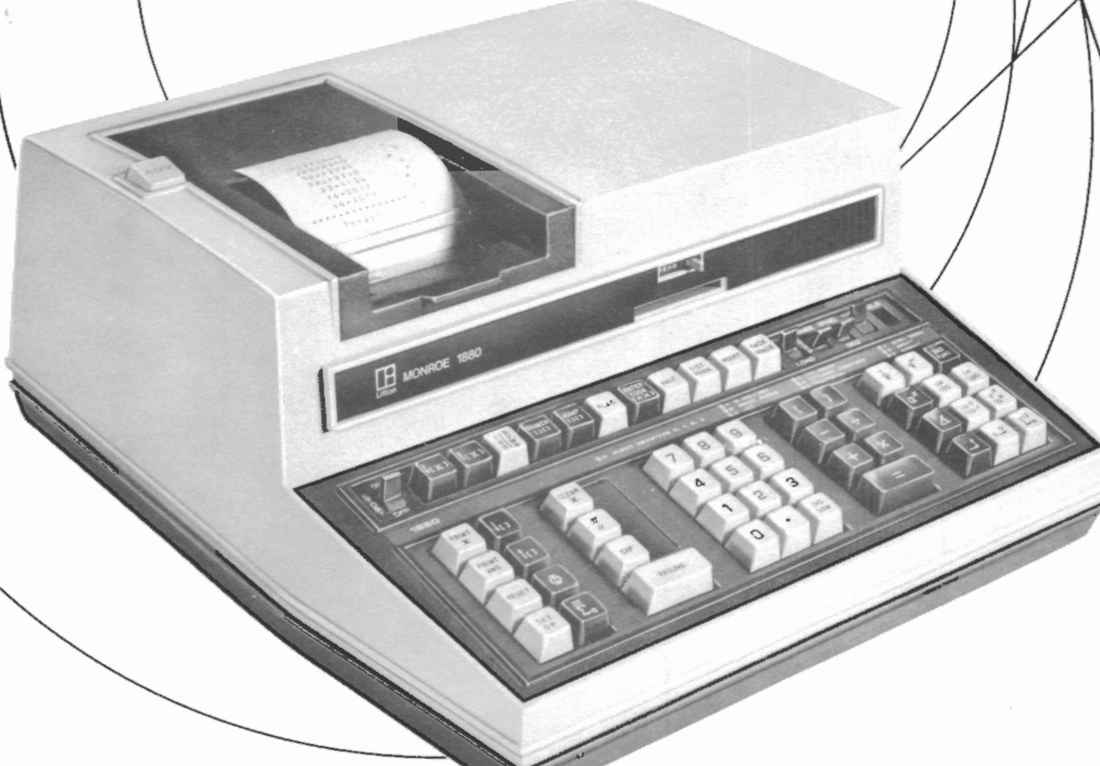
PROGRAMMING REFERENCE MANUAL

Litton MONROE

model 1880 Scientific Programmable Printing Calculator



Monroe, The Calculator Company

CONTENTS

CONTENTS (Cont.)

ILLUSTRATIONS

ILLUSTRATIONS (Cont.)

TABLES

INTRODUCTION

This manual introduces basic programming techniques and capabilities of the Monroe Model 1880 Scientific Calculator. Before reading this material, you should be thoroughly familiar with the keyboard operations of the calculator as described in the Model 1880 Scientific Calculator Operating Instructions Manual. Additionally, a basic introduction to programming is provided by the Monroe primer, Fundamentals of Programming.

General topics of discussion in this reference manual include the capacity and storage scheme of the calculator; non-keyboard, as well as keyboard, instructions; typical programming techniques, such as branching and jumping; different methods of memory addressing; and general procedures for program execution. Details of the calculator architecture and macro-instruction repertoire are presented in the Advanced Programming Reference Manual for the Model 1800 Series Programmable Calculators.

I. CALCULATOR FUNCTIONS

The operations of the calculator fall into six functional categories:

- Control of calculator operations
- Input of data and instructions
- Storage of data and instructions
- Execution of a program
- Arithmetic computations
- Output of data, instructions, and messages

The operation of these functions is shown schematically in figure 1-1. Each function is discussed in the following paragraphs.

CONTROL

Calculator operations are controlled both by keyboard manipulations and by program instructions. Typical keyboard-controlled functions are printing data, setting the decimal point format, and clearing registers. Keys and switches for these controls are explained in the Operating Instructions Manual. Typical control instructions set and reset internal flags. These instructions are explained in section III.

INPUT

Data may be loaded into data storage and instructions into program memory from the keyboard, from magnetic cards, or from peripheral devices. Card reader input always goes directly to memory, whereas the keyboard can give instructions directly to the control and arithmetic unit.

STORAGE

Data and instructions are stored into and accessed from the calculator's storage registers. Memory contents are lost when the calculator is turned off, but are retained if the Power switch is set to the STDBY position. The calculator contains four kinds of storage registers: working registers, scratch pad registers, main data memory registers, and program memory registers. These registers are described in section II.

PROGRAM EXECUTION

A program must be stored in memory before it can be executed. Programs may be entered from peripheral devices or directly from the keyboard or magnetic cards. Regardless of the input mode, the program must be loaded, beginning at a proper point

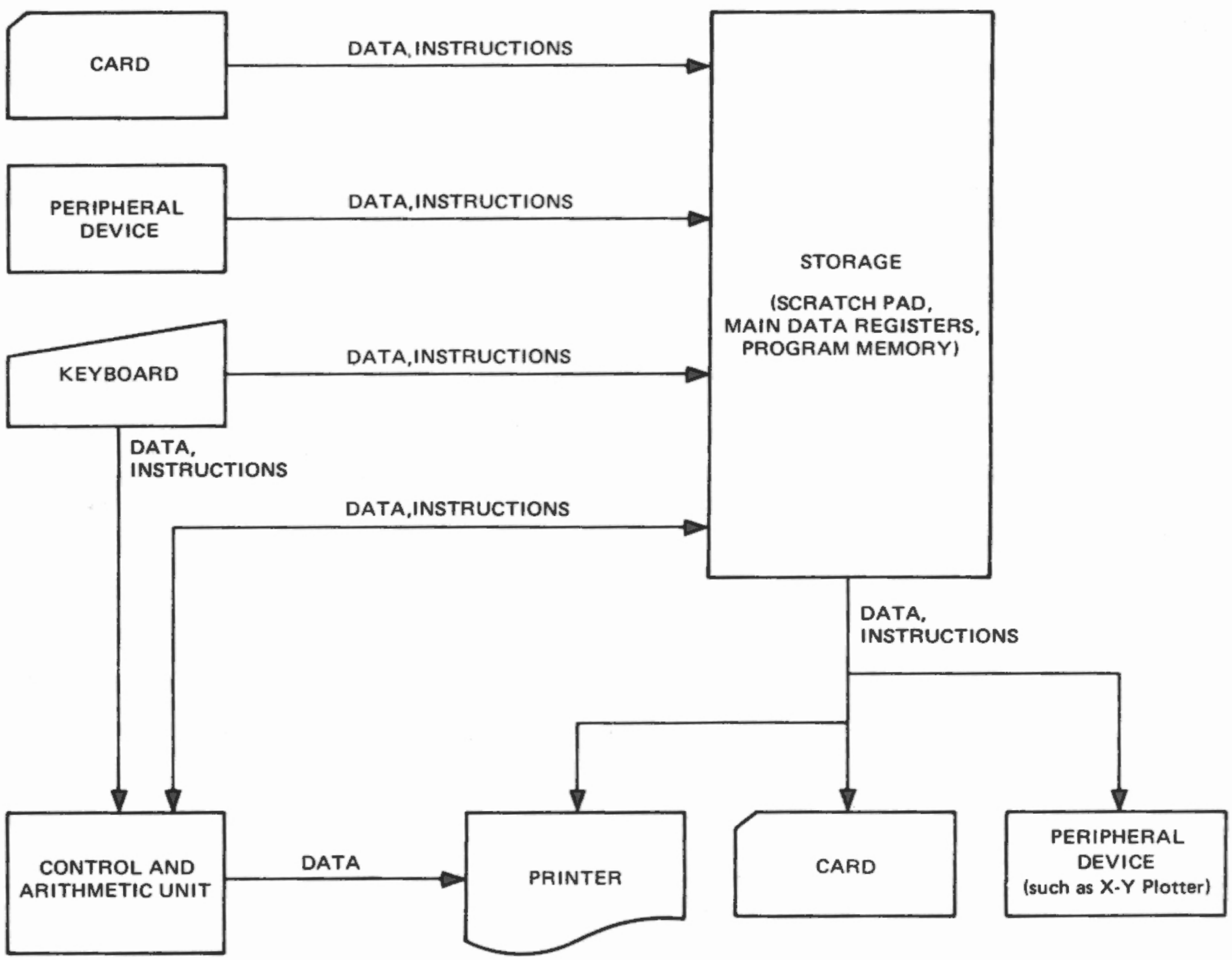


Figure 1-1. Calculator Functions

in memory. (If you are not familiar with techniques for program loading and execution, see section V for detailed loading procedures.) The calculator will perform the programmed operations. If data is to be entered from the keyboard during the course of program execution, a Halt instruction in the program will temporarily suspend program operation so that the operator may key in the necessary data. Depressing the RESUME key continues program execution. The calculator fetches data and executes instructions in memory as directed; branches or jumps according to instructions, flag settings, or SENSE switch settings; performs computations as programmed; and, outputs the results either to the printer or to other peripheral devices.

ARITHMETIC FUNCTIONS

The functional unit referred to as the control and arithmetic unit includes many complex operations and functions. This unit performs the operations necessary to carry out keyboard and non-keyboard instructions and to provide results for display on the printer tape.

OUTPUT

The calculator has two types of output: calculated results and memory contents; that is, data and instructions. The results of calculations are normally printed by the printer or written on magnetic cards (see figure 1-1). However, data and instructions can be transferred from memory to peripheral devices such as an X – Y Plotter.

II. PROGRAM AND DATA STORAGE

As mentioned in section I, the calculator has working registers, scratch pad registers, main data memory registers, and program memory registers.

WORKING REGISTERS

Working registers are used in arithmetic computations. Except for the entry register (commonly called the E-register), they are not available to the user. All input data from the keyboard and output data to the printer go through the E-register, as shown in figure 2-1. The E-register is also used in arithmetic computations.

The E-register will accommodate a 13-digit signed number (mantissa), with a signed 2-digit exponent. When a number from the E-register is stored into a scratch pad register or main data register (see below), the number also remains unchanged in the E-register. (Similarly, a number entered into the E-register from a scratch pad or main data register remains unchanged in the scratch pad register or main data register.)

Either the [PRINT x] or the [PRINT ANS] key will print the contents of the E-register, regardless of any PRINT switch setting. The [PRINT ANS] printout is identified by a letter A to the right of the data; [PRINT ANS] rounds the number to the decimal place selected; the number is also rounded in the E-register.

Two keyboard keys clear (set to zero) the E-register. The [CLEAR x] key clears the E-register without affecting any operations already in progress. The [RESET] key clears the E-register and nullifies any operation in progress.

Special function 0 (key combination [Φ] [0], "Clear Register"; see the Operating Instructions Manual) clears the E-register, as well as scratch pad registers 0, 1, 2, and 3. Finally, changing the Power switch from STDBY to ON retains memory contents, but also acts as a reset operation; that is, it clears the E-register and nullifies any operation in progress.

SCRATCH PAD REGISTERS

Ten scratch pad registers, numbered 0 through 9, are available to the user. Scratch pad registers are accessed from the keyboard by using numeral keys 0 through 9.

Data is entered into a scratch pad register from the E-register by depressing the [↓( )] key and the numeral key for the number of the desired scratch pad register. After a number from the E-register has been stored into a scratch pad register, it still remains in the E-register.

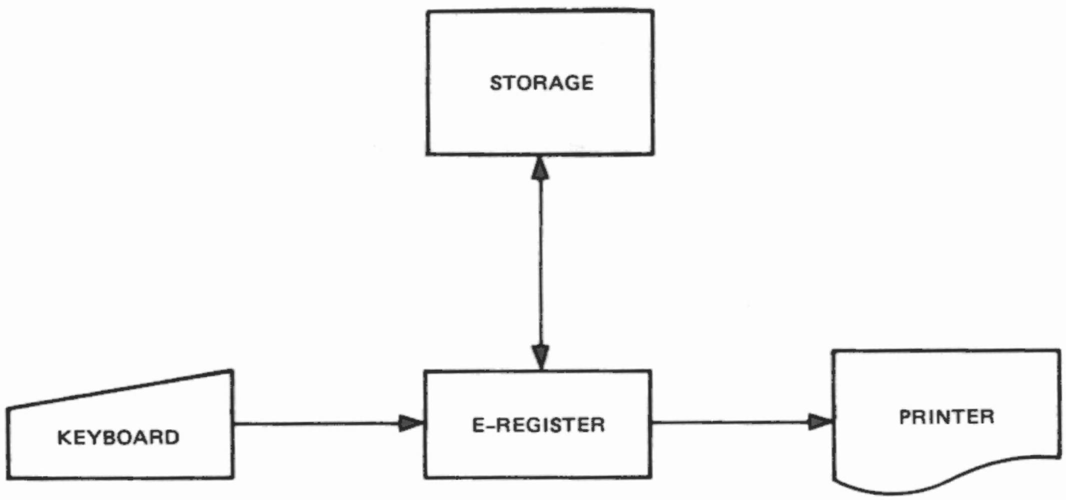


Figure 2-1. Entry Register Functions

Data is retrieved from a scratch pad register and returned to the E-register by depressing the [↑( )] key and the numeral key for the number of the scratch pad register that contains the desired data. After a number has been returned to the E-register from a scratch pad register, the number remains in the accessed scratch pad register.

Scratch pad registers are cleared by entering a 0 into the E-register, and then storing the 0 into the selected scratch pad registers. (Key combination [Φ] [0] clears scratch pad registers 0, 1, 2, and 3.)

Register arithmetic operations are described in the Operating Instructions Manual.

MAIN DATA MEMORY REGISTERS

Basic main data memory has 64 registers, numbered 00 through 63. This configuration can be expanded to 512 registers, in increments of 64. Main data memory registers may be accessed from the keyboard by using numeral keys 0 through 9 and keys [•], [CHG SIGN], [EXP], and [π e].

Data from the E-register is stored into a main data memory register by depressing the [↓( )( )] key and the appropriate numeral keys for main data memory registers 00 through 99. For registers 100 through 199, the keyboard sequence is [↓( )( )] [•] [n] [n], where [n] represents a numeral key. Similarly, registers 200 through 299 are accessed by the keyboard sequence [↓( )( )] [CHG SIGN] [n] [n]; registers 300 through 399, by keys [↓( )( )] [EXP] [n] [n]; and registers 400 through 499, by keys [↓( )( )] [π e] [n] [n]. Data registers 500 through 511 are accessed by using indirect addressing, as discussed in section IV of this manual. After a number from the E-register has been stored in main data memory, it still remains in the E-register.

Data is retrieved from a main data memory register and returned to the E-register by depressing [↑( )( )] and the appropriate numeral keys for main data memory registers 00 through 99. To recall data from main data memory registers 100 through 199, the keyboard sequence is [↑( )( )] [•] [n] [n], where [n] represents a numeral key. Similarly, data is recalled from registers 200 through 299 by the keyboard sequence [↑( )( )] [CHG SIGN] [n] [n]; from registers 300 through 399, by keys [↑( )( )] [EXP] [n] [n]; and from registers 400 through 499, by keys [↑( )( )] [π e] [n] [n]. Data registers 500 through 511 are accessed by using indirect addressing, as described in section IV of this manual. After a number has been returned to the E-register, it still remains in its main data memory register.

PROGRAM MEMORY

Basic program memory has 512 locations. This configuration can be expanded to 4096 locations, in increments of 512. Since program memory is used primarily for storing instructions that are part of a program, the individual locations where codes are stored are referred to as program steps. Every tenth step is designated a branch point.

Each step may hold one 3-digit code, which may represent an instruction, an address, or one digit of a data constant. The calculator has two kinds of instruction codes: keyboard codes and non-keyboard codes. They are explained in the following paragraphs.

KEYBOARD CODES

Most keys have corresponding keyboard codes. Keyboard codes are stored in successive program memory locations by depressing individual keys when the calculator is set for loading (RUN/STEP/LOAD switch in the LOAD position). For example, the [+] key stores code 021. A complete list of keyboard codes is given in appendix A.

NON-KEYBOARD CODES

In addition to its repertoire of keyboard instructions, the calculator also accepts non-keyboard codes. These codes are 3-digit codes that represent macro instructions. They provide the user with an additional set of calculations to be performed during programmed operation.

Non-keyboard codes are stored in program memory by setting the RUN/STEP/LOAD switch to LOAD, and then depressing the [ENTER CODE 000] key and the three numeral keys of the code. For example, depressing [ENTER CODE 000] [0] [4] [5] enters the non-keyboard code for the "Absolute Value" operation. Non-keyboard codes are explained in detail in section III of this manual. A list of these codes is given in appendix B.

MEMORY ADDRESSING

A program step number or a main data memory register number is called an address, because the number identifies the place in memory where the program step or the item of data is stored. When a step or register is to be addressed for storing information or finding what information is already stored, the address is placed in a counter that "points" to the location of the step or register. The counter for the program memory is called the program counter or P-counter. The main data memory counter is called the data counter. Figure 2-2 shows the function of the program and data counters. Addressing techniques for program and main data memory are discussed in the following paragraphs.

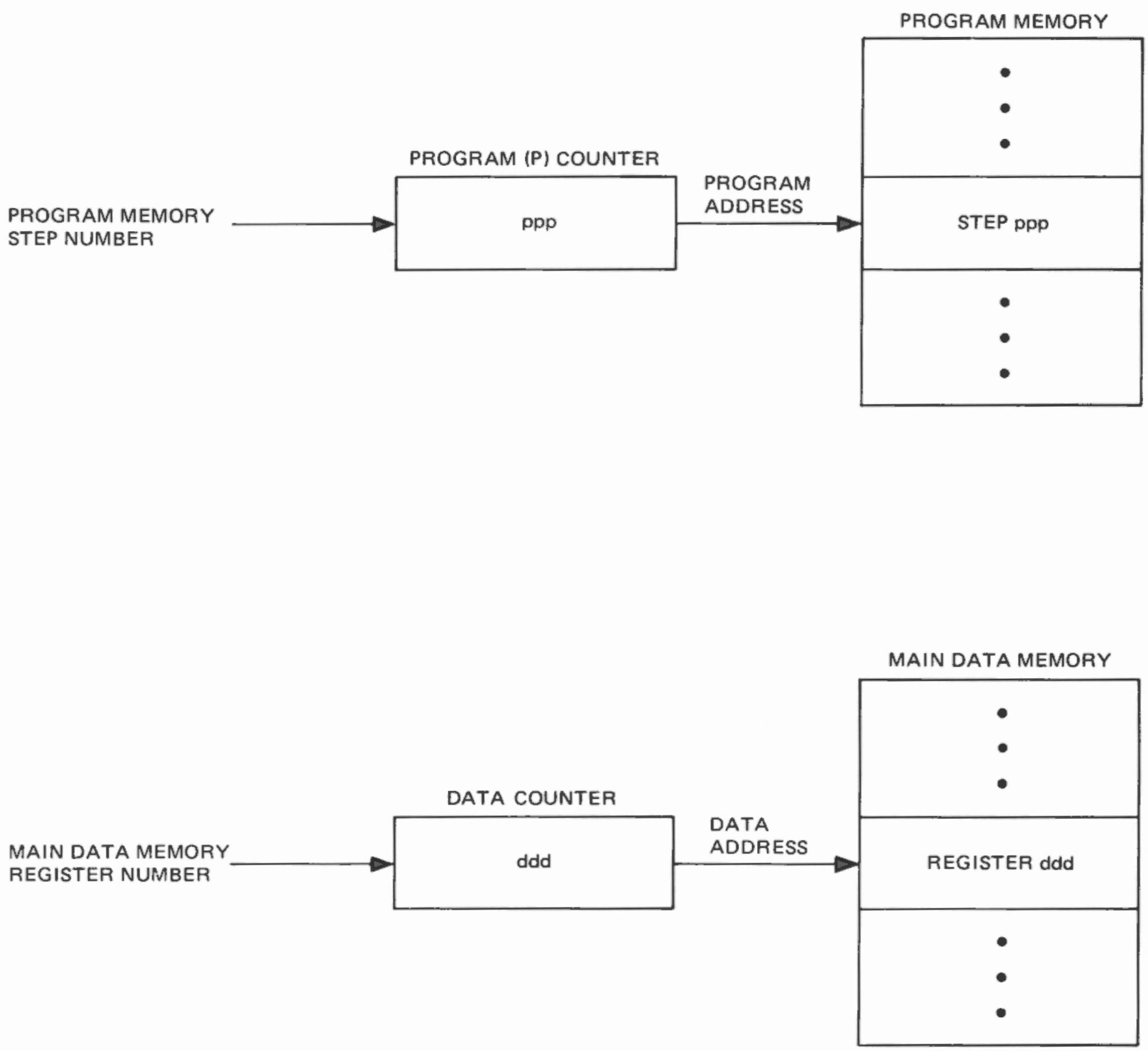


Figure 2-2. Program and Data Counters

ADDRESSING PROGRAM MEMORY

Program steps are numbered sequentially, starting from step 0. Only branch point addresses, that is, step numbers that are multiples of 10, may be set into the program counter. (Intermediate steps may be addressed symbolically or with machine instructions. For further information, see sections III and IV of this manual.)

Branch point addresses are selected by depressing either the [BRANCH ( ) ( )] or the [JUMP ( ) ( )] key and the two numeral keys that designate the desired branch point. The [BRANCH ( ) ( )] instruction automatically saves the address of the instruction following it; hence, the Branch instruction is used as an entry to subroutines. The [JUMP ( ) ( )] instruction does not save an address. (See section IV for discussions of branching and jumping techniques.) Branch points 0 through 399 are addressed as shown in table 2-1. (Branch points 400 through 409 can be addressed only through special codes, discussed in the Advanced Programming Reference Manual.) Two typical program addressing operations are shown in figure 2-3.

ADDRESSING MAIN DATA MEMORY

Any main data memory register number may be set into the data counter by using the [↓( )( )] or [↑( )( )] key and the numeral keys that correspond to the desired register. The numeral keys for registers 00 through 99 are [0] [0] through [9] [9]. The keys for registers 100 through 199 are [•] [0] [0] through [•] [9] [9]; those for registers 200 through 299 are [CHG SIGN] [0] [0] through [CHG SIGN] [9] [9]; those for registers 300 through 399 are [EXP] [0] [0] through [EXP] [9] [9]; and those for registers 400 through 499 are [π e] [0] [0] through [π e] [9] [9]. Two typical main data memory addressing operations are shown in figure 2-4. If you don't know which main data memory registers are available when you load and execute a program, indirect addressing may be used. This technique permits you to select available registers at the time you run your program. For information on this programming technique, see Indirect Data Addressing in section IV.

Table 2-1. Branch Point Designations

Keys	Step No.	Branch Point
0 0	0	0
0 1	10	1
0 2	20	2
0 3	30	3
0 4	40	4
0 5	50	5
0 6	60	6
0 7	70	7
0 8	80	8
0 9	90	9
1 0	100	10
1 1	110	11
⋮	⋮	⋮
9 9	990	99

Keys	Step No.	Branch Point
• 0 0	1000	100
• 0 1	1010	101
• 0 2	1020	102
• 0 3	1030	103
• 0 4	1040	104
• 0 5	1050	105
• 0 6	1060	106
• 0 7	1070	107
• 0 8	1080	108
• 0 9	1090	109
• 1 0	1100	110
• 1 1	1110	111
⋮	⋮	⋮
• 9 9	1990	199
CHG SIGN 0 0	2000	200
CHG SIGN 0 1	2010	201
CHG SIGN 0 2	2020	202
CHG SIGN 0 3	2030	203
CHG SIGN 0 4	2040	204
⋮	⋮	⋮
CHG SIGN 9 9	2990	299
EXP 0 0	3000	300
EXP 0 1	3010	301
⋮	⋮	⋮
EXP 9 9	3990	399

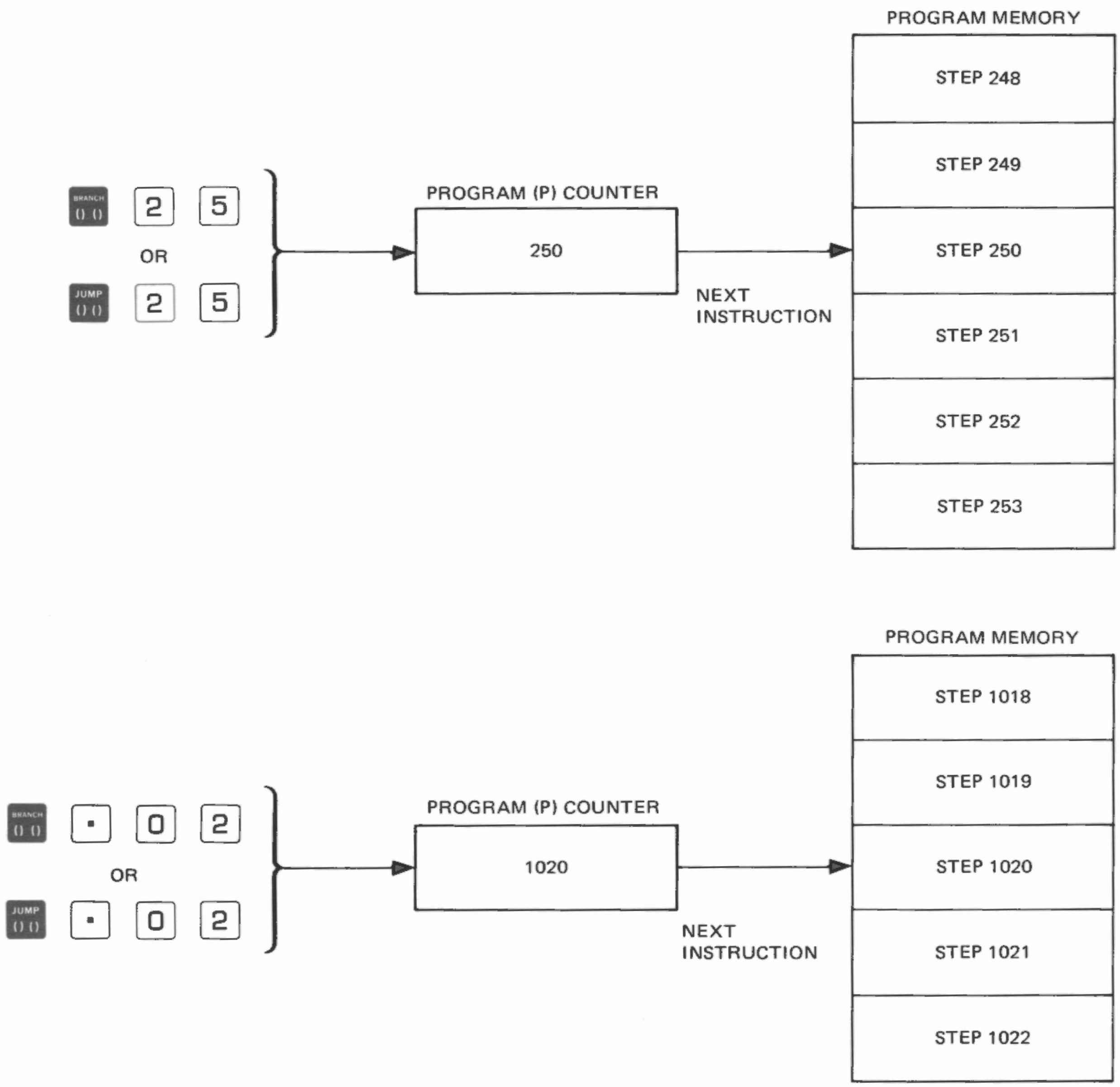


Figure 2-3. Addressing Program Memory

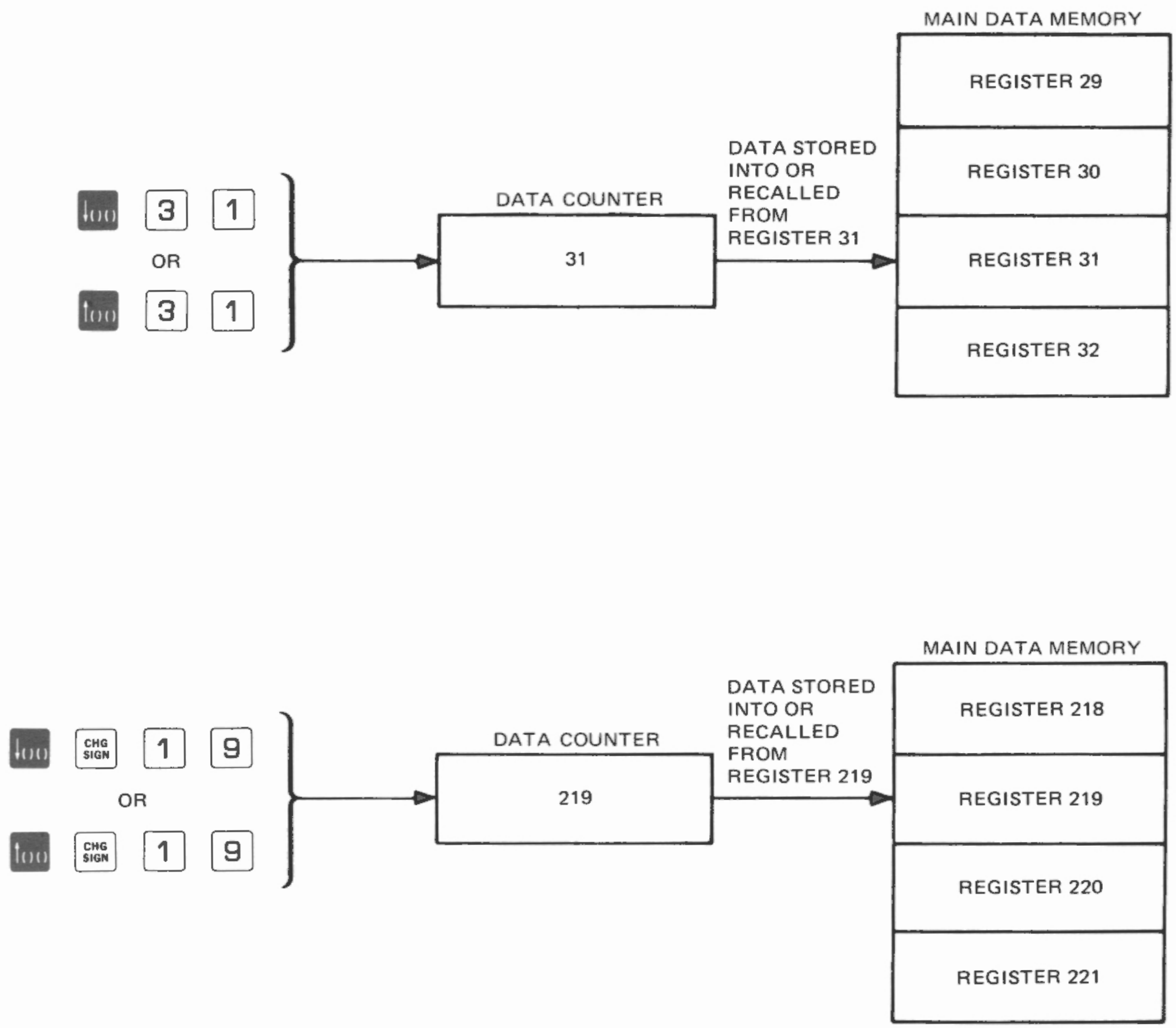


Figure 2-4. Addressing Main Data Memory

III. PROGRAMMABLE INSTRUCTIONS

As outlined in section II, the calculator has a repertoire of non-keyboard code commands. Although several of these instructions duplicate operations that are available from the keyboard, others are unique. A program normally includes both keyboard instructions and non-keyboard codes.

Non-keyboard codes are three-digit codes that are accessed from the keyboard after the RUN/STEP/LOAD switch has been set to LOAD and the [ENTER CODE 000] key has been depressed. The following paragraphs detail the operations and key sequences of the codes as they relate to register arithmetic, functions, and control. A list of the non-keyboard instructions and their corresponding machine codes is given in appendix B. Appendix C lists keyboard and non-keyboard codes in code numerical order.

REGISTER ARITHMETIC

The following non-keyboard codes provide for register arithmetic, supplementing the keyboard register arithmetic discussed in the Model 1880 Operating Instructions Manual. (It will be useful to study the register arithmetic techniques performed in the example of figure 4-14. These techniques will prove useful if there is a need for saving program steps.)

ADD TO MAIN DATA MEMORY

[ENTER CODE 000] [1] [2] [3] The Add to Main Data Memory instruction adds the number in the E-register to one of the main data storage registers. The storage register must be specified by the codes that follow this instruction. For example:

[ENTER CODE 000] [1] [2] [3]	Add to Main Data Memory
[1] [4]	Specifies data register 14

EXCHANGE MAIN DATA MEMORY

[ENTER CODE 000] [1] [2] [2] The Exchange Main Data Memory instruction exchanges data between the E-register and a main data storage register. The storage register must be specified by the digits that follow this instruction. For example:

[ENTER CODE 000] [1] [2] [2]	Exchange Main Data Memory
[2] [6]	Specifies data register 26

ADD TO SCRATCH PAD MEMORY

[ENTER CODE] [1] [1] [3] The Add to Scratch Pad Memory instruction adds the number in the E-register to one of the scratch pad registers or the pointer register. (The pointer register is a special register that holds the address of the desired main data register. It is accessed by using the [•] key. Specific use of the pointer register is described in <u>Indirect Data Addressing</u>, section IV.) The scratch pad register or the storage register is specified by the digit that follows this instruction.

For example:

[ENTER CODE] [1] [1] [3]	Add to Scratch Pad Memory
[6]	Scratch pad register 6
or	
[•]	Pointer register

EXCHANGE SCRATCH PAD MEMORY

[ENTER CODE] [1] [1] [2] The Exchange Scratch Pad Memory instruction exchanges data between the E-register and a scratch pad register or the pointer register. The scratch pad register or the storage register is specified by the digit that follows this instruction. For example:

[ENTER CODE] [1] [1] [2]	Exchange Scratch Pad Memory
[4]	Scratch pad register 4
or	
[•]	Pointer register

TOTAL SCRATCH PAD MEMORY

[ENTER CODE] [1] [1] [4] The Total Scratch Pad Memory instruction copies into the E-register the number in a scratch pad register or the pointer register and sets the scratch pad or pointer register to zero. The scratch pad register or the pointer register is specified by the digit that follows this instruction. For example:

[ENTER CODE] [1] [1] [4]	Recall into E-Register from Scratch Pad Memory
[7]	Scratch pad register 7
or	
[•]	Pointer register

FUNCTIONS

The following functions are provided by non-keyboard codes.

TANGENT

ENTER CODE [0] [7] [3] The Tangent instruction calculates the tangent of the angle (degrees or grads, depending on the position of the GRAD/DEG switch) in the E-register. Both the angle and the tangent are printed. The angle may be positive or negative, of any magnitude. Full accuracy is retained regardless of the magnitude of the angle.

Executing this function with angles whose tangents are outside the range 10^{+99} to 10^{-99} causes an error.

ARC TANGENT (arctan)

ENTER CODE [1] [0] [3] The Arc Tangent instruction calculates the arc tangent (radians) of the number in the E-register. Both the number and its arc tangent are printed. The arc tangent must be in the range $-\pi/2$ to $+\pi/2$.

SQUARE

ENTER CODE [0] [5] [3] The Square instruction calculates the square of the number in the E-register. Both the number and its square are printed.

ABSOLUTE VALUE

ENTER CODE [0] [4] [5] The Absolute Value instruction makes the sign of the number in the E-register positive.

ADD (ACCUMULATOR REGISTER)

ENTER CODE [0] [4] [1] The Add instruction adds the contents of the E-register to the contents of a special accumulator register. The number in the E-register is not changed, and that number is printed.

SUBTRACT (ACCUMULATOR REGISTER)

ENTER CODE [0] [4] [2] The Subtract instruction subtracts the contents of the E-register from the contents of a special accumulator register. The number in the E-register is not changed, and that number is printed.

SUBTOTAL (ACCUMULATOR REGISTER)

ENTER CODE [0] [4] [3] The Subtotal instruction copies the contents of the special accumulator register into the E-register and prints that number. The contents of the accumulator register are not altered.

TOTAL (ACCUMULATOR REGISTER)

ENTER CODE [0] [4] [0] The Total instruction copies the contents of the special accumulator register into the E-register and prints that number. Then the accumulator register is cleared.

INCREMENT ENTRY

ENTER CODE [1] [5] [1] The Increment Entry instruction increases the contents of the E-register by 1.

DECREMENT ENTRY

ENTER CODE [1] [5] [2] The Decrement Entry instruction decreases the contents of the E-register by 1.

CONTROL

The following non-keyboard codes control various operations of the calculator.

PRINT ENABLE

ENTER CODE [1] [5] [5] The Print Enable instruction enables normal keyboard instruction printing from the user program passing print control to the PRINT switch. The Print Enable instruction may be revoked only by the Print Disable instruction (below). When the calculator is turned on, Print Enable status is established.

PRINT DISABLE

ENTER CODE [1] [5] [4] The Print Disable instruction disables printing from the user program. It disables the PRINT switch so that keyboard instruction printing from the user program cannot occur with the PRINT switch on, except for specific, programmed Print or Identifier instructions. Printing in response to direct keyboard operation is not changed. The Print Disable instruction may be revoked only by the Print Enable instruction. During a Halt, print control returns to the PRINT switch.

RECALL D.P.

ENTER CODE [1] [5] [7] The Recall D.P. instruction recalls the previous decimal point setting, making it the current setting.

SET FLAG 1

ENTER CODE [0] [1] [6] The Set Flag 1 instruction sets program flag 1. A program may interrogate the flag and conditionally branch or jump, depending on its setting. Flag 1 is reset only by the Reset Flag 1 instruction (below), although the flag may be set from the keyboard, as well as from a program.

SET FLAG 2

ENTER CODE [0] [1] [7] The Set Flag 2 instruction sets program flag 2. A program may interrogate the flag and conditionally branch or jump, depending on its setting. Flag 2 is changed only by the Set Flag 2 or Reset Flag 2 instruction (below).

RESET FLAG 1

ENTER CODE [1] [6] [6] The Reset Flag 1 instruction resets program flag 1.

RESET FLAG 2

ENTER CODE [1] [6] [7] The Reset Flag 2 instruction resets program flag 2.

DOT PRINT

ENTER CODE [1] [7] [6] The Dot Print instruction prints a line of dots. The Dot Print instruction is not affected by the Print Disable instruction.

IDENTIFIER

ENTER CODE [1] [7] [7] The Identifier instruction prints a numeric label for the contents of the E-register. The numeric label, or "identifier," is printed in a left-justified format, with insignificant trailing zeros suppressed. Negative identifiers are printed in red, with a minus sign. When the Identifier instruction is preceded by an operation that inputs a number to the E-register, with no complex operations (such as Log or a^x) intervening between the number input and the Identifier instruction, the Identifier instruction will automatically restore the number that was in the E-register before the Identifier entry. The Identifier instruction is not affected by the Print Disable instruction.

Recommended usages of the Identifier instruction are outlined below.

Entered Identifier:

1. A calculated or entered number, C, is in the E-register.
2. Enter the Identifier; C is saved automatically.

3. Execute the Identifier instruction. The entered Identifier is printed, left-justified.

4. C is restored to the E-register.

For example, the following short program will halt twice, first for input of the number C, second for input of the Identifier. When [RESUME] is depressed, the identifier will be printed, left-justified, preceding C. To execute the program:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [0] [0].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress the following keys:

   [HALT]

   [HALT]

   [ENTER CODE ( )( )( )] [1] [7] [7]

   [PRINT x]

   [JUMP ( ) ( )]

   [0]

   [0]
5. Set the RUN/STEP/LOAD switch to RUN.
6. Depress [RESUME].
7. To enter C, depress

   [1] [2] [3] [4] [5] [•].
8. Depress [RESUME].
9. To enter the identifier, depress:

   [6] [7] [8] [•] [0] [0] [0] [0]

   (Notice that the trailing zeros are dropped.)
10. Depress [RESUME]

```
STEP      CODE      SYMBOL

0000      056
0001      056
0002      177
0003      060
0004      126        Ju
0005      000         0
0006      000         0
678•
 12,345•0000
```

NOTE: See Section V, page 5-1, Loading a Program, for details on printouts.

Calculated Identifier:

1. The result of a calculation, C, is in the E-register.

2. Execute 1, ↓( ), +, n, ↑( ), n. (This is a simple identifier incrementing sequence.) The n specifies the register where the Identifier number is stored, the "1" (or any other number you may enter) is added to that Identifier number, and the sum is recalled to the E-register. When the "1" is entered, C is saved automatically.

3. Execute the Identifier instruction. The calculated identifier number is printed, left-justified.

4. C is restored to the E-register.

For example, the following program will halt twice for input of values that are added to generate the calculated number, C, in the E-register. C is then printed, followed by the input increment, 1.000. Because scratch pad register 4, which contains the identifier number, is zero for the first run, the increment is the identifier, printed left-justified, without trailing zeros. During the second run, the first identifier is added to the increment, giving an identifier of 2. The Advance key (Code 065) is depressed three times to automatically separate the two runs of the program. To execute the program:

(NOTE: All examples in this manual assume a decimal point setting of 4.)

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [0] [0].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress the following keys:

[HALT]
[+]
[HALT]
[=]
[PRINT x]
[1]
[.]
[0]
[0]
[0]
[0]
[↓( )]
[+]
[4]
[↑( )]
[4]
[ENTER CODE ( )( )( )] [1] [7] [7]

0000	056	
0001	021	+
0002	056	
0003	020	=
0004	060	
0005	001	1
0006	012	
0007	000	0
0008	000	0
0009	000	0
0010	000	0
0011	110	↓
0012	021	+
0013	004	4
0014	111	↑
0015	004	4
0016	177	

[PRINT x]

[ADV]

[ADV]

[ADV]

[JUMP ( ) ( )]

[0]

[0]

5. Set the RUN/STEP/LOAD switch to RUN.

6. Depress [RESUME].

7. To enter the first value used to calculate C, depress

   [1] [2] [•] [3] [4] [5]

8. Depress [RESUME].

9. To enter the second value used to calculate C, depress

   [6] [7] [•] [8] [9] [0]

10. Depress [RESUME].

11. To enter the values used to calculate C in run 2, depress

    [9] [8] [•] [7] [6] [5]

    [RESUME]

    [4] [3] [•] [2] [1] [0]

    [RESUME]

Note the new identifier, 2.

```
   0017        060
   0018        065
   0019        065
   0020        065
   0021        126      Ju
   0022        000        0
   0023        000        0
      12•3450           +
      67•8900           =
      80•2350          *
      80•2350
       1•0000          ↓+◄
       1•0000          ↑ ◄
1•
      80•2350



      98•7650           +
      43•2100           =
     141•9750          *
     141•9750
       1•0000          ↓+◄
       2•0000          ↑ ◄
2•
     141•9750
```

IV. PROGRAMMING TECHNIQUES

This section explains how common programming techniques are used with the Monroe Model 1880 Scientific Calculator.

The coding sheet used to write a program for the Monroe Model 1880 Scientific Calculator is shown in figure 4-1. Step numbers and commands (that is, key symbols or abbreviations) are entered in their respective columns for each instruction. The "Symbol" column is used to list symbols used in symbolic addressing. Symbolic addressing is explained later in this section. The "Comments" column provides space for general explanatory remarks.

The following paragraphs discuss initializing the calculator; storing, recalling, and exchanging data; jumping and branching; decision-making processes; indirect and symbolic addressing; and indexing.

INITIALIZATION

To ensure the validity of data within your program, initialize, that is, set to zero or a constant value, the registers used in your program. Either the [CLEAR x] or the [RESET] key will clear (set to zero) the E-register (see Working Registers in section II for additional functions of these keys). To clear a scratch pad register, store a zero in it. Special function 0 ([Φ] [0]) will clear the E-register and scratch pad registers 0, 1, 2, and 3. Main data memory registers are cleared by storing zeros in them.

When the calculator is turned on, all registers are cleared, program memory is filled with NOOP (no operation) codes, a reset is executed, the decimal point is set to 2, and Print Enable is activated.

All programs that set flags or the SENSE switch should return them to their normal state at the end of the program. If you are loading your program into a calculator already turned on, or if you will be loading your program after the calculator has just completed operations from a previously stored program, remember that proper resettings may not have been made to the calculator. In such a case, it is advisable to execute Print Enable, Reset Flag 1, and Reset Flag 2 instructions (codes 155, 166, and 167, respectively). In addition to these precautions, check the keyboard for positioning of the PRINT and SENSE switches as required by your program.

Finally, if your program uses symbolic program addressing (discussed in detail later in this section), you should determine whether a previously stored program uses the same symbols that you used in your program. Procedures for testing for duplication of symbols are presented under Symbolic Program Addressing in this section.

TITLE ______________________ PROGRAMMER ______________________ Litton MONROE

STEP			SYMBOL	COMMAND	COMMENTS
	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				
	0				
	1				
	2				
	3				
	4				
	5				
	6				
	7				
	8				
	9				

NOTE: This simplified sheet shows only the decimal step column, headed STEP. The CRS column (not shown) is for advanced programming. The actual sheets contain 120 steps each.

Figure 4-1. Monroe Model 1880 Coding Sheet

STORING, RECALLING, AND EXCHANGING DATA

You can often save time or program steps by putting constants and variable data into storage registers at the beginning of your program. For example, if variables a, b, c, and d are to be used at various points in the program, it is easier to enter them at the beginning of the program and store them for recall when needed. A constant that is entered in the program memory requires a step for each digit, whereas an entire number can be stored in one data register. If the constant is used more than once, you can save program steps by storing the constant in a data register and recalling it when necessary.

The coding sequence in figure 4-2 illustrates several means of storing and recalling data. The keyboard sequence below outlines the steps required to load and execute that coding sequence, using 9 for a and 7 for b. The Branch instruction (operation 3) will cause the program to be loaded beginning at branch point 0. Depressing the [RESUME] key after setting the RUN/STEP/LOAD switch to RUN, (operation 6), causes the loaded program to begin execution, or to continue if execution had been temporarily suspended by a HALT instruction. The program begins (or continues) at the immediately loaded (or current) address. If the program is to be executed starting at a branch point other than the last loaded branch point, a [BRANCH 00] instruction containing the desired beginning address must precede [RESUME]. Any proper address may be used in such a branch, including symbolic addresses.

1. Set the PRINT switch to PRINT.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH 00] [0] [0].
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-2, steps 00 through 28.

0000	056		
0001	110	↓	
0002	001		1
0003	060		
0004	056		
0005	120	↓	
0006	000		0
0007	001		1
0008	060		
0009	005		5
0010	003		3
0011	012		
0012	010		8
0013	002		2
0014	004		4
0015	021	+	
0016	111	↑	
0017	001		1
0018	023	X	
0019	121	↑	
0020	000		0
0021	001		1
0022	020	=	
0023	110	↓	
0024	002		2
0025	061	A	
0026	126	Ju	
0027	000		0
0028	000		0

TITLE ______________________ PROGRAMMER ______________________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
0	0		HALT	START; ENTER a
	1		↓ ( )	STORE a IN SCRATCH PAD
	2		1	REGISTER 1
	3		PRINT X	PRINT a
	4		HALT	ENTER b
	5		↓ ( ) ( )	
	6		0	STORE b IN MAIN DATA REGISTER 1
	7		1	
	8		PRINT X	PRINT b
	9		5	
1	0		3	
	1		.	CONSTANT 53.824 IN E-REGISTER
	2		8	
	3		2	
	4		4	
	5		+	
	6		↑ ( )	ADD CONSTANT TO a, RECALLED
	7		1	FROM SCRATCH PAD REGISTER 1
	8		X	
	9		↑ ( ) ( )	MULTIPLY SUM BY b, RECALLED
2	0		0	FROM MAIN DATA REGISTER 1
	1		1	
	2		=	TERMINATE ARITHMETIC OPERATION
	3		↓ ( )	STORE RESULT IN SCRATCH PAD
	4		2	REGISTER 2
	5		PRINT ANS	PRINT RESULT
	6		JUMP ( ) ( )	
	7		0	RETURN TO START FOR ENTRY
	8		0	OF NEW a VALUE
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

NOTE: The use of the SYMBOL column for symbolic addressing is discussed on page 4-25.

Figure 4-2. Storing and Recalling Data

6. Set the RUN/STEP/LOAD switch to RUN.

7. Depress [RESUME].

8. To enter a, depress [9].

9. Depress [RESUME].

10. To enter b, depress [7].

11. Depress [RESUME].

```
      9.00    ↓ 1
      9.00
      7.00    ↓ 01
      7.00
     53.82     +
      9.00    ↑ 1
      9.00     X
      7.00    ↑ 01
      7.00     =
    439.76    *
    439.76    ↓ 2
    439.77    A
```

In addition to storing and recalling data, you may also exchange data between the E-register and a main data storage register. The exchange is programmed by depressing the [ENTER CODE] [1] [2] [2] keys, followed by the numeral keys of the main data register to be exchanged. In the example above, a constant was entered, manipulated arithmetically, and an answer printed. Figure 4-3 modifies this, using the calculated result as a new constant.

The answer in the E-register is exchanged with the old constant (53.824) in main data register 2.* On subsequent reiterations of the program, no new constant is entered at the first HALT. Instead, the program jumps ahead for input of new values for a and b, permitting execution of the program using the previously calculated answer as the constant for the next run. This procedure is shown below.

1. Set the PRINT switch to PRINT.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH] [0] [0].
4. Set the RUN/STEP/LOAD switch to LOAD.

*Note, that, to effect this exchange, the constant is now stored in a register rather than as program steps.

TITLE ______ PROGRAMMER ______ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
0	0		HALT	ENTER INITIAL CONSTANT
	1		↓( ) ( )	
	2		0	STORE CONSTANT IN MAIN
	3		2	DATA REGISTER 2
	4		JUMP ( ) ( )	
	5		0	JUMP TO STEP 10
	6		1	
	7	ENTER CODE	377	(NO OPERATION)
	8	ENTER CODE	377	(NO OPERATION)
	9	ENTER CODE	377	(NO OPERATION)
1	0		HALT	ENTER a
	1		↓( )	STORE a IN SCRATCH
	2		1	PAD REGISTER 1
	3		PRINT X	PRINT a
	4		HALT	ENTER b
	5		↓( ) ( )	
	6		0	STORE b IN MAIN
	7		1	DATA REGISTER 1
	8		PRINT X	PRINT b
	9		↑( ) ( )	
2	0		0	RECALL CONSTANT FROM MAIN
	1		2	DATA REGISTER 2
	2		+	
	3		↑( )	ADD CONSTANT TO a, RECALLED
	4		1	FROM SCRATCH PAD REGISTER 1
	5		×	
	6		↑( ) ( )	MULTIPLY SUM BY b, RECALLED FROM
	7		0	MAIN DATA REGISTER 1
	8		1	
	9		=	TERMINATE ARITHMETIC OPERATION
3	0		↓( )	STORE RESULT IN SCRATCH
	1		2	PAD REGISTER 2
	2		PRINT ANS	PRINT RESULT
	3	ENTER CODE	122	
	4		0	EXCHANGE ANSWER IN E-REGISTER AND
	5		2	CONSTANT IN MAIN DATA REGISTER 2
	6		PRINT X	PRINT E-REGISTER CONTENTS TO VERIFY EXCHANGE
	7		JUMP ( ) ( )	
	8		0	RETURN TO HALT: DEPRESS RESUME
	9		1	FOR SUBSEQUENT CALCULATIONS

Figure 4-3. Exchanging E-Register and Main Data Memory

5. Depress the keys shown on the coding sheet, figure 4-3, steps 00 through 39. No operation instructions (non-keyboard code 377) are used to advance to step 10. (The program counter counts up by one each time a key is depressed, except for codes following the [ENTER CODE] key. Note that input of non-keyboard codes advances the program counter only after the fourth key depression.)
6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress [RESUME].
8. To enter the initial constant, depress

   [5] [3] [•] [8] [2] [4].
9. Depress [RESUME].
10. To enter a, depress [9].
11. Depress [RESUME].
12. To enter b, depress [7].
13. Depress [RESUME].

BRANCHING

Any set of instructions arranged in the proper sequence to cause the calculator to perform a desired operation may be called a "routine." A "subroutine" is a routine that is a part, or subsection, of another routine. Subroutines are often used to perform a calculation that will be repeated many times during the execution of the program. To save memory space, the calculation is programmed only once, as a subroutine, and the program is directed to divert, or "branch," to the subroutine each time the calculation is required.

Program branches are made with the Branch or Jump instruction. A Branch instruction automatically saves the address of the instruction following it (that is, the return address) in a special memory unit called "program storage" (P-store). A Resume

0000	056		
0001	120	↓	
0002	000		0
0003	002		2
0004	126	Ju	
0005	000		0
0006	001		1
0007	377		
0008	377		
0009	377		
0010	056		
0011	110	↓	
0012	001		1
0013	060		
0014	056		
0015	120	↓	
0016	000		0
0017	001		1
0018	060		
0019	121	↑	
0020	000		0
0021	002		2
0022	021	+	
0023	111	↑	
0024	001		1
0025	023	X	
0026	121	↑	
0027	000		0
0028	001		1
0029	020	=	
0030	110	↓	
0031	002		2
0032	061	A	
0033	122	↕	
0034	000		0
0035	002		2
0036	060		
0037	126	Ju	
0038	000		0
0039	001		1
53.8240		↓	02
9.0000		↓	1
9.0000			
7.0000		↓	01
7.0000			

instruction at the end of the subroutine causes the return address in P-store to be copied into the program counter after the subroutine is executed. Thus, the program will continue automatically from the point where it was diverted to the subroutine. This process is shown schematically in figure 4-4.

The coding form in figure 4-5 contains a short program with a subroutine. Notice that the store instructions given are for scratch pad registers, not main data memory. After entry and storage of a, b, c, and d, the program branches to a subroutine that prints the number, squares it, adds the constant 32 to the squared number, and prints the sum. The Resume instruction at the end of the subroutine causes a return to the main program, which then performs various arithmetic operations using the data entered and the values formed by the subroutine. The subroutine performs a valid function that could be used as part of the mathematical calculations in the solution of a working equation. To observe the operation of the subroutine in the calculator, load and execute the program as follows:

1. Set the PRINT switch to the off position.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress BRANCH 0 0.
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-5, steps 00 through 58.

53.8240	↑	02
53.8240	+	
9.0000	↑	1
9.0000	X	
7.0000	↑	01
7.0000	=	
439.7680	*	
439.7680	↓	2
439.7680	A	
53.8240	↕	02
53.8240		

0000	056		
0001	110	↓	
0002	002		2
0003	127	Br	
0004	000		0
0005	007		7
0006	110	↓	
0007	003		3
0008	056		
0009	110	↓	
0010	004		4
0011	127	Br	
0012	000		0
0013	007		7
0014	110	↓	
0015	005		5
0016	056		
0017	110	↓	
0018	006		6
0019	127	Br	
0020	000		0
0021	007		7
0022	110	↓	
0023	007		7
0024	056		
0025	110	↓	
0026	010		8
0027	127	Br	
0028	000		0
0029	007		7
0030	110	↓	

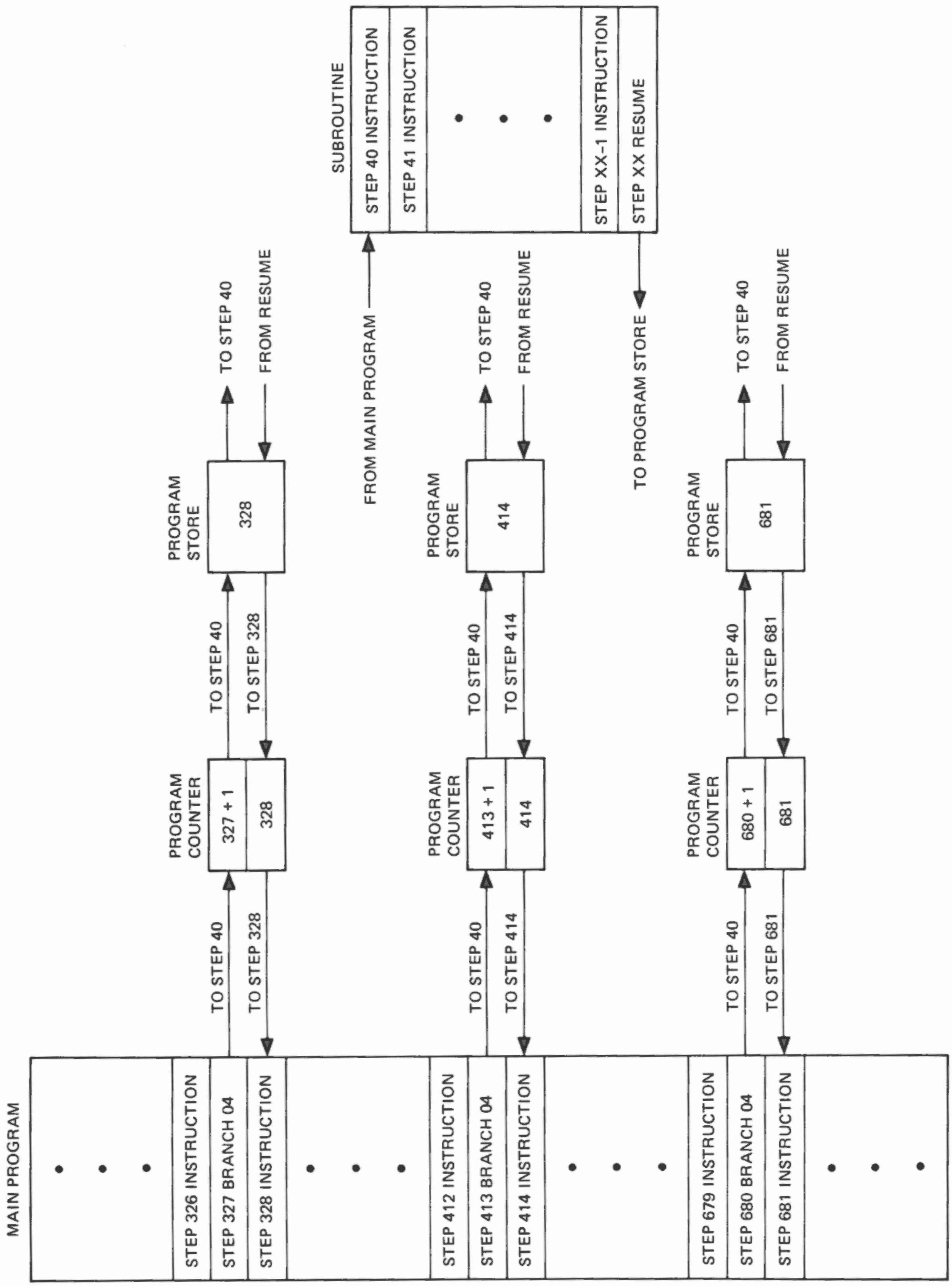


Figure 4-4. Program Branches and Returns

TITLE ______________ PROGRAMMER ______________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
0	0		HALT	ENTER a
	1		↓ ( )	STORE a IN REGISTER 2
	2		2	
	3		BRANCH ( ) ( )	
	4		0	BRANCH TO SUBROUTINE AT STEP
	5		7	70 (BRANCH POINT 7)
	6		↓ ( )	STORE $a^2 + 32$ IN REGISTER 3
	7		3	
	8		HALT	ENTER b
	9		↓ ( )	STORE b IN REGISTER 4
1	0		4	
	1		BRANCH ( ) ( )	
	2		0	BRANCH TO SUBROUTINE
	3		7	
	4		↓ ( )	STORE $b^2 + 32$ IN REGISTER 5
	5		5	
	6		HALT	ENTER c
	7		↓ ( )	STORE c IN REGISTER 6
	8		6	
	9		BRANCH ( ) ( )	
2	0		0	BRANCH TO SUBROUTINE
	1		7	
	2		↓ ( )	STORE $c^2 + 32$ IN REGISTER 7
	3		7	
	4		HALT	ENTER d
	5		↓ ( )	STORE d IN REGISTER 8
	6		8	
	7		BRANCH ( ) ( )	
	8		0	BRANCH TO SUBROUTINE
	9		7	
3	0		↓ ( )	STORE $d^2 + 32$ IN REGISTER 9
	1		9	
	2		+	
	3		↑ ( )	ADD (c^2+32) TO (d^2+32)
	4		7	
	5		÷	
	6		↑ ( )	DIVIDE BY a
	7		2	
	8		X	MULTIPLY BY b
	9		↑ ( )	

Figure 4-5. Subroutine Example (1 of 2)

STEP		SYMBOL	COMMAND	COMMENTS
4	0		4	
	1		=	TERMINATE ARITHMETIC OPERATIONS
	2		PRINT ANS	PRINT RESULT
	3		↑( )	$a^2 + 32$ IN E-REGISTER
	4		3	
	5		+	
	6		↑( )	ADD $b^2 + 32$
	7		5	
	8		÷	
	9		↑( )	DIVIDE BY c
5	0		6	
	1		×	
	2		↑( )	MULTIPLY BY d
	3		8	
	4		=	TERMINATE ARITHMETIC OPERATIONS
	5		PRINT ANS	PRINT RESULT
	6		JUMP ( ) ( )	
	7		0	RETURN TO BEGINNING OF PROGRAM
	8		0	
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
7	0		PRINT X	PRINT CONTENTS OF E-REGISTER
	1		×	SQUARE THE NUMBER
	2		+	
	3		3	ADD 32 TO SQUARED NUMBER
	4		2	
	5		=	TERMINATE ARITHMETIC OPERATIONS
	6		PRINT ANS	PRINT RESULT
	7		RESUME	RETURN TO MAIN PROGRAM
	8			
	9			

Figure 4-5. Subroutine Example (2 of 2)

6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress [BRANCH] [0] [7].
8. Set the RUN/STEP/LOAD switch to LOAD.
9. Depress the keys shown on the coding sheet, figure 4-5, steps 70 through 77.
10. Set the RUN/STEP/LOAD switch to RUN.
11. Depress [BRANCH] [0] [0].
12. Depress [RESUME].
13. To enter a depress [6].
14. Depress [RESUME].
15. To enter b, depress [7].
16. Depress [RESUME].
17. To enter c, depress [8].
18. Depress [RESUME].
19. To enter d, depress [9].
20. Depress [RESUME].

Notice that the subroutine has printed a, $a^2 + 32$, b, $b^2 + 32$, c, $c^2 + 32$, d, $d^2 + 32$, and the results of the two calculations.

0031	011	9
0032	021	+
0033	111	↑
0034	007	7
0035	024	÷
0036	111	↑
0037	002	2
0038	023	X
0039	111	↑
0040	004	4
0041	020	=
0042	061	A
0043	111	↑
0044	003	3
0045	021	+
0046	111	↑
0047	005	5
0048	024	÷
0049	111	↑
0050	006	6
0051	023	X
0052	111	↑
0053	010	8
0054	020	=
0055	061	A
0056	126	Ju
0057	000	0
0058	000	0
0070	060	
0071	023	X
0072	021	+
0073	003	3
0074	002	2
0075	020	=
0076	061	A
0077	057	
	6.0000	
	68.0000	A
	7.0000	
	81.0000	A
	8.0000	
	96.0000	A
	9.0000	
	113.0000	A
	243.8333	A
	167.6250	A

Subroutines may operate within other subroutines. The calculator accepts this type of subroutine "nesting" to six levels; that is, the calculator will accept six Branch instructions before it will require a Resume instruction to return to the next higher-level subroutine.

JUMPING

A Jump instruction can be used to set the program counter to any branch point in program memory. One application of the Jump instruction is to loop to the starting point of a program after each execution. Jump may also be used to distribute a program in convenient locations in memory when a sequential series of steps is not available. For example, assume that a program is stored, beginning at branch point 2, and you want to load the storing-and-recalling-data example (figure 4-2) program, beginning at branch point 0. You can use the Jump instruction to bypass the previously stored program. In the example in figure 4-6, the program jumps to branch point 5. Notice that the program beginning at branch point 0 operates as if it were stored in consecutive locations. The program returns to branch point 0 to allow entry of additional data. Note that the Jump instruction does not store the address required to return to the normal sequence; the return must be specifically indicated with a second Jump instruction.

In the following example, two programs are loaded, and the program beginning at branch point 0 is executed.

1. Set the PRINT switch to PRINT.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress BRANCH [0] [0].
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-6, steps 00 through 17.

0000	056	
0001	110	↓
0002	001	1
0003	060	
0004	056	
0005	120	↓
0006	000	0
0007	001	1
0008	060	
0009	005	5
0010	003	3
0011	012	
0012	010	8
0013	002	2
0014	004	4
0015	126	Ju
0016	000	0
0017	005	5

TITLE ______ PROGRAMMER ______ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
0	0		HALT	BRANCH POINT 0, START; ENTER a
	1		↓( )	STORE a IN SCRATCH PAD
	2		1	REGISTER 1
	3		PRINT X	PRINT a
	4		HALT	ENTER b
	5		↓( )( )	
	6		0	STORE b IN MAIN
	7		1	DATA REGISTER 1
	8		PRINT X	PRINT b
	9		5	
1	0		3	
	1		.	CONSTANT 53.824 IN E-REGISTER
	2		8	
	3		2	
	4		4	
	5		JUMP ( )( )	
	6		0	JUMP TO BRANCH POINT 5
	7		5	TO CONTINUE PROGRAM
	8			
	9			
2	0		(	
	1		5	
	2		X	
	3		HALT	
	4		PRINT X	
	5		)	
	6		+	
	7		(	
	8		2	
	9		X	PROGRAM THAT IS
3	0		HALT	TO BE BYPASSED
	1		PRINT X	
	2		)	
	3		X	
	4		HALT	
	5		PRINT X	
	6		÷	
	7		HALT	
	8		PRINT X	
	9		=	

Figure 4-6. Jump Instruction Example (1 of 2)

TITLE ______________________ PROGRAMMER ______________________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
4	0		PRINT ANS	
	1		BRANCH ( ) ( )	
	2		0	
	3		8	
	4			
	5			
	6			
	7			
	8			
	9			
5	0		↓ ( ) ( )	
	1		0	STORE CONSTANT IN MAIN
	2		2	DATA REGISTER 2
	3		+	
	4		↑ ( )	ADD CONSTANT TO a, RECALLED
	5		1	FROM SCRATCH PAD REGISTER 1
	6		X	
	7		↑ ( ) ( )	MULTIPLY SUM BY b, RECALLED FROM
	8		0	MAIN DATA REGISTER 1
	9		1	
6	0		=	TERMINATE ARITHMETIC OPERATION
	1		↓ ( )	STORE RESULT IN SCRATCH
	2		2	PAD REGISTER 2
	3		PRINT ANS	PRINT RESULT
	4		JUMP ( ) ( )	
	5		0	JUMP TO BEGINNING OF PROGRAM,
	6		0	BRANCH POINT 0
	7			
	8			
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

Figure 4-6. Jump Instruction Example (2 of 2)

6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress [BRANCH ( ) ( )] [0] [5].
8. Set the RUN/STEP/LOAD switch to LOAD.
9. Depress the keys shown on the coding sheet, figure 4-6, steps 50 through 66.
10. Set the RUN/STEP/LOAD switch to RUN.
11. Depress [BRANCH ( ) ( )] [0] [0].
12. Depress [RESUME].
13. To enter a, depress [9].
14. Depress [RESUME].
15. To enter b, depress [7].
16. Depress [RESUME].

```
0050        120      ↓
0051        000         0
0052        002         2
0053        021       +
0054        111      ↑
0055        001         1
0056        023       X
0057        121      ↑
0058        000         0
0059        001         1
0060        020       =
0061        110      ↓
0062        002         2
0063        061      A
0064        126       Ju
0065        000         0
0066        000         0
       9•0000        ↓  1
       9•0000
       7•0000        ↓  01
       7•0000
      53•8240        ↓  02
      53•8240         +
       9•0000        ↑  1
       9•0000         X
       7•0000        ↑  01
       7•0000         =
     439•7680        *
     439•7680        ↓  2
     439•7680        A
```

INDIRECT DATA ADDRESSING

Paragraphs under Memory Addressing in section II of this manual describe direct data addressing. The method is "direct" because it transfers data from the E-register to explicitly specified main data memory registers. Another method of data addressing is known as "indirect" data addressing. Indirect addressing is used to select main data registers without actually specifying register numbers in the program. This feature is used for storing and recalling data and for register arithmetic. The indirect addressing technique is convenient when you don't know which main data registers will be available when your program is executed. It also permits arraying of data using n-count incrementing or decrementing of main data register numbers.

Indirect addressing uses a register "pointer," instead of an instruction, to direct the data flow. The number in the pointer register specifies the desired main data register. A number is stored in the pointer register with the [↓( )] and [•] keys as if the pointer were scratch pad register 10 and the [•] key represented the 10. For example, to store the number 20 in the pointer register, depress:

[2] [0] [↓( )] [•]

With the PRINT switch on, the printout is:

```
20•0000        ↓
```

and the pointer register assumes the following state:

POINTER REGISTER

20

Main data register 20, because it is specified by the pointer register, is the data register that the program will access when the indirect addressing command is executed.

After a register number has been stored in the pointer register, you can use the [IND | SYMB] key instead of the numeral keys for the main data register. For example, to store the number 45 in register 20 with direct addressing, you would use the keyboard sequence:

[4] [5] [↓( )] [2] [0]

In indirect addressing, with 20 already in the pointer register, the following entries would perform the same operation:

[4] [5] [↓( )] [IND | SYMB]

Note that, when using indirect addressing, register numbers above 99 are addressed without use of [•], [CHG SIGN], [EXP], or [π e] by simply storing the full register number in the pointer register. For example, if 45 is to be stored in register 312 indirectly, 312 would be placed in the pointer register and [4] [5] [↓( )] [IND | SYMB] depressed. The indirect addressing process is shown schematically in figure 4-7.

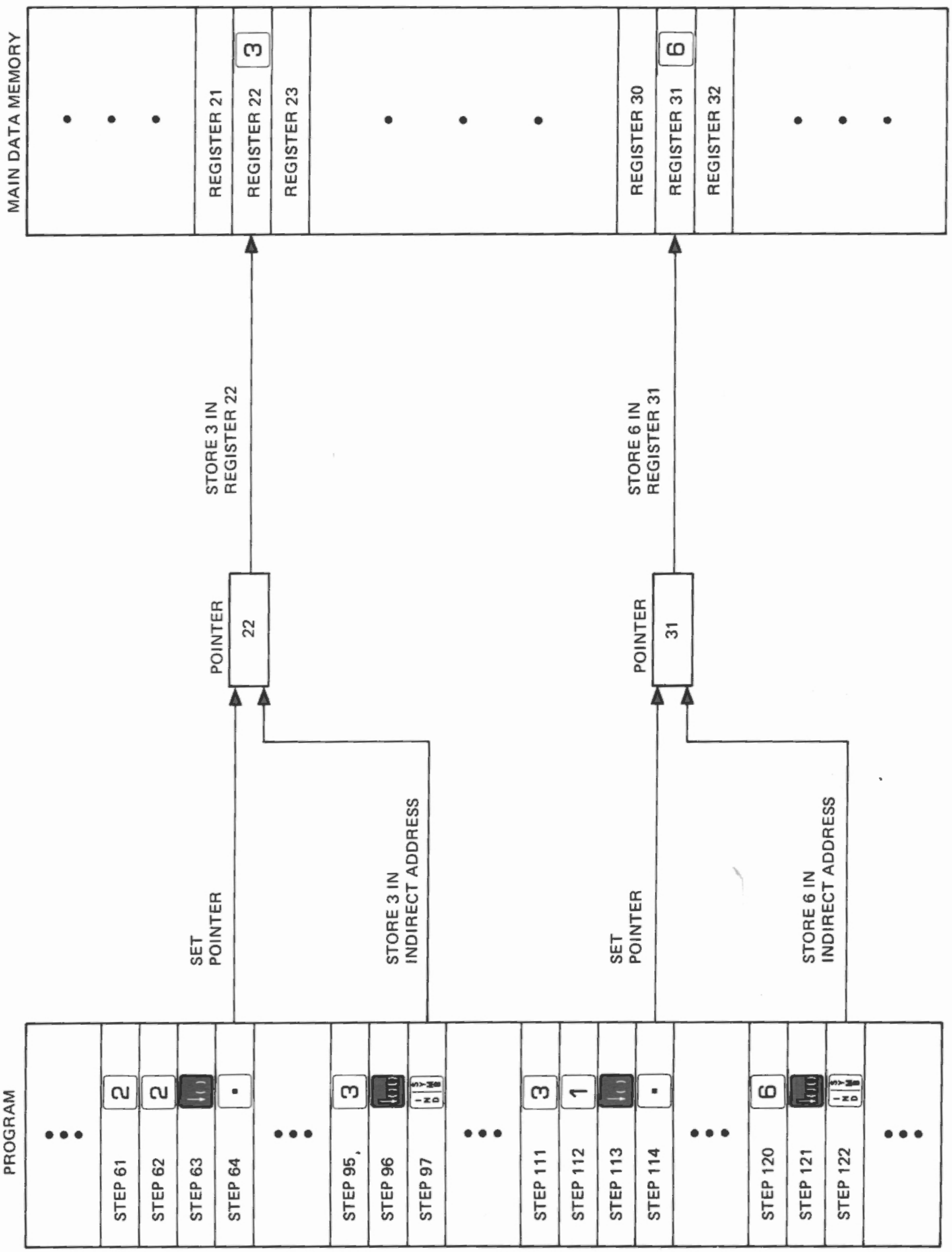


Figure 4-7. Indirect Addressing

Indirect addressing may also be used to perform register arithmetic. After a register number has been stored in the pointer register, the [IND SYM] key replaces the register number in the program. For example, to add 25 to the contents of main data register 60 with the keyboard, you would depress:

[2] [5] [↓( )] [+] [6] [0]

With indirect addressing, you would store 60 in the pointer register, and execute the following keyboard sequence:

[2] [5] [↓( )] [+] [IND SYM]

In like manner, the [IND SYM] key substitutes for the data register number for register subtraction, multiplication, and division.

Indirect addressing enables you to select any main data register that is available when you execute the program. Include in your program a Halt instruction, followed by [↓( )] [•]. When the program is executed and the halt occurs, manually enter the number of the selected register; when you depress [RESUME], the register number is stored in the pointer.

Follow the procedure detailed below to store the indirect-addressing sample program in figure 4-8, beginning at branch point 4, and to execute it, using main data register 15 and data values 2 and 3.

1. Set the PRINT switch to PRINT.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH ( )( )] [0] [4].
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-8.

0040	056		
0041	110	↓	
0042	012		
0043	120	↓	
0044	000		0
0045	010		8
0046	126	Ju	
0047	000		0
0048	005		5
0049	377		
0050	056		
0051	120	↓	
0052	067		
0053	056		
0054	120	↓	
0055	021	+	
0056	067		
0057	121	↑	
0058	067		
0059	061	A	
0060	001		1
0061	121	↑	
0062	021	+	
0063	000		0
0064	010		8
0065	120	↓	
0066	012		
0067	120	↓	
0068	000		0
0069	010		8

TITLE ______________________ PROGRAMMER ______________________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
4	0		HALT	ENTER INITIAL REGISTER NUMBER (ADDRESS)
	1		↓( )	STORE NUMBER (ADDRESS)
	2		•	IN REGISTER POINTER
	3		↓( )( )	
	4		0	STORE COUNT NUMBER IN MAIN
	5		8	DATA REGISTER 8
	6		JUMP( )( )	
	7		0	JUMP TO SUBROUTINE AT STEP 50
	8		5	
	9	ENTER CODE	377	(NO OPERATION)
5	0		HALT	ENTER DATA a
	1		↓( )( )	STORE ACCORDING TO REGISTER
	2		IND/SYMB	POINTER
	3		HALT	ENTER DATA b
	4		↓( )( )	
	5		+	ADD TO DATA a IN MEMORY, STORED
	6		IND/SYMB	ACCORDING TO REGISTER POINTER
	7		↑( )( )	RECALL SUM ACCORDING TO
	8		IND/SYMB	REGISTER POINTER
	9		PRINT ANS	PRINT SUM
6	0		1	
	1		↑( )( )	
	2		+	INCREMENT COUNT NUMBER IN MAIN
	3		0	DATA REGISTER 8 BY 1
	4		8	
	5		↓( )( )	STORE NEW ADDRESS IN
	6		•	REGISTER POINTER
	7		↓( )( )	
	8		0	STORE NEW COUNT NUMBER IN
	9		8	MAIN DATA REGISTER 8
	0		• • •	(OTHER INSTRUCTIONS TO "TEST" THE
	1			SUM FOR UPPER LIMIT. IF LESS
	2			THAN LIMITING VALUE, BRANCH TO
	3			05 TO ENTER MORE DATA;
	4			OTHERWISE, BRANCH TO OTHER
	5			PART OF MAIN PROGRAM.)
	6			
	7			
	8			
	9			

Figure 4-8. Indirect Addressing Example

6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress BRANCH ( ) ( ) 0 4.
8. Depress RESUME.
9. To enter the desired register number, depress 1 5.
10. Depress RESUME.
11. To enter the first item, depress 2.
12. Depress RESUME.
13. To enter the second item, depress 3.
14. Depress RESUME.

```
15.0000    ↓
15.0000    ↓   08
 2.0000    ↓ I
 3.0000    ↓ I /
 5.0000    ↑ I
 5.0000    A
16.0000    ↑ + 08
 0.1000    ↓   08
```

Notice that the number recalled at the end of the program is 5; you entered 2 into register 15, then added 3 to the value (2) in register 15. Additionally, the pointer register was incremented by 1; the next main data register to be used would be 16.

SYMBOLIC PROGRAM ADDRESSING

Symbolic program addressing makes it possible to locate a program anywhere in program memory at the time of loading. It also permits branching or jumping to any step in a program, without being constrained to branch points.

The IND SYM instruction defines a keyboard or non-keyboard entry as a symbol, rather than an operating instruction. The IND SYM instruction must always be entered in the step preceding the symbol. The combination IND SYM α defines a symbolic address at that point in the program. (α represents either a keyboard symbol such as √, or a non-keyboard symbol such as IND SYM ENTER CODE ( )( )( ) 165.) When the same symbol is again defined in a Branch or Jump instruction, the program is told to branch or jump to that symbolic address.

The Branch or Jump instruction has the following forms:

BRANCH ( ) ( ) IND SYM α

or

JUMP ( ) ( ) IND SYM α

In non-symbolic branching or jumping, the numerals following the Branch or Jump instruction tell the calculator where to get the next instruction. Consequently, this type of program must be stored in the section of memory specified by the numerals. However, at the time the program is loaded, that section may already contain information that should be retained in the calculator. Figure 4-9 shows how symbolic addressing permits flexible program location. The instructions in sequence A must be stored as shown, because steps 81 and 82 specify branch point 7 as the step to which the program jumps. Sequences B and C, which contain the same instructions as A, can be loaded into any desired block of memory steps because, instead of an explicit address, a symbol is used to identify the point in the program to which the jump is made. Irrespective of the location in memory where the program is stored, a branch or jump to the symbol sets the program counter to the step containing the [IND | SYM] instruction for defining that symbol. A symbol may be placed in any memory location, regardless of branch points.

You may use as many branches or jumps and as many symbols as you wish in your program, to a maximum of 95 symbols. Since the last defined symbol is the one accessed by the calculator, do not use the same symbol in any one program to represent more than one location, or the same symbol in different, existing programs, both of which will be used in the same calculation.

To avoid duplicating any of these symbols, you also need to know what symbols are stored in other programs in memory. If a program operation sheet is available for the program in memory, you may find all symbols used listed on the sheet. If not, it may be necessary to test for duplicate symbols. To test for duplicate symbols, depress [JUMP ( ) ( )] [IND | SYM], and the key you intend to use for your symbol. Then put the RUN/STEP/LOAD switch in the LOAD position and depress [LIST PROG] twice. If the symbol appears on the printout, a previously stored program is using the symbol. Use a different symbol in your program.

If your symbol is a non-keyboard symbol generated by [IND | SYM] [ENTER CODE ( )( )( )] [n] [n] [n], an instruction sequence for testing for symbol duplication must be loaded into the calculator. The search sequence, which may be loaded at any available branch point, is:

[JUMP ( ) ( )] [IND | SYM] [ENTER CODE ( )( )( )] [n] [n] [n]

where n represents the numerals of the non-keyboard symbol.

As an exercise in using search sequences, load non-keyboard symbols 110 at steps 000, 001, and 002, and 111 at steps 010, 011, and 012 as follows:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [JUMP ( ) ( )] [0] [0].

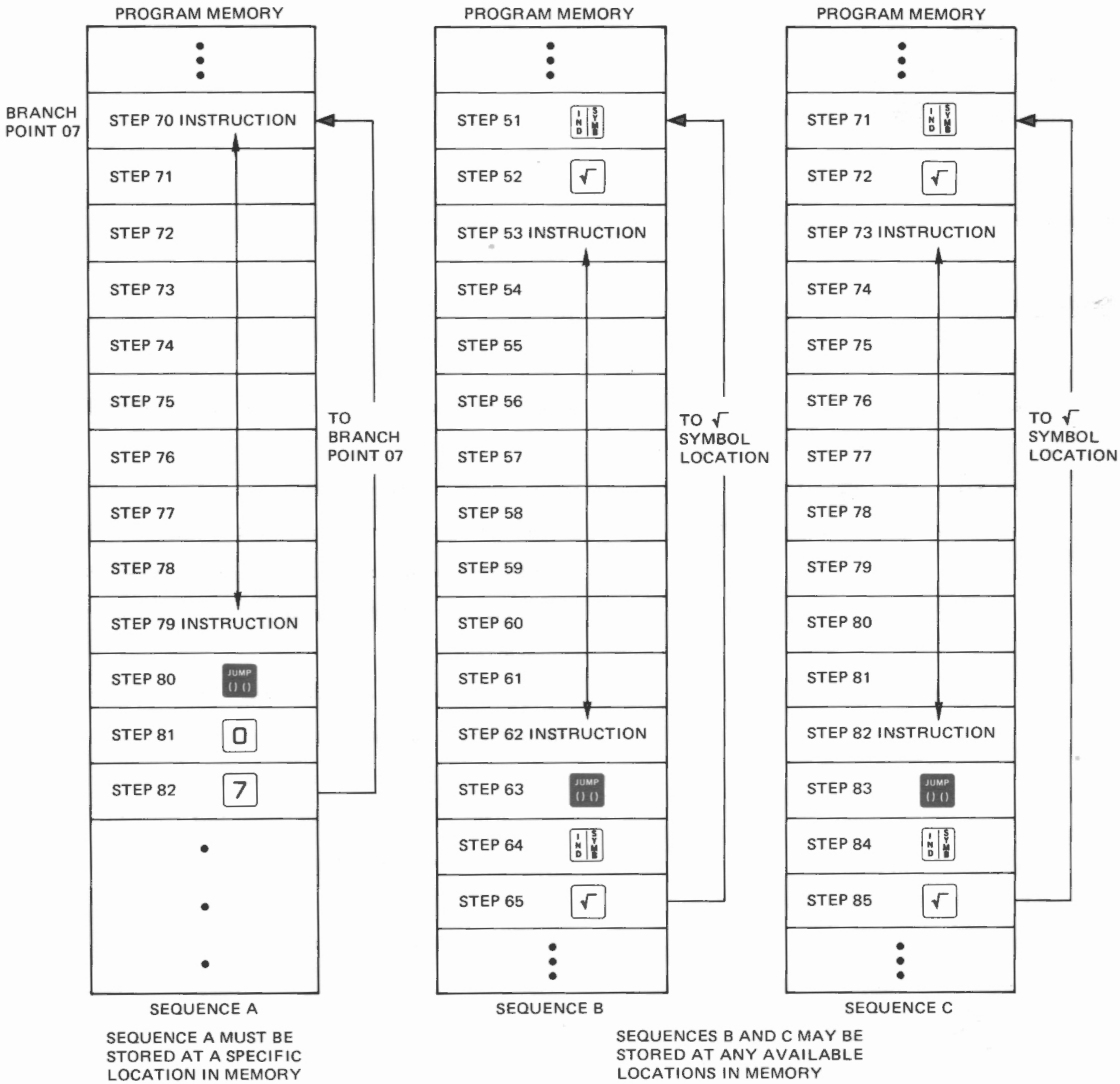


Figure 4-9. Symbolic Addressing

3. Set the RUN/STEP/LOAD switch to LOAD.

4. Depress [IND SYMB] [ENTER CODE] [1] [1] [0]

5. Set the RUN/STEP/LOAD switch to RUN.

6. Depress [JUMP] [0] [1]

7. Set the RUN/STEP/LOAD switch to STEP.

8. Depress [IND SYMB] [ENTER CODE] [1] [1] [1]

To use the search sequence to test for these symbols, the following procedure should be followed:

Keyboard Input	Explanation
1. Set the RUN/STEP/LOAD switch to RUN.	
2. Depress [JUMP] [2] [1].	(Assume branch point 21 is available for search sequence.)
3. Set the RUN/STEP/LOAD switch to LOAD.	
4. Depress [JUMP] [IND SYMB] [ENTER CODE] [1] [1] [0].	Search sequence for symbol 110.
5. Set the RUN/STEP/LOAD switch to STEP.	This procedure is used to step to the duplicate symbol. (See <u>Testing a Program</u>, section V, for details on step mode.) Note that, in the step mode, the program instructions will be printed in red.
6. Depress [RESUME], repeatedly. • • •	No more than six [RESUME] depressions are necessary to cause printing of the search sequence and the duplicate symbol. The search sequence (including the symbol entered in the search sequence) is printed first, followed by the sought-after duplicate symbol, if it was in memory.
7. Set the RUN/STEP/LOAD switch to RUN.	
8. Depress [JUMP] [2] [1]	(Can use same branch point again.)
9. Set the RUN/STEP/LOAD switch to LOAD.	
10. Depress [JUMP] [IND SYMB] [ENTER CODE] [1] [1] [1]	Search sequence for symbol 111.
11. Set the RUN/STEP/LOAD switch to STEP.	
12. Depress [RESUME], repeatedly. • • •	Use the [RESUME] key as necessary.

There are 95 symbols available for use in symbolic program addressing. A valid symbol may be any three-digit combination of the number 0 through 7 up to number 137 (with the exception of number 066). The calculator does not accept an 8 or a 9 in a memory code, but keyboard digits 8 or 9 are permissible, since their memory codes are 010 and 011, respectively. For example, 081 and 119 are invalid codes. Most three-digit combinations represent key codes, and so may be entered directly from the keyboard by depressing the appropriate key. As previously mentioned, non-keyboard symbols are entered with the [ENTER CODE] key as follows:

[IND SYMB] [ENTER CODE] [n] [n] [n]

Special care is needed when defining codes 120 through 137 as symbols. Numeric symbols that fall within the range [IND SYMB] [1] [2] [0] through [IND SYMB] [1] [3] [7] must not be immediately followed by another [IND SYMB] [α], where [α] represents some other symbol that is being defined. It is recommended that at least two instructions or data digits separate defined symbols in the 120-137 range from the next sequential [IND SYMB] instruction. For example, two [CHG SIGN] instructions may be entered, one after the other, to provide the required two-step separation:

[JUMP] [IND SYMB] [ENTER CODE] [1] [2] [3] [CHG SIGN]

[CHG SIGN] [IND SYMB] [ENTER CODE] [α]

Figure 4-10 shows a program that contains two symbolic addresses. The symbols are the Plus instruction and the numeric symbol 102. Notice that the symbolic addresses are written in the "Symbol" column on the coding sheet. This convention makes symbolic addresses easy to spot on the coding sheet. Since the numeric symbol 102 is a non-keyboard code, use the

0000	066	
0001	110	↓
0010	066	
0011	111	↑
0210	126	Ju
0211	067	
0212	110	↓
0210	126	Ju
0211	067	
0212	110	↓
0000	066	
0001	110	↓
0000	126	Ju
0001	067	
0002	111	↑
0000	126	Ju
0001	067	
0002	111	↑
0010	066	
0011	111	↑

TITLE ______________________ PROGRAMMER ______________________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
3	0	+	IND/SYMB	SYMBOLIC ADDRESS +
	1		+	
	2		(	
	3		5	
	4		X	
	5		HALT	
	6		PRINT X	
	7		JUMP ( ) ( )	
	8		IND/SYMB	JUMP TO SYMBOLIC ADDRESS 102
	9	ENTER CODE	102	
4	0			
	1			
	2			(RESERVED FOR ANOTHER PROGRAM)
	3			
	4			
	5	102	IND/SYMB	SYMBOLIC ADDRESS 102
	6	ENTER CODE	102	
	7		)	
	8		+	
	9		(	
5	0		2	
	1		X	
	2		HALT	
	3		PRINT X	
	4		)	
	5		X	
	6		HALT	
	7		PRINT X	
	8		÷	
	9		HALT	
6	0		PRINT X	
	1		=	
	2		PRINT ANS	
	3		BRANCH ( ) ( )	
	4		IND/SYMB	BRANCH TO SYMBOLIC ADDRESS +
	5		+	
	6			
	7			
	8			
	9			

Figure 4-10. Symbolic Addressing Example

[ENTER CODE] key and the numerals 102 to enter it. After the program has been loaded, program execution is started with the normal Branch or Jump instruction, followed by the [IND SYMB] and [+] keys. After the program has been executed, the Branch, Indirect/Symbolic, and Plus instructions cause the calculator to transfer automatically back to the step containing the [IND SYMB] instruction that defines [+] as a symbol. The symbolic Jump instruction operates automatically within the program. The Jump, Indirect/Symbolic, and 102 instructions define as the symbol the numeric code 102. When the next Indirect/Symbolic instruction and code 102 are encountered, the calculator puts into the program counter the address containing the Indirect/Symbol instruction, and the program continues, beginning with that Indirect/Symbol instruction.

As an exercise in symbolic program addressing, load the sample program, beginning at branch point 3 (step 30), and execute it with data entries of 1, 2, 3, and 4. Note that a code 377 on the printout indicates that steps 40 through 44 contain NOOPs (no operation); NOOPs are automatically placed in all memory steps when the calculator is turned ON.

1. Set the PRINT switch to the off position.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH] [0] [3].

0030	066	
0031	021	+
0032	026	(
0033	005	5
0034	023	X
0035	056	
0036	060	
0037	126	Ju
0038	067	
0039	102	
0040	377	
0041	377	
0042	377	
0043	377	
0044	377	
0045	066	
0046	102	
0047	027	)
0048	021	+
0049	026	(
0050	002	2
0051	023	X
0052	056	
0053	060	
0054	027	)
0055	023	X
0056	056	
0057	060	
0058	024	÷
0059	056	
0060	060	
0061	020	=
0062	061	A
0063	127	br
0064	067	
0065	021	+
	1.0000	
	2.0000	
	3.0000	
	4.0000	
	6.7500	A

4. Set the RUN/STEP/LOAD switch to LOAD.

5. Depress the keys shown on the coding sheet, figure 4-10, steps 30 through 38. Depress [ENTER CODE] [1] [0] [2] for indirect address 102, step 39.

6. Depress [LIST PROG] 5 times to pass steps 40 through 44 (area reserved for another program).

7. Depress [IND] [ENTER CODE] [1] [0] [2].

8. Depress the keys shown in steps 47 through 65.

9. Set the RUN/STEP/LOAD switch to RUN.

10. Depress [RESUME].

11. To enter the first item, depress [1].

12. Depress [RESUME].

13. To enter the second item, depress [2].

14. Depress [RESUME].

15. To enter the third item, depress [3].

16. Depress [RESUME].

17. To enter the fourth item, depress [4].

18. Depress [RESUME].

To demonstrate that the program will operate in any section of memory, load the program, beginning at step 50, by depressing [BRANCH] [0] [5]. Execute the program in the same manner. Notice that although the addresses are different, the results are the same as when the program was stored, beginning at branch point 3.

0050	066	
0051	021	+
0052	026	(
0053	005	5
0054	023	X
0055	056	
0056	060	
0057	126	Ju
0058	067	
0059	102	
0060	377	
0061	377	
0062	377	
0063	377	
0064	377	
0065	066	
0066	102	
0067	027	)
0068	021	+
0069	026	(
0070	002	2
0071	023	X
0072	056	
0073	060	
0074	027	)
0075	023	X
0076	056	
0077	060	
0078	024	÷
0079	056	
0080	060	
0081	020	=
0082	061	A
0083	127	Br
0084	067	
0085	021	+
	1.00	
	2.00	
	3.00	
	4.00	
	6.75	A

DECISION-MAKING

The calculator has a feature that permits a choice between two or more possible sets of instructions. The decision is based on variable factors specified in the program. An instruction that tests these factors is a conditional, or decision-making, instruction. Usually the idea of an "if" is inherent in a conditional instruction. For example, an instruction might cause a branch to a certain memory step <u>if</u> a manual switch is set; or the calculator might perform a repetitive calculation that decreases the value in a certain register, and <u>if</u> the contents of the register has reached zero, the program might branch to another calculation. The program might also compare the contents of two registers by subtracting one from the other and then choose one of two paths, depending on whether the result is positive.

The calculator responds to several types of conditional instructions. The following paragraphs describe branches or jumps that may be performed with conditional instructions entered from the keyboard.

SENSE SWITCH DECISIONS

The keyboard SENSE switch establishes a condition that is tested by a decision-making instruction in the program. The decision-making function is performed by keyboard input of any of the following keying sequences:

BRANCH ( ) ( ) × n n

JUMP ( ) ( ) × n n

BRANCH ( ) ( ) × IND | SYMB α

JUMP ( ) ( ) × IND | SYMB α

where $\boxed{n}$ represents a numeral of the step to which the program branches, and $\boxed{\alpha}$ represents the symbol to which the program branches (either keyboard symbol or non-keyboard symbol). Upon encountering the Sense instruction, the calculator determines the position of the SENSE switch and decides which path to follow. The branch takes place only if the SENSE switch is in the up position. If the SENSE switch is down, the program ignores the branch and continues with the instruction following the numeral entries.

Since the position of the SENSE switch is controlled by the operator, remember to set the switch to the down position at the end of the calculation unless you want to execute the branch in the next calculation.

The SENSE switch may be used to signal the end of data entry so that the program can begin computation. Another use of the switch is to select one of two separate calculation routines in the program. The second use is shown schematically in the flowchart in figure 4-11. The program solves these equations:

$$X_n = X_1 + \frac{(X_2 - X_1)}{(Y_2 - Y_1)} (Y_n - Y_1)$$

$$Y_n = Y_1 + \frac{(Y_2 - Y_1)}{(X_2 - X_1)} (X_n - X_1)$$

Procedures for loading and executing the program are detailed below. You can enter X_n and solve for Y_n or you can enter Y_n and solve for X_n. Set the SENSE switch to the up position if you are solving for X_n. Leave the switch down if solving for Y_n.

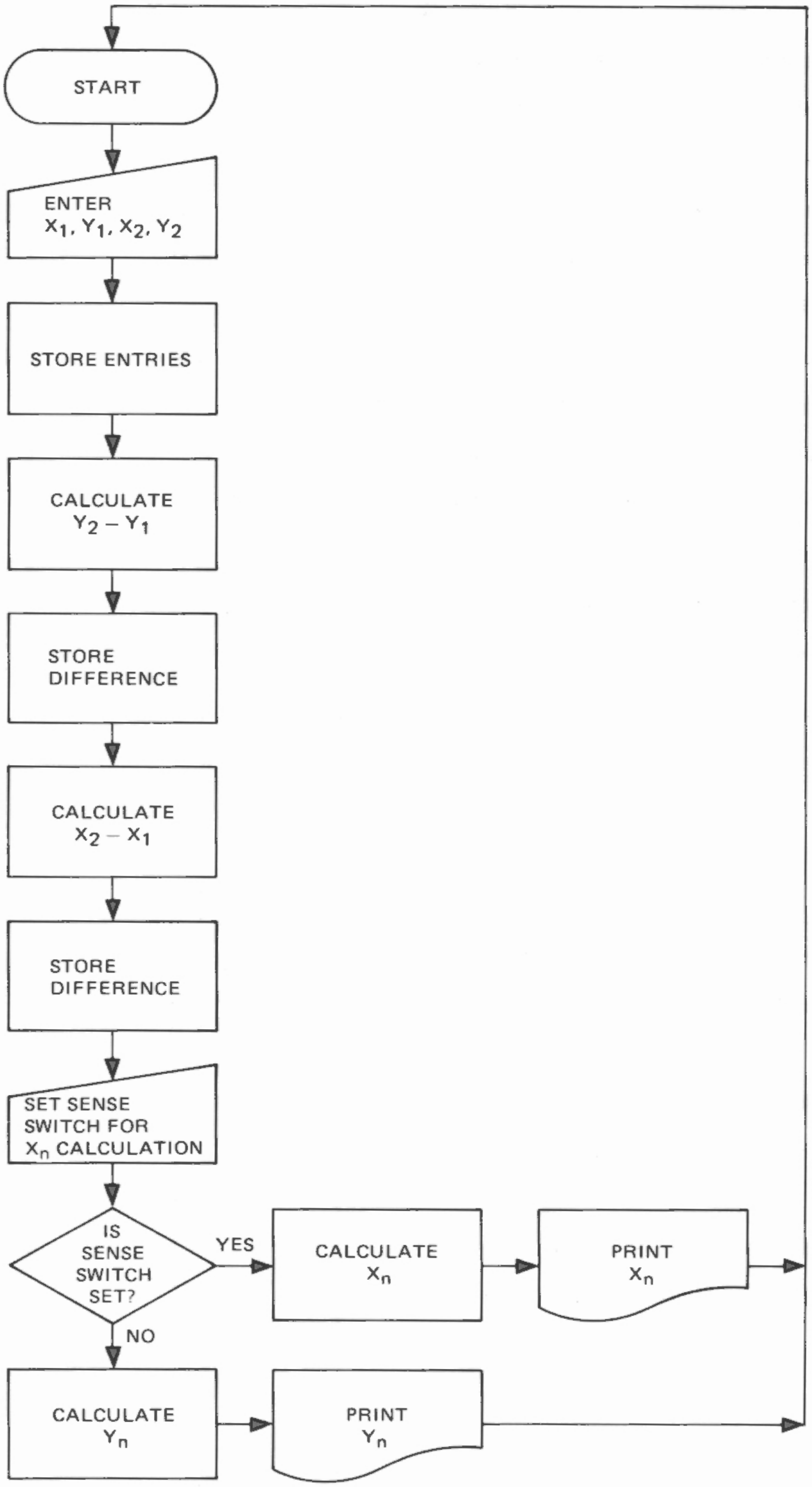


Figure 4-11. Sense Switch Flowchart

The coding sheet for the sample program is shown in figure 4-12. Each function in the flowchart is performed by a simple routine in the program. Once you have entered and stored the initial values of X_1, Y_1, X_2, and Y_2 and computed $(Y_2 - Y_1)$ and $(X_2 - X_1)$, you can enter any number of values of X_n or Y_n and solve for the unknown value, using the SENSE switch to determine which calculation is to be performed. If the switch is down, the program continues in its normal sequence and calculates Y_n. If the switch is up, the program branches to the X_n calculation. The functions performed by the sample program are noted on the coding sheet. Notice that scratch pad registers 1 through 4 are used for temporary storage of initial values and of intermediate results. Symbolic addressing is used so that you can store the program at any desired branch point.

The following variables will be used in executing the program:

$X_1 = 1$ $\quad$ $Y_1 = 3$

$X_2 = 2$ $\quad$ $Y_2 = 4$

$Y_n = 5$

Use the following procedure to load the sample program, beginning at branch point 8, and execute it:

1. Set the PRINT switch to PRINT.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH ( ) ( )] [0] [8].
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-12, in the order given.

0080	056	
0081	110	↓
0082	001	1
0083	056	
0084	110	↓
0085	002	2
0086	056	
0087	110	↓
0088	003	3
0089	056	
0090	110	↓
0091	004	4
0092	022	−
0093	111	↑
0094	002	2
0095	020	=
0096	110	↓
0097	004	4
0098	111	↑
0099	003	3
0100	022	−
0101	111	↑
0102	001	1
0103	020	=
0104	110	↓
0105	003	3
0106	066	
0107	052	F
0108	056	
0109	127	Br
0110	023	X
0111	067	
0112	055	√
0113	022	−
0114	111	↑
0115	001	1
0116	023	X
0117	111	↑
0118	004	4
0119	024	÷
0120	111	↑
0121	003	3
0122	021	+
0123	111	↑
0124	002	2
0125	020	=
0126	061	A

TITLE		PROGRAMMER	
STEP	**SYMBOL**	**COMMAND**	**COMMENTS**
80		HALT	ENTER X_1
81		↓( )	
82		1	
83		HALT	ENTER Y_1
84		↓( )	
85		2	STORE INITIAL VALUES
86		HALT	ENTER X_2
87		↓( )	
88		3	
89		HALT	ENTER Y_2
90		↓( )	
91		4	
92		—	
93		↑( )	CALCULATE $(Y_2 - Y_1)$
94		2	
95		=	
96		↓( )	STORE $(Y_2 - Y_1)$ IN REGISTER 4
97		4	
98		↑( )	
99		3	
100		—	CALCULATE $(X_2 - X_1)$
101		↑( )	
102		1	
103		=	
104		↓( )	STORE $(X_2 - X_1)$ IN REGISTER 3
105		3	
106	2ND FUNC	IND/SYMB	
107		2ND FUNC	
108		HALT	SENSE SWITCH UP: ENTER Y_n
109		BRANCH( )( )	SWITCH DOWN: ENTER X_n
110		X	
111		IND/SYMB	
112		√	
113		—	
114		↑( )	
115		1	
116		X	SOLVE FOR Y_n
117		↑( )	
118		4	
119		÷	

MONROE Litton

Figure 4-12. Sense Switch Example (1 of 2)

TITLE ____________ PROGRAMMER ____________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
12	0		↑( )	
	1		3	
	2		+	
	3		↑( )	
	4		2	
	5		=	
	6		PRINT ANS	PRINT Y_n
	7		JUMP ( )( )	
	8		IND / SYMB	GO BACK TO ENTRY OF X_n OR Y_n
	9		2ND FUNC	
13	0	√	IND/SYMB	
	1		√	
	2		—	
	3		↑( )	
	4		2	
	5		×	
	6		↑( )	
	7		3	
	8		÷	SOLVE FOR X_n
	9		↑( )	
14	0		4	
	1		+	
	2		↑( )	
	3		1	
	4		=	
	5		PRINT ANS	PRINT X_n
	6		BRANCH ( )( )	
	7		IND / SYMB	
	8		2ND FUNC	
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

Figure 4-12. Sense Switch Example (2 of 2)

6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress [RESUME].
8. To enter X_1, depress [1].
9. Depress [RESUME].
10. To enter Y_1, depress [3].
11. Depress [RESUME].
12. To enter X_2, depress [2].
13. Depress [RESUME].
14. To enter Y_2, depress [4].
15. Depress [RESUME].
16. Set the SENSE switch to the up position.
17. To enter Y_n, depress [5].
18. Depress [RESUME].
19. Reposition the SENSE switch to the down position unless you are going to calculate another X_n.

. FLAG KEY DECISIONS

The [FLAG] key performs the same function as the SENSE switch and operates in the same way except for one important difference: the [FLAG] key is momentary and the corresponding internal switch cannot be reset manually. Only a non-keyboard instruction in the program can reset the [FLAG] key (see section III). This feature is an advantage when you want to reset the key automatically, without having to remember to unlatch a switch on the keyboard.

The decision-making function is performed by keyboard input of any of the following keying sequences:

[BRANCH ( ) ( )] [FLAG] [n] [n]

[JUMP ( ) ( )] [FLAG] [n] [n]

[BRANCH ( ) ( )] [FLAG] [IND SYMB] [α]

[JUMP ( ) ( )] [FLAG] [IND SYMB] [α]

```
0127    126      Ju
0128    067
0129    052      F
0130    066
0131    055    √
0132    022      -
0133    111     ↑
0134    002        2
0135    023      X
0136    111     ↑
0137    003        3
0138    024      ÷
0139    111     ↑
0140    004        4
0141    021      +
0142    111     ↑
0143    001        1
0144    020      =
0145    061     A
0146    127    Br
0147    067
0148    052      F
   1.0000       ↓  1
   3.0000       ↓  2
   2.0000       ↓  3
   4.0000       ↓  4
   4.0000        -
   3.0000       ↑  2
   3.0000        =
   1.0000       *
   1.0000       ↓  4
   2.0000       ↑  3
   2.0000        -
   1.0000       ↑  1
   1.0000        =
   1.0000       *
   1.0000       ↓  3
   5.0000        -
   3.0000       ↑  2
   3.0000        X
   1.0000       ↑  3
   1.0000        ÷
   1.0000       ↑  4
   1.0000        +
   1.0000       ↑  1
   1.0000        =
   3.0000       *
   3.0000       A
```

where $\boxed{n}$ represents a numeral of the step to which the program branches and $\boxed{\alpha}$ represents the symbol to which the program branches (either keyboard symbol or non-keyboard symbol). Upon encountering the Flag instruction, the calculator determines the state of the flag and decides which path to follow. The branch takes place only if the [FLAG] key has been depressed. If the [FLAG] key has not been depressed, the program ignores the branch and continues with the instruction following the second numeral entry.

The routine to which the program branches must contain a Reset Flag instruction, machine code 166, so that you will be able to control the state of the flag the next time the program is executed. The Reset Flag instruction is not available on the keyboard and, therefore, must be loaded with the [ENTER CODE] key (see section III).

A flowchart for a sample program that tests the flag and SENSE switch is shown in figure 4-13. The program also demonstrates register arithmetic techniques.

The program solves the engineering problem of calculating heat transfer coefficients from test data. Though this example is merely illustrative, the same programming techniques can be applied to complex problems requiring many branching options, program steps, and data registers.

In a series of tests for determining the heat transfer coefficient of an organic fluid, the fluid was passed through an electrically heated tube, heavily insulated on the outer surface. There was a slight drift in tube wall temperatures during each run, and an average value of thermocouple readings taken throughout the run is to be used for computations.

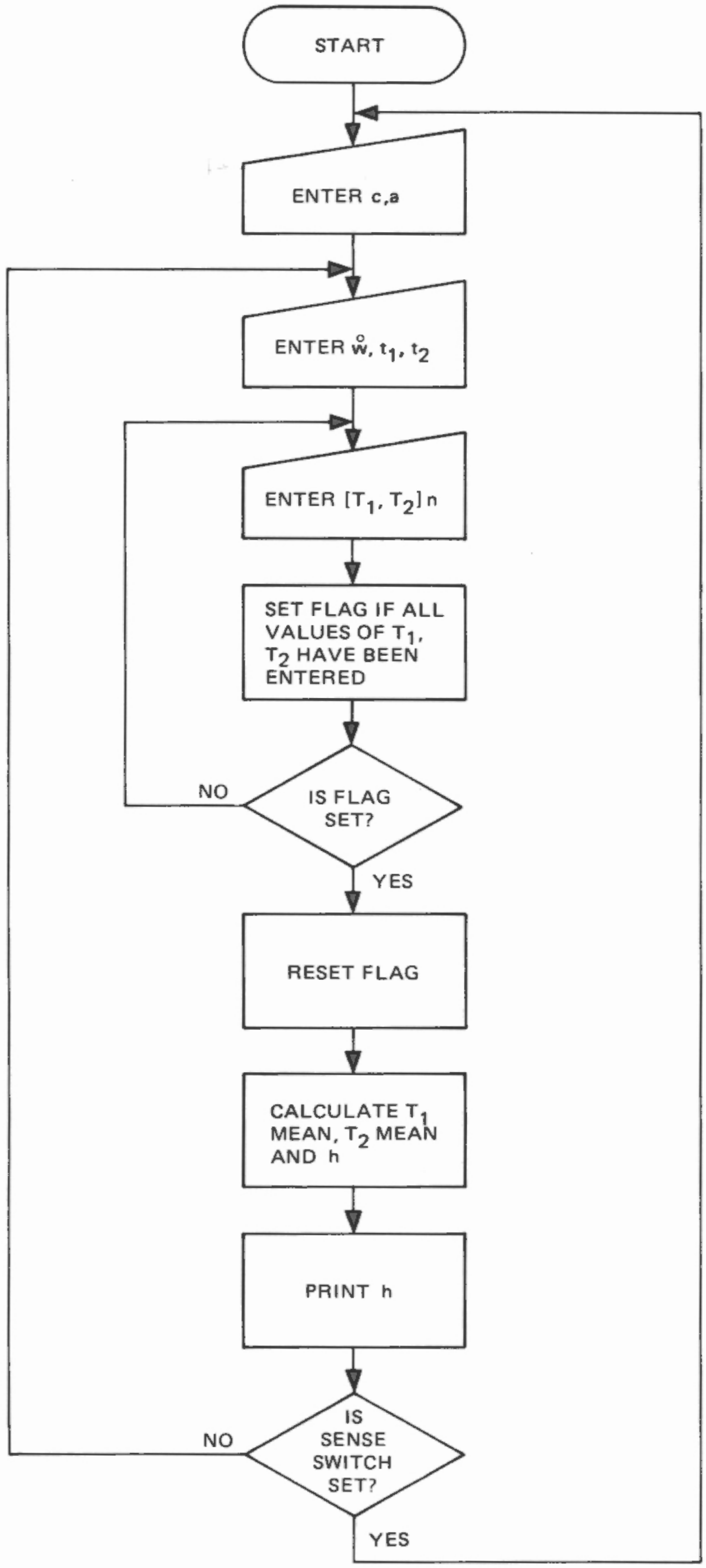


Figure 4-13. Flag Key Flowchart

The heat transfer coefficient h was determined with the following expression:

$$h = \frac{\mathring{w}\, c\, (t_2 - t_1) \ln \left[\frac{T_2 - t_2}{T_1 - t_1}\right]}{A\ [(T_2 - t_2) - (T_1 - t_1)]}$$

where

where	h	is the heat transfer coefficient, BTU/hr ft^2 °F
	$\mathring{w}$	is the organic fluid flow rate, lb/hr
	A	is the tube inner surface, ft^2
	c	is the organic fluid specific heat, BTU/lb °F
	T_2	is the tube exit wall temperature, °F
	T_1	is the tube inlet wall temperature, °F
	t_2	is the organic fluid outlet temperature, °F
	t_1	is the organic fluid inlet temperature, °F

The following table of test data is from a series of runs with fixed power input but varying organic fluid flow rate:

Run No.	$\mathring{w}$	t_1	t_2	T_1	T_2
1	1010	73.4	98.4	554.0	575.1
	1010	73.4	98.4	556.3	579.1
	1010	73.4	98.4	555.4	581.3
	1010	73.4	98.4	553.2	579.6
2	1840	73.5	87.2	383.7	398.4
	1840	73.5	87.2	382.3	397.6
	1840	73.5	87.2	383.1	399.2

For these runs, the organic fluid under test had a c value of 0.50; the tube inner surface area (A value) was 0.26 ft^2.

As shown in figure 4-13, after entering (and printing) c, A, $\mathring{w}$, t_1 and t_2 for the run under consideration, the series of T_1 and T_2 values for that run are entered. After the last set of T_1, T_2 entries, the user depresses the FLAG key, indicating the end of data entry for that run. The FLAG is tested; prior to entry of the last set of data, the FLAG is not set and the program returns ("NO" route) for the next set of T_1, T_2 entries. After the last data entry, the FLAG is set and the program moves ahead ("YES" route), resetting the FLAG immediately, in preparation for the next run of test data. After resetting the FLAG the mean values of T_1 and T_2 are calculated and stored. These values are then used in calculating h. After printing h, the program tests the SENSE switch. If the SENSE switch is in the down position ("NO" route), the program returns for entry of new $\mathring{w}$, t_1 and t_2 values for the next run. If the SENSE switch is in the up position, the program returns for entry of a new c, a new A, or both, indicating that test data is now to be entered for a different organic fluid (different specific heat) or a different heated tube size, or both.

When all T_1, T_2 entries for a run have been entered, and the next run is to use a different c and/or A, the user both depresses the [FLAG] key and moves the SENSE switch up. (The SENSE switch must be moved to the down position again at the start of the following run, since that switch can only be reset manually.)

The coding sheet for the sample program is shown in figure 4-14. The first step defines symbol 5. Other symbolic addresses are defined as =, Σ^2_n, and 7. Steps 0 through 10 involve entry, printing, and storing of c and A. Step 11 defines symbol =. The following steps involve entry, printing, and storing of $\mathring{w}$, t_1 and t_2. Step 29 defines symbol Σ^2_n followed by a Halt and Flag test to see if all values of T_1, T_2 have been entered. T_1 and T_2 are entered and accumulated preparatory to calculating $\Sigma T/n$, that is, T mean. Calculation of T mean and h are performed following the definition of symbol 7 at step 50. Before studying the steps in detail, notice that the problem is divided into units, each headed by a symbolic address. On testing for FLAG or SENSE, a Jump to these units may be made and repeated calculations performed.

Before the operating procedure for a sample calculation is presented, a few details might be examined. At step 32, a Halt permits entry of T_1 in the first, second, or nth temperature-set entry. However, if all values of T_1 and T_2 have been entered, FLAG is depressed at that Halt instead. Then, when [RESUME] is depressed, the program tests the FLAG at step 34. With FLAG depressed, the program jumps to symbolic address 7 (step 50), where the FLAG is reset and the calculations proceed. If FLAG had not been depressed after entering T_1 and depressing [RESUME], the program would have "fallen through" to step 37, printing T_1.

Steps 63 through 106 use a considerable amount of register arithmetic. At step 63, $\mathring{w}$ is recalled to the E-register from scratch pad register 6. A register multiplication is then performed (steps 65 - 67) producing the product, $\mathring{w}c$, in the E-register. At step 66, a normal multiplication is started, that is, $\mathring{w}c$ x .... At that point, $\mathring{w}c$ is in both the E-register and a non-user internal register, which holds the number to be multiplied ($\mathring{w}c$). Consequently, when the content of register 8 (t_2) is recalled to the E-register (steps 69, 70), though $\mathring{w}c$ is lost from the E-register and superseded by t_2, $\mathring{w}c$ is retained in the non-user internal register. A register subtraction is then performed (steps 71 - 73) between the E-register (t_2) and register 7 (t_1) resulting in ($t_2 - t_1$) in the E-register, while $\mathring{w}c$ is still held in the non-user internal register. The start of a third multiplication is performed at step 74. In effect, the following has been performed: $\mathring{w}c \times (t_2 - t_1) \times$ ....

TITLE ____________ PROGRAMMER ____________ MONROE

STEP	SYMBOL	COMMAND	COMMENTS
0 0	5	IND/SYMB	SYMBOLIC ADDRESS 5
1		5	
2		ADV	ADVANCE A SPACE
3		HALT	ENTER C
4		PRINT X	PRINT C
5		↓( )	STORE C IN SCRATCH
6		4	PAD REGISTER 4
7		HALT	ENTER A
8		PRINT X	PRINT A
9		↓( )	STORE A IN SCRATCH
1 0		5	PAD REGISTER 5
1	=	IND/SYMB	SYMBOLIC ADDRESS =
2		=	
3	ENTER CODE	176	PRINT A LINE OF DOTS
4	ENTER CODE	166	RESET PROGRAM FLAG 1
5		Φ	CLEAR REGISTERS E, 0, 1,
6		0	2, AND 3
7		HALT	ENTER $\mathring{\omega}$
8		PRINT X	PRINT $\mathring{\omega}$
9		↓( )	STORE $\mathring{\omega}$ IN SCRATCH
2 0		6	PAD REGISTER 6
1		HALT	ENTER t_1
2		PRINT X	PRINT t_1
3		↓( )	STORE t_1 IN SCRATCH
4		7	PAD REGISTER 7
5		HALT	ENTER t_2
6		PRINT X	PRINT t_2
7		↓( )	STORE t_2 IN SCRATCH
8		8	PAD REGISTER 8
9	Σ_n^2	IND/SYMB	SYMBOLIC ADDRESS Σ_n^2
3 0		Σ_n^2	
1		ADV	ADVANCE A SPACE
2		HALT	ENTER T_1
3		JUMP( ) ( )	Q: HAVE ALL T_s BEEN ENTERED?
4		FLAG	IF YES, GO TO IND SYMB 7.
5		IND/SYMB	IF NO, CONTINUE ENTRIES.
6		7	
7		PRINT X	PRINT T_1
8	ENTER CODE	113	ACCUMULATE T_1 IN SCRATCH
9		1	PAD REGISTER 1

Figure 4-14. Flag Key Example (1 of 3)

TITLE ______________ PROGRAMMER ______________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
4	0		HALT	ENTER T_2
	1		PRINT X	PRINT T_2
	2	ENTER CODE	113	ACCUMULATE T_2 IN SCRATCH
	3		2	PAD REGISTER 2
	4		1	
	5	ENTER CODE	113	ACCUMULATE n IN SCRATCH
	6		3	PAD REGISTER 3
	7		JUMP ( ) ( )	
	8		IND / SYMB	CONTINUE ENTERING T_s
	9		Σ_n^2	
5	0	7	IND/SYMB	SYMBOLIC ADDRESS 7 - ALL T_s HAVE
	1		7	BEEN ENTERED. FIND MEAN AND CONTINUE.
	2	ENTER CODE	166	RESET PROGRAM FLAG 1
	3		↑( )	RECALL n INTO THE E-REGISTER
	4		3	
	5		↓( )	
	6		÷	CALCULATE $\bar{T}_1 = \frac{\Sigma T_1}{n}$, STORE
	7		1	ANSWER IN SCRATCH PAD REGISTER 1
	8		↓( )	
	9		÷	CALCULATE $\bar{T}_2 = \frac{\Sigma T_2}{n}$, STORE
6	0		2	ANSWER IN SCRATCH PAD REGISTER 2
	1	ENTER CODE	176	PRINT A LINE OF DOTS
	2	ENTER CODE	176	PRINT A LINE OF DOTS
	3		↑( )	
	4		6	
	5		↑( )	
	6		X	
	7		4	
	8		X	
	9		↑( )	
7	0		8	
	1		↑( )	
	2		−	
	3		7	
	4		X	
	5		(	
	6		(	
	7		↑( )	
	8		2	
	9		↑( )	

Figure 4-14. Flag Key Example (2 of 3)

TITLE ______ PROGRAMMER ______ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
8	0		—	
	1		8	
	2		÷	
	3		↑( )	
	4		1	
	5		↑( )	
	6		—	
	7		7	
	8		)	
	9		Ln LOG	
9	0		)	
	1		÷	
	2		↑( )	
	3		5	CALCULATE h
	4		÷	
	5		↑( )	
	6		2	
	7		↑( )	
	8		—	
	9		8	
10	0		↑( )	
	1		—	
	2		1	
	3		↑( )	
	4		+	
	5		7	
	6		=	
	7		PRINT X	
	8	ENTER CODE	176	PRINT A LINE OF DOTS
	9		ADV	ADVANCE A SPACE
11	0		JUMP ( ) ( )	
	1		X	IF SENSE IS UP, GO TO C ENTRY
	2		IND / SYMB	
	3		5	
	4		JUMP ( ) ( )	
	5		IND / SYMB	IF SENSE IS DOWN, GO TO ω̊ ENTRY
	6		=	
	7			
	8			
	9			

Figure 4-14. Flag Key Example (3 of 3)

The entire chain, corresponding to the equation for h, follows:

$$\uparrow 6 \times \uparrow 4 \times (\uparrow 8 - \uparrow 7)\ \ln \left[\left[\frac{\uparrow 2 - \uparrow 8}{\uparrow 1 - \uparrow 7}\right]\right] \frac{1}{\uparrow 5\,(\uparrow 2 - \uparrow 8 - \uparrow 1 + \uparrow 7)} =$$

The sequence $\uparrow 2 - \uparrow 8 - \uparrow 1 + \uparrow 7$ (steps 95 - 105) is $T_2 - t_2 - (T_1 - t_1)$ of the equation, in which $-(T_1 - t_1)$ is expressed as $-T_1 + t_1$.

In steps 110 - 113, if the SENSE switch is in the up position, a jump is made to symbolic address 5 for entry of new values of c and/or A. If the SENSE switch is in the down position, the program "falls through" to step 114, where a jump is made to symbolic address = for entry of new values of $\mathring{w}$, t_1 and t_2.

Load the sample program, beginning at branch point 0 and execute the program using the data of runs 1 and 2 in the data table.

1. Set the PRINT switch and SENSE switch to the down (off) position.
2. Depress [BRANCH] [0] [0].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress the keys shown on the coding sheet, figure 4-14, using the [ENTER CODE] key where necessary.
5. Set the RUN/STEP/LOAD switch to RUN.
6. Depress [RESUME].
7. To enter c, depress [•] [5].
8. Depress [RESUME].
9. To enter A, depress [•] [2] [6].
10. Depress [RESUME].
11. To enter $\mathring{w}$, depress [1] [0] [1] [0].
12. Depress [RESUME].
13. To enter t_1, depress [7] [3] [•] [4].
14. Depress [RESUME].
15. To enter t_2, depress [9] [8] [•] [4].
16. Depress [RESUME].
17. To enter first value of T_1, depress [5] [5] [4].
18. Depress [RESUME].

0000	066	
0001	005	5
0002	065	
0003	056	
0004	060	
0005	110	↓
0006	004	4
0007	056	
0008	060	
0009	110	↓
0010	005	5
0011	066	
0012	020	=
0013	176	
0014	166	
0015	116	[illegible]
0016	000	0
0017	056	
0018	060	
0019	110	↓
0020	006	6
0021	056	
0022	060	
0023	110	↓
0024	007	7
0025	056	
0026	060	
0027	110	↓
0028	010	8

19. To enter first value of T_2, depress [5] [7] [5] [•] [1].

20. Depress [RESUME].

21. To enter second value of T_1, depress [5] [5] [6] [•] [3].

22. Depress [RESUME].

23. To enter second value of T_2, depress [5] [7] [9] [•] [1].

24. Depress [RESUME].

25. To enter third value of T_1, depress [5] [5] [5] [•] [4].

26. Depress [RESUME].

27. To enter third value of T_2, depress [5] [8] [1] [•] [3].

28. Depress [RESUME].

29. To enter fourth value of T_1, depress [5] [5] [3] [•] [2].

30. Depress [RESUME].

31. To enter fourth value of T_2, depress [5] [7] [9] [•] [6].

32. Depress [RESUME].

33. Since all T_1, T_2 values have been entered, depress [FLAG].

34. Depress [RESUME].

After two lines of dots for separation, the answer, h = 100.9830 is printed. The program is now ready for run 2, starting with entry of $\mathring{w}$ = 1840. The printouts for the second run are shown. The answer for run 2 is 156.1928.

To illustrate the SENSE switch option, assume a third run with data identical to that of run 1, but with a different organic fluid having a specific heat of 0.43. Then, after entering the last T_2

0029	066	
0030	047	Σ
0031	065	
0032	056	
0033	126	Ju
0034	016	
0035	067	
0036	007	7
0037	060	
0038	113	+
0039	001	1
0040	056	
0041	060	
0042	113	+
0043	002	2
0044	001	1
0045	113	+
0046	003	3
0047	126	Ju
0048	067	
0049	047	Σ
0050	066	
0051	007	7
0052	166	
0053	111	↑
0054	003	3
0055	110	↓
0056	024	÷
0057	001	1
0058	110	↓
0059	024	÷
0060	002	2
0061	176	
0062	176	
0063	111	↑
0064	006	6
0065	111	↑
0066	023	X
0067	004	4
0068	023	X
0069	111	↑
0070	010	8
0071	111	↑
0072	022	−

value in run 2, depress [FLAG] and also move the SENSE switch to the up position. After calculating the h for run 2 (h = 156.1928), the program will be ready for entry of the new value of c, followed by entry of A (still equal to 0.26), etc. The printouts are shown. Remember to move the SENSE switch down if you wish to try run 2 again, with the new value of c.

DECISIONS BASED ON E-REGISTER CONTENTS

The calculator can test the content of the E-register and make the following three types of conditional branches or jumps. In the sample keying sequences, [n] represents a numeral of the step to which the program branches, and [α] represents the symbol to which the program branches (either keyboard symbol or non-keyboard symbol).

1. Branch or jump if the content of the E-register is positive:

   [BRANCH ( )] [+] [n] [n]
   [JUMP ( )] [+] [n] [n]
   [BRANCH ( )] [+] [IND SYMB] [α]
   [JUMP ( )] [+] [IND SYMB] [α]

2. Branch or jump if the content of the E-register is zero:

   [BRANCH ( )] [=] [n] [n]
   [JUMP ( )] [=] [n] [n]
   [BRANCH ( )] [=] [IND SYMB] [α]
   [JUMP ( )] [=] [IND SYMB] [α]

3. Branch or jump if the content of the E-register is negative:

   [BRANCH ( )] [−] [n] [n]
   [JUMP ( )] [−] [n] [n]
   [BRANCH ( )] [−] [IND SYMB] [α]
   [JUMP ( )] [−] [IND SYMB] [α]

0073	007		7
0074	023		X
0075	026		(
0076	026		(
0077	111		↑
0078	002		2
0079	111		↑
0080	022		−
0081	010		8
0082	024		÷
0083	111		↑
0084	001		1
0085	111		↑
0086	022		−
0087	007		7
0088	027		)
0089	050	ℓg	
0090	027		)
0091	024		÷
0092	111		↑
0093	005		5
0094	024		÷
0095	111		↑
0096	002		2
0097	111		↑
0098	022		−
0099	010		8
0100	111		↑
0101	022		−
0102	001		1
0103	111		↑
0104	021		+
0105	007		7
0106	020		=
0107	060		
0108	176		
0109	065		
0110	126		Ju
0111	023		X
0112	067		
0113	005		5
0114	126		Ju
0115	067		
0116	020		=

If the specified condition is not met, the program continues in its normal sequence. The example below illustrates the application of E-register tests.

The flowchart in figure 4-15 shows a solution to the quadratic equation

$$ax^2 + bx + c = 0$$

in which the E-register content is tested twice. At the first test, if the value a (in the E-register) is zero, the program calculates $-c/b$. Otherwise, the program goes to the imaginary or two-value solution. When

$$(-b/2a)^2 - c/a$$

has been computed and the result placed in the E-register, the E-register content is again tested. If the value is negative, the program branches to the solution for imaginary numbers. If the E-register content is positive or zero, the program continues with the normal sequence and computes the two real values.

The coding sheet in figure 4-16 shows the instructions used to solve the problem. Individual functions and routines are explained in the "Comments" column. Symbolic addressing is used, where:

[√] is the symbol for the starting location

[Ln LOG] is the symbol for the a = 0 routine

[SET D.P.] is the symbol for the imaginary solution routine

The following variables will be used in the execution of the program:

Run No.	a	b	c
1	2	4	6
2	2	1	0

```
         0•5000
         0•2600
• • • • • • • • • • • • • • • •
     1,010•0000
        73•4000
        98•4000

       554•0000
       575•1000

       556•3000
       579•1000

       555•4000
       581•3000

       553•2000
       579•6000
• • • • • • • • • • • • • • • •
• • • • • • • • • • • • • • • •
       100•9830
• • • • • • • • • • • • • • • •

• • • • • • • • • • • • • • • •
     1,840•0000
        73•5000
        87•2000

       383•7000
       398•4000

       382•3000
       397•6000

       383•1000
       399•2000

• • • • • • • • • • • • • • • •
• • • • • • • • • • • • • • • •
       156•1928
• • • • • • • • • • • • • • • •

         0•4300
         0•2600
• • • • • • • • • • • • • • • •
     1,010•0000
        73•4000
        98•4000

       554•0000
       575•1000

       556•3000
       579•1000

       555•4000
       581•3000

       553•2000
       579•6000

• • • • • • • • • • • • • • • •
• • • • • • • • • • • • • • • •
        86•8454
• • • • • • • • • • • • • • • •
```

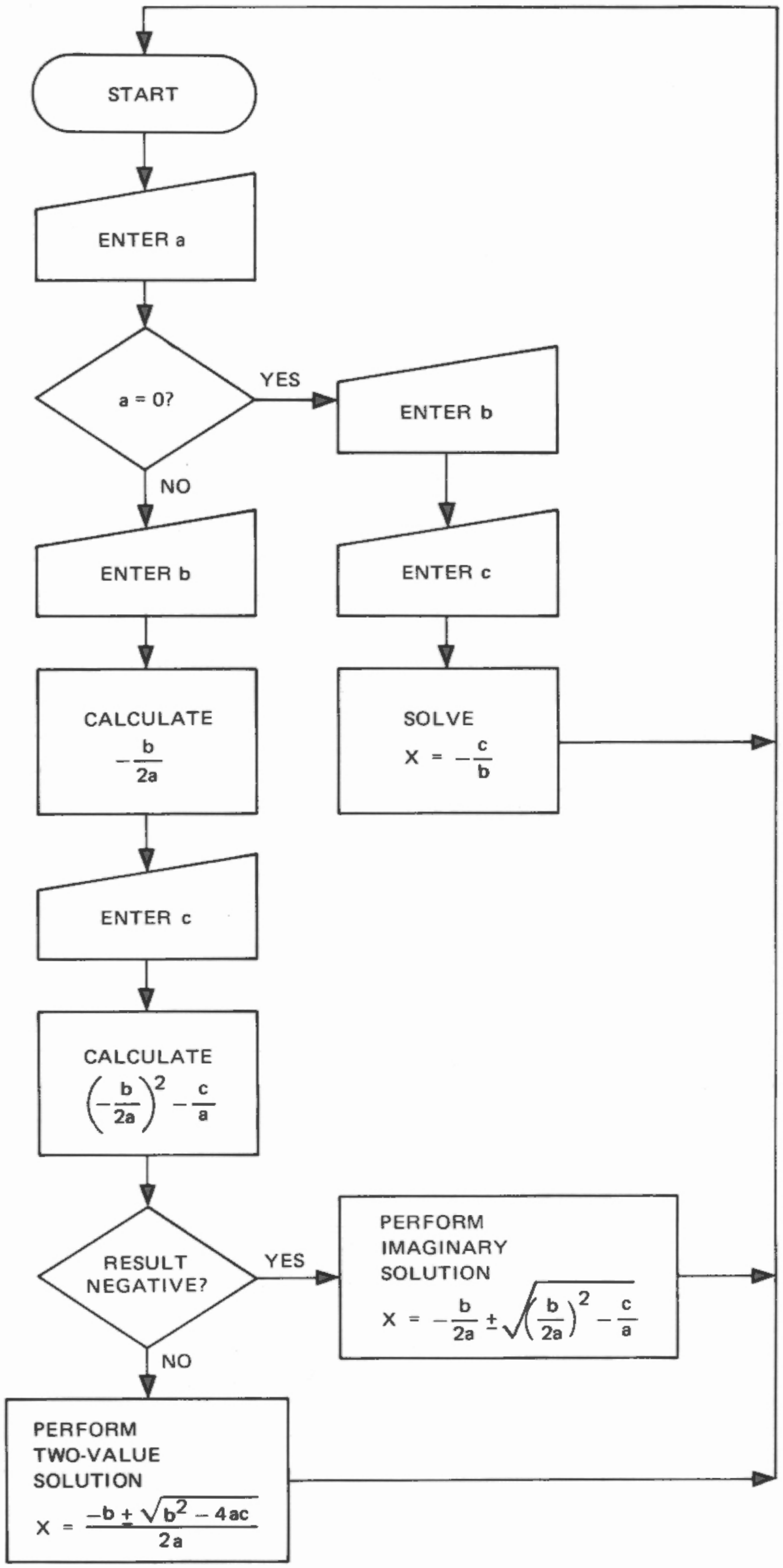


Figure 4-15. Flowchart for Branching on E-Register Contents

TITLE ____________ PROGRAMMER ____________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
	0	$\sqrt{\ }$		IND/SYMB
	1			$\sqrt{\ }$
	2		HALT	ENTER a
	3		JUMP ( ) ( )	
	4		=	GO TO $a = .0$ ROUTINE
	5		IND/SYMB	
	6		Ln LOG	
	7		↓( )	$a \neq 0$; STORE a IN SCRATCH
	8		1	PAD REGISTER 1
	9		HALT	ENTER b
	0		÷	
	1		↑( )	
	2		1	
	3		÷	CALCULATE $-\left(\frac{b}{2a}\right)$
	4		2	
	5		=	
	6		CHG SIGN	
	7		↓( )	STORE $-\left(\frac{b}{2a}\right)$ IN SCRATCH
	8		2	PAD REGISTER 2
	9		HALT	ENTER c
	0		÷	
	1		↑( )	CALCULATE $\frac{c}{a}$
	2		1	
	3		=	
	4		↓( )	STORE $\frac{c}{a}$ IN SCRATCH PAD
	5		1	REGISTER 1
	6		↑( )	
	7		2	
	8		×	
	9		−	CALCULATE $\left(-\frac{b}{2a}\right)^2 - \frac{c}{a}$
	0		↑( )	
	1		1	
	2		=	
	3		↓( )	STORE IN SCRATCH
	4		1	PAD REGISTER 1
	5		JUMP ( ) ( )	
	6		−	$E < 0$; GO TO IMAGINARY
	7		IND/SYMB	NUMBER CALCULATION
	8		SET D.P.	
	9		2	

Figure 4-16. E-Register Decision Example (1 of 3)

TITLE ______________ PROGRAMMER ______________ Litton MONROE

STEP	SYMBOL	COMMAND	COMMENTS
0	ENTER CODE	177	IDENTIFIER "2" IMPLIES TWO-ANSWER SOLUTION
1		PRINT X	PRINT $(\frac{-b}{2a})^2 - \frac{c}{a}$, FOR CHECKOUT PURPOSES
2		↑( )	
3		1	
4		$\sqrt{\ }$	
5		↓( )	
6		1	CALCULATE $-\frac{b}{2a} + \sqrt{(\frac{b}{2a})^2 - \frac{c}{a}}$
7		+	
8		↑( )	
9		2	
0		=	
1		PRINT ANS	PRINT X_2
2		↑( )	
3		2	
4		−	CALCULATE $\frac{-b}{2a} - \sqrt{(\frac{b}{2a})^2 - \frac{c}{a}}$
5		↑( )	
6		1	
7		=	
8		PRINT ANS	PRINT X_1
9		JUMP ( ) ( )	
0		IND/SYMB	RETURN TO START OF PROGRAM
1		$\sqrt{\ }$	
2	SET D.P.	IND/SYMB	
3		SET D.P.	
4		3	IDENTIFIER "3" IMPLIES X IS IMAGINARY
5	ENTER CODE	177	
6		PRINT X	PRINT $(\frac{-b}{2a})^2 - \frac{c}{a}$, FOR CHECKOUT
7		↑( )	
8		1	CALCULATE $\sqrt{\left\lvert(\frac{b}{2a})^2 - \frac{c}{a}\right\rvert}$, ABSOLUTE VALUE
9		CHG SIGN	
0		$\sqrt{\ }$	
1		PRINT ANS	PRINT IMAGINARY PART OF X
2		↑( )	
3		2	PRINT REAL PART OF X
4		PRINT ANS	
5		JUMP ( ) ( )	
6		IND/SYMB	RETURN TO START OF PROGRAM
7		$\sqrt{\ }$	
8	LN LOG	IND/SYMB	
9		LN LOG	$a = 0$ ROUTINE

Figure 4-16. E-Register Decision Example (2 of 3)

TITLE ____________ PROGRAMMER ____________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
	0		HALT	ENTER b
	1		↓( )	
	2		1	
	3		HALT	ENTER C
	4		÷	
	5		↑( )	
	6		1	CALCULATE $-\frac{c}{b}$
	7		=	
	8		CHG SIGN	
	9		↓( )	
	0		0	
	1		1	
	2	ENTER CODE	177	IDENTIFIER "1" IMPLIES ONE-VALUE SOLUTION
	3		↑( )	
	4		0	
	5		PRINT ANS	PRINT X
	6		JUMP ( ) ( )	
	7		IND/SYMB	RETURN TO START OF PROGRAM
	8		√	
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

Figure 4-16. E-Register Decision Example (3 of 3)

1. Set the PRINT switch to the off position.
2. Set the RUN/STEP/LOAD switch to RUN.
3. Depress [BRANCH] [2] [9].
4. Set the RUN/STEP/LOAD switch to LOAD.
5. Depress the keys shown on the coding sheet, figure 4-16.
6. Set the RUN/STEP/LOAD switch to RUN.
7. Depress [RESUME].
8. To enter a, of run 1, depress [2].
9. Depress [RESUME].
10. To enter b, depress [4].
11. Depress [RESUME].
12. To enter c, depress [6].
13. Depress [RESUME]. (Note the printing of identifier "3.", indicating an imaginary solution.)
14. To enter a of run 2, depress [2].
15. Depress [RESUME].
16. To enter b, depress [1].
17. Depress [RESUME].
18. To enter c, depress [0].
19. Depress [RESUME]. (Note the printing of identifier "2.", indicating a two-answer solution.)

0290	066	
0291	055	√
0292	056	
0293	126	Ju
0294	020	=
0295	067	
0296	050	Lg
0297	110	↓
0298	001	/
0299	056	
0300	024	÷
0301	111	↑
0302	001	/
0303	024	÷
0304	002	2
0305	020	=
0306	013	−
0307	110	↓
0308	002	2
0309	056	
0310	024	÷
0311	111	↑
0312	001	/
0313	020	=
0314	110	↓
0315	001	/
0316	111	↑
0317	002	2
0318	023	X
0319	022	−
0320	111	↑
0321	001	/
0322	020	=
0323	110	↓
0324	001	/
0325	126	Ju
0326	022	−
0327	067	

0328	117	
0329	002	2
0330	177	
0331	060	
0332	111	↑
0333	001	1
0334	055	√
0335	110	↓
0336	001	1
0337	021	+
0338	111	↑
0339	002	2
0340	020	=
0341	061	A
0342	111	↑
0343	002	2
0344	022	−
0345	111	↑
0346	001	1
0347	020	=
0348	061	A
0349	126	Ju
0350	067	
0351	055	√
0352	066	
0353	117	
0354	003	· 3
0355	177	
0356	060	
0357	111	↑
0358	001	1
0359	013	−
0360	055	√
0361	061	A
0362	111	↑
0363	002	2
0364	061	A
0365	126	Ju
0366	067	
0367	055	√
0368	066	
0369	050	lg
0370	056	
0371	110	↓

0372	001	1
0373	056	
0374	024	÷
0375	111	↑
0376	001	1
0377	020	=
0378	013	−
0379	110	↓
0380	000	0
0381	001	1
0382	177	
0383	111	↑
0384	000	0
0385	061	A
0386	126	Ju
0387	067	
0388	055	√

3.

−2.0000	
1.4142	A
−1.0000	A

2.

0.0625	
0.0000	A
−0.5000	A

INDEXING

The Monroe Model 1880 Scientific Calculator contains an index register whose basic function is to modify instructions. Basic principles of indexing are explained in the Monroe primer, Fundamentals of Programming. Indexing as related to the calculator is described in the Advanced Programming Reference Manual.

V. PROGRAM EXECUTION

The following paragraphs describe procedures for loading and manipulating programs from the keyboard. Note that, when the calculator is turned on, the following conditions are established:

- All registers are cleared.
- Program memory is filled with NOOP (no operation) codes.
- The decimal point is set to 2.
- A Reset instruction is executed.
- Print Enable is turned on.

LOADING A PROGRAM

A program must be loaded, starting at a specified branch point in program memory. You might select a block of memory steps beginning, for example, at step 50. A typical loading procedure is given below, with an explanation of each step in the procedure. The procedure loads a program to evaluate the equation:

$$\frac{((5 \times b) + (2 \times c))a}{d}$$

when values for a, b, c, and d are input. The coding sheet for the program is shown in figure 5-1.

Keyboard Input	Explanation
1. Set the RUN/STEP/LOAD switch to RUN or STEP.	Prepares the calculator to set a branch point address into the program counter.
2. Depress [BRANCH () ()] (or [JUMP () ()]) [0] [5].	Sets the program counter to 50 (branch point 5).
3. Set the RUN/STEP/LOAD switch to LOAD.	Prepares the calculator to load information into program memory.
4. Depress [( ].	Loads an Open Parenthesis instruction into step 50. Notice the printout: 0050 026 (which shows the program address and the left-parenthesis code (see appendix A). The program counter automatically counts to 51.

TITLE ______________ PROGRAMMER ______________ Litton MONROE

STEP		SYMBOL	COMMAND	COMMENTS
5	0		(	
	1		5	
	2		X	
	3		HALT	ENTER b
	4		PRINT X	
	5		)	
	6		+	
	7		(	
	8		2	
	9		X	
6	0		HALT	ENTER c
	1		PRINT X	
	2		)	
	3		X	
	4		HALT	ENTER a
	5		PRINT X	
	6		÷	
	7		HALT	ENTER d
	8		PRINT X	
	9		=	
7	0		PRINT ANS	PRINT RESULT
	1		BRANCH ( ) ( )	
	2		0	BRANCH TO BEGINNING OF PROGRAM,
	3		0	BRANCH POINT 0
	4			
	5			
	6			
	7			
	8			
	9			
	0			
	1			
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			

Figure 5-1. Program for Stepped Testing

Keyboard Input

5. Depress [5].

Explanation

Loads a digit 5, code 005, into step 51. Notice the printout:

0051	005	5

The program counter automatically counts to 52.

Keyboard Input

6. Depress the remaining keys:

- [×]
- [HALT]
- [PRINT x]
- [)]
- [+]
- [(]
- [2]
- [×]
- [HALT]
- [PRINT x]
- [)]
- [×]
- [HALT]
- [PRINT x]
- [÷]
- [HALT]
- [PRINT x]
- [=]
- [PRINT ANS]
- [BRANCH () ()]
- [0]
- [5]

Explanation

Loads the remaining instructions into steps 52 through 73. The program counter counts up by one each time a key is depressed. Addresses, instruction codes, and print symbols (if any), are printed as follows:

0052	023	X
0053	056	
0054	060	
0055	027	)
0056	021	+
0057	026	(
0058	002	2
0059	023	X
0060	056	
0061	060	
0062	027	)
0063	023	X
0064	056	
0065	060	
0066	024	÷
0067	056	
0068	060	
0069	020	=
0070	061	A
0071	127	Br
0072	000	0
0073	005	5

Notice that after each instruction is loaded, the program counter contains the number of the next step to be loaded. After loading the program, set the RUN/STEP/LOAD switch to RUN. This setting ends the loading operation and prepares the calculator to execute the program.

If the program had contained non-keyboard codes, they would have been entered with the [ENTER CODE 000] key. Note that, when entering non-keyboard codes, the program counter advances only after every fourth key depression, as in:

Keyboard Input	Program Counter	Memory Contents	Step Number
	69	. . .	68
[ENTER CODE 000] [1] [6] [6]	70	166	69
[ENTER CODE 000] [1] [7] [6]	71	176	70
[ENTER CODE 000] [1] [7] [7]	72	177	71

If you enter the wrong number at any time during program loading, correct the error by using the [BACK SPACE] key. For example, if you had depressed [3] instead of [2] at step 57 in the program you just loaded, depress [BACK SPACE] and then key in the correct number as follows:

[(]
[5]
[×]
[HALT]
[)]
[+]
[(]
[3] Wrong key
[BACK SPACE]
[2] Correct key
[×] Continue loading program

0050	026	(
0051	005	5
0052	023	X
0053	056	
0054	027	)
0055	021	+
0056	026	(
0057	003	3
0056	026	(
0057	002	2
0058	023	X

If you discover your error after you have loaded several subsequent steps, backspace as many times as necessary to reach the step immediately preceding the incorrect step. When you have corrected the error, use the [LIST PROG] key to print the correct codes following the corrected step in order to advance the program counter to the step where you discovered your error. For example, if you depressed [6] instead of [5] at step 51 in the program just loaded, but you did not notice the error until after you depressed [+], you could correct the error as follows:

[(]	
[6]	Wrong key
[×]	
[HALT]	
[)]	
[+]	Error detected
[BACK SPACE]	Backspace five times to step before error
[BACK SPACE]	
[BACK SPACE]	
[BACK SPACE]	
[BACK SPACE]	
[5]	Correct error
[LIST PROG]	List codes entered correctly
[LIST PROG]	
[LIST PROG]	
[LIST PROG]	
[(]	Continue loading program

0050	026	(
0051	006	6
0052	023	X
0053	056	
0054	027	)
0055	021	+
0054	027	)
0053	056	
0052	023	X
0051	006	6
0050	026	(
0051	005	5
0052	023	X
0053	056	
0054	027	)
0055	021	+
0056	026	(

VERIFYING A PROGRAM

The paragraphs below present three methods of determining whether a program has been loaded correctly.

VERIFYING DURING LOADING

One way to verify that you have depressed the correct keys is to look at the tape that was printed during loading. The printout shows the address, the instruction code, and the corresponding print symbol, if any, of each instruction that was loaded. For example, if you addressed branch point 2 and loaded [HALT], [↓( )], [1], [HALT], [↓( )], [2], the printout would be as shown.

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [0] [2].
3. Set the RUN/STEP/LOAD switch to LOAD.

4. Depress the following keys:

[HALT]
[↓( )]
[1]
[HALT]
[↓( )]
[2]

```
0020   056
0021   110   ↓
0022   001      1
0023   056
0024   110   ↓
0025   002      2
```

LISTING A PROGRAM

If the original printed tape is not available, you can print all or any part of your program by using the [LIST PROG] key with the RUN/STEP/LOAD switch set to LOAD. To list the program steps, one by one, depress [LIST PROG] and release immediately. The address, code, and symbol are printed. Depress and release [LIST PROG] for every step you wish to verify. To verify the instructions loaded according to the previous paragraph, proceed as follows:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [0] [2].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress and release [LIST PROG] six times.

```
0020   056
0021   110   ↓
0022   001      1
0023   056
0024   110   ↓
0025   002      2
```

If you want to list program steps continuously, depress and hold the [LIST PROG] key until after the first line is printed. Starting at the current location, the address, code, and symbol (if any) of each instruction in sequence is printed until the [HALT] key is depressed or the RUN/STEP/LOAD switch is moved from the LOAD position. To demonstrate, list the instructions stored, beginning at branch point 2, and verified according to the previous paragraph:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [0] [2].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress and hold the [LIST PROG] key. After the first instruction is printed, release [LIST PROG]. After the sixth line is printed, depress [HALT].

```
0020   056
   1   110   ↓
   2   001      1
   3   056
   4   110   ↓
   5   002      2
```

DETERMINING CURRENT PROGRAM ADDRESS

If at any time you want to know the address currently in the program counter, proceed as follows:

1. Make sure the Idle light is on and not flashing.
2. Set the RUN/STEP/LOAD switch to RUN or STEP.
3. Depress [LIST PROG]. The address, instruction code, and print symbol of the instruction in the location currently specified by the program counter are printed. Repeated use of [LIST PROG] causes the same address, code, and symbol to be printed, in RUN or STEP mode. The contents of the E-register do not change.
4. To execute the program beginning with the current location, depress [RESUME].

TESTING A PROGRAM

If you have verified that your program is correctly stored and you are still getting erroneous results, test the program by executing it, step by step, observing the intermediate results in the printout and comparing the results with a longhand solution of the problem. In step-by-step execution, the program instructions are printed in red.

Step-by-step execution is accomplished by selecting the program's starting location in the usual manner, setting the RUN/STEP/LOAD switch to STEP, and then depressing [RESUME] each time a step is to be executed. In the step mode, the program counter advances by one each time [RESUME] is depressed, instead of advancing automatically after each step. For each step executed in the step mode, the address, code, and print symbol are printed. If execution of that step normally causes printing, the appropriate printout appears on the tape with the PRINT switch up.

As an exercise in program testing, load the sample program shown in figure 5-1 at Branch 00 and then execute the program, step by step, as follows:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH 00] [0] [0].
3. Set the RUN/STEP/LOAD switch to STEP.
4. To execute the Open Parenthesis instruction in step 00, depress [RESUME].
5. To place the constant 5 in the E-register at step 01, depress [RESUME].
6. To execute the Multiply instruction in step 02, depress [RESUME].

```
0000      026     (
     0.0000       (
0001      005       5
0002      023     X
     5.0000       X
```

7. To execute the Halt instruction in step 03, depress RESUME.
8. To enter b, depress 3.
9. To execute the Print X instruction in step 04, depress RESUME.
10. To execute the Close Parenthesis instruction in step 05, depress RESUME.
11. To execute the Plus instruction in step 06, depress RESUME.
12. To execute the Open Parenthesis instruction in step 07, depress RESUME.
13. To place the constant 2 in the E-register at step 08, depress RESUME.
14. To execute the Multiply instruction in step 09, depress RESUME.
15. To execute the Halt instruction in step 10, depress RESUME.
16. To enter c, depress 4.
17. To execute the Print X instruction in step 11, depress RESUME.
18. To execute the Close Parenthesis instruction in step 12, depress RESUME.
19. To execute the Multiply instruction in step 13, depress RESUME.
20. To execute the Halt instruction in step 14, depress RESUME.
21. To enter a, depress 2.
22. To execute the Print X instruction in step 15, depress RESUME.
23. To execute the Divide instruction in step 16, depress RESUME.

```
0003      056
0004      060
     3.0000
0005      027      )
     3.0000        )
    15.0000       *
0006      021      +
    15.0000        +
0007      026     (
    15.0000       (
0008      002      2
0009      023      X
     2.0000        X
0010      056
0011      060
     4.0000
0012      027      )
     4.0000        )
     8.0000       *
0013      023      X
     8.0000        X
0014      056
0015      060
     2.0000
0016      024      ÷
     2.0000        ÷
```

24. To execute the Halt instruction in step 17,

    depress [RESUME].

25. To enter d, depress [5].

26. To execute the Print X instruction in step 18,

    depress [RESUME].

27. To execute the Equals instruction in step 19,

    depress [RESUME].

28. To execute the Print Answer instruction in step 20,

    depress [RESUME].

29. To execute the Branch instruction in step 21,

    depress [RESUME].

30. To specify the first digit of the branch point,

    depress [RESUME].

31. To specify the second digit of the branch point,

    depress [RESUME].

```
0017        056
0018        060
      5•0000
0019        020        =
      5•0000           =
      9•2000           *
0020        061        A
      9•2000           A
0021        127      br
0022        000          0
0023        000          0
```

Notice that the step number and instruction code are printed first (in red), followed by the result of executing the instruction (in black).

An attempted illegal operation causes an error condition. In the error condition, the calculator suspends operations, the keyboard becomes inoperative, and the message ERROR is printed, regardless of the PRINT switch setting. Additionally, the error condition is signaled by the flashing idle light. The error condition may be relieved by depressing the [RESET] or [CLEAR x] key. Use of these keys does not change the program address.

Operations with numbers outside the range of the calculator cause the calculator to go into the overflow condition. In the overflow condition, calculator operation is halted, the keyboard becomes inoperative, and the message OVERFLOW is printed, regardless of the PRINT switch setting. As an additional signal, the idle light flashes. To recover from the overflow condition, depress the [RESET] or [CLEAR x] key. If overflow occurs in a program, the program stops at the instruction that caused the overflow. The [RESET] or [CLEAR x] key used for recovery does not change the program address.

See Error and Overflow in the <u>Operating Instructions Manual</u> for a detailed listing of conditions causing error or overflow.

CHANGING MEMORY CONTENTS

After a program has been loaded into memory, additional instructions may be inserted or existing instructions changed. Two methods can be used to change an instruction in a program stored in memory. One method involves listing the program up to the instruction to be changed and then entering the corrected instruction. The alternative is to list the program until the instruction to be changed is printed, then to use the [BACK SPACE] key to return to the address to be changed. For example, assume that program memory locations 190 through 196 contain the constant 6.01324 and that this number must be changed to 6.01314.

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH] [1] [9].
3. Set the RUN/STEP/LOAD switch to LOAD.

List-Only Method

4. Depress [LIST PROG].
5. Depress [LIST PROG].
6. Depress [LIST PROG].
7. Depress [LIST PROG].
8. Depress [LIST PROG].
9. To load code 001 in step 0195, depress [1].
10. Depress [LIST PROG].

0190	006	6
0191	012	
0192	000	0
0193	001	1
0194	003	3
0195	001	1
0196	004	4

List-and-Backspace Method

4. Depress [LIST PROG].
5. Depress [LIST PROG].
6. Depress [LIST PROG].
7. Depress [LIST PROG].
8. Depress [LIST PROG].
9. Depress [LIST PROG].
10. To correct the above code, depress [BACK SPACE].
11. To load code 001 in step 0195, depress [1].
12. Depress [LIST PROG].

0190	006	6
0191	012	
0192	000	0
0193	001	1
0194	003	3
0195	002	2
0194	003	3
0195	001	1
0196	004	4

You can insert instructions into a program by addressing the nearest branch point with a Branch or Jump instruction and using the [LIST PROG] or the [BACK SPACE] key to increase or reduce the contents of the program counter as required. The [INSERT] key creates a space for the additional instruction. Instructions following the insert are automatically moved forward as necessary. Programs using fixed addresses may be changed by using [INSERT], but branch points and addresses must be renumbered by the programmer, where necessary. Only programs with symbolic addressing may be changed by the following simple insertion procedure. For example, assume that locations 190 to 196 contain the constant 26.8143 and that you wish to insert a 5 between the 1 and 4 so that the constant becomes 26.81543:

1. Set the RUN/STEP/LOAD switch to RUN.
2. Depress [BRANCH ( ) ( )] [1] [9].
3. Set the RUN/STEP/LOAD switch to LOAD.
4. Depress [LIST PROG].
5. Depress [LIST PROG].
6. Depress [LIST PROG].
7. Depress [LIST PROG].
8. Depress [LIST PROG].
9. To create a space after step 0194, depress [INSERT].
10. To fill the space with a 5, depress [5].
11. Depress [LIST PROG].
12. Depress [LIST PROG].

0190	002	2
0191	006	6
0192	012	
0193	010	8
0194	001	1
0194	001	1
0195	005	5
0196	004	4
0197	003	3

The last two instructions have been shifted forward one location (program step). To verify that the number is stored correctly,

13. Set the RUN/STEP/LOAD switch to RUN.
14. Depress [BRANCH ( ) ( )] [1] [9].
15. Set the RUN/STEP/LOAD switch to LOAD.
16. Depress [LIST PROG] eight times.

0190	002	2
0191	006	6
0192	012	
0193	010	8
0194	001	1
0195	005	5
0196	004	4
0197	003	3

To ensure an updated symbol table and proper recording of instruction codes, it is strongly recommended that a program in memory that has been edited be transferred to a magnetic card and then reloaded into memory prior to execution of the edited version.

WRITING ON MAGNETIC CARDS

The Monroe Model 1880 Scientific Calculator has an integral magnetic-card device. This device permits both writing programs and data onto magnetic cards and reading programs and data from magnetic cards. Each card edge may contain up to 256 program instructions or data for up to 32 data registers.

When writing onto a magnetic card, a verify total is generated; this total is basically a summation of the instruction codes or data. This verify total is written onto the card. When a card is read into the calculator, the verify total is recalculated and compared with the total written on the card. If the two do not agree (due to loss of data on the card from scratching or the like) ERROR is printed and the idle light flashes. The error may be cleared and a second READ attempted.

The following paragraphs outline procedures for writing programs or data onto a magnetic card and for reading programs or data from a magnetic card. Remember that program memory is accessed at branch points. Every tenth program step is a branch point. Program memory branch points are accessed with the [BRANCH 00] or [JUMP 00] key, followed by the two numeral keys that correspond to the desired branch point. For branch points 100 through 199, precede the numeral keys with the [•] key. Similarly, branch points 200 through 299 are preceded by the [CHG SIGN] key, and branch points 300 through 399 by the [EXP] key as shown in table 2-1. Branch points 400 through 409 are accessed by using special codes that are explained in the Advanced Programming Reference Manual.

Main data memory registers, on the other hand, are accessed with the [↓00] and [↑00] keys, followed by the appropriate numeral keys for main data memory registers 00 through 99. For main data memory registers 100 through 199, precede the numeral keys with the [•] key. Similarly, main data memory registers 200 through 299 are preceded by the [CHG SIGN] key, registers 300 through 399 by the [EXP] key, and registers 400 through 499 by the [π/e] key. Registers 500 through 511 are accessed by using indirect addressing techniques.

WRITING A PROGRAM ONTO A MAGNETIC CARD

Before you can record a program from memory onto a magnetic card, you must know its address and the number of steps in the program. You can then record the program with the following procedure:

1. Depress [BRANCH 00] and the numeral keys of the branch point where the program begins.
2. Enter the number of steps (N) in the program into the E-register by depressing the numeral keys for the number of steps. If N isn't entered, a full 256 steps will be written onto the card from memory, or 512 steps if both card sides are used.
3. Set the card device switch to WRITE.

4. Insert the A SIDE arrow edge of the magnetic card into the card slot. The card will be pulled into the device and then ejected.
5. If the program has more than 256 instructions (steps), insert the B SIDE arrow edge of the card into the card slot to record the remainder of the program.
6. If the program has more than 512 steps, insert the A SIDE arrow edge of a second card into the card slot. Continue until all the program has been recorded.
7. After the program has been recorded, set the card device switch back to READ unless additional programs are to be recorded.

Remember that if the last step of a program that starts with step 0000 is step 0058, that program contains 59 steps, not 58, since step 0000 must be included.

WRITING DATA ONTO A MAGNETIC CARD

You can record data from the data registers onto magnetic cards in the same manner as you record a program. Before you begin, you must know which data register you want to start recording from and the number of data registers to be read. Then use the following procedure:

1. Depress [↑( )( )] and the numeral keys of the first data register to be read.
2. Enter into the E-register the number of registers (N) to be read by depressing the numeral keys for the number of registers. If N isn't entered, the contents of 32 registers will be written onto the card, or 64 registers, if both card sides are used.
3. Set the card device switch to WRITE.
4. Insert the A SIDE arrow edge of the magnetic card into the card slot. The card will be pulled into the device and then ejected.
5. If more than 32 registers are to be recorded, insert the B SIDE arrow edge of the card into the card slot to record the remainder of the registers.
6. After all the registers have been recorded, reset the card device switch to READ unless additional cards are to be written.

READING MAGNETIC CARDS

Programs or data may be read from magnetic cards into calculator memory. As discussed under writing on magnetic cards, every tenth program step is a branch point. Program memory branch points are accessed with the [BRANCH ( ) ( )] or [JUMP ( ) ( )] key, followed by the two numeral keys that correspond to the desired branch point. For branch points 100 through 199, precede the numeral keys with the [•] key. Similarly, branch points 200 through 299 are preceded by the [CHG SIGN] key, and branch points 300 through 399 by the [EXP] key. Branch points 400 through 409 are accessed by using special codes that are explained in the Advanced Programming Reference Manual.

Main data memory registers, on the other hand, are accessed with the [↓00] and [↑00] keys, followed by the appropriate numeral keys for main data memory registers 00 through 99. For main data memory registers 100 through 199, precede the numeral keys with the [•] key. Similarly, main data memory registers 200 through 299 are preceded by the [CHG SIGN] key, registers 300 through 399 by the [EXP] key, and registers 400 through 499 by the [π e] key. Registers 500 through 511 are accessed by using indirect addressing techniques.

READING A PROGRAM FROM A MAGNETIC CARD

Before you read a magnetic card and load its program into memory, make sure that a program is not already stored in that location of memory. (Once a program has been loaded into memory, it remains there until either a new program is written over it or power is removed from the calculator.) Also, make sure that the number of steps in your program will not run into another program that is already stored and is to remain in the calculator. To determine whether a program is stored at the branch point you are to use, depress [BRANCH 00] and the numeral keys for the branch point you are to use. Then depress [LIST PROG]. If an address and a three-digit code (other than 377) are printed on the tape, a program is stored at that branch point.

If no program is stored at the branch point, read and load your program with the following procedure:

1. Depress [BRANCH 00] and the numeral keys of the branch point you are to use for your program.
2. Set the card device switch to READ.
3. Insert the edge of the card with the A SIDE arrow into the slot on the card device. The card will be pulled into the card device and then ejected. If the other edge of the card is to be loaded, turn the card around and insert the edge with the B SIDE arrow into the card device slot. Load any additional cards the same way.
4. After the cards have been read and loaded, replace the cards in their protective envelopes.

READING DATA FROM A MAGNETIC CARD

To read and load data from a magnetic card into data registers:

1. Depress [↓00] and the numeral keys of the first data register to be used.
2. Set the card device switch to READ.
3. Feed the card into the card device.

APPENDIX A. KEYBOARD CODES

Key	Operation	Code	Print Symbol (If Any)	See Page(s)
ADV	Paper Advance	065		*
↓( )( )	Store in Main Data Memory	120	↓	2-6, 4-17, 5-12 *
↑( )( )	Recall from Main Data Memory	121	↑	2-6, 5-12 *
IND SYMB	Define Symbol	066		4-21
IND SYMB	Indirect Address/Symb Jump, Branch	067		4-17, 4-21
BRANCH ( )( )	Branch to Program Memory	127	Br	1-3, 2-6, 4-3, 4-7, 5-11, 5-12, 5-13 *
JUMP ( )( )	Jump to Program Memory	126	Ju	1-3, 2-6, 4-7, 4-13, 4-22, 5-11, 5-12, 5-13
FLAG	Set Program Flag 1	016		1-3, 4-35
HALT	Halt	056		1-3, 3-6, 5-6 *
PRINT x	Print Entry Register Contents	060		2-1 *
PRINT ANS	Print Answer	061	A	2-1 *
RESET	Reset	062	Λ	2-1, 4-1 *
SET D.P.	Set Decimal Point	117		*
↓( )	Store in Scratch Pad Memory	110	↓	2-1, 4-16 *
↑( )	Recall from Scratch Pad Memory	111	↑	2-3 *
Φ	Special Function	116	Φ	2-1 *
Φ 0	Clear Registers (E, 0, 1, 2, and 3)	116 000	CL	2-1, 4-1 *
Φ 1	Deg/Min/Sec Input	116 001	D,M,S, or G	*
Φ 2	Print Angle	116 002	D,M,S, or G	*
Φ 3	Sum-Square Backout	116 003	− Σ	*
Φ 4	Standard Deviation/Mean**	116 004	SD	*
Φ 5	Integer/Fraction**	116 006	I	*
Φ 6	Factorial	116 006	!	*
Φ 7	Hyperbolic Sine/Cosine**	116 007	Φ 7	*

*See the Model 1880 Scientific Programmable Calculator Operating Instructions manual.

**Latter operation stored in Second Function register, accessed with the 2ND FUNC key.

Key	Operation	Code	Print Symbol (If Any)	See Page(s)
Φ 8	Arc Hyperbolic Sin	116 010	Φ 8	*
Φ 9	Arc Hyperbolic Cosine	116 011	Φ 9	*
$\bar{\Sigma}_0$	Equals Sum-Zero	037	$\bar{\Sigma}_0$	*
CLEAR x	Clear Entry Register	063		2-1, 4-1
π / e	π/e Constants**	015		2-3, 2-6, 5-12 *
EXP	Exponent	014		2-3, 2-6, 2-7, 5-12 *
RESUME	Resume	057		1-3, 4-3, 4-7, 4-19, 5-7 *
0 - 9	Numeral Keys	000-011		*
•	Decimal Point	012		2-3, 2-6, 2-7, 4-16, 5-12, 5-13 *
CHG SIGN	Change Sign	013		2-3, 2-6, 2-7, 4-25, 5-12, 5-13 *
(	Left Parenthesis	026	(	*
)	Right Parenthesis	027	)	*
−	Minus	022	−	4-45 *
÷	Divide	024	÷	*
+	Plus	021	+	4-45 *
×	Multiply	023	X	4-29 *
=	Equals	020	=	4-45 *
$\frac{1}{x}$	Invert	054	$1/x$	*
a^x	Raise to Power	025	a^x	*
∠	To Polar Coordinates	031	∠	*
⌟	To Rectangular Coordinates	030	⌟	*
√	Square Root	055	√	*
SIN COS	Sine/Cosine**	070	S	*
SIN^{-1} COS^{-1}	Arc Sine/Arc Cosine**	071	$\bar{S}$	*
R→°	Radians to Degrees	072	R°	*

*See the Model 1880 Scientific Programmable Calculator Operating Instructions manual.

**Latter operation stored in Second Function register, accessed with the 2ND FUNC key.

Key	Operations	Code	Print Symbol (If Any)	See Page(s)
2ND FUNC	Second Function	052	F 2	*
Ln LOG	Logarithm, base e/base 10**	050	ℓg	*
e^x 10^x	Antilogarithm, base e/base 10**	051	ℓg'	*
Σ^2_n	Sum-Square	047	Σ	*

*See the Model 1880 Scientific Programmable Calculator Operating Instructions manual.

**Latter operation stored in Second Function register, accessed with the 2ND FUNC key.

APPENDIX B. NON-KEYBOARD CODES

Operation	Code	Print Symbol (If Any)	See Page(s)
Add to Main Data Memory	123	+	3-1
Exchange Main Data Memory	122	↕	3-1
Add Scratch Pad Memory	113	+	3-2
Exchange Scratch Pad Memory	112	↕	3-2
Total Scratch Pad Memory	114	✱	3-2
Tangent	073	t	3-3
Arc Tangent (Arctan)	103		3-3
Square	053	x^2	3-3
Integer/Fraction	044	I	*
Absolute Value	045	$\|x\|$	3-3
Add (Accumulator Register)	041	+	3-3
Subtract (Accumulator Register)	042	−	3-3
Subtotal (Accumulator Register)	043	◇	3-4
Total (Accumulator Register)	040	✱	3-4
Increment Entry	151		3-4
Decrement Entry	152		3-4
Print Enable	155		3-4, 4-1
Print Disable	154		3-4
Recall Decimal Point	157		3-4
Set Program Flag 1	016		3-5, 4-1
Set Program Flag 2	017		3-5, 4-1
Reset Program Flag 1	166		3-5, 4-1
Reset Program Flag 2	167		3-5, 4-1
Dot Print	176		3-5
Identifier	177		3-5
No Operation (NOOP)	377		4-1, 4-7, *

*See the Model 1880 Scientific Programmable Calculator Operating Instructions manual.

APPENDIX C. KEYBOARD AND NON-KEYBOARD CODES, NUMERICAL SEQUENCE

Code	Operation	Key
000	Numeral Zero	0
001	Numeral One	1
002	Numeral Two	2
003	Numeral Three	3
004	Numeral Four	4
005	Numeral Five	5
006	Numeral Six	6
007	Numeral Seven	7
010	Numeral Eight	8
011	Numeral Nine	9
012	Decimal Point	•
013	Change Sign	CHG SIGN
014	Exponent	EXP
015	π/e Constants	$\frac{\pi}{e}$
016	Set Program Flag 1	FLAG
017	Set Program Flag 2	none
020	Equals	=
021	Plus	+
022	Minus	−
023	Multiply	×
024	Divide	÷
025	Raise to Power	a^x
026	Left Parenthesis	(
027	Right Parenthesis	)
030	To Rectangular Coordinates	⌟
031	To Polar Coordinates	∡
037	Equals Sum-Zero	$\bar{\Sigma}_0$

Code	Operation	Key
040	Total (Accumulator Register)	none
041	Add (Accumulator Register)	none
042	Subtract (Accumulator Register)	none
043	Subtotal (Accumulator Register)	none
044	Integer/Fraction	none
045	Absolute Value	none
047	Sum-Square	Σ_x^2
050	Logarithm, base e/base 10	Ln LOG
051	Antilogarithm, base e/base 10	e^x 10^x
052	Second Function	2ND FUNC
053	Square	none
054	Invert	$\frac{1}{x}$
055	Square Root	$\sqrt{\ }$
056	Halt	HALT
057	Resume	RESUME
060	Print Entry Register Contents	PRINT x
061	Print Answer	PRINT ANS
062	Reset	RESET
063	Clear Entry Register	CLEAR x
065	Paper Advance	ADV
066	Define Symbol	IND | SYMB
067	Indirect Address/Symbolic Jump or Branch	IND | SYMB
070	Sine/Cosine	SIN COS
071	Arc Sine/Arc Cosine	SIN^{-1} COS^{-1}
072	Radians to Degrees	R→°
073	Tangent	none
103	Arc Tangent	none
110	Store in Scratch Pad Memory	↓( )
111	Recall from Scratch Pad Memory	↑( )
112	Exchange Scratch Pad Memory	none

Code	Operation	Key
113	Add Scratch Pad Memory	none
114	Total Scratch Pad Memory	none
116	Special Function (in Conjunction with a Numeral Key)	Φ
117	Set Decimal Point	SET D.P.
120	Store in Main Data Memory	↓( )( )
121	Recall from Main Data Memory	↑( )( )
122	Exchange Main Data Memory	none
123	Add to Main Data Memory	none
126	Jump to Program Memory	JUMP ( ) ( )
127	Branch to Program Memory	BRANCH ( ) ( )
151	Increment Entry	none
152	Decrement Entry	none
154	Print Disable	none
155	Print Enable	none
157	Recall Decimal Point	none
166	Reset Program Flag 1	none
167	Reset Program Flag 2	none
176	Dot Print	none
177	Identifier	none
377	NOOP (No Operation)	none