMBLER PROGRAM

6-1 INTRODUCTION

6-2 This section contains a sample program using the symbolic coding of the SEL 810A computer. It is intended to assemble this program using the SEL 810A/810B Assembler. The program is a sample only to demonstrate the use of the Assembler. The sample coding can be punched either on paper tape or cards for input during the assembly. It may be run in either the one-pass or two-pass mode.

6-3 PROGRAM INPUT

6-4 The programming example is intended to be simple so that only use of the Assembler may be demonstrated. There is no attempt to teach programming.

The example will order constants from different storage locations, sum them and restore them in different memory locations.

6-5 PROGRAM CODING

6-6 The coding sheets are shown in figure 6-1.

6-7 ASSEMBLER OUTPUT

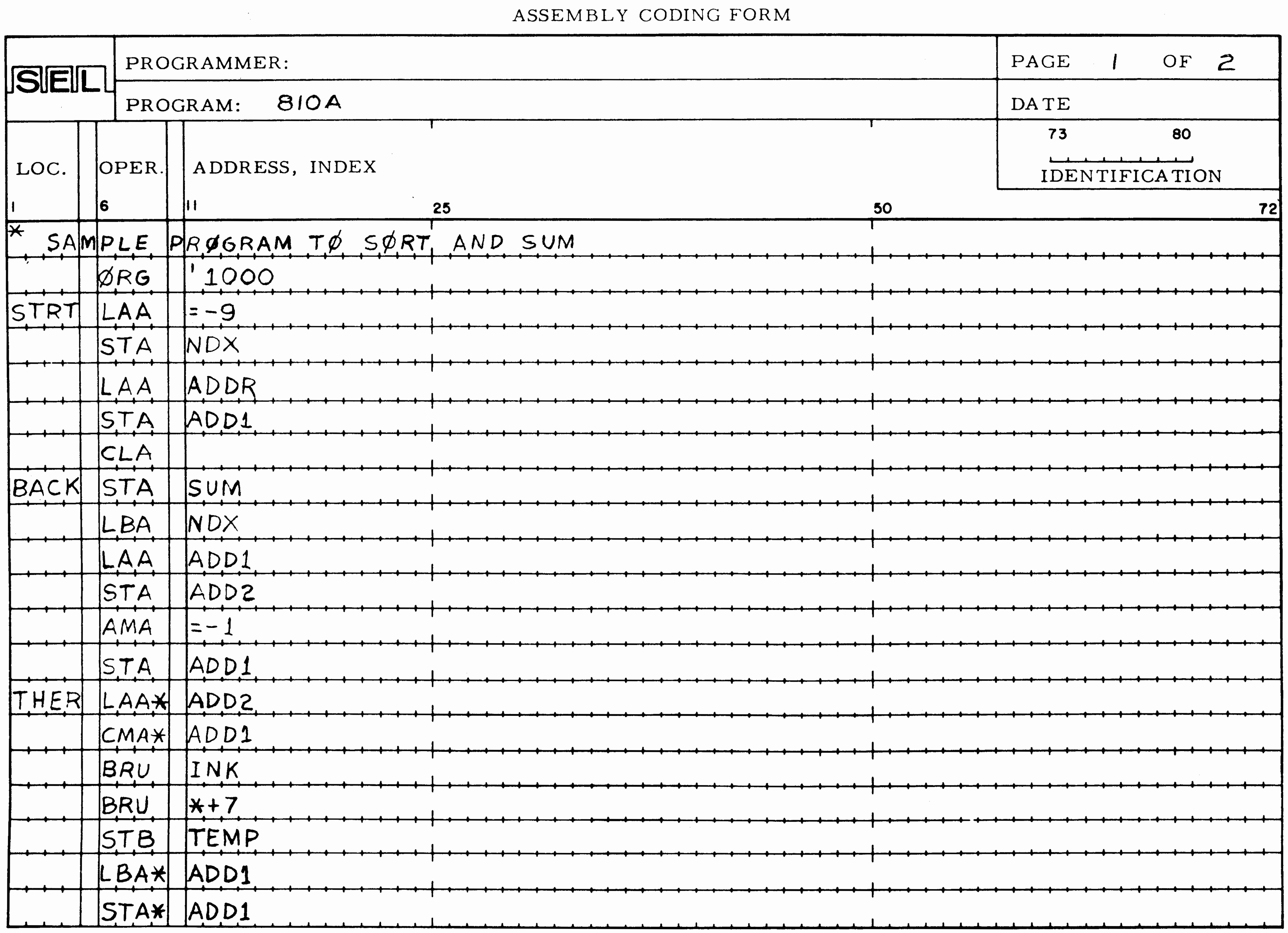
6-8 The machine code is shown along with its symbolic code in table 6-1 Assembler output.

Table 6-1. Assembler Output

```
0001   00000   00000000               *  Sample Program To Sort And Sum
0002   01000   60001000                  ORG     '1000
0003   01000   01077767   STRT           LAA     =-9
0004   01001   03001040                  STA     NDX
0005   01002   01001035                  LAA     ADDR
0006   01003   03001036                  STA     ADD1
0007   01004   00000003                  CLA
0008   01005   03001041   BACK           STA     SUM
0009   01006   02001040                  LBA     NUX
0010   01007   01001036                  LAA     ADD1
0011   01010   03001037                  STA     ADD2
0012   01011   05077777                  AMA     =-1
0013   01012   03001036                  STA     ADD1
0014   01013   01201037   THER           LAA*    ADD2
0015   01014   15201036                  CMA*    ADD1
0016   01015   11001024                  BRU     INK
0017   01016   11001025                  BRU     *+7
0018   01017   04001042                  STB     TEMP
0019   01020   02201036                  LBA*    ADD1
0020   01021   03201036                  STA*    ADD1
0021   01022   04201037                  STB*    ADD2
0022   01023   02001042                  LBA     TEMP
0023   01024   00000026   INK            IBS
0024   01025   11001013                  BRU     THER
0025   01026   01201036                  LAA*    ADD1
0026   01027   05001041                  AMA     SUM
0027   01030   14001040                  IMS     NDX
0028   01031   11001005                  BRU     BACK
0029   01032   05001043                  AMA     DATA
0030   01033   03001041                  STA     SUM
0031   01034   00000000                  HLT
0032   01035   25601055   ADDR           DAC     DATA + 10, 1
0033   01036   25400000   ADD1           DAC     **
```

Table 6-1. Assembler Output (Cont'd)

```
0034     01037     25400000     ADD2      DAC      **
0035     01040A    00000000     NUX       ***
0036     01041     00000001     SUM       BSS      1
0037     01042     00000001     TEMP      BSS      1
0038     01043     00050000     DATA      DATA     5B3,'1234, 3. 142C12, -1, 2. E5,
                                                   ''ABCD''
0038     01044     00001234
0038     01045     00000031
0038     01046     00010550
0038     01047     00177777
0038     01050     00060650
0038     01051     00000022
0038     01052     00140702
0038     01053     00141704
0039     01054     60401000               END      STPT
```

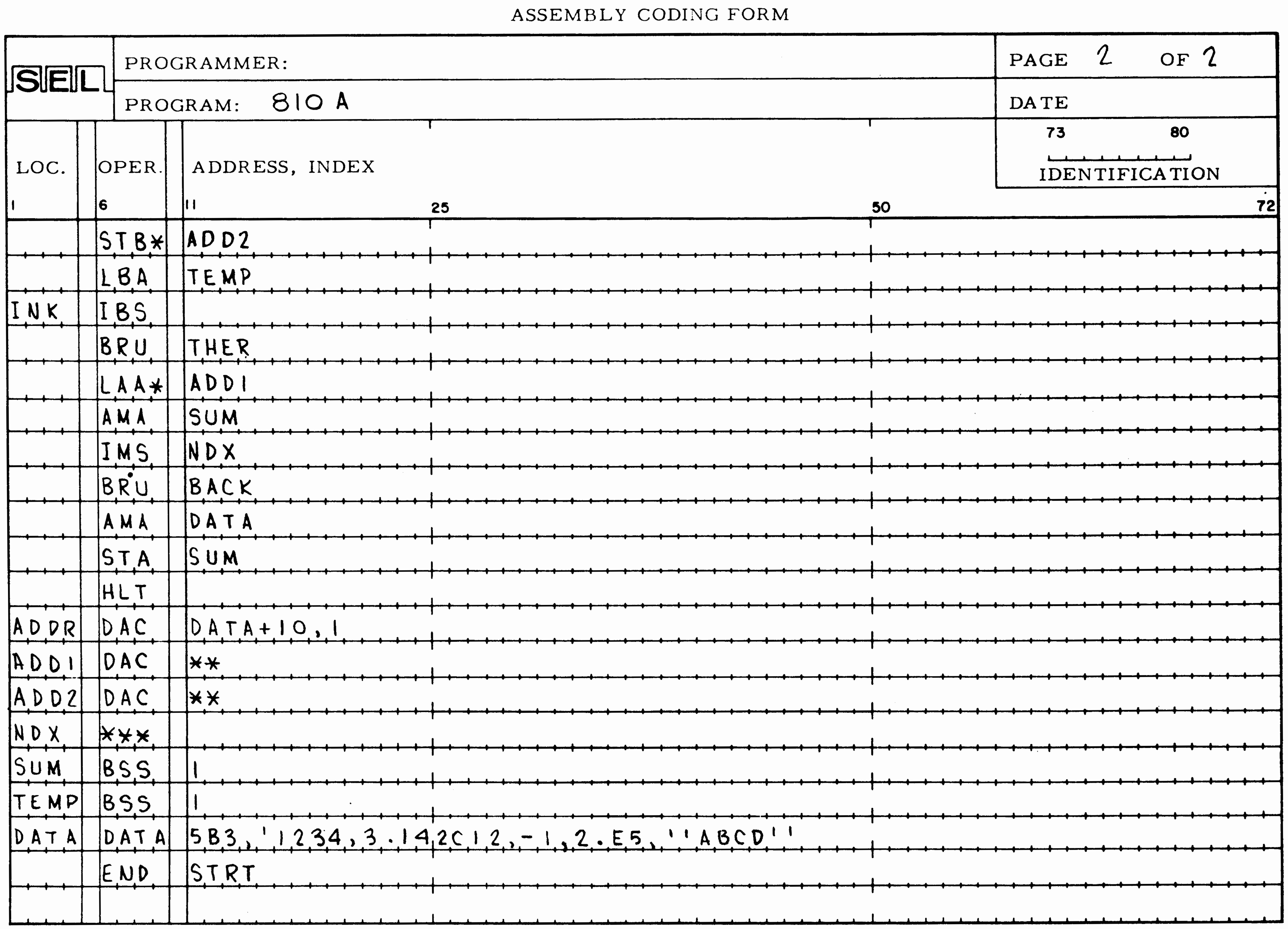


ASSEMBLY CODING FORM

SEL	PROGRAMMER:	PAGE 1 OF 2
	PROGRAM: 810A	DATE

```
LOC.  OPER.  ADDRESS, INDEX                                      IDENTIFICATION
1     6      11           25                      50          72  73        80
*   SAMPLE PRØGRAM TØ SØRT AND SUM
      ØRG    '1000
STRT  LAA    =-9
      STA    NDX
      LAA    ADDR
      STA    ADD1
      CLA
BACK  STA    SUM
      LBA    NDX
      LAA    ADD1
      STA    ADD2
      AMA    =-1
      STA    ADD1
THER  LAA*   ADD2
      CMA*   ADD1
      BRU    INK
      BRU    *+7
      STB    TEMP
      LBA*   ADD1
      STA*   ADD1
```

Figure 6-1. Sample Assembler Program



ASSEMBLY CODING FORM

SEL

PROGRAMMER:

PROGRAM: 810 A

PAGE 2 OF 2

DATE

73 80
IDENTIFICATION

LOC. 1	OPER. 6	ADDRESS, INDEX 11
	STB*	ADD2
	LBA	TEMP
INK	IBS	
	BRU	THER
	LAA*	ADD1
	AMA	SUM
	IMS	NDX
	BRU	BACK
	AMA	DATA
	STA	SUM
	HLT	
ADDR	DAC	DATA+10,1
ADD1	DAC	**
ADD2	DAC	**
NDX	***	
SUM	BSS	1
TEMP	BSS	1
DATA	DATA	5B3,'1234,3.14,2C12,-1,2.E5,''ABCD''
	END	STRT

Figure 6-1. Sample Assembler Program (Continued)

APPENDIX A

SEL 810A/810B INSTRUCTION LIST SUMMARY

CLASS	MNEMONIC	OP CODE	FUNCTION
ARITHMETIC:	AMA	05	ADD MEMORY TO A
	AMB	16	ADD MEMORY TO B
	SMA	06	SUBTRACT MEMORY FROM A
	MPY	07	MULTIPLY B TIMES MEMORY
	DIV	10	DIVIDE A AND B BY MEMORY
	RNA'	00-01	ROUND A BY MSB IN B
	●OVS'	00-37	SET OVERFLOW
LOAD/STORE:	LAA	01	LOAD A FROM MEMORY
	LBA	02	LOAD B FROM MEMORY
	STA	03	STORE MEMORY FROM A
	STB	04	STORE MEMORY FROM B
	LCS'	00-31	LOAD CONTROL SWITCHES IN A
	●LIX'	00-45	LOAD INDEX REGISTER
	●STX'	00-44	STORE INDEX REGISTER
BRANCH/SKIP:	BRU	11	UNCONDITIONAL BRANCH
	SPB	12	STORE PLACE AND BRANCH
	SNS	13-4	SKIP IF CONTROL SWITCH NOT SET
	IMS	14	INCREMENT MEMORY AND SKIP IF 0
	CMA	15	COMPARE MEMORY AND A (3 WAY) n + 1 if (A)< (M) n + 2 if (A)=(M) n + 3 if (A)>(M)
	IBS'	00-26	INCREMENT B(INDEX) AND SKIP IF 0
	SAZ'	00-22	SKIP IF A IS ZERO
	SAP'	00-24	SKIP IF A IS POSITIVE
	SAN'	00-23	SKIP IF A IS NEGATIVE
	SOF'	00-25	SKIP NO OVERFLOW
	SAS'	00-21	SKIP ON A SIGN (3 WAY) n + 1 (-), n + 2 (0), n + 3(+)
	SNO'	00-32	SKIP IF A IS NORMALIZED
	LOB'	00-36	LONG BRANCH
	●SXB'	00-50	SKIP IF INDEX POINTER IS SET TO B
	●IXS'	00-51	INCREMENT INDEX A AND SKIP IF ≥ 0
LOGICAL:	ABA'	00-27	AND A AND B
	OBA'	00-30	OR AND B
	NEG'	00-02	NEGATE A
	ASC'	00-20	COMPLEMENT A SIGN
	CNS'	00-34	CONVERT NUMBER SYSTEM

CLASS	MNEMONIC	OP CODE	FUNCTION
REGISTER CHANGE:	CLA'	00-03	CLEAR A
	TAB'	00-05	TRANSFER A TO B
	IAB'	00-06	INTERCHANGE A AND B
	CSB'	00-07	TRANSFER B SIGN TO CARRY AND CLEAR B SIGN TO POSITIVE
	TBA'	00-04	TRANSFER B TO A
	●TBP	00-40	TRANSFER B-ACCUMULATOR TO PROTECT REGISTER
	●TPB	00-41	TRANSFER PROTECT REGISTER TO B-ACCUMULATOR
	●TAX'	00-52	TRANSFER A-ACCUMULATOR TO HARDWARE INDEX REGISTER
	●TXA'	00-53	TRANSFER HARDWARE INDEX REGISTER TO A-ACCUMULATOR
	●XPX'	00-46	SET INDEX POINTER TO INDEX REGISTER
	●XPB'	00-47	SET INDEX POINTER TO B-ACCUMULATOR
	TBV'	00-42	TRANSFER B-ACCUMULATOR TO VBR
	TVB'	00-43	TRANSFER VBR TO B-ACCUMULATOR
SHIFT:	RSA'	00-10	RIGHT SHIFT A
	LSA'	00-11	LEFT SHIFT A
	FRA'	00-12	RIGHT SHIFT A AND B
	FLA'	00-17	LEFT SHIFT A AND B
	RSL'	00-15	RIGHT LOGICAL SHIFT A
	FRL'	00-14	FULL ROTATE LOGICAL A AND B LEFT
	LSL'	00-16	LEFT LOGICAL SHIFT A
	FLL'	00-13	LEFT LOGICAL SHIFT A AND B
CONTROL:	HLT'	00-00	HALT
	NOP'	00-33	NO OPERATION
	TOI'	00-35	TURN OFF INTERRUPT
	PIE'	130600	ENABLE INTERRUPT
	PID'	130601	DISABLE INTERRUPT
	●●PON'	002040	PROTECT BIT ON
	●●POF'	002041	PROTECT BIT OFF
INPUT/OUTPUT	CEU'	13. 0IM. 0	COMMAND EXTERNAL UNIT
	TEU'	13. 0IM. 2	TEST EXTERNAL UNIT
	AOP'	1700	ACCUMULATOR WORD OUT TO UNIT
	AIP'	1702	ACCUMULATOR WORD IN FROM UNIT
	MOP'	17. 0IM. 4	MEMORY WORD OUT TO UNIT
	MIP'	17. 0IM. 6	MEMORY WORD IN FROM UNIT

NOTES:

(a) ● = 810B only

(b) ●● = 810A only

(c) ' = Augmented

(d) Underlined instructions require optional hardware.

CLASS	MNEMONIC	OP CODE	FUNCTION
REGISTER CHANGE:	CLA'	00-03	CLEAR A
	TAB'	00-05	TRANSFER A TO B
	IAB'	00-06	INTERCHANGE A AND B
	CSB'	00-07	TRANSFER B SIGN TO CARRY AND CLEAR B SIGN TO POSITIVE
	TBA'	00-04	TRANSFER B TO A
	●<u>TBP</u>	00-40	TRANSFER B-ACCUMULATOR TO PROTECT REGISTER
	●<u>TPB</u>	00-41	TRANSFER PROTECT REGISTER TO B-ACCUMULATOR
	●<u>TAX'</u>	00-52	TRANSFER A-ACCUMULATOR TO HARDWARE INDEX REGISTER
	●<u>TXA'</u>	00-53	TRANSFER HARDWARE INDEX REGISTER TO A-ACCUMULATOR
	●<u>XPX'</u>	00-46	SET INDEX POINTER TO INDEX REGISTER
	●<u>XPB'</u>	00-47	SET INDEX POINTER TO B-ACCUMULATOR
	<u>TBV'</u>	00-42	TRANSFER B-ACCUMULATOR TO VBR
	<u>TVB'</u>	00-43	TRANSFER VBR TO B-ACCUM-ULATOR
SHIFT:	RSA'	00-10	RIGHT SHIFT A
	LSA'	00-11	LEFT SHIFT A
	FRA'	00-12	RIGHT SHIFT A AND B
	FLA'	00-17	LEFT SHIFT A AND B
	RSL'	00-15	RIGHT LOGICAL SHIFT A
	FRL'	00-14	FULL ROTATE LOGICAL A AND B LEFT
	LSL'	00-16	LEFT LOGICAL SHIFT A
	FLL'	00-13	LEFT LOGICAL SHIFT A AND B
CONTROL:	HLT'	00-00	HALT
	NOP'	00-33	NO OPERATION
	TOI'	00-35	TURN OFF INTERRUPT
	PIE'	130600	ENABLE INTERRUPT
	PID'	130601	DISABLE INTERRUPT
	●●PON'	002040	PROTECT BIT ON
	●●POF'	002041	PROTECT BIT OFF
INPUT/OUTPUT	CEU'	13. 0IM. 0	COMMAND EXTERNAL UNIT
	TEU'	13. 0IM. 2	TEST EXTERNAL UNIT
	AOP'	1700	ACCUMULATOR WORD OUT TO UNIT
	AIP'	1702	ACCUMULATOR WORD IN FROM UNIT
	MOP'	17. 0IM. 4	MEMORY WORD OUT TO UNIT
	MIP'	17. 0IM. 6	MEMORY WORD IN FROM UNIT

NOTES:

(a) ● = 810B only

(b) ●● = 810A only

(c) ' = Augmented

(d) Underlined instructions require optional hardware.

APPENDIX B
LOADING THE ASSEMBLER

B. 1 INTRODUCTION

The SEL 810A/810B ASSEMBLER PAPER TAPE PROGRAM is in Relocatable Format. The ASSEMBLER must be loaded with the Relocatable LOADER as described in Section IV of the SEL 810A OPERATING INSTRUCTION MANUAL. If the LOADER is not resident in memory prior to assembly, follow these initial procedures:

1. Load into memory a binary non-relocatable paper tape of the LOADER.

The operating procedures to load a binary non-relocatable paper tape, using either the MANUAL BOOTSTRAP or the BINARY PAPER TAPE READ PROGRAM, may be found in Section 3. 3 of the OPERATING INSTRUCTION MANUAL.

2. The ASSEMBLER has an origin address of 00000; the operating procedures assume this origin. A new origin must be entered into the PROGRAM COUNTER if the ASSEMBLER is relocated.

B. 2 ASSEMBLING A PAPER TAPE SOURCE PROGRAM

Step 1. Turn the Console Keyboard LINE, OFF, LOCAL switch to the LINE position.

Step 2. Place the Paper Tape Reader START, STOP, FREE toggle switch in the START position.

Step 3. Place the paper tape source program in position on the Paper Tape Reader.

Step 4. Raise the appropriate SENSE switches on the Console Control Panel to designate mode of assembly and I/O devices to be used.

Step 5. Choose assembly mode ---

a. TWO-PASS MODE---FOLLOW STEPS 6-10.

Step 6. If a TWO-PASS assembly mode is designated, ready the Console Paper Tape Punch by depressing the "ON" button located on the punch unit.

Step 7. Raise NO PROGRAM ADVANCE, depress START.

NOTE

IF A TWO PASS ASSEMBLY IS DESIGNATED, the assembly will HALT the computer upon recognizing the "END" punched in the paper tape source program.

Step 8. Reposition the source program paper tape into the Paper Tape Reader to allow the ASSEMBLER to complete PASS TWO of the assembly.

Step 9. Depress START toggle on the Console Control Panel.

B. 3 SELECTION OF ASSEMBLER OPTIONS BY SENSE SWITCH SETTINGS

The many options available for use by the ASSEMBLER are controlled by the SENSE SWITCH settings on the SEL 810A Console Control Panel. The following SENSE SWITCH settings will enable the ASSEMBLER to perform the desired mode of assembly and indicate the type of devices to be used for source program input and object program output. SENSE SWITCHES are the ENTRY TOGGLES operating in the RAISED state. When the ENTRY TOGGLES are returned to their center position they are said to be in a NEUTRAL state.

SENSE SWITCH	STATE	FUNCTION
0	RAISED	TWO PASS ASSEMBLY
0	NEUTRAL	ONE PASS ASSEMBLY
1	RAISED	NO LISTING
1	NEUTRAL	LISTING PROVIDED
2	RAISED	NO OBJECT OUTPUT
2	NEUTRAL	OBJECT OUTPUT PROVIDED
3	RAISED	LIST THE ERROR LINES ONLY
3	NEUTRAL	LIST ACCORDING TO SENSE SWITCH 1

SENSE SWITCH	STATE	FUNCTION
4	RAISED	LISTING ON CONSOLE KEYBOARD PRINTER
4	NEUTRAL	LISTING ON HI-SPEED (OPTIONAL) PRINTER
5	RAISED	SOURCE INPUT FROM PAPER TAPE READER
5	NEUTRAL	SOURCE INPUT FROM CARD READER
6	RAISED	OBJECT OUTPUT ON ASR-33 PAPER TAPE PUNCH
6	NEUTRAL	OBJECT OUTPUT ON HI-SPEED (OPTIONAL)PAPER TAPE PUNCH
7	RAISED	LIST THE SYMBOL TABLE
7	NEUTRAL	SYMBOL TABLE NOT LISTED
8	RAISED	SOURCE INPUT FROM ASR-33 PAPER TAPE READER
8	NEUTRAL	SOURCE INPUT FROM HI-SPEED PAPER TAPE READER
9	RAISED	SOURCE INPUT FROM CONSOLE KEYBOARD
9	NEUTRAL	CHECK OTHER SWITCHES
10*	SEE NOTE BELOW	
10*		
11	RAISED	MAGNETIC TAPE SOURCE INPUT
12	RAISED	MAGNETIC TAPE OBJECT OUTPUT

NOTE

THE FOLLOWING PROCEDURE MUST BE FOLLOWED TO OBTAIN A LISTING AND/OR OBJECT PAPER TAPE OUTPUT WHEN THE CONSOLE KEYBOARD PRINTER IS USED TO OUTPUT THE OBJECT PAPER TAPE DURING ASSEMBLY:

1. TWO-PASS MODE --- Process the source input from any device other than the Console Keyboard with SENSE SWITCHES 0, 1, 6, raised on the second pass to produce output on paper tape.

If an assembly listing is required, set SENSE SWITCH 1 to its NEUTRAL position and set SENSE SWITCHES 2 and 10 to the RAISED position.

Depress the START toggle to produce an assembly listing.

2. SENSE SWITCHES 1, 2, 3, in the RAISED state will produce no output other than a listing of the source program statement line in error.

APPENDIX C

SEL 810A SAMPLE ERROR PRINTOUT

```
0001    00000      40000004        NAME ERROR, C
0001    00000      00000000
0001    00000      01211022
0001    00000      03611040
0002    00000      00000000        REL
0003    00000      00000000      *This Program Was Written to Provide A
0004    00000      00000000      *Sample Error Listing
0005    00000      00000000      *
0006    000000     01000000      A     LAA    B     Undefined Symbol
0007    00001M     05100004      A     AMA    C     Multiple Defined Symbol
0008    00002A     03000000            STA          No Address
0009    00003Q     00100004            ADD    C     Undefined Operation
0010    00004      00001130      C     BSS    600   Overflows Map
0011    01134      00000500            DATA   5825  Data Conversion Error
0012    01136E     70001136      1BC   BES    1     Miscellaneous Error
0013    01136      70400000            END
M        A            00000
U        B            00000
         C            00004
         1BC          01136
```

APPENDIX D
UPDATE, DEBUG, LOADER PROCEDURES

D. 1 UPDATE INSTRUCTIONS

The UPDATE program is designed to allow the operator to easily correct or modify a symbolic source program by providing the following functions:

1. Deletion of a specified line or a group of lines.

2. Insertion of a new or replacement line or lines.

3. List the source program complete with line reference numbers.

All references to the symbolic source tape are made by referring to a sequence number. The sequence number is present on all assembly typeouts and on all typeouts generated by the UPDATE porgram.

Each function is initiated by a keyword entry on the Console Keyboard Printer. The functions are all initiated by a type-in consisting of the SLASH CHARACTER (/), (KEYWORD AND) TERMINATOR (CARRIAGE RETURN).

The keyword processor of the UPDATE program looks only at the first character, digits 0 through 0, and the terminator (carriage return): All other characters are ignored.

Note that there must be a space between the 20 and 29 below.

Example:

```
/D20 29 CR

/DELETE ALL LINES BETWEEN 20 and
 29 CR

/I33 CR

/INSERT CORRECTIONS AFTER LINE 33
 CR
```

D. 2 DEBUG INSTRUCTIONS

The DEBUG program is a utility program designed to help a programmer debug a program while it is in memory. The following functions are provided:

1. Type the contents of specified memory in octal or command format.

2. Modify the specified memory. Input being in octal format or command format.

3. Dump specified memory areas onto paper tape in a self-loading (non-relocatable) format.

4. Load binary tape.

5. Enter breakpoints in order to "leap-frog" trace a program.

6. Clear specified areas of memory to zero.

7. Search memory for references to specified areas.

8. Initiate branches (or HALT and BRANCH) to any part of memory.

Each of these functions are initiated by typing a keyword through the Console Typewriter Keyboard. This keyword consists of a letter, an address (or addresses) and a terminator (CARRIAGE RETURN).

When a keyword requires two addresses (a lower and upper boundary), separate the addresses with a space or comma.

If an error is generated during input of the keyword but before the keyword was terminated, type a slash character (/). This will cause the keyword in error to be completely ignored, cause the Console Keyboard to generate a carriage return, and will then ask for a new keyword input. If the computer detects an error, it will initiate the same action automatically.

The keyword input portion of the DEBUG program looks only at the first character, digits 0 through 7, and the terminator (carriage return). Leading zeros on octal entries or addresses are not necessary.

For example the operator types:

```
S20 CR or
T140-153 CR
```

The sense switches are as follows:

Sense Switch 0	RAISED High Speed Paper Tape Reader Load and High Speed Paper Tape Punch Dump
Sense Switch 0	NEUTRAL ASR-33 Paper Tape Reader Load and Paper Tape Dump

D. 3 LOADER OPERATING INSTRUCTIONS

This procedure assumes that the LOADER program has been read into memory according to the SEL 810A Operating Instructions Manual Section IV.

NOTE

The LOADER uses the ENTRY TOGGLES as SENSE switches. SENSE switches are the ENTRY TOGGLES in a raised state.

The main program must be loaded first. Once loaded, the subroutine library may be loaded for automatic integration by the software LOADER. These 11 steps are as follows:

Step 1. Depress HALT toggle on the Console Control Panel.

Step 2. Depress computer MASTER CLEAR toggle.

Step 3. Place the program to be loaded in the Paper Tape Reader.

Step 4. Rotate the Keyboard Printer LOCAL, LINE, OFF switch to the LINE position.

Step 5. Raise the START, STOP, FREE toggle on the Paper Tape Reader to the START position.

Step 6. Set the PROGRAM COUNTER to the starting address for LOADER.

Step 7. Set the A-Accumulator to the program starting address.

NOTE

a) A-Accumulator is set to zero if the program is absolute.

b) A-Accumulator is set to the relocation base if the program is relative.

Step 8. Set the B-Accumulator to the MAP starting address. MAP must be greater than 10 if library routines are to be called.

NOTE

B-Accumulator is set to 777 if the ASSEMBLER is being loaded into MAP zero.

Step 9. Set SENSE switches by raising the ENTRY TOGGLES to the desired SENSE switch setting.

The Entry Toggle settings before Step 10 are as follows:

ENTRY TOGGLE	STATE	FUNCTIONS TO BE PERFORMED
0	RAISED	INPUT ACCEPTED FROM OPTIONAL HIGH SPEED PAPER TAPE READER.
0	NEUTRAL	INPUT ACCEPTED FROM ASR-33 PAPER TAPE READER
1	RAISED	PRINTED OUTPUT WILL LIST ALL SUBROUTINES ON THE KEYBOARD PRINTER.
2	RAISED	WILL LIST ALL SUBROUTINES NOT LOADED.
3	RAISED	INPUT FROM MAGNETIC TAPE.

Step 10. Raise "No Program Advance Toggle".

Step 11. Depress the START toggle switch.

D. 4 OPERATOR COMMUNICATIONS

When Loading is Complete

1. If SENSE switch No. 2 is RAISED, the typeout will include a list of subroutines that are missing.

2. If SENSE switch No. 1 is RAISED, the typeout will include a list of all the subroutines.

3. If no subroutines were required, the typeout will contain "LCEJ" and the program will arrive at a normal HALT.

4. The LOADER will HALT after each library paper tape load. Depressing START when more subroutines are needed will cause the LOADER to search for a new or next library record on paper tape.

5. Depressing START after all subroutines are loaded will cause a branch to the first instruction of the main program just loaded.

EOJ - Typeout

Whenever an EOJ ($) block is encountered, the loader types out EJ. If no external subroutines have been requested by the main program, it halts. When an EOJ ($) code is encountered at the end of a library input file, the loader will do the following according to the settings of sense switches Nos. 1 and 2;

1. Both neutral: No subroutine name is listed.

2. No. 1 raised, No. 2 neutral: All subroutine names are listed.

3. No. 1 neutral, No. 2 raised: Only unloaded subroutine names are listed.

4. Both Raised: All subroutine names are listed.

By depressing the start button, the loader will either transfer control to the start location of the loaded program (if the number of undefined subroutine calls remaining is 0), or read more information from the next library file.

D. 5 RECOVERY PROCEDURES

"CL" Checksum Error

1. Each block of input is concluded with a logical difference checksum. An error in the checksum causes the LOADER to typeout "CK" with a program HALT.

2. Loading resumes with depressing of the START toggle.

"MO" Memory Overflow

1. "MO" will be typed followed by a pro gram HALT when the program exceeds the available memory space.