# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>SECTION</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>II. GENERAL DESCRIPTION</td>
<td>3</td>
</tr>
<tr>
<td>A. Program</td>
<td>3</td>
</tr>
<tr>
<td>B. Symbolic Instruction Format</td>
<td>3</td>
</tr>
<tr>
<td>1. Symbolic Addresses</td>
<td>5</td>
</tr>
<tr>
<td>a. Symbolic Labels</td>
<td>5</td>
</tr>
<tr>
<td>b. Program Point Addresses</td>
<td>5</td>
</tr>
<tr>
<td>c. Absolute Addresses</td>
<td>6</td>
</tr>
<tr>
<td>d. Special Addresses</td>
<td>6</td>
</tr>
<tr>
<td>2. Location Field</td>
<td>6</td>
</tr>
<tr>
<td>3. Operation Field</td>
<td>6</td>
</tr>
<tr>
<td>a. Indirect Address</td>
<td>7</td>
</tr>
<tr>
<td>4. Variable Field (Address, Tag Field)</td>
<td>7</td>
</tr>
<tr>
<td>5. Comments Field</td>
<td>8</td>
</tr>
<tr>
<td>C. Literals</td>
<td>8</td>
</tr>
<tr>
<td>D. Data Items</td>
<td>9</td>
</tr>
<tr>
<td>1. Octal Integers</td>
<td>9</td>
</tr>
<tr>
<td>2. Double Precision Decimal Integers</td>
<td>9</td>
</tr>
<tr>
<td>3. Double Precision Floating Point Numbers</td>
<td>9</td>
</tr>
<tr>
<td>4. Alphanumeric Data</td>
<td>10</td>
</tr>
<tr>
<td>III. PSEUDO-OPERATIONS</td>
<td>11</td>
</tr>
<tr>
<td>A. BCI</td>
<td>11</td>
</tr>
<tr>
<td>B. BSS</td>
<td>12</td>
</tr>
<tr>
<td>C. CALL</td>
<td>12</td>
</tr>
<tr>
<td>D. DEC</td>
<td>12</td>
</tr>
<tr>
<td>E. EQU</td>
<td>13</td>
</tr>
<tr>
<td>F. END</td>
<td>13</td>
</tr>
<tr>
<td>G. EVEN</td>
<td>14</td>
</tr>
<tr>
<td>H. JMP</td>
<td>14</td>
</tr>
<tr>
<td>I. NAME</td>
<td>14</td>
</tr>
</tbody>
</table>
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 declaration, 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.
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 "1" 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 '*1 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 '*1 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:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>LDL</td>
<td>1</td>
</tr>
<tr>
<td>ADL</td>
<td>-5</td>
</tr>
<tr>
<td>ANL</td>
<td>'77</td>
</tr>
<tr>
<td>ANL</td>
<td>=A</td>
</tr>
<tr>
<td>XOL</td>
<td>+40</td>
</tr>
</tbody>
</table>

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 item 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 $-33554432 = -2^{24} \leq I = 2^{24} - 1 = 33554431$.

<table>
<thead>
<tr>
<th>Even Location Word 1:</th>
<th>SIGN VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 11</td>
</tr>
<tr>
<td></td>
<td>0 11</td>
</tr>
</tbody>
</table>

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 \(2^{-2048} \leq F \leq 2^{2048}\). The mantissa of the number must be less than \(2^{24-1}=33554431\).

<table>
<thead>
<tr>
<th>Even Location Word 1:</th>
<th>0</th>
<th>1</th>
<th>11</th>
<th>Sign of exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word 2:</td>
<td>0</td>
<td>11</td>
<td>Exponent</td>
<td></td>
</tr>
<tr>
<td>Word 3:</td>
<td>0</td>
<td>11</td>
<td>Mantissa (most significant)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&quot; (least significant)</td>
<td></td>
</tr>
</tbody>
</table>

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.)
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 even 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:

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:

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

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:

<table>
<thead>
<tr>
<th>NAME</th>
<th>A, B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>PAR</td>
</tr>
<tr>
<td></td>
<td>Æ</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>DEC</td>
</tr>
</tbody>
</table>

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:

\[
\begin{align*}
\text{ABC} &= '2200 \\
\text{DEF} &= \text{A} + 7
\end{align*}
\]

ABC has an operation code of '2200
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**

```
   0 1 2 3 4 5 6 7 8 9 10 11
   OP I R Y
```

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:

<table>
<thead>
<tr>
<th>MNN</th>
<th>Name</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code I R Y</td>
<td>is the SPL mnemonic assigned to the instruction</td>
<td></td>
</tr>
<tr>
<td>MNN</td>
<td>Name</td>
<td>is the instruction name.</td>
</tr>
<tr>
<td>Timing</td>
<td>is the number of machine cycles used by the instruction. Each level of indirect addressing adds one cycle to the instruction.</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>is the octal operation code of the instruction; and in some cases the complete 4-digit octal code instruction word will be given.</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>is present if the instruction can optionally contain an indirect address bit. Otherwise, the numeric value of I is placed in this position.</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>is present if the instruction can optionally contain a mode bit.</td>
<td></td>
</tr>
</tbody>
</table>
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.
### NOTATION

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>DEFINITION</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>The main arithmetic register, or Accumulator</td>
</tr>
<tr>
<td>X</td>
<td>The index register and left hand Accumulator extension</td>
</tr>
<tr>
<td>Y</td>
<td>Operand address of instruction</td>
</tr>
<tr>
<td>M</td>
<td>Effective operand address of instruction</td>
</tr>
<tr>
<td>P</td>
<td>The location counter. Contains the address of the instruction to be executed.</td>
</tr>
<tr>
<td>( )</td>
<td>Contents of. Signifies the contents of the symbol enclosed within the parentheses.</td>
</tr>
<tr>
<td></td>
<td>Replacement designator. The value on the left is placed into the value on the right.</td>
</tr>
<tr>
<td>+</td>
<td>Add</td>
</tr>
<tr>
<td>-</td>
<td>Subtract</td>
</tr>
<tr>
<td>/</td>
<td>Divide</td>
</tr>
<tr>
<td>*</td>
<td>Multiply</td>
</tr>
<tr>
<td>(\text{&amp;})</td>
<td>Logical AND</td>
</tr>
<tr>
<td>(\text{|})</td>
<td>Logical OR</td>
</tr>
<tr>
<td>(\oplus)</td>
<td>Exclusive OR</td>
</tr>
<tr>
<td>(</td>
<td>)</td>
</tr>
<tr>
<td></td>
<td>One's complement</td>
</tr>
<tr>
<td>(\text{&lt;})</td>
<td>Less than</td>
</tr>
<tr>
<td>(\text{&gt;})</td>
<td>Greater than</td>
</tr>
<tr>
<td>(=)</td>
<td>Equal</td>
</tr>
<tr>
<td>(\equiv)</td>
<td>Equivalent by definition</td>
</tr>
</tbody>
</table>
B. DATA TRANSFER INSTRUCTIONS

**LDA** Load A  
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  
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  
10 I R Y  
(M) → X  
The operand is placed into the index register. The contents of M remains unchanged.

**STX** Store Index  
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 \((M) = 0\), \((P) + 2 \rightarrow P\), otherwise \((P) + 1 \rightarrow P\).

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

\[ \begin{align*}
\text{Timing: 2} \\
\text{Forms the logical AND of the operand and the contents of A and places the result in A. The contents of memory at M remains unchanged.}
\end{align*} \]

**XOR**

Exclusive OR

\[ \begin{align*}
\text{Timing: 2} \\
\text{Forms the exclusive OR of the operand and the contents of A and places the result in A. The contents of M remains unchanged.}
\end{align*} \]

**D. CONTROL AND TEST INSTRUCTIONS**

**HLT**

Halt

\[ \text{Timing: 1} \]

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.
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 A11 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 A11 enters A0.

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 $A_0$ 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 $A_0$, is rotated right one binary place. Bit CO enters $A_0$ 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 $X_0$. 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 $X_0$. Bit $X_{11}$ enters $A_0$.

**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 $X_0$. Vacated bit position $A_0$ 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 A₀ and A₁₁ preceding X₀, is rotated right one binary place. Bit CO enters A₀, bit A₁₁ enters X₀ and bit X₁₁ enters CO.

**SAL** Short Arithmetic Shift Left Timing: 1

0016

(A) left 1 place → A

The contents of A is shifted left one binary place. Bit A₀ is lost. Vacated bit position A₁₁ is filled with a zero.

If the sign of A, A₀, changes, the overflow flip-flop is set.

**SRL** Short Rotate Left Timing: 1

0012

(A) left cycle 1 place → A

The contents of A is rotated left one binary place. Bit A₀ enters A₁₁.

**SLL** Short Logical Shift Left Timing: 1

0116

(A) left 1 place → A

The contents of A is shifted left one binary place. Bit A₀ is lost. Vacated bit position A₁₁ is filled with a zero.

**SCL** Short Circulate Left Timing: 1

0112

(A, CO) left 1 place → A

The coupled A- and CO-registers, with CO following bit A₁₁, is rotated left one binary place. Bit CO enters A₁₁ and bit A₀ 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
X₀ enters A₁₁. Bit A₀ is lost. Vacated bit position X₁₁ is filled
with a zero.

If the sign of A, A₀, 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
X₀ enters A₁₁. Bit A₀ enters X₁₁.

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
A₀ is lost. Bit X₀ enters A₁₁. Vacated bit position X₁₁ 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 X₁₁
and A₁₁ preceding X₀, is rotated left one binary place. Bit CO
enters X₁₁, bit X₀ enters A₁₁ and Bit A₀ 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

\(30\ Y\)

\((A_0-5) \rightarrow A_0-5,\ (A_6-11) \odot Y \rightarrow A_6-11\)

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.

XOR EXCLUSIVE OR Literal

\(31\ Y\)

\((A_0-5) \rightarrow A_0-5,\ (A_6-11) \odot \bar{Y} \oplus (A_6-11) \odot Y = (A_6-11) \oplus Y \rightarrow A_6-11\)

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

\(32\ Y\)

\(Y_0 \equiv \text{Sign-bit of } Y \rightarrow A_0-5\)
\(Y \equiv Y_0-5 \rightarrow A_6-11\)

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 \( Y_0 = \text{Sign-bit of} \ Y \rightarrow Y_{0-5}, \ Y = Y_{0-5} \rightarrow Y_{6-11}, \) then
\( (A) + Y' \rightarrow A \)

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-OPEARATE 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-in-
structions by assembling the proper codes from the information supplied
in the following format.

\[ \text{OP} \ F, T \]

where:

- \( \text{OP} \) 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 0 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 0.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Timing</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIN</td>
<td>Set Interrupt Control ON</td>
<td>1</td>
</tr>
<tr>
<td>0322</td>
<td>Sets the Interrupt Control Register bit to 1.</td>
<td></td>
</tr>
<tr>
<td>OPT</td>
<td>Overflow Test</td>
<td>1</td>
</tr>
<tr>
<td>0024</td>
<td>If Overflow Register ON, reset it to OFF, and (P)+1 → P; otherwise (P)+2 → P.</td>
<td></td>
</tr>
<tr>
<td>SBK</td>
<td>Set Bank Flag</td>
<td>1</td>
</tr>
<tr>
<td>0026</td>
<td>Sets inhibit IB-Register Flag ON.</td>
<td></td>
</tr>
<tr>
<td>RBK</td>
<td>Restore Bank Flag</td>
<td>1</td>
</tr>
<tr>
<td>0126</td>
<td>Resets the Inhibit IB-Register flag to OFF.</td>
<td></td>
</tr>
<tr>
<td>CLI</td>
<td>Clear Interrupt</td>
<td>1</td>
</tr>
<tr>
<td>0226</td>
<td>Resets the interrupt.</td>
<td></td>
</tr>
<tr>
<td>COT</td>
<td>CO Carry-Out Test</td>
<td>1</td>
</tr>
<tr>
<td>0032</td>
<td>If (CO) = 0, (P)+1 → P; otherwise, if (CO) = 1, (P)+2 → P.</td>
<td></td>
</tr>
<tr>
<td>XSS</td>
<td>Index Status Set</td>
<td>1</td>
</tr>
<tr>
<td>0034</td>
<td>Sets the Index Status Register to 1.</td>
<td></td>
</tr>
<tr>
<td>XSR</td>
<td>Index Status Reset</td>
<td>1</td>
</tr>
<tr>
<td>0234</td>
<td>Resets the Index Status Register to Ø.</td>
<td></td>
</tr>
</tbody>
</table>
LIA Load IB Register

Timing: 1

0036

(A9_{11}) \rightarrow IB

Stores (A9_{11}) into the IB Register

AAX Logical AND A with X

Timing: 1

0242

(X) \& (A) \rightarrow A

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

AOX Logical OR A with X

Timing: 1

0042

(X) \oplus (A) \rightarrow 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 \rightarrow P;
otherwise, if OFF, (P)+2 \rightarrow P.

IOT Input/Output Error Test

Timing: 1

0046

IF I/O Error Register ON, reset it to OFF, and (P)+1 \rightarrow P;
otherwise, if OFF, (P)+2 \rightarrow P.

LAS Load from Switch Register

Timing: 1

0050

(Switch) \rightarrow 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 (I4=0 for program bank, I4=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 40008, 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:

\[ -1 \leq \frac{(A, X)}{(N)} \leq 1 \]

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

(A0-5) → A0-5,
(Shift) → A6-11

The contents of the shift Register is copied into A6-11. A0-5 remains unchanged.

**LSH** Load Shift Register

Timing: 1

0266

(A0-5) → A0-5,
(A6-11) → Shift

The contents of A6-11 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

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>OP</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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=n1, L2=n2 (C/R)

where L1 and L2 represents labels in the source tape, or the "*" symbol (see below), n1 and n2 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 n1 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 \( L_1 = n_1 \) is encountered. At this point all lines from \( L_1 = n_1 \) to \( L_2 = n_2 \), 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 \( L_1 \) or \( L_2 \) with the following meaning:

For example:

\[
/D*+1,*+5 \text{ C/R}
\]

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

<table>
<thead>
<tr>
<th>mnemonic</th>
<th>description</th>
<th>page</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCI</td>
<td>Alphanumeric Character Data</td>
<td>11</td>
</tr>
<tr>
<td>BSS</td>
<td>Reserve Data Storage</td>
<td>12</td>
</tr>
<tr>
<td>CALL</td>
<td>Call</td>
<td>12</td>
</tr>
<tr>
<td>DEC</td>
<td>Double Precision Decimal or Floating Point Data</td>
<td>12</td>
</tr>
<tr>
<td>END</td>
<td>Program End</td>
<td>13</td>
</tr>
<tr>
<td>EQU</td>
<td>Symbol Equivalence</td>
<td>13</td>
</tr>
<tr>
<td>EVEN</td>
<td>Make next location Even</td>
<td>14</td>
</tr>
<tr>
<td>JMP</td>
<td>Jump</td>
<td>14</td>
</tr>
<tr>
<td>NAME</td>
<td>Program Name</td>
<td>14</td>
</tr>
<tr>
<td>ORG</td>
<td>Program Origin</td>
<td>14</td>
</tr>
<tr>
<td>PAR</td>
<td>Parameter String</td>
<td>15</td>
</tr>
</tbody>
</table>
## APPENDIX B

### SCC 650 MACHINE INSTRUCTIONS

#### Data Transfer

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDA 60</td>
<td>Load A</td>
<td>20</td>
</tr>
<tr>
<td>STA 34</td>
<td>Store A</td>
<td>20</td>
</tr>
<tr>
<td>LDX 10</td>
<td>Load Index</td>
<td>20</td>
</tr>
<tr>
<td>STX 14</td>
<td>Store Index</td>
<td>20</td>
</tr>
</tbody>
</table>

#### Arithmetic

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD 44</td>
<td>Add</td>
<td>21</td>
</tr>
<tr>
<td>SUB 64</td>
<td>Subtract</td>
<td>21</td>
</tr>
<tr>
<td>MIN 40</td>
<td>Memory Increment</td>
<td>21</td>
</tr>
</tbody>
</table>

#### Logical

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND 74</td>
<td>Logical AND</td>
<td>22</td>
</tr>
<tr>
<td>XOR 54</td>
<td>Exclusive OR</td>
<td>22</td>
</tr>
</tbody>
</table>

#### Control and Test

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>HLT 0000</td>
<td>Halt</td>
<td>22</td>
</tr>
<tr>
<td>NOP 0002</td>
<td>No Operation</td>
<td>23</td>
</tr>
<tr>
<td>JMF 20</td>
<td>Jump Forward</td>
<td>23</td>
</tr>
<tr>
<td>JMB 24</td>
<td>Jump Backward</td>
<td>23</td>
</tr>
<tr>
<td>JSL 50</td>
<td>Jump and Store Location</td>
<td>23</td>
</tr>
<tr>
<td>JRT 70</td>
<td>Return Jump</td>
<td>24</td>
</tr>
</tbody>
</table>

#### Shift

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAR 0014</td>
<td>Short Arithmetic Shift Right</td>
<td>24</td>
</tr>
<tr>
<td>SRR 0010</td>
<td>Short Rotate Right</td>
<td>24</td>
</tr>
<tr>
<td>SLR 0114</td>
<td>Short Logical Shift Right</td>
<td>24</td>
</tr>
<tr>
<td>SCR 0110</td>
<td>Short Circulate Right</td>
<td>25</td>
</tr>
<tr>
<td>LAR 0214</td>
<td>Long Arithmetic Shift Right</td>
<td>25</td>
</tr>
<tr>
<td>LRR 0210</td>
<td>Long Rotate Right</td>
<td>25</td>
</tr>
<tr>
<td>LLR 0314</td>
<td>Long Logical Shift Right</td>
<td>25</td>
</tr>
<tr>
<td>LCR 0310</td>
<td>Long Circulate Right</td>
<td>25</td>
</tr>
<tr>
<td>SAL 0016</td>
<td>Short Arithmetic Shift Left</td>
<td>26</td>
</tr>
<tr>
<td>SRL 0012</td>
<td>Short Rotate Left</td>
<td>26</td>
</tr>
<tr>
<td>SLL 0116</td>
<td>Short Logical Shift Left</td>
<td>26</td>
</tr>
<tr>
<td>SCL 0112</td>
<td>Short Circulate Left</td>
<td>26</td>
</tr>
<tr>
<td>LAL 0216</td>
<td>Long Arithmetic Shift Left</td>
<td>27</td>
</tr>
<tr>
<td>LRL 0212</td>
<td>Long Rotate Left</td>
<td>27</td>
</tr>
<tr>
<td>LLL 0316</td>
<td>Long Logical Shift Left</td>
<td>27</td>
</tr>
<tr>
<td>LCL 0312</td>
<td>Long Circulate Left</td>
<td>27</td>
</tr>
<tr>
<td>Register Change</td>
<td>Page</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>CLA 0003</td>
<td>Clear A</td>
<td>28</td>
</tr>
<tr>
<td>CLX 0007</td>
<td>Clear X</td>
<td>28</td>
</tr>
<tr>
<td>CAX 0240</td>
<td>Copy A to X</td>
<td>28</td>
</tr>
<tr>
<td>CXA 0140</td>
<td>Copy X to A</td>
<td>28</td>
</tr>
<tr>
<td>XAX 0040</td>
<td>Exchange A and X</td>
<td>28</td>
</tr>
<tr>
<td>XHA 0020</td>
<td>Exchange half A</td>
<td>29</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Literal</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ANL 00</td>
<td>Logical AND Literal</td>
</tr>
<tr>
<td>XOL 01</td>
<td>EXCLUSIVE OR Literal</td>
</tr>
<tr>
<td>LDL 02</td>
<td>Load A Literal</td>
</tr>
<tr>
<td>ADL 03</td>
<td>Add Literal</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Micro-Operate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SRA F,T</td>
<td>Select Register A (0 ≤ F ≤ 7; T=F,N,Z)</td>
</tr>
<tr>
<td>SRX F,T</td>
<td>Select Register X (0 ≤ F ≤ 7; T=F,N,Z)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmed-Operate</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>XHA 0020</td>
<td>Exchange Halves of A</td>
</tr>
<tr>
<td>SCF 0022</td>
<td>Set Carry-Out (CO) Register OFF</td>
</tr>
<tr>
<td>SCN 0122</td>
<td>Set Carry-Out (CO) Register ON</td>
</tr>
<tr>
<td>SIF 0222</td>
<td>Set Interrupt Control OFF</td>
</tr>
<tr>
<td>SIN 0322</td>
<td>Set Interrupt Control ON</td>
</tr>
<tr>
<td>OPT 0024</td>
<td>Overflow Test</td>
</tr>
<tr>
<td>SBK 0026</td>
<td>Set Bank Flag</td>
</tr>
<tr>
<td>RBK 0126</td>
<td>Restore Banks</td>
</tr>
<tr>
<td>COT 0022</td>
<td>Carry-Out (CO) Register Test</td>
</tr>
<tr>
<td>XSS 0034</td>
<td>Index State Set ON</td>
</tr>
<tr>
<td>XSR 0234</td>
<td>Index State Set OFF</td>
</tr>
<tr>
<td>LIA 0036</td>
<td>Load Indirect Extension (IB) Register</td>
</tr>
<tr>
<td>AAX 0242</td>
<td>A and X Registers Logical Product</td>
</tr>
<tr>
<td>AOX 0042</td>
<td>A OR X Registers Logical Sum</td>
</tr>
<tr>
<td>MPT 0044</td>
<td>Memory Protect Test</td>
</tr>
<tr>
<td>IOT 0046</td>
<td>Input/Output Error Test</td>
</tr>
<tr>
<td>LAS 0050</td>
<td>Load Switch Register</td>
</tr>
<tr>
<td>ADC 0054</td>
<td>Add with Carry</td>
</tr>
<tr>
<td>SST 0060</td>
<td>Store Status Register</td>
</tr>
<tr>
<td>LST 0260</td>
<td>Load Status Register</td>
</tr>
<tr>
<td>CLI 0226</td>
<td>Clear Interrupt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extended Operations</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH 0066</td>
<td>Store Shift Register</td>
</tr>
<tr>
<td>LSH 0266</td>
<td>Load Shift Register</td>
</tr>
<tr>
<td>SSH 0266</td>
<td>Load Shift Register</td>
</tr>
<tr>
<td>RPT 0078</td>
<td>Repeat</td>
</tr>
<tr>
<td>NOR 0072</td>
<td>Normalize AX Registers</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------</td>
</tr>
<tr>
<td>MPY 074</td>
<td>Multiply</td>
</tr>
<tr>
<td>DVD 174</td>
<td>Divide</td>
</tr>
<tr>
<td>LDD 076</td>
<td>Load Double Length AX Registers</td>
</tr>
<tr>
<td>STD 176</td>
<td>Store Double Length AX Registers</td>
</tr>
<tr>
<td>TTA 04D</td>
<td>Transmit from A or Skip</td>
</tr>
<tr>
<td>TFA 04D</td>
<td>Transmit to A or Skip</td>
</tr>
<tr>
<td>SDF 54D</td>
<td>Skip No Device Flag</td>
</tr>
<tr>
<td>SNL 70D</td>
<td>Select with no leader</td>
</tr>
<tr>
<td>SWL 74D</td>
<td>Select with leader</td>
</tr>
<tr>
<td>TMR 64D</td>
<td>Terminate</td>
</tr>
<tr>
<td>DST 50D</td>
<td>Input Device Status Test</td>
</tr>
<tr>
<td>EXU 60D</td>
<td>Execute Command in A</td>
</tr>
</tbody>
</table>
## APPENDIX C

### SCC 650 STANDARD CHARACTER CODES

<table>
<thead>
<tr>
<th>Character</th>
<th>ASCII</th>
<th>6-Bit Code</th>
<th>Character</th>
<th>ASCII</th>
<th>6-Bit Code</th>
<th>Character</th>
<th>ASCII</th>
<th>6-Bit Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>~</td>
<td>260</td>
<td>00100100</td>
<td>-</td>
<td>255</td>
<td>01111111</td>
<td>40</td>
<td>01010000</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>261</td>
<td>00100101</td>
<td>J</td>
<td>41</td>
<td>01000001</td>
<td>60</td>
<td>01110000</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>262</td>
<td>00100110</td>
<td>K</td>
<td>42</td>
<td>01000010</td>
<td>70</td>
<td>01110100</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>263</td>
<td>00100111</td>
<td>L</td>
<td>43</td>
<td>01000011</td>
<td>80</td>
<td>01111000</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>264</td>
<td>00101000</td>
<td>M</td>
<td>44</td>
<td>01000100</td>
<td>90</td>
<td>01111100</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>265</td>
<td>00101001</td>
<td>N</td>
<td>45</td>
<td>01000101</td>
<td>100</td>
<td>00001000</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>266</td>
<td>00101010</td>
<td>O</td>
<td>46</td>
<td>01000110</td>
<td>110</td>
<td>00001001</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>267</td>
<td>00101011</td>
<td>P</td>
<td>47</td>
<td>01000111</td>
<td>120</td>
<td>00001010</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>270</td>
<td>00101100</td>
<td>Q</td>
<td>50</td>
<td>01001000</td>
<td>130</td>
<td>00001011</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>271</td>
<td>00101101</td>
<td>R</td>
<td>51</td>
<td>01001001</td>
<td>140</td>
<td>00001100</td>
<td></td>
</tr>
<tr>
<td>SPACE</td>
<td>12</td>
<td>240</td>
<td>CHAR.RET.</td>
<td>52</td>
<td>215</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>13</td>
<td>275</td>
<td>¥</td>
<td>53</td>
<td>244</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>:</td>
<td>14</td>
<td>247</td>
<td>*</td>
<td>54</td>
<td>252</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>15</td>
<td>272</td>
<td>]</td>
<td>55</td>
<td>335</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&gt;</td>
<td>16</td>
<td>276</td>
<td>i</td>
<td>56</td>
<td>273</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>17</td>
<td>246</td>
<td>()</td>
<td>57</td>
<td>245</td>
<td>(%)</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>+</td>
<td>20</td>
<td>253</td>
<td>¥</td>
<td>60</td>
<td>277</td>
<td>(?)</td>
<td>95</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>21</td>
<td>301</td>
<td>/</td>
<td>61</td>
<td>257</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>22</td>
<td>302</td>
<td>S</td>
<td>62</td>
<td>323</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>23</td>
<td>303</td>
<td>T</td>
<td>63</td>
<td>324</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>24</td>
<td>304</td>
<td>U</td>
<td>64</td>
<td>325</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>25</td>
<td>305</td>
<td>V</td>
<td>65</td>
<td>326</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>26</td>
<td>306</td>
<td>W</td>
<td>66</td>
<td>327</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>27</td>
<td>307</td>
<td>X</td>
<td>67</td>
<td>330</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>30</td>
<td>310</td>
<td>Y</td>
<td>70</td>
<td>331</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>31</td>
<td>311</td>
<td>Z</td>
<td>71</td>
<td>332</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BACKSPACE</td>
<td>32</td>
<td>293</td>
<td>(EO)</td>
<td>72</td>
<td>211</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>33</td>
<td>241</td>
<td>\</td>
<td>73</td>
<td>254</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>)</td>
<td>34</td>
<td>251</td>
<td>(</td>
<td>74</td>
<td>250</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[</td>
<td>35</td>
<td>333</td>
<td>\</td>
<td>75</td>
<td>243</td>
<td>(#)</td>
<td>91</td>
<td></td>
</tr>
<tr>
<td>\</td>
<td>36</td>
<td>274</td>
<td>\</td>
<td>76</td>
<td>334</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>≠</td>
<td>37</td>
<td>241</td>
<td>(≠)</td>
<td>77</td>
<td>300</td>
<td>(≠)</td>
<td>91</td>
<td></td>
</tr>
</tbody>
</table>

**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 — 641-3556

Pasadena, California
180 East California Blvd.
213 — 661-2651
(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