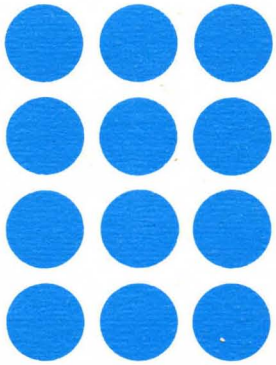
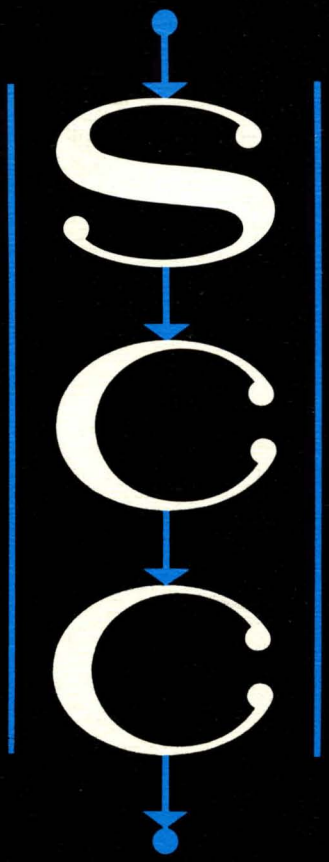
Scientific Control Corporation

# SPL Assembler Manual
## 650





# SCIENTIFIC CONTROL CORPORATION

# 650 SPL REFERENCE MANUAL

TABLE OF CONTENTS

# II.

## GENERAL DESCRIPTION

Programs written in SPL assembly language are processed into object programs by the SPL assembler. The SPL assembler accepts assembly language programs from either the paper tape reader or the typewriter. A listing of the assembled programs may be obtained on the typewriter. The assembled programs are punched onto paper tape in one of two forms: Relocatable binary format or Absolute binary format.

### A. PROGRAM

An SPL assembly language program consists of a series of one or more lines, the last of which must be an END directive.

A line contains a symbolic assembler delcaration, a machine instruction, or a symbolic instruction.

A program may contain fixed or relative instructions and addresses. Fixed instructions and addresses refer to a fixed memory location when the program is loaded. Relative instructions and addresses are relative to the program and are relocated by the Loader when the program is loaded. The type fixed or relative is defined by an ORG declaration.

### B. SYMBOLIC INSTRUCTION FORMAT

A symbolic instruction line consists of a location field, an operation field, a variable field (address, tag field), and a comments field. A line is one logical unit and the fields within a line are defined as being in fixed character positions or columns.

For paper tape and typewriter, a line consists of a string of up to 80 character positions terminated by a carriage return. The fields within a line may be defined as starting at a fixed character position or may be defined by a tab character to the position.

The format of a line is as shown on the coding form, on the following page.

**SCIENTIFIC CONTROL SYSTEMS, INC.**
**14008 DISTRIBUTION WAY**
**DALLAS, TEXAS 75234**

# 80 COLUMN CODING AND DATA FORM

NAME

PROGRAM | UNIT | DATE

ROUTINE | EXT. | PAGE OF

| STATEMENT NO. | CONT. | FORTRAN STATEMENT | FORTRAN | IDENTIFICATION |
|---|---|---|---|---|
| LOCATION | | OPERATION | ADDRESS, TAG | COMMENTS | SPL | |

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

## 1. Symbolic Addresses

A symbolic address is a collection of characters which serves as a name for a location used by the program. The assembly process will assign a unique location to each symbol appearing in the program. We shall distinguish four types of addresses which may appear in a program: (1) Symbolic, (2) Program point, (3) Absolute, and (4) Special.

### a. Symbolic Labels

A symbolic label consists of one to six non-blank, alphanumeric characters. The first character must be an alphabetic character (the currency symbol, '$', is considered to be alphabetic). Each symbol used must receive a memory location as its assignment. This may be accomplished in one of two manners - it may appear in the location field or it may be defined by the EQU pseudo-operation described later. In either case, a symbol must be defined precisely once.

### b. Program Point Addresses

Program point addresses provide the assembler with short-term memory for symbols as opposed to the long-term memory provided by symbolic labels. When written in the location field, the program point takes the form of a decimal point followed by a single letter, in the address field, by + or - followed by a single letter. The operand address "+L" refers to the next cell to be defined as ".L"; the operand address "-L" refers to the cell most recently defined as ".L". Program point addresses may be defined only by ap-

pearing in the location field and may be re-defined in this manner indefinitely.

c. Absolute Addresses

An absolute address (machine address) consists of one to five decimal digits or 1-5 octal digits preceded by a "!" character. If desired, leading zeros need not be written. The assembler will use the given number rather than treat the address as symbolic. Absolute addresses may not appear in the location field.

d. Special Address

The symbol '*' acting as an address indicates the address of the instruction being assembled. The symbol '**' forces a zero address, but indicates that the address may be changed by the program. Special address may not appear in the location field.

2. Location Field

The location field occupies column 1-6; column 7 is always blank. The location field may contain any allowable symbolic address as a label, or may be left blank.

As asterisk character '*' in column one of the location field defines the line as being a comment line. A comment line is not processed by SPL except for listing purposes. Comment lines may appear anywhere within a program.

3. Operation Field

Operation codes are written using the standard mnemonic abbreviations. They should be written starting in column 8 of the operation

field. If desired, a 1 or 2 digit octal operation code may be written. The operation field will also be used for pseudo-instructions which will be described later.

a. Indirect Address

The specification that the operand address of an instruction is an indirect address is signified by the presence of an asterisk character '*' immediately following the operation code.

4. Variable Field (address, Tag Field)

The variable field begins in column 16 and is terminated by the first blank character after column 16.

The variable field consists of a symbolic address or a symbolic address followed by a '+' or '-' followed by a decimal or octal integral increment. Octal increments are written with an "'" followed by an octal digit string. The variable field may be left blank if not required by the instruction. Following the variable field, a comma followed by an X or D may occur if other than the relative or relative indirect mode of addressing is required. The D indicates the direct mode while the X indicates the indexed mode.

The index mode designated (X) does not cause a modification to the current line instruction and is introduced only to make the symbolic code line more readable.

5. Comments Field

The comments field follows the variable field and may extend to column 80 of a line.

The comments field may contain programmer remarks and is not processed by SPL except for listing purposes.

C. LITERALS

A literal is a symbol referenced as a constant to be defined by SPL. A literal may appear only in the variable field of a line and only for the four mnemonic literal instructions, ANL, XOL, LDL, and ADL. It cannot be indexed, and may not appear in an expression.

A literal is a one to two digit optionally signed decimal or octal integer or any single character preceded by the '=' character.

For the literal instructions, LDL and ADL, the 6-bit literal address Y is considered to be a signed integer, with $-'40=-32 \leq Y \leq 31='37$. (The apostrophe prefixed number indicates an octal number as written for SPL.) Negative values have the 6-bit two's complement of the integer value placed in the address portion of the literal instruction.

For the literal instructions ANL and XOL, the 6-bit literal address Y is considered to be a non-negative integer, with $00 \leq Y \leq 63='77$.

If the literal address is represented with an '=' followed by a single character, then SPL places the equivalent 6-bit binary configuration for that character as the literal instruction address.

Examples:

```
LDL   1
ADL   -5
ANL   '77
ANL   =A
XOL   '40
```

D. DATA ITEMS

Three types of data items are processed by SPL: Octal, Decimal, and alphanumeric. They are specified by pseudo-operations PAR, DEC and BCI (see III PSEUDO OPERATIONS).

1. Octal Integers

An octal data items consists of one word containing from one to four octal digits. The octal data item is converted to binary. If the item is preceded by a minus sign, the two's complement of the binary number is generated.

2. Double Precision Decimal Integers

Double precision decimal integers are represented internally as two word two's complement data items. The most significant part of the number is contained in the first word and the least significant part is contained in the next successive word. The first word must be at an even machine location. A double precision, I, must be in the range $-33554432=-2^{24} \leq I \leq 2^{24}-1=33554431$.

| | SIGN | | VALUE | |
|---|---|---|---|---|
| Even Location Word 1: | 0 | 1 | 11 | most significant |
| : | 0 | | 11 | least significant |

3. Double Precision Floating Point Numbers

A floating point number is a decimal number which is expressed as either of the following:

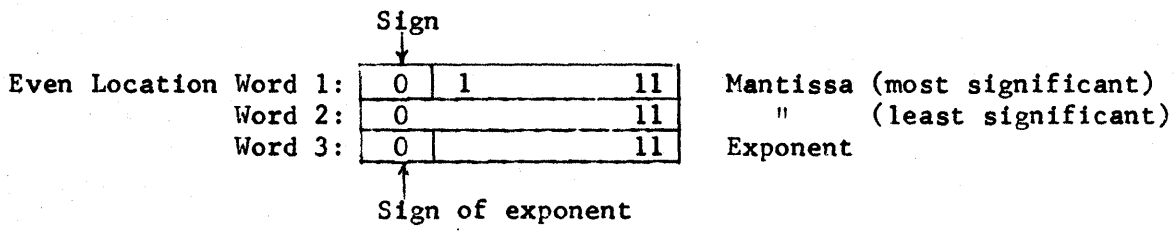
A signed or unsigned decimal number containing a decimal point optionally followed by an exponent part consisting of the letter E followed by a signed or unsigned decimal integer.

A signed or unsigned decimal integer optionally containing a decimal point, followed by an exponent part consisting of the letter E followed by a signed or unsigned decimal integer.

The number following the letter E is the power of ten to which the number is to be raised when it is converted.

Examples: 1.57
1.57E6
157E-6

Double precision floating point numbers are represented internally as three word two's complement data items. The data item consists of a 2-word 23 bit mantissa plus sign, where most significant part is carried in an even machine location, followed by a 1-word twelve bit signed binary exponent. The magnitude of a double precision floating point number, F, must be in the range $2^{-2048} \leq F < 2^{2048}$. The mantissa of the number must be less than $2^{24}-1=33554431$.




4. Alphanumeric Data

An alphanumeric data item is composed of from one to two characters. These characters are converted to six-bit character codes and stored as one word data items. (The character codes are listed in Appendix B.

# III

## PSEUDO-OPERATIONS

Pseudo-operations are operation codes which direct the assembler to perform operations on the program. Pseudo-operations do not normally cause actual machine instructions to be output, but cause memory allocations, constant definitions, symbolic definitions, or external program linkages.

Pseudo-operations are written similar to normal operations, with any exception to be noted under the description of the particular pseudo-operation. The location field of any pseudo-operation may contain a symbolic name which (with the exception of EQU) will be assigned the next address in sequence, prior to any effects the pseudo-operation might have on subsequent address assignments.

A. BCI

The BCI pseudo-operation specifies words of data expressed as 6-bit BCD characters packed two per word.

The variable field contains a word count followed by a comma followed by a string of characters. The word count specifies the number of 2 character words present in the character string. Blanks are counted as significant characters in the string.

Examples:

```
A    BCI    2, DATA
B    BCI    6, DATA WORDS
```

B. BSS

The BSS pseudo-operation declares a data area which is reserved by the program.

The variable field contains a symbolic address which defines the number of words to be reserved. The value of the symbolic address must have an absolute value.

Examples:

```
A   BSS   5
C   EQU   '75
      .
      .
B   BSS   200
    BSS   C
```

C. CALL

The CALL pseudo-operation creates a link to an external subprogram CALL generates a two word link to an object time transfer routine and indicates, to the Loader, the name of the required subprogram.

Programs using CALL must not be absolute and may not use direct addresses '75 - '77.

D. DEC

The DEC pseudo-operation specifies words of data expressed as double precision decimal or floating point numbers. The first word of the data will be set to an <u>even</u> location in the machine.

The variable field contains one or more double precision, decimal integers or floating point numbers separated by commas. Negative numbers

will be converted to two's complement. As many items as desired may be specified in the variable field. The format of decimal numbers is described in Section II, D., 2.

Examples:

```
A   DEC   1,6
B   DEC   25, 1, 7E-8, 2E5, 3.
C   DEC   -27
```

E. EQU

The EQU pseudo-operation specifies the equivalence of symbols within a program.

The location field contains a label which is set as having the value or address specified in the variable field.

The variable field contains a symbolic address. Any symbolic labels must be defined prior to appearing within the EQU pseudo-operation. That is, any symbolic labels appearing in the variable field must have appeared in the location field of a previous instruction.

Examples:

```
A   EQU   X
B   EQU   Y + '100
C   EQU   B-20
```

F. END

The END pseudo-operation signifies the end of the program and must be present. The variable field defines the execution address of the program. This field may be left blank if a subprogram is being assembled.

Example:

```
START       LDA   X
             .
             .
             .
            END   START    Designates the end of the
                           program, and an execution
                           address at START.
```

G. EVEN

The EVEN pseudo-operation causes the location of the next SPL instruction to be set even. This may result in one memory location being skipped at most.

H. JMP

The JMP pseudo-operation permits programmers to write jump instructions without deciding whether the desired location is forward or backward.

A JMF or JMB will be generated.

I. NAME

The NAME pseudo-operation specifies and defines the names of locations in the program which may be referenced by external programs.

The variable field contains a list of all labels contained within the program which may be referenced by external programs.

Example:

```
        NAME    A, B
A       PAR     Ø
        .
        .
        .
B       DEC     2
```

The labeled locations, A and B, may be referenced by external programs.

J. ORG

The ORG pseudo-operation defines the origin or loading address of a sequence of instructions or data. This may be either a relative or absolute value, but relocatable programs may not use absolute 'ORG' declarations except for areas 00000-00077 and 07701-07777.

The variable field defines the starting address of following instructions or data and may contain any symbolic address type, except that symbolic labels must have been previously defined.

The first line of every program should be an 'ORG' pseudo-operation, to indicate the type of program being assembled. 'ORG n' indicates an absolute program starting at location n, where n is a decimal or octal address. 'ORG *' indicates a relocatable program of less than 4096 words in length, while 'ORG *+n' indicates a partially relocatable program starting at location n in a bay. If the initial ORG line is omitted, 'ORG *+Ø' is assumed.

Examples:

```
        ORG         *       Indicates a relocatable program
A       LDA         B       A is assigned address relative Ø
B       ORG         *+2Ø    Defines origin as relative forward
          .                 2Ø
          .
          .
        ORG         B+1     Defines origin as relative 2
```

K. PAR

The PAR pseudo-operations specifies words of data expressed as symbolic addresses, or decimal or octal integers.

The variable field contains one or more symbolic address. The resulting word will contain the 12-bit address assigned to the corresponding symbolic address.

Examples:

```
PAR   A, B+2, C, 2, '4Ø
PAR   '27, *+5
```

(Note: In the second example, the second word would be assigned an address equal to its address plus 5, not the starting address of PAR plus 5.)

## L. OPERATION DEFINITIONS

In order to expand the allowable symbolic operation code set, symbolic operation codes not defined by the assembler may be defined by writing the symbolic operation to be defined in the op-code field with a variable field started by an "=" signed and followed by any symbolic address (normally an octal number). If a symbolic label is used, it must be absolute and have been previously defined. In any case, the resulting 12-bit value will be used as the basic operation code, to which the symbolic address is added when the new symbol is used as an operation.

Example:

```
ABC='22ØØ       ABC has an operation code of '22ØØ
DEF=A+7         DEF has an operation code consisting
                    of the current address of A plus seven.
```

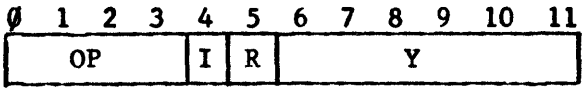
# IV

# MACHINE INSTRUCTIONS

This section describes the instruction repertoire of the SCC 650 computer.

The word format of a machine instruction is as follows:

Basic Instruction Format




where

OP is the basic 4-bit operation code

I is the indirect address bit

R is the mode bit

Y is the 6-bit instruction operand address

The descriptions of each of the instructions is headed by the information in the following format:

MNM Name Timing

Code I R Y

MNM is the SPL mnemonic assigned to the instruction

Name is the instruction name.

Timing is the number of machine cycles used by the instruction. Each level of indirect addressing adds one cycle to the instruction.

Code is the octal operation code of the instruction; and in some cases the complete 4-digit octal code instruction word will be given.

I is present if the instruction can optionally contain an indirect address bit. Otherwise, the numeric value of I is placed in this position.

R is present if the instruction can optionally contain a mode bit.

| MNM | Name | Timing |
|---|---|---|
| | Y | is present if the instruction contains an operand address. Otherwise, the octal value of Y is placed in this position. |

Table IV, A. contains a list of symbols and their definitions used in the instruction description. The instructions are categorized into sections according to function.

Operand is defined as being the contents of the effective operand address.

A. TABLE

NOTATION

| SYMBOL | DEFINITION |
|---|---|
| A | The main arithmetic register, or Accumulator |
| X | The index register and left hand Accululator extension |
| Y | Operand address of instruction |
| M | Effective operand address of instruction |
| P | The location counter. Contains the address of the instruction to be executed. |
| ( ) | Contents of. Signifies the contents of the symbol enclosed within the parentheses. |
| $\longrightarrow$ | Replacement designator. The value on the left is placed into the value on the right. |
| + | Add |
| - | subtract |
| / | Divide |
| x | Multiply |
| $\odot$ | Logical AND |
| $+$ | Logical OR |
| $\neq$ | Exclusive OR |
| $\lvert\ \rvert$ | Absolute value |
| $-$ | One's complement |
| < | Less than |
| > | Greater than |
| = | Equal |
| $\equiv$ | Equivalent by definition |

B. DATA TRANSFER INSTRUCTIONS

LDA Load A Timing: 2

60 I R Y

(M)→A

The operand is placed in A. The contents of memory at M remains unchanged. (See also literal instruction LDL.)

STA Store A Timing: 2

34 I R Y

(A)→M

The contents of A are placed into memory at M.

The contents of A remains unchanged.

LDX Load Index Timing: 2

10 I R Y

(M)→X

The operand is placed into the index register. The contents of M remains unchanged.

STX Store Index Timing: 2

14 I R Y

(X)→M

The contents of the index register is placed into M. The contents of the index register remains unchanged.

C. ARITHMETIC INSTRUCTIONS

Arithmetic operations are performed in two's complement arithmetic. Overflow and carry conditions cause the setting of machine flip-flops which may be tested or used for double precision arithmetic. The flip-flops and conditions are as follows:

Overflow flip-flop. Any arithmetic operation which causes the sign of the result to be wrong (i.e., a carry into the sign position) causes this flip-flop to be set. For example, addition of two positive numbers resulting in a negative number causes the overflow flip-flop to be set.

Carry flip-flop - Any addition or subtraction which causes a carry from bit position 0 (i.e., a 13 bit result) causes the carry flip-flop to be set.

ADD Add Timing: 2

44 I R Y

(A) + (M)→A

The operand is added to the contents of A and the sum is placed in A. The contents of M remains unchanged.

This instruction may cause the overflow, and carry flip-flops to be set.

SUB Subtract Timing: 2

64 I R Y

(A) - (M)→A

The operand is subtracted from the contents of A and the difference is placed in A. The operand at M remains unchanged.

This instruction may cause the overflow, and carry flip-flops to be set.

MIN Memory Increment Timing: 3

40 I R Y

(M) + 1→M

If (M) = 0, (P) + 2 → P, otherwise (P) + 1 → P.

One is added to the operand and placed into memory at M. If the result is zero the next instruction is skipped. Otherwise, the next instruction is taken in sequence.

This instruction will never affect the contents of the carry or overflow flip-flops.

AND Logical AND Timing: 2

74 I R Y

(A) ⊙ (M) → A

Forms the logical AND of the operand and the contents of A and places the result in A. The contents of memory at M remains unchanged.

XOR Exclusive OR Timing: 2

54 I R Y

(A) ⊙ (M) ⊕ (A) ⊙ (M) = (A) ⊖ (M) → A

Forms the exclusive OR of the operand and the contents of A and places the result in A. The contents of M remains unchanged.

D. CONTROL AND TEST INSTRUCTIONS

HLT Halt Timing: 1

0000

Halt

The computer halts awaiting manual intervention from the console. P contains the address of the next instruction following the HLT instruction.

The address, index, and indirect fields of this instruction are not used.

NOP No Operation Timing: 1

0002

No operation is performed.

JMF Jump Forward Timing: 1

20 I R Y

M → P

The next instruction is taken at the effective operand address, M, where if I=0 and:

R=0, then M=Y;
R=1, Index State =0, then M=(P)+Y;
R=1, Index State =1, then M=(X)+Y;

while if I=1 and:

R=0, then M=(Y);
R=1, Index State=0, then M=((P)+Y);
R=1, Index State=1, then M=((X)+Y).

JMB Jump Backward Timing: 1

24 I R Y

M → P

Same as for Jump Forward above with -Y replacing Y.

JSL Jump and Store Location Counter Timing: 3

50 I R Y

(P)+1 → M, Status → M+1,
M+2 → P

The location counter address plus one, (P)+1, is placed in the effective operand address, M, and the Status Register is placed in M+1.

The next instruction is taken from location M+2.

JRT Return Jump Timing: 3

70 I R Y

(M+1)→Status,

(M)→P

The contents of M+1 are placed into the Status Register. The next instruction is taken from location M, where the effective address M is as described in Jump Backward above.

E. SHIFT INSTRUCTIONS

All shift instructions are 1 bit shift and use bits 4-11 of the instruction word to indicate the type of shift.

SPL automatically sets the shift type bits in the instruction according to the mnemonic.

SAR Short Arithmetic Shift Right Timing: 1

0014

(A) right 1 place→A

The contents of A is shifted right one binary place. Bit $A_{11}$ is lost. Bit 0 of A is not shifted but is copied into the vacated bit position, bit 1, on its right.

SRR Short Rotate Right Timing: 1

0010

(A) rotate right 1 place→A

The contents of A is rotated right one binary place. Bit $A_{11}$ enters $A_0$.

SLR Short Logical Shift Right Timing: 1

0114

(A) right 1 place→A

The contents of A is shifted right one binary place Bit $A_{11}$ is lost. Vacated bit position $A_o$ is filled with a zero.

SCR Short Circulate Right Timing: 1

0110

(CO,A) right 1 place→A

The coupled CO- and A-registers, with CO preceding bit $A_o$, is rotated right one binary place. Bit CO enters $A_o$ and bit $A_{11}$ enters CO.

LAR Long Arithmetic Shift Right Timing: 1

0214

(A,X) right 1 place→A,X

The contents of A and X are shifted right one binary place. Bit $A_{11}$ enters $X_o$. Bit $X_{11}$ is lost. Bit 0 of A is not shifted but is copied into the vacated bit position bit 1 on its right.

LRR Long Rotate Right Timing: 1

0210

(A,X) right cycle 1 place→A,X

The contents of A and X are rotated right one binary place. Bit $A_{11}$ enters $X_o$. Bit $X_{11}$ enters $A_o$.

LLR Long Logical Shift Right Timing: 1

(A,X) right 1 place→A,X

The contents of A and X are shifted right one binary place. Bit $X_{11}$ is lost. Bit $A_{11}$ enters $X_o$. Vacated bit position $A_o$ is filled with a zero.

LCR Long Circulate Right Timing: 1

0310

(CO,A,X,) right 1 place→A,X

The coupled CO-, A- and X-registers, with CO preceding bit $A_0$ and $A_{11}$ preceding $X_0$, is rotated right one binary place. Bit CO enters $A_0$, bit $A_{11}$, enters $X_0$ and bit $X_{11}$ enters CO.

SAL Short Arithmetic Shift Left Timing: 1

0016

(A) left 1 place $\rightarrow$ A

The contents of A is shifted left one binary place. Bit $A_0$ is lost. Vacated bit position $A_{11}$ is filled with a zero.

If the sign of A, $A_0$, changes, the overflow flip-flop is set.

SRL Short Rotate Left Timing: 1

0012

(A) left cycle 1 place $\rightarrow$ A

The contents of A is rotated left one binary place. Bit $A_0$ enters $A_{11}$.

SLL Short Logical Shift Left Timing: 1

0116

(A) left 1 place $\rightarrow$ A

The contents of A is shifted left one binary place. Bit $A_0$ is lost. Vacated bit position $A_{11}$ is filled with a zero.

SCL Short Circulate Left Timing: 1

0112

(A,CO) left 1 place $\rightarrow$ A

The coupled A- and CO-registers, with CO following bit $A_{11}$, is rotated left one binary place. Bit CO enters $A_{11}$ and bit $A_0$ enters CO.

LAL Long Arithmetic Shift Left Timing: 1

0216

(A,X) left 1 place→ A,X

The contents of A and X are shifted left one binary place. Bit $X_0$ enters $A_{11}$. Bit $A_0$ is lost. Vacated bit position $X_{11}$ is filled with a zero.

If the sign of A, $A_0$, changes, the overflow flip-flop is set.

LRL Long Rotate Left Timing: 1

0212

(A,X) left cycle 1 place→ A,X

The contents of A and X are rotated left one binary place. Bit $X_0$ enters $A_{11}$. Bit $A_0$ enters $X_{11}$.

LLL Long Logical Shift Left Timing: 1

0316

(A,X) left 1 place→ A,X

The contents of A and X are shifted left one binary place. Bit $A_0$ is lost. Bit $X_0$ enters $A_{11}$. Vacated bit position $X_{11}$ is filled with a zero.

LCL Long Circulate Left Timing: 1

0312

(A,X,CO) left 1 place→ A,X

The coupled A-, X- and CO-registers, with CO following bit $X_{11}$ and $A_{11}$ preceding $X_0$, is rotated left one binary place. Bit CO enters $X_{11}$, bit $X_0$ enters $A_{11}$ and Bit $A_0$ enters CO.

F. REGISTER CHANGE INSTRUCTIONS

Register change instructions use the operand address portion of the instructions to specify operations. Indexing and indirect addressing are not permitted. SPL automatically sets the operand address bits from the mnemonic operation code.

CLA Clear A Timing: 1

0003

0⟶A

The contents of A are set to zero

CLX Clear X Timing: 1

0007

0⟶X

The contents of X are set to zero

CAX Copy A to X Timing: 1

0240

(A)⟶X

The contents of A are placed into the index register. A is unchanged.

CXA Copy X to A Timing: 1

0140

(X)⟶A

The contents of the index register are placed into A. X is unchanged.

XAX Exchange X and A Timing: 1

0040

(X)⟶A, (A)⟶X

The original contents of X are placed into A.

The original contents of A are placed into X.

XHA Exchange Half A Timing: 1

0020

$(A_{0-5}) \rightarrow A_{6-11}$, $(A_{6-11}) \rightarrow A_{0-5}$

The original contents of $A_{0-5}$ are placed into $A_{6-11}$.

The original contents of $A_{6-11}$ are placed into $A_{0-5}$.

G. LITERAL INSTRUCTIONS

The literal instruction uses the literal value Y, with or without its sign- or most significant-bit extended, as its operand.

ANL Logical AND Literal Timing: 1

3Ø Y

$(A_{0-5}) \rightarrow A_{0-5}$,
$(A_{6-11}) \odot Y \rightarrow A_{6-11}$

Forms the logical AND of the address Y and the contents of $A_{6-11}$ and places the result in $A_{6-11}$. $A_{0-5}$ remains unchanged.

XOL EXCLUSIVE OR Literal Timing: 1

31 Y

$(A_{0-5}) \rightarrow A_{0-5}$
$(A_{6-11}) \odot \overline{Y} \oplus \overline{(A_{6-11})} \odot Y = (A_{6-11}) \ominus Y \rightarrow A_{6-11}$

Forms the EXCLUSIVE OR of the address Y and the contents of $A_{6-11}$ and places the result in $A_{6-11}$. $A_{0-5}$ remains unchanged.

LDL Load A Literal Timing: 1

32 Y

$Y_o \equiv$ Sign-bit of $Y \rightarrow A_{0-5}$

$Y \equiv Y_{0-5} \rightarrow A_{6-11}$

The most significant bit of the 6-bit address Y (considered as the sign-bit) is placed into $A_{0-5}$ and Y is placed into $A_{6-11}$, i.e., Y with sign extended is placed in A.

ADL Add Literal Timing: 1

33 Y

(Let $Y_0 \equiv$ Sign-bit of $Y \rightarrow Y'_{0-5}$, $Y \equiv Y_{0-5} \rightarrow Y'_{6-11}$,) then
(A)+Y'→ A

The 6-bit address Y, with its most significant bit taken as the sign-bit, extended, as Y', is added to the contents of A and the sum is placed in A.

This instruction may cause the overflow, and carry flip-flops to be set.

## H. MICRO-OPERATE INSTRUCTIONS

The "SCC 650" provides the ability in a one word "micro-instruction" to perform an operation involving the A and/or X registers, and then, optionally, test the result. SPL facilitates the usage of micro-instructions by assembling the proper codes from the information supplied in the following format.

OP F,T

where:

OP is the mnemonic SRA or SRX if the A or X is to be the "selected register." It is convenient to denote the selected register by SR, and the unselected register by USR.

F is a one digit code which specifies the desired function, as follows:

F=0 - test the SR only.

1 - increment the SR by one.

2 - an add is performed with SR and USR; the result is left in the SR.

3 - the SR is exclusively OR'd with USR replacing the SR.

4 - one's complement of SR replaces SR.

5 - two's complement of SR replaces SR.

6 - one's complement of SR exclusively OR'd with USR replaces SR.

7 - SR subtracted from USR replaces SR.

T is a one letter mnemonic (P,N, or Z) which is present if the result of the function F is to be tested, as follows:

T=G - skip the next instruction if the SR is greater than zero.

N - skip the next instruction if the SR is negative.

Z - skip the next instruction if the SR is zero.

I. PROGRAMMED-OPERATE INSTRUCTION CODE

SCF Set CO OFF Timing: 1

0022

Resets the carry-out bit to be Ø in the CO Register

SCN Set CO ON Timing: 1

0122

Sets the carry-out bit to 1 in the CO Register.

SIF Set Interrupt Control OFF Timing: 1

0222

Resets the Interrupt Control Register bit to be Ø.

SIN Set Interrupt Control ON Timing: 1

0322

Sets the Interrupt Control Register bit to 1.

OFT Overflow Test Timing: 1

0024

If Overflow Register ON, reset it to OFF, and (P)+1⟶P; otherwise (P)+2⟶P.

SBK Set Bank Flag Timing: 1

0026

Sets inhibit IB-Register Flag ON.

RBK Restore Bank Flag Timing: 1

0126

Resets the Inhibit IB-Register flag to OFF.

CLI Clear Interrupt Timing: 1

0226

Resets the interrupt.

COT CO Carry-Out Test Timing: 1

0032

If (CO)=Ø, (P)+1⟶P; otherwise, if (CO)=1, (P)+2⟶P.

XSS Index Status Set Timing: 1

0034

Sets the Index Status Register to 1.

XSR Index Status Reset Timing: 1

0234

Resets the Index Status Register to Ø.

LIA Load IB Register Timing: 1

0036

$(A_{9-11}) \longrightarrow IB$

Stores $(A_{9-11})$ into the IB Register

AAX Logical AND A with X Timing: 1

0242

$(X) \odot (A) \longrightarrow A$

The logical product of $(X) \odot (A)$ is placed in A.

AOX Logical OR A with X Timing: 1

0042

$(X) \oplus (A) \longrightarrow A$

The logical sum of $(X) \oplus (A)$ is placed in A.

MPT Memory Protect Test Timing: 1

0044

If Memory Protect ON, reset it to OFF, and $(P)+1 \longrightarrow P$;

otherwise, if OFF, $(P)+2 \longrightarrow P$.

IOT Input/Output Error Test Timing: 1

0046

IF I/O Error Register ON, reset it to OFF, and $(P)+1 \longrightarrow P$;

otherwise, if OFF, $(P)+2 \longrightarrow P$.

LAS Load from Switch Register Timing: 1

0050

$(Switch) \longrightarrow A$.

The content of the Switch Register is placed in A.

ADC Add Carry Timing: 1

0054

(A) + (CO)⟶A

The CO Register bit is added to the contents of A and the sum is placed in A. The contents of CO remains unchanged.

SST Store Status Register Timing: 1

0060

(Status)⟶A

The contents of the Status Register are placed in A. The contents of the Status Register remain unchanged.

LST Load Status Register Timing: 1

0260

(A)⟶ Status

The contents of A are placed into the Status Register. The contents of A remain unchanged.

J. 650 EXTENDED OPERATION CODES

1. Arithmetic

The extended arithmetic operations use the A- and X-registers as a double-length working register with A being most significant and X least significant.

The effective memory locations referenced (M), is obtained by using the 12 bit address following the operation with the bank specified by bit #4 of the instruction ($I_{4=0}$ for program bank, $I_{4=1}$ for indirect bank). The program counter will be advanced by two to bypass this second word.

LDD Load Double-Length Timing: 3

0076, 0276

(M)→A, (M+1)→X

The double-length operand at M, M+1 is placed in the A- and X-registers, respectively. The contents of memory at M and M+1 remain unchanged.

STD Store Double-length Timing: 3

0176, 0376

(A)→M, (X)→M+1

The double-length A- and X-registers are placed into memory at M and M+1, respectively. The contents of A and X remain unchanged.

MPY Multiply Timing: 9

0074, 0274

(A) x (M)→A, X

The contents of A is multiplied by the operand and the resultant 23 bit product is placed in the A and X registers. A contains the most significant bits, X contains the least significant bits, and bit 0 of A is the sign of the result.

If both numbers have the value $4000_8$, the overflow flip-flop is set and the product is set to zero.

DVD Divide Timing: 9

0174, 0374

(A,X) / (M)→A, Remainder→X

The contents of A and X are treated as a 23 bit dividend and are divided by the operand. The quotient is placed in A and the remainder is placed in X. The contents of M remains unchanged.

Overflow occurs and the overflow flip-flop is set if the following relationship is not satisfied:

$$-1 \leq \frac{(A,X)}{(M)} \leq 1$$

2. Shifts

The extended shift operations make use of a six bit, two's complement shift counter.

NOR Normalize Timing: 1+ count

0072

The instruction following this command is repeated until the sign of A and bit #1 of A are different, or until the shift counter is zero. After each repetition, the shift count is decremented by one. This instruction is normally followed by a long arithmetic shift left (LAL).

SSH Store Shift Register Timing: 1

0066

$(A_{0-5}) \longrightarrow A_{0-5}$,
(Shift) $\longrightarrow A_{6-11}$

The contents of the shift Register is copied into $A_{6-11}$. $A_{0-5}$ remains unchanged.

LSH Load Shift Register Timing: 1

0266

$(A_{0-5}) \longrightarrow A_{0-5}$,
$(A_{6-11}) \longrightarrow$ Shift

The contents of $A_{6-11}$ is copied into the Shift Register. (A) remains unchanged.

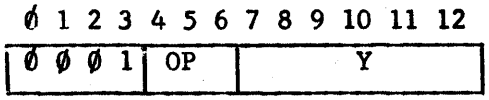
RPT Repeat Timing: 1

0070

The instruction following this command is repeated until the shift count is zero. After each repetition, the shift count is decreased by one.

K. INPUT/OUTPUT INSTRUCTIONS

Input-Output Instruction Format




Where OP is the operation code and Y is the device selection code.

TTA Transmit to A or Skip Timing: 1

Ø Y

If device (Y) ready, (Device Y)⟶A, (P) + 1⟶P; otherwise, if if not ready, (P) + 2⟶P. The contents of the I/O Device Y buffer is tranferred into A and the next instruction is executed. The device buffer is cleared and is ready for reloading by the external device. If the device is not ready, the next instruction is skipped and A is not loaded.

TFA Transmit from A or Skip Timing: 1

1 Y

If device (Y) ready, (A)⟶Device Y, (P) + 1⟶P; otherwise, if not ready, (P) + 2⟶P. If the device is ready the operand, (A), is transferred to the I/O device Y and the next instruction is executed. If the device is not ready, the next instruction in sequence is skipped.

DST Input Device Status Timing: 1

2 Y

If device (Y) is ready, (Device (Y) status)⟶A, P+1⟶P; otherwise, if not ready, (P)+2⟶P. The status of the selected device is transmitted to (A) if the device is ready and the next instruction is executed. If the device is not ready, the next instruction in sequence is skipped.

SDF Skip No Device Flag Timing: 1

3 Y

If Device Flag=1, (P)+1⟶P; Otherwise, if Device Flag=0, (P)+2⟶P. This instruction tests the selected Device Flag. If ON, the next instruction is executed. Otherwise, if OFF, the next instruction in sequence is skipped.

EXU Execute Command in A Timing: 1

4 Y

The external device (Y) executes the command in (A)

TMR Terminate Timing: 1

5 Y

The selected device (Y) is inactivated.

SNL Select with no leader Timing: 1

6 Y

The selected device (Y) is activated with no leader

SWL Select with leader Timing: 1

7 Y

The selected device (Y) is activated with leader being generated or read.

# V

# ASSEMBLER OPERATION

When the assembler is loaded and ready to begin assembly, the following message is typed on the console typewriter:

ASSEMBLE

The operator may then type in any of the following characters to specify the operation:

| | |
|---|---|
| A | The object program is to be output in Absolute binary format |
| R | The object program is to be output in Relocatable binary format |
| P | The program is to be input from paper tape. |
| T | The program is to be input from the typewriter |
| E | Punch an end-of-job record on the paper tape. |
| TAB | Punch length of blank tape |

The assembly processing begins when the operator types in a carriage return. If options are not specifically designated. the R and P options are assumed.

Options may be designated during an assembly by settings of breakpoint switches. These settings are:

| | | |
|---|---|---|
| Breakpoint 1 | ON | Do not list the assembled program. (errors are always listed). |
| Breakpoint 2 | ON | No object program is to be output. |

When the options have been specified and the carriage return has been typed in, the assembler accepts the program from the specified input device and processes the first assembly pass until and END declaration is processed.

The assembler signals it is ready for the second pass by typing the following message on the typewriter:

' PASS 2

When the operator has loaded the paper tape into the reader, processing of the second pass may be started by the operator flipping the RUN switch.

During the second pass the program is listed if specified and the object program, if specified, is output. Assembly of the program is complete when the END declaration is processed.

The assembler then halts. If the RUN switch is flipped, the assembler initiates itself and signals it is ready for a new assembly by returning to the 'ASSEMBLE' typeout point.

A. PROGRAM LISTING

A listing of the assembled program is typed on the typewriter in the following format:

FORMAT

LLLLLLSIIIISSVVV. . .

where

L = Octal location assigned to the instruction

S = Space character

I = Generated instruction

V = Symbolic statement that was processed to obtain the octal information

B. ERROR INDICATIONS

Error indications are always listed and consist of an '*' followed by a single character to indicate the error type.

Pass #1 error indications are followed by a typeout of the form SSLLLLLL+NN, where SS is two spaces, L-L is the last non program point label encountered and NN is the number (octal) of statements since that label.

Pass #2 error indications occur just prior to the listing of the statement in which the error occurred. If listing is not being performed, the Pass #1 format will be followed.

A list of error indicating characters follows.

A Indirect address specified incorrectly

B Statement not at beginning of program

C Not allowed in Bootstrap Load Format

D Improper literal

E Multiply defined symbol

F Illegal mnemonic

G Variable field not vacant

H Assembler Dictionary full

I Undefined Symbol

J Variable Field Error

K Index specified incorrectly

C. ASSEMBLY INFORMATION

Assembly information is always listed on the typewriter and upon completion of assembly.

| | |
|---|---|
| n ERRORS | Where n is the number of errors contained in the program |
| RANGE a | Where a is the highest assembled octal location in the program |
| UNDEFINED ... list | Where list is a list of all undefined symbols in the program or is the word NONE |

| | |
|---|---|
| EXTERNAL...list | Where list is a list of all external symbols in the program or is the word NONE |
| UNREFERENCED...list | Where list is a list of all symbols defined but unreferenced within the program, or is the word NONE |

Example:

```
00004  ERRORS
RANGE  01743
UNDEFINED...NONE
EXTERNAL.....
     SIN
     SQRT
     COS
     LOG
UNREFERENCED....
     A30
     K125
     ALPHX
```

D. SOURCE TAPE UPDATING PROCEDURE

On the first pass, if the typewriter keyboard input is specified, then the keyboard data may be UPDATE control commands as well as SPL instructions. An UPDATE control command has the following general form, beginning in column 1 with no spaces.

/SL1=n,L2=n2 (C/R)

where L1 and L2 represents labels in the source tape, or the "*" symbol (see below), $n_1$ and $n_2$ are optional integer increments, and X may be the characters "A", "D", "L", or a (C/R).

If X = "A", it means that the source tape is to be read, punched without change into the new source tape, and assembled until $n_1$ lines following the line with label L1 have been processed. Then the reading stops and SPL instructions may now be inserted via the typewriter key-

board, followed by a new UPDATE command. The second field (,L2+C2) is omitted if X = "A".

If X = "D", or "L" the source tape is read as before (but if X = "L", it is not punched or assembled) until the line denoted by L1=$n_1$ is encountered. At this point all lines from L1=$n_1$ to L2=$n_2$, inclusive, are listed but not punched or assembled. Then instructions and/or UPDATE commands may be input via the typewriter.

If X = "CARRIAGE RETURN", the remaining portion of the source tape will be read, punched and assembled without change.

Asterisks notation

One may type "*" in lieu of L1 or L2 with the following meaning:
For example:

```
/D*+1,*+5 C/R
```

would effectively delete the first five lines of a source tape, if the tape was positioned at the beginning. That is, *+1 refers to the first line that will next be read by the reader, *+2 the next, etc. Therefore, *+0 or * without an addend have no meaning and, indeed, is illegal.

Keyboard Mistakes

If a mistake is made at the keyboard while typing either an SPL instruction or an UPDATE command, it may be rectified by typing backspace followed by carriage return, to delete the line with no effect.

# APPENDIX A

## MNEMONIC OPERATION CODES

### SCC 650 PSEUDO-OPERATIONS

# APPENDIX B

## SCC 650 MACHINE INSTRUCTIONS

Register Change



Literal



Micro-Operate



Programmed-Operate



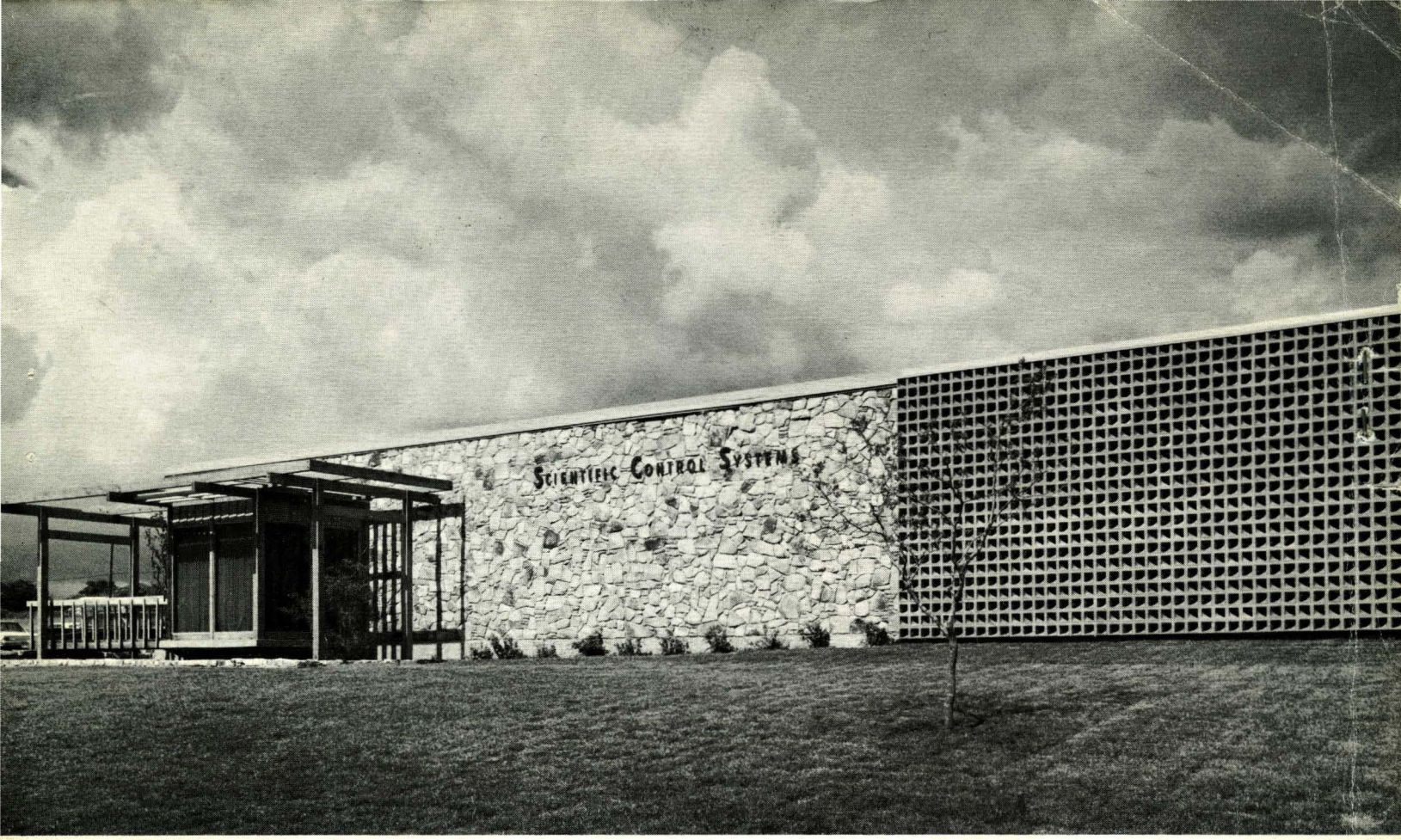
Extended Operations

APPENDIX C

## SCC 650 STANDARD CHARACTER CODES

| Character | 6-Bit Code | ASCII | | Character | 6-Bit Code | ASCII | |
|---|---|---|---|---|---|---|---|
| Ø | ØØ | 26Ø | | - | 4Ø | 255 | |
| 1 | Ø1 | 261 | | J | 41 | 312 | |
| 2 | Ø2 | 262 | | K | 42 | 313 | |
| 3 | Ø3 | 263 | | L | 43 | 314 | |
| 4 | Ø4 | 264 | | M | 44 | 315 | |
| 5 | Ø5 | 265 | | N | 45 | 316 | |
| 6 | Ø6 | 266 | | O | 46 | 317 | |
| 7 | Ø7 | 267 | | P | 47 | 320 | |
| 8 | 1Ø | 27Ø | | Q | 5Ø | 321 | |
| 9 | 11 | 271 | | R | 51 | 322 | |
| SPACE | 12 | 24Ø | | CHAR.RET. | 52 | 215 | |
| = | 13 | 275 | | $ | 53 | 244 | |
| : | 14 | 247 | | * | 54 | 252 | |
| ≠ | 15 | 272 | | ] | 55 | 335 | |
| > | 16 | 276 | | ; | 56 | 273 | |
| √ | 17 | 246 | (&) | Δ | 57 | 245 | (%) |
| + | 2Ø | 253 | | ↑ | 6Ø | 277 | (?) |
| A | 21 | 3Ø1 | | / | 61 | 257 | |
| B | 22 | 3Ø2 | | S | 62 | 323 | |
| C | 23 | 3Ø3 | | T | 63 | 324 | |
| D | 24 | 3Ø4 | | U | 64 | 325 | |
| E | 25 | 3Ø5 | | V | 65 | 326 | |
| F | 26 | 3Ø6 | | W | 66 | 327 | |
| G | 27 | 3Ø7 | | X | 67 | 33Ø | |
| H | 3Ø | 31Ø | | Y | 7Ø | 331 | |
| I | 31 | 311 | | Z | 71 | 332 | |
| BACKSPACE | 32 | 2Ø3 | (EOM) | TAB | 72 | 211 | |
| - | 33 | 241 | | ' | 73 | 254 | |
| ) | 34 | 251 | | ( | 74 | 25Ø | |
| [ | 35 | 333 | | ∩ | 75 | 243 | (# ) |
| < | 36 | 274 | | \ | 76 | 334 | |
| ≠ | 37 | 241 | (!) | ≠ | 77 | 3ØØ | (@) |

NOTE: Normal input conversion will delete all other ASCII codes. Normal output conversion to teletype will output a '52 (Carriage Return) as a Line Feed, Carriage Return (212,215).




SCC maintains complete support activities for its users. Installation and maintenance services are available through SCC offices strategically located throughout the United States. For pre-procurement demonstration of hardware and programs in Dallas, contact local sales office or the Marketing Department in Dallas.

Arlington, Massachusetts
30 Park Avenue
617 — 648-2922
(Boston)

Skokie, Illinois
125 Old Orchard Arcade
312 — 675-6700
(Chicago)

Houston, Texas
7800 Westglen Drive
713 — 782-9851

Seattle, Washington
1806 South Bush Place
206 — 324-7911

Midland Park, New Jersey
36 Central Avenue
201 — 652-6750
(New York)

Crofton, Maryland
Village Green
301 — 647-6431
(Baltimore)

Orlando, Florida
2319 E. South Street
305 — 841-3556

Pasadena, California
180 East California Blvd.
213 — 681-2651
(Los Angeles)

Other SCC products include: telemetry systems and airborne signal conditioning equipment such as amplifiers, demodulators and converters.

# Scientific Control Corporation

14008 Distribution Way • Dallas, Texas 75234 • 214 — 241-2111



Printed in U.S.A.