OPERATING and PROGRAMMING MANUAL

HEWLETT-PACKARD 9830A CALCULATOR

9830A CALCULATOR SHOWN WITH 9866A PRINTER

HEWLETT–PACKARD CALCULATOR PRODUCTS DIVISION
P.O. Box 301, Loveland, Colorado 80537, Tel. (303) 667-5000
(For World-wide Sales and Service Offices see rear of manual.)

MANUAL SUMMARY

CHAPTER 1: GENERAL OPERATING INFORMATION

To familiarize the user with the Model 30, various topics are introduced.

CHAPTER 2: USING THE MODEL 30 KEYBOARD

The capabilities of the Model 30, strictly as a calculator, are described. Editing capabilities are presented. Mathematical topics include variables, trigonometric operations, and logical evaluation.

CHAPTER 3: PROGRAMMING AND PROGRAMMING STATEMENTS

The programming language, BASIC, is presented. Programming statements are individually discussed. Examples, using particular statements, are given.

CHAPTER 4: PROGRAMMING-RELATED INFORMATION

Program viewing, editing, and debugging are discussed individually. Useful keys and commands, related to these topics, are presented. The Model 30 memory is described.

CHAPTER 5: USING A TAPE CASSETTE

Tape cassettes are described. Each tape cassette command is individually discussed.

CHAPTER 6: SPECIAL FUNCTION KEYS

The uses of these keys as typing-aids, as functions, and as programs are discussed.

APPENDIX A: CALCULATOR AND PRINTER INSTALLATION PROCEDURE

Initial turn-on and inspection procedures are presented. *This appendix should be read immediately upon arrival of the calculator and printer.*

APPENDIX B: GENERAL OWNER'S INFORMATION

Equipment supplied, service information, etc. are all described.

APPENDIX C: ROM OPTIONS

Available ROM's are described and a general installation procedure is given.

APPENDIX D: MODEL 60 CARD READER

For those who have purchased this accessory, a general description is given.

APPENDIX E: ERROR CODES

The error codes and messages are listed along with some additional explanations.

APPENDIX F: PRINTER OPERATING PROCEDURES

Printing commands and examples are given. A table, which shows the ASCII characters that can be output using the FORMAT B statement, is included.

PREFACE

This book is organized both:

- With a logical arrangement of topics so that it can be read straight through, if desired; and
- As a reference guide, with major topics 'tabbed' at the right side of appropriate pages.

The quick index to the right of this page separates the tabbed information by chapter.

It is expected that the tabbed material will be most beneficial as a reference guide to Chapter 3, where each program statement is tabbed, and to Chapter 5, where each tape command is tabbed.

Wherever possible, major topics are self-contained; that is, it's not necessary to read an entire chapter to extract one idea.

However, since Chapter 1 discusses the special features of the calculator, you should understand the material in this chapter before going on to isolated topics.

TABLE OF CONTENTS

CHAPTERS

CHAPTER 3: PROGRAMMING AND PROGRAMMING STATEMENTS

(Continued)

CHAPTERS

CHAPTER 6: SPECIAL FUNCTION KEYS

(Continued)

APPENDICES

FIGURES

TABLES

CAUTION

The inspection procedure for the Model 30 is presented in Appendix A of this book. Please refer there:

- If the calculator and printer have not been inspected, or
- If there is any doubt regarding the compatibility of the calculator power requirements to the voltage in your area.

Otherwise, the calculator and printer could be severely damaged.

Chapter 1

GENERAL OPERATING INFORMATION

It is assumed here that the Model 30 is set up to operate in your area – if there is any doubt whatsoever, please refer to Appendix A of this book.

This chapter contains background material for the remaining chapters in this book.

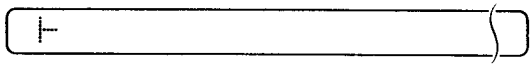
PREPARING THE CALCULATOR FOR USE

There are a few things you should check each time you plan to use the calculator.

If the calculator is turned off:

- Set the OFF/ON switch to the ON position. 

- When the following display appears, the calculator is ready to operate:



- If the display remains blank, refer to Appendix A for plug-in instructions.

If the calculator is turned on and the display is blank:

- Hold the STOP key down until STOP appears on the display.
- If the display remains blank, turn the calculator off, then a few seconds later, turn it on again.
- If the display still remains blank, refer to Appendix A for plug-in instructions.

If the calculator is turned on and the display is not blank:

- If you wish to do calculations, just follow the arithmetic rules discussed in Chapter 2.
- If you wish to do programming, first check with other users to ensure that nothing important is in memory (memory is the term used to denote the area in which a user can put pertinent information – such as a program); if there is no need to retain the information in memory, turn the calculator off, then a few seconds later, on again. This procedure erases whatever was previously in memory.

CHARACTERISTICS OF THE MODEL 30

THE KEYBOARD

A representation of the Model 30 keyboard is shown in Figure 1-1. The keyboard is divided into specified areas as described below.

- Alphanumeric Keys – This area acts very much like a standard typewriter keyboard. If, for instance, you want to display the dollar sign, $, you must have the SHIFT key held down when you press (4). It differs from a standard typewriter in that letters always appear in upper case on the display.
- Numeric Keys – This area conveniently locates a set of numbers to the left of the five arithmetic keys. If, however, you feel more comfortable using the other set of numbers (in the alphanumeric region), go right ahead. It makes no difference to the calculator.
- Arithmetic Keys – The standard arithmetic operators, addition, subtraction, multiplication, division, and exponentiation, are located here.
- Special Function Keys – The keys in the upper left-hand region of the keyboard, f_0 through f_9, add considerable flexibility to the calculator. These keys are discussed in Chapter 6.
- The rest of the keys in the upper half of the keyboard are helpful in a variety of ways. Some are especially useful as editing tools, while others have more specific uses. The keys are described, in appropriate places, throughout this book.

Keys that are the same color have similar operating characteristics. As you become more familiar with the Model 30, these similarities will become evident.

Here are a few more topics related to keyboard operations:

- Spacing – In general it makes no difference what spacing you use either in calculating or in programming; for example, 4+6 or 4 + 6 are interpreted the same by the calculator. There are, however, certain instances when you can set a specific spacing for outputting purposes, say, in PRINT and DISP statements, which are discussed in Chapter 3.
- Repetition of Operations – Most keys, when held down for about two seconds, rapidly repeat their operation. With certain editing keys, this is a particularly useful feature.
- Upper and Lower Case Alphabetics – The display on the calculator shows letters only in upper case. But if you have a printer with lower case capabilities, information can be printed that way; more about that later when the PRINT statement is discussed.
- @ Symbol – If you ever need this symbol, you can obtain it by pressing the RESULT key when the SHIFT key is depressed.
- ⊢ Symbol – When the display is clear and awaiting inputs, '⊢' is located in the left-hand corner of the display; the first character then keyed in replaces this symbol in the display. '⊢' also appears at the end of a program line that is returned to the display from memory.

THE DISPLAY

Up to 32 characters can appear on the display at any one time.

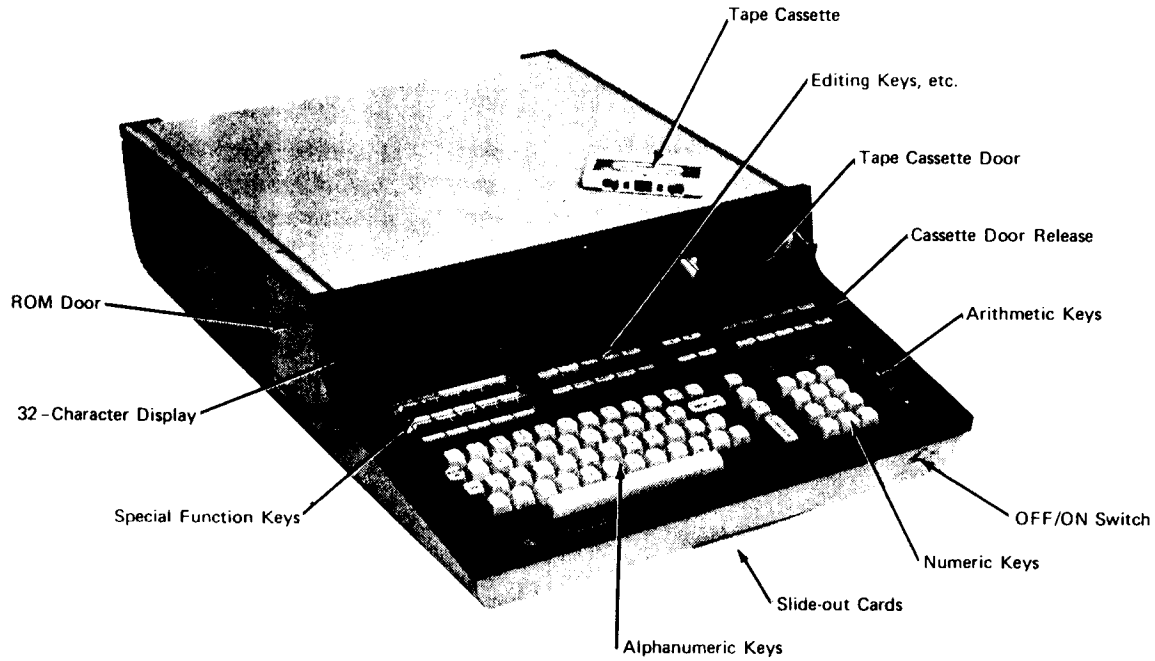


Figure 1-1. The Model 30 Calculator

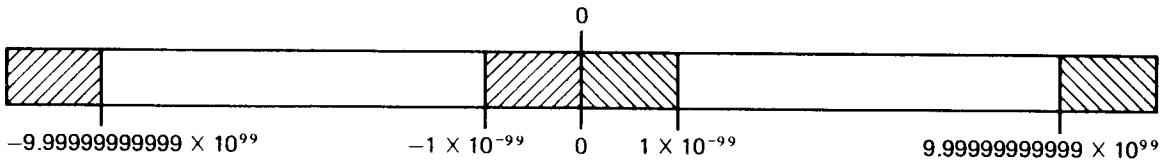
LINE LENGTH

Even though only 32 characters can be displayed at any one time, up to 80 characters can be keyed into the same expression. After the 32nd character is keyed in, additional characters that are keyed in cause the display region to be shifted to the left. After the 72nd character is keyed in, a soft beep occurs, informing the user that only eight more characters can be keyed in.

Any portion of the line can be reviewed on the display as you will see in Chapter 2.

RANGE

The calculating range of the Model 30 is $-9.99999999999 \times 10^{99}$ through -1×10^{-99}, 0, and 1×10^{-99} through $9.99999999999 \times 10^{99}$. The pictorial representation below shows an exaggerated representation of the non-defined (cross hatched) areas.



MEMORY

The calculator has an allocated amount of user memory. The amount of memory is expressed in 'words', which is a computer-oriented measuring stick. By keying in a few programs and seeing how much (or how little) memory is used by these programs, you will soon have a good feel for just how large a program you can input. (For the computer oriented, you'll be interested to know that each word is made up of 16 bits – two bytes.) The calculator memory is discussed in Chapter 4.

CHARACTERISTICS OF THE MODEL 30
(continued)

ERROR MESSAGES

Occasionally the word, ERROR, followed by a number, will appear on the display at the same time that the calculator makes a soft beeping sound. This indicates that an error has occurred. The number references an error message that helps to pinpoint the cause of the error.

If an error message appears during an attempt to run a program, the program line in which the error occurs will be referenced, also.

A complete listing of the error messages appears both on a slide-out card underneath the Model 30 and in Appendix E of this book. An explanation of recoverable versus non-recoverable errors is given in Appendix E.

A PREVIEW OF THE KEYS

CLEAR

In general, when the CLEAR key is pressed, whatever was previously on the display will be erased; but the following exceptions may occur:

- If, when the CLEAR key is pressed, the following display appears:

  KEY-

  the Special Function mode has somehow been previously accessed (see Chapter 6). To get out of this mode and to clear the display, press:

  END

- If the display 'blanks out' while the calculator is turned on, hold the STOP key down until STOP appears on the display; then you can press the CLEAR key and continue inputting. If your display continues to go blank, you may find that your printer is not properly attached to the calculator; in this case, see Appendix A.
- If you can't locate your problem and yet you seem to have no control over the keyboard, turn the calculator off and then, in a few seconds, on again. (This procedure may be the last resort in some cases, though; for if you had previously put something into the user's memory area, it is erased by turning the Model 30 off.)

EXECUTE

Almost every sequence of keys that is pressed to accomplish a goal needs to be completed with the EXECUTE key; when this key is pressed, the previous instructions to the calculator are performed. One major exception to this rule comes about when inputting program lines to memory (see below). Other exceptions are keys, such as PRT ALL and RECALL, which are immediately executed when pressed.

END OF LINE

At the completion of each program line that is keyed in, the END OF LINE key must be pressed to put the information in memory.

Chapter 2

USING THE MODEL 30 KEYBOARD

Most keys, when pressed, cause the character or characters represented on the particular keys to be displayed. There are, of course, exceptions. As you learn about the various keys, it will become apparent why a particular key acts as it does. Three keys have already been discussed:

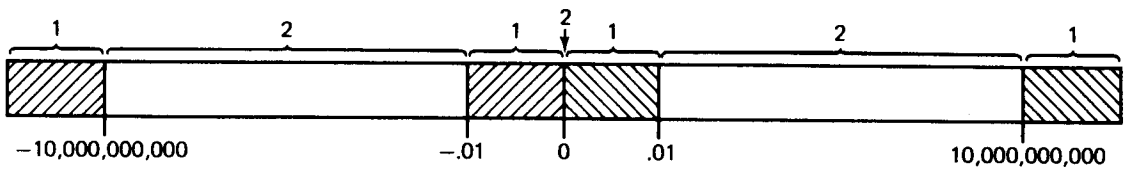
- CLEAR – which erases the display;
- EXECUTE – which performs the operation indicated in the display;
- END OF LINE – which puts program lines into memory.

In this chapter, several other keys and operations will be discussed. Non-programming (calculator mode) applications are emphasized. Many of the tools discussed, however, can also be used in the 'programming mode' as shown in Chapter 4.

THE FORMAT KEYS

When the calculator is turned on, it assumes a 'standard' operating form. The results of arithmetic calculations then appear in one of two ways, depending on the size of the number, as shown in the following representation:

1. The form of numbers in this range is ±Z.ZZZZZE±ZZ.
2. Numbers in this range appear as commonly written.



For both the very small and the very large numbers, a scientific (or floating point) notation is used, where ±Z.ZZZZZE±ZZ corresponds to the mathematical representation, $\pm Z.ZZZZZ \times 10^{\pm ZZ}$. This representation will be discussed in greater detail with the subject matter to follow shortly – FLOAT N.

STD

As mentioned, the calculator initially assumes a 'standard' operating mode. But the operating mode can be changed either to 'fixed' or to 'floating' point (as discussed next). The standard mode can then be returned if the STD key is pressed.

If you need specified output formats, either fixed or floating-point notation should be used. But for normal calculations, the standard mode is often most convenient since the outputs of calculations appear in an easily readable manner.

(continued)

THE FORMAT KEYS
(continued)

Here are the rules that outputs in standard mode follow:

- No more than 10 digits total can be output, with no more than 9 digits to the right of the decimal point; for example:

  9876543210.6 would be output as 9876543211 (notice the tenth digit is rounded); and

  .0123456789 would be output as 0.012345679 (at least one digit is always displayed to the left of the decimal point; so, in this case, the ninth digit to the right of the decimal point is rounded from 8 to 9).

- Excess zeros to the right of the decimal point are suppressed; for example:

  32.1000 would be output as 32.1; but

  32.111199999 would be rounded to 32.11120000 (in this case, the zeros are output since they were the result of rounding).

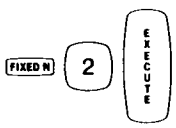
- As shown on the pictorial on page 2-1, numbers outside the range [−10,000,000,000 through −.01, 0, +.01 through +10,000,000,000] are output in a scientific notation. The form is the same as float-5 notation.

Here are some numbers and their output form when standard notation is used:

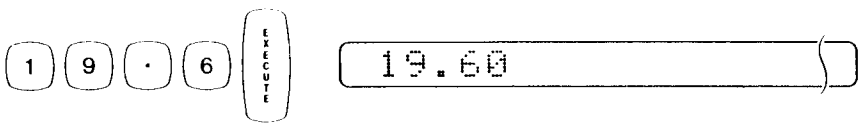
	Standard Output
12.03	12.03
18.7654	18.7654
15	15
−832.600	−832.6
987654321.234	987654321.2
123456789123.0	1.23457E+11
−.0004	−4.00000E−04

FIXED N

Fixed-point notation is commonly used for a variety of problems. In 'fixed-point' the user can specify the number of digits to appear to the right of the decimal point; in payroll problems for instance, two digits to the right of the decimal point would be required – hence 'fixed-2 notation'. This notation can be obtained if you press:



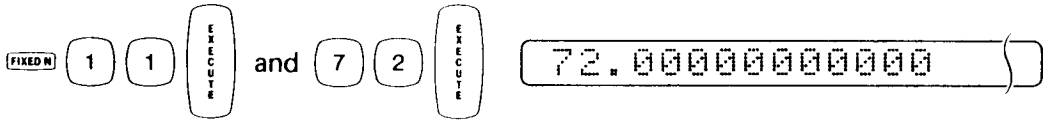
Now the results of calculations will appear with two digits to the right of the decimal point. For instance, if you press:



The results of other calculations now will appear with two digits to the right of the decimal point, also. However, the result of a calculation could be too large to appear in fixed-point form. In such circumstances, the calculator reverts to floating-point (in our case, it would revert to float-2). Floating-point notation is described at the conclusion of the fixed-point discussion.

Fixed-0 through fixed-11 notation can be specified. In each case the numeric indicates the number of digits that will appear to the right of the decimal point. (However, in fixed-0, since there are no digits to the right of the decimal point, the decimal point is omitted.)

Here's another example. Press:



Here are some numbers and their output form if, say, fixed-3 notation is used:

	Fixed-3 Output
18	18.000
.000006	0.000
−2.7531	−2.753
4.56789	4.568
1234567891234.5	1.235E+12†

† *If fixed-point notation is specified and the result of a calculation has more than 12 digits to the left of the decimal point, the calculator temporarily reverts to floating-point. Notice, however, that in this example, three digits are retained to the right of the decimal point to parallel the desired fixed-point notation.*

Occasionally a result, slightly different from the expected, may appear on the display; the two topics following the floating-point discussion – Significant Digits and Rounding – explain these situations.

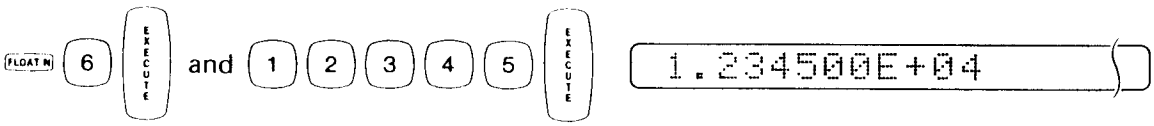
FLOAT N

Floating-point (often called scientific notation) is a convenient form when the results of calculations are either very large or very small. Its form is as follows:

$$\pm Z.Z \cdots ZE\pm ZZ$$

- The first non-zero digit of a number is the first digit displayed (if the number is negative, a minus sign precedes this digit);
- A decimal point follows the first digit;
- Some digits follow the decimal point; the amount is determined by the specified floating-point form (e.g., in float-5, five digits would follow the decimal point);
- Then the character 'E' appears, followed by a sign and two digits – this is the exponent, representing a positive or negative power of ten, which indicates the direction and the number of places that the decimal point would have to be moved to express the number in fixed-point form.

Here's an example. Press:



If the sign following the E is positive (as in this example), then the decimal point can be moved to the right the number of places indicated by the exponent (in this case, 4) to express the number in fixed-point form.

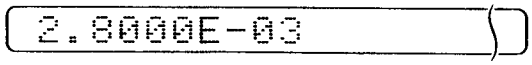
(continued)

THE FORMAT KEYS

(continued)

Here's another example. Press:

FLOAT N 4 EXECUTE and · 0 0 2 8 EXECUTE



Since the sign of the exponent in this example is minus, the decimal point can be moved to the left three places (as indicated by the exponent '03') to express the number in fixed-point form.

Float-Ø through float-11 can all be specified. In each case, the numeric indicates the number of digits that will appear immediately to the right of the decimal point. (However, in float-Ø, since there are no digits to the right of the decimal point, the decimal point is omitted.)

Here are some numbers and their output form if, say, float-2 notation is used:

	Float-2 Output
−3.2	−3.20E+00
271	2.71E+02
26.3777	2.64E+01
.0004	4.00E−04
2.4E78†	2.40E+78

† *This entry could be keyed in by using either of the following two keys:*

ENTER EXP *or the alphabetic key* E

In numerical inputs, these two keys are interpreted as a floating-point input. For instance, the previous example can be keyed in and executed as:

2 · 4 ENTER EXP 7 8

For the number, 6000, the following entries would all be equivalent:

6 0 0 0 or 6 E 3 or 6 0 ENTER EXP 2

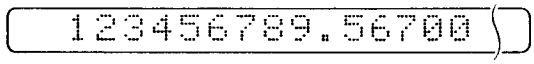
When using ENTER EXP or E , be sure to first enter a number, since 1 is not assumed. Keying in E without a preceding number results in error 40.

SIGNIFICANT DIGITS

The number of digits output in a calculation depends on the notation that is used. However, regardless of the notation, the Model 30 internally retains 12 significant digits for the output; that is, it calculates to 12-point accuracy.

The 12-point accuracy can always be displayed on outputs if either fixed-11 or float-11 notation is specified. But in any notation, if more than twelve digits are to be output, all digits after the twelfth are truncated and replaced by zeros; for example, in fixed-5 notation, when executed, the number 123456789.56789 (having 14 digits) would be

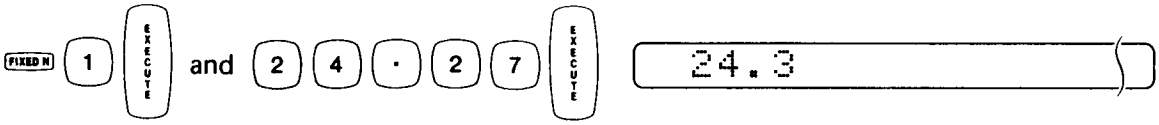
displayed as:



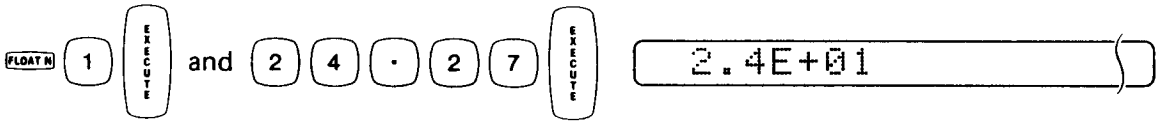
Notice that the last two digits of the number (8 and 9) are ignored and zeros are displayed in their place – since the '7' preceding them was the twelfth significant digit.

ROUNDING

If the result of a calculation has fewer than thirteen digits, it is rounded if there are more digits to the right of the decimal point than the specified notation will allow. The rounding is performed as follows: the first excess digit is checked; if its value is 5 or above, the digit immediately preceding it is incremented (rounded-up) by one. For example, press:



The '2' is rounded to '3' since it is the last allowable digit to be displayed in this notation and since the digit following it is a '7'. Now key in:



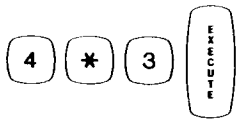
In this example, the digit checked is the one following the 4. Since its value (2) is less than 5, it, together with the following digits, is truncated (rounded-down).

ARITHMETIC

The five basic numerical operations and their respective symbols are as follows: addition (+), subtraction (–), multiplication (*), division (/), and exponentiation (↑).

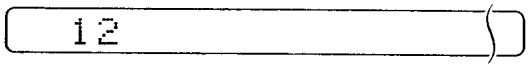
CALCULATING

To perform basic calculations, such as 4 multiplied by 3, simply do the following. Press:

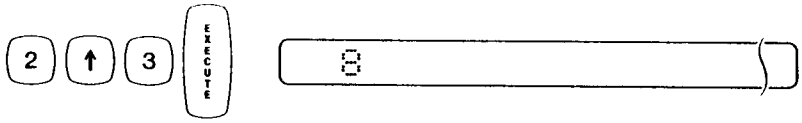


The EXECUTE key must be pressed to evaluate any keyed-in expression.

In standard notation, the following result appears on the display:

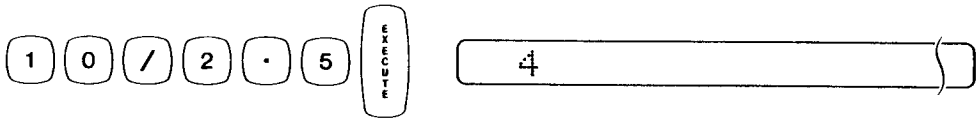


To raise 2 to the third power (2^3), press:



Notice, however, that 2^{-3} must be keyed in as 2↑(-3); that is, parentheses are required to separate the minus sign from the exponentiation symbol.

To divide 10 by 2.5, press:



ARITHMETIC HIERARCHY

In a mathematical expression (which is a sequence of numbers separated by numeric symbols), there is a prescribed order of execution; the order of execution, called the hierarchy, is the same as the order commonly used in standard arithmetic; it is:

1. ↑ highest precedence
2. * /
3. + – lowest precedence

The order of execution is from highest precedence to lowest precedence.

When an expression contains two or more symbols at the same level in the hierarchy, the order of execution is from left to right.

The prescribed order of execution can be altered if parentheses are used; operations within parentheses are performed prior to operations outside parentheses regardless of the hierarchy. Furthermore, parentheses can be nested (i.e., parentheses within parentheses are allowed); when nesting occurs, the 'deepest' nest is calculated first; e.g., in (5*(4–2)), the portion (4–2) is simplified first.

In the following two examples, the internal step-by-step procedure that the calculator goes through, when executing an expression, is presented. By being able to predict the order of execution in the calculator, you will be capable of keying in quite complex expressions. Here is the expression:

4+6/2*(3+7)

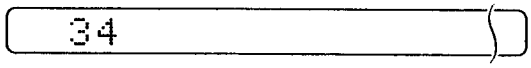
1. First, the operation inside the parentheses is evaluated, thus reducing the expression to:

   4+6/2*10

2. Then the calculator looks for the symbol with the highest precedence; since both / and * have the same precedence, the calculator evaluates these two operations from left to right:

   4+3*10
   4+30

3. Finally, the addition (having the lowest precedence) is performed; the result is then displayed in whatever format was previously specified. In standard notation it would appear as:



Fortunately, when the EXECUTE key is pressed, the expression is evaluated considerably faster than the previous explanation might imply. Here's an expression that is somewhat more complex:

2*3↑3+4*((5−3)/2+1)

1. Taken step-by-step, this equation also is easily simplified; first, the expression in the most deeply nested parentheses, (5−3), is evaluated:

   2*3↑3+4*(2/2+1)

2. Then the expression in the remaining parentheses is simplified, and since division is above addition in the hierarchy, it is performed first:

   2*3↑3+4*(1+1)

3. These parentheses are then totally eliminated:

   2*3↑3+4*2

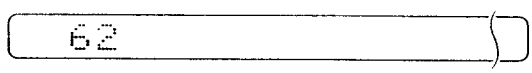
4. Now the operation with the highest precedence in the hierarchy, exponentiation, is performed; this reduces the expression to:

   2*27+4*2

5. The two multiplication operations can then be evaluated:

   54+8

6. So if the original expression is keyed in and the EXECUTE key is pressed, the result will appear on the display; in standard notation it will appear as:



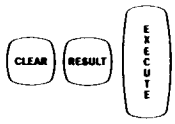
NOTE

Using parentheses for 'implied' multiplication is not allowed; hence, 4(5−2) must appear as 4*(5−2).

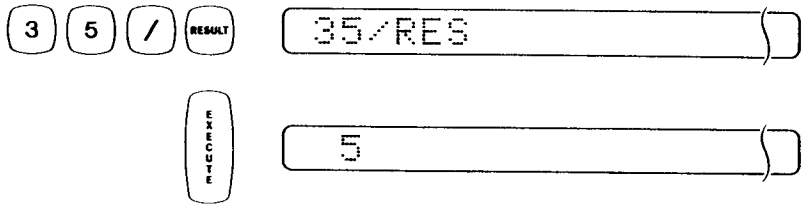
ARITHMETIC
(continued)

An extremely useful key, when performing arithmetic operations, is the RESULT key. It has two specific applications:

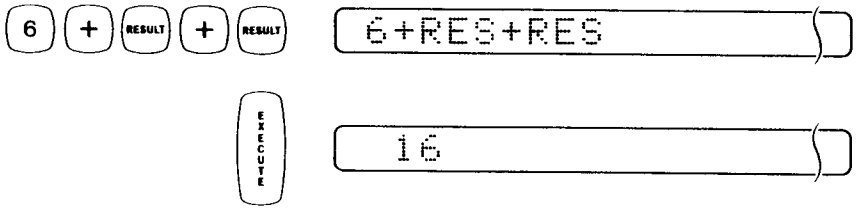
- It can be used to display the numerical value of the last arithmetic statement that was executed. Just press:



- It can be pressed during arithmetic operations, allowing you to use the result of the last calculation in the current expression. For example, by executing 3+4, 7 is displayed; then by pressing:



The previous result of 7 was divided into 35, leaving a current result of 5. Now by keying in:



The previous result (this time, 5) was added twice to the number 6.

VARIABLES

Often it is convenient to assign values to letters and then use these letters in expressions. In programs a letter can have its value constantly updated – hence, the term 'variable'. In some cases a letter (variable) is used as an intermediate storage device; for instance, to keep the current summation of a group of numbers being added together. Other times, a variable in a particular formula may have several different values inserted into the formula to determine what effect these values have on the result. A letter can also be used for convenience; just as π is used to designate the value 3.14159265. . . , an equally cumbersome number that you are using over and over can be designated by a letter.

SIMPLE VARIABLES

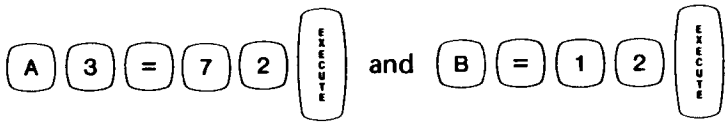
On the Model 30 the following simple variables are allowed:

- Any letter – from A through Z.
- Any letter immediately followed by a digit – from Ø through 9.

For instance, acceptable simple variables are: B3, G, HØ, M, M9, Y.

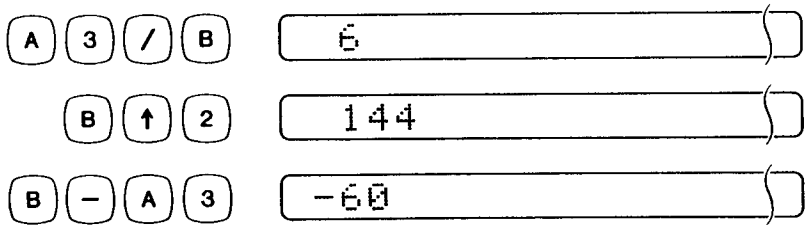
In all, 286 simple variables are allowed.

In either the calculator or programming mode†, simple variables can be assigned values. Below, two variables are assigned values. Press:



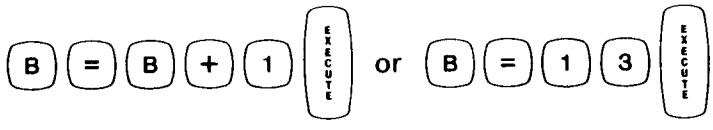
In the assignment statement, the variable name appears first, followed by the equals sign, and then the value assigned to the variable.

The variables A3 and B now are assigned the values 72 and 12, respectively. If we press the keys shown below and then press the EXECUTE key, in standard notation the following displays will appear:



In these examples, the variable names are keyed into the arithmetic expressions and when the expressions are executed, the previously assigned values of the variables are used. So the first example is really just 72/12.

Variables can be reassigned values, either indirectly or directly; for instance, to reassign B equal to 13, press either:



Then in any additional arithmetic expression, B is equal to 13.

ARRAY VARIABLES

Another type of variable is the array variable. To use array variables in the calculator mode, you must 'initialize' the calculator as discussed on page 4-14. Arrays become quite useful in the programming mode, especially when manipulating matrices. Array variables can be either one or two dimensional; for instance:

- A(3) is a one-dimensional array, containing up to three specific items – A(1), A(2), and A(3);
- A(3,3) is a two-dimensional array, containing up to nine specific items – A(1,1), A(1,2), A(1,3), A(2,1), A(2,2), A(2,3), A(3,1), A(3,2), A(3,3).

(continued)

† The 'calculator mode' refers to non-programming operations whereas the 'programming mode' refers to programming operations.

VARIABLES
(continued)

On the Model 30 the following array variables are allowed:

- Any letter – from A through Z – followed by
- Either one or two numbers enclosed within parentheses (where 256 is the maximum size of either number).

When two numbers are enclosed within the parentheses, they must be separated by commas.

Arrays are discussed more completely on page 3-36.

EDITING IN CALCULATOR MODE

In the 'calculator mode' (that is, for non-programming operations), several editing keys are available. But before these keys are discussed, a brief introduction to the operating characteristics of the display is needed.

The calculator, by necessity, keeps track of where it is located on the display with a mechanism, which we will refer to as a 'pointer'. The pointer determines the location of the next character to be keyed in.

During normal keyboard operations, with characters being keyed into the display from left to right, the pointer isn't visible to the user. But, when the normal left to right sequence is altered, the pointer becomes visible. For, when the pointer is located at a previously keyed-in character, the character space flashes alternately with the character that's there.

The editing keys are as follows:

- BACK which moves the pointer one character space to the left each time it's pressed (until the leftmost portion of the display is accessed).
- FORWARD which moves the visible pointer one character space to the right each time it's pressed (until the pointer becomes invisible).
- INSERT which opens up a character space immediately to the left of the visible pointer.
- SHIFT INSERT which deletes the character space where the visible pointer is located (SHIFT must be held down when INSERT is pressed).

Each of these keys, when held down for about two seconds, will rapidly repeat its operation.

Here's an example using these editing keys:

Press the keys:

1 2 3 3 4 5 6 7 8 9

Now let's add the two quantities:

```
 −1234
+56789
```

So we have to add a minus sign in front of the 1, delete one of the 3's and put a plus sign in front of the 5.

a) First let's locate the pointer at the leftmost character space. Press [BACK] and keep it held down until the pointer is located at the 1. (Since the pointer cannot move past this location, it must stop there.)

b) Now press [INSERT]. A character space opens up to the left of the 1. The minus sign can then be keyed in.

```
-1233456789
 ^
```
†

c) Now press [FORWARD] twice. This locates the pointer at the first 3, which can now be deleted by pressing the keys – SHIFT INSERT. This deletes both the 3 and the character space. (With the pointer located at the first 3, the space bar could have been pressed to delete the 3; however, the blank character space would have remained.)

```
-123456789
   ^
```

d) The pointer is now located at the remaining 3. To put a plus sign between the 4 and 5, it is necessary to press [FORWARD] twice; this locates the pointer at the 5. Then press [INSERT]. A space is opened up to the left of the 5. The plus sign can then be keyed in.

```
-1234+56789
     ^
```

e) The expression can be immediately executed regardless of the pointer's position.

[EXECUTE]

```
55555
```


ADDITIONAL KEYS

In the calculator mode, the following keys also can be used:

- [→] which, for displays of greater than 32 characters, allows you to review the beginning of the display.
- [←] which, for displays of greater than 32 characters, allows you to review the end of the display.
- [RECALL] which allows you to review the previous expression or key sequence that was executed.
- [PRT ALL] which allows you to keep a printed record of whatever you're doing.
- [SCRATCH] which erases memory.

In the following paragraphs, these keys are discussed in detail.

(continued)

† The arrow shows the position of the blinking indicator.

ADDITIONAL KEYS

(continued)

← →

With these two keys, an input of up to 80 characters can be viewed on the 32-character display.

Key in the following expression, which repeats each numerical digit eight times:

11111111+22222222+33333333+44444444+55555555+66666666+77777777+88888888+99999999

NOTE 1 NOTE 2

Note 1:

Notice that when the fifth '4' is keyed in, the 32-character display has been filled. Each additional character then pushes the display region one character space to the left, thus eliminating the leftmost character from the display.

Note 2:

Notice also that when the last '+' sign is keyed in, the calculator makes a soft beeping sound, informing the user that 72 of the allowable 80 characters have been keyed in.

After the last '9' has been keyed in, the display looks like this:

```
66666+77777777+88888888+99999999
```

To view any other portion of the expression, hold [→] (the right arrow) down for a few seconds; the display region is rapidly pushed to the right. When the portion of the display that you want to see is visible, just release this key. If you are at the end of the display and you want to view the beginning, just hold [→] down for about five seconds and the beginning of the display should be visible; it doesn't matter if you hold the key down longer because once the leftmost 32 characters are displayed, [→] has no more effect.

To view the end of the display, you can use [←] (the left arrow). It performs the reverse operation of the right arrow. When [←] is pressed, the display region is pushed to the left.

Of course, either key can be used to shift the display region one character space at a time. So to push the display region three spaces to the right, just press [→] three times.

In all cases after the eightieth character has been keyed in, it is impossible to input an eighty-first character. But the expression can be executed at any time; in this case, the result is:

```
49999995
```

RECALL

When the RECALL key is pressed, the last thing that was executed by the calculator is recalled to the display. The original expression can then be viewed, edited and re executed if necessary. For example, press:

(9)(↑)(( )(4)(−)(( )(7)(*)(·)(5)( ))

If you execute this expression, the display will read:

```
ERROR 16
```

A quick check of the error codes shows that a right parenthesis is missing from the expression. If RECALL is then pressed, the original expression is recalled to the display:

```
9↑(4-(7*.5)
```

The required right parenthesis can then be added to balance the equation:

```
9↑(4-(7*.5))
```

Then the corrected expression can be executed:

```
 3
```

The PRT ALL (Print-All) key is used to obtain a printed record of your operations.

In the calculator mode with print-all in effect, everything in the display that is executed is recorded on the printer. The result of the execution is also printed.

Pressing PRT ALL either establishes or cancels print-all:

- If ON appears in the display, print-all is established.
- If OFF appears, print-all is cancelled.

By pressing PRT ALL a second time, the ON or OFF designation is reversed.

If the printer isn't connected to the calculator, be sure print-all is OFF; otherwise, the display will 'blank out' when the EXECUTE key is pressed (if this occurs, hold the STOP key down until STOP appears on the display to regain control of the calculator). Printer plug-in instructions are located in Appendix A of this book.

Here are some examples with print-all ON:

Keyboard Entries	Printouts
FIXED N 2 EXECUTE	FIXED2
3 6 5 * 2 4 EXECUTE	365*24 8760.00
3 + (4 * 5 EXECUTE	3+(4*5 ERROR 16
RECALL) EXECUTE	3+(4*5) 23.00

ADDITIONAL KEYS
(continued)

SCRATCH

Different portions of memory can be erased with the following variations of the SCRATCH command:

SCRATCH EXECUTE — erases all variables from memory and program lines from mainline memory [programs on Special Function keys (see Chapter 6) are not affected]. In many cases, this command effectively erases all the memory.

SCRATCH A EXECUTE — erases all of memory; it is the same as turning the calculator off and then on again.

SCRATCH V EXECUTE — erases the values of all variables.

SCRATCH K EXECUTE — erases all Special Function keys.

SCRATCH f(x) — erases the particular Special Function key that is pressed.

FUNCTIONS

The following paragraphs describe the mathematical and trigonometric functions in the Model 30.

MATHEMATICAL FUNCTIONS

The following mathematical functions can be executed:

ABS (expression)†	determines the absolute value of the expression;
EXP (expression)	raises the constant, e, to the power of the computed expression;
INT (expression)	gives the expression an integer value ≤ the value of the expression;
LGT (expression)	determines the logarithm of a positively valued expression to the base 10;
LOG (expression)	determines the logarithm of a positively valued expression to the base 'e';
RND (expression)	gives a random number between Ø and 1; the expression is a dummy argument;

† Throughout this text, the word 'expression' includes constants (like 43.2), variables (like A), expressions (like 43.2*A), and other functions (like INT A).

SGN (expression)	returns a 1 if the expression is greater than zero, returns a Ø if the expression equals zero, or returns a −1 if the expression is less than zero;
SQR (expression)	computes the square root of a positively valued expression.

In general, it is wise to separate the expression by parentheses although parentheses are optional if the expression is either a positive constant, a variable, or another function.

As can be seen by the examples to follow, most of these functions are relatively easy to use; however, the RND function warrants the additional explanation given it.

In the standard mode, execute these key sequences:

- (A)(B)(S)(()(2)(−)(7)()) → 5

  which gives a positive value to the expression.

- (E)(X)(P)(1) → 2.718281828

  which is the value of the constant, e, raised to the first power.

- (I)(N)(T)(()(6)(·)(3)(*)(4)()) → 25

  whereas,

  (I)(N)(T)(()(−)(6)(·)(3)(*)(4)()) → −26

  The integer value returned must be less than or equal to the result of the expression. So for a result of 25.2, 25 is returned; whereas for a result of −25.2, −26 is returned.

- (L)(G)(T)(2) → 0.30102996

  (L)(O)(G)(2) → 0.693147181

  LGT is used for determining logarithms to the base 10, whereas LOG is used for determining logarithms to the base 'e' (natural logs). The following key sequences show how the result of an expression can be determined if logarithms are used.

  Here's a practical example. Suppose we wish to solve for X, where $X=144^{111}/121^{108}$. This equation cannot be solved by using regular mathematics, since the range of the calculator would be exceeded. But this equation can be solved by using logarithms and their mathematical rules. Using $Y=\log_{10} X$, key in and execute:

  Y=111*LGT144−108*LGT121

  Then to solve for X, execute:

  X↑Y → 4.33927E+14

(continued)

FUNCTIONS
(continued)

Whereas using $Y=\log_e X$, key in and execute:

Y=111*LOG144−108*LOG121

Then to solve for X, execute:

X=EXPY

```
 4.33927E+14
```

- (R)(N)(D)(x)†

```
 0.039710839
```

A random number between Ø and 1 is displayed each time RND x is executed.

Often it is useful to have random numbers appear as integers that are within a certain range. For instance, random integers from Ø through 999 are output if you execute:

(I)(N)(T)(()(R)(N)(D)(x)†(*)(1)(0)(0)(0)())

Now each time this expression is executed (this is easily accomplished if you press RECALL EXECUTE), a new random number is generated.

Each time the calculator is turned on, the random number generator uses the same 'seed' (that is, the number upon which the sequencing is based). This causes the sequence of numbers to be the same each time the RND function is initially used. But you can avoid the repetition if you put in your own 'seed'. This is accomplished by executing:

(R)(N)(D)(()(−)(n)())

The negative sign causes the seed to be changed while the number input for 'n' becomes the new seed.

Now that the seed is generated, random numbers can once again be obtained by the method previously discussed. In this way, a new seed is used each time.

Some seeds do make better random number generators than others. To obtain a good seed, n should be a number between Ø and 1; the more digits the number has, the better − (up to 11); and it is often preferable to make the last digit either a 1, 3, 7, or 9.

Here's an example of a good seed. Execute:

(R)(N)(D)(()(−)(.)(1)(2)(3)(4)(5)(6)(7)(8)(9)())

The seed that the Model 30 itself uses is: $(2-\pi/2)$.

† where 'X' can be any positive number. The positive number used has absolutely no effect on the random number that is generated.

- (S)(G)(N)(()(6)(*)(7)()) 1

but

(S)(G)(N)(()(−)(6)(*)(7)()) −1

Since the result of 6*7 is positive, a '1' is returned; and since the result of −6*7 is negative, a '−1' is returned.

- (S)(Q)(R)(1)(2)(1) 11

but

(S)(Q)(R)(()(−)(1)(2)(1)()) ERROR 52

For the square root of a negative number is an 'imaginary' number.

TRIGONOMETRIC FUNCTIONS

The following trigonometric functions can be executed:

SIN (expression)†	which determines the sine of an expression.
COS (expression)	which determines the cosine of an expression.
TAN (expression)	which determines the tangent of an expression.
ATN (expression)	which determines the arctangent of an expression.

The angle is always assumed to be in radians, unless otherwise stated. However, the angle can be expressed in any of the following three ways:

DEG	which calculates angles in degrees.
GRAD	which calculates angles in grads.
RAD	which calculates angles in radians.

When one of these three forms is specified, all calculations assume the specified form. To revert to one of the other two forms, it is necessary to execute the desired mnemonic. Then, if all the memory is erased or if the calculator is initialized (see page 4-14), radians are once again assumed.

As mentioned before, it is wise to separate the expression by parentheses. In addition to the parentheses guidelines given previously in the "Mathematical Functions" section, be sure to use parentheses when evaluating the tangent of variable D. The reason TAN D causes an error, although TAN (D) does not, is apparent in the "Logical Operators" section, page 2-20.

The value of π can be obtained if you key in:

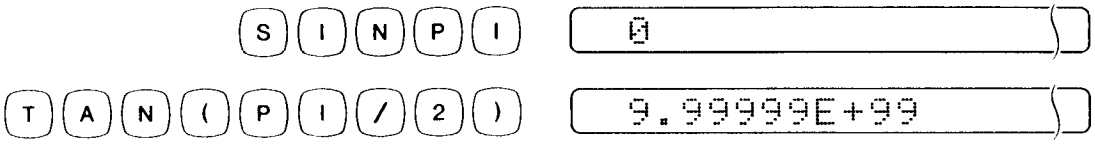
(P)(I) When an expression encounters this, 3.14159265360 is assumed.

(Continued)

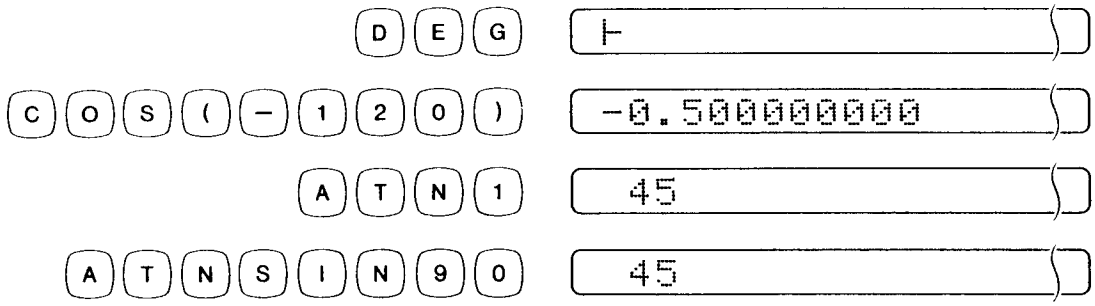
† Like the mathematical functions, it is generally wise to separate the expression by parentheses although parentheses are optional if the expression is either a positive constant, a variable, or another function.

FUNCTIONS
(Continued)

Here are some examples; execute the following key sequences:



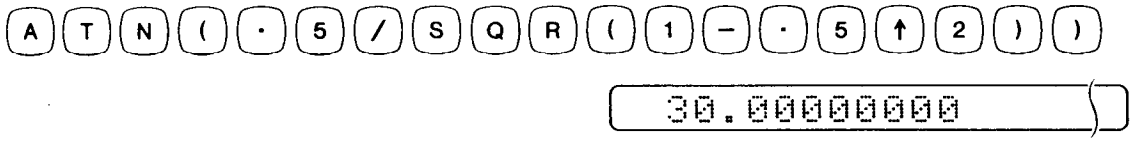
The last result approximates positive infinity. Now to compute angles in degrees, execute:



The arcsine and arccosine functions are not directly obtainable on the Model 30. However, they can be easily obtained by using the following formulas – where x is the point at which the function is evaluated.

Arcsine x=ATN(x/SQR(1−x↑2))
Arccosine x=ATN(SQR(1−x↑2)/x)

Here's an example; with the calculator still in degrees, the arcsine of .5 can be determined if you execute:



MATHEMATICAL HIERARCHY

When functions are included in the arithmetic hierarchy†, they have the highest precedence. So the order of execution is as follows:

1. Functions — highest precedence
2. Exponentiation
3. Multiplication, Division
4. Addition, Subtraction — lowest precedence

All other rules affecting the order of execution remain the same.

† The arithmetic hierarchy was presented on page 2-6.

LOGICAL EVALUATION

In logical evaluation, expressions† can be compared by using relational operators and/or logical operators. If the comparison is 'true', the value '1' is returned; if the comparison is 'false', the value 'Ø' is returned.

RELATIONAL OPERATORS

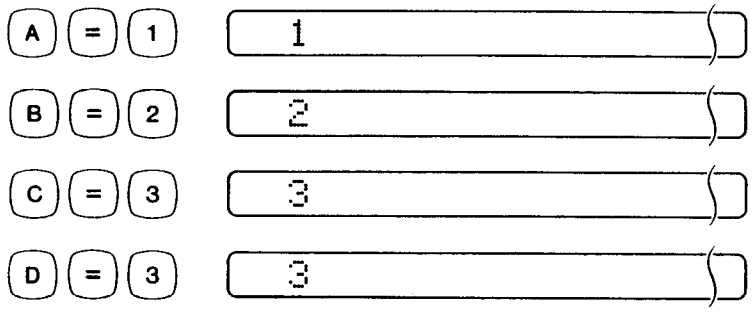
These operators are used to determine the logical relationship between two expressions. They are as follows:

equality	=
inequality	# or <>‡
greater than	>
less than	<
greater than or equal to	>=
less than or equal to	<=

The equals sign is also used in assignment statements as shown on page 2-9. In assignment statements, the variable is to the left of the equals sign and the value to be assigned to this variable is to the right of the equals sign. If the equals sign is used in such a way that it might be either an assignment or relational operator, the Model 30 assumes it is an assignment operator.

Here are a few examples using relational operators:

First let's assign values to the variables A, B, C, and D. Execute:



Now execute these logical expressions:

A<B	which is true, so 1 is displayed.
B<A	which is false, so Ø is displayed.
B#C	which is true, so 1 is displayed.
C#D	which is false, so Ø is displayed.
3=C	which is true, so 1 is displayed.
4=A	which is false, so Ø is displayed.
A=4	which is an assignment statement and not a logical statement; so A is assigned the value of 4.

Relational operators are very important in programming, especially when used in the IF statement, discussed on page 3-9.

† An expression can be a constant (like 7.2), a variable (like B), or an arithmetic expression (like 7.2*SQR6).

‡ # corresponds to a shifted 3; < corresponds to a shifted ,; and > corresponds to a shifted .

LOGICAL EVALUATION
(Continued)

LOGICAL OPERATORS

The logical operators, often called 'Boolean' operators, are as follows:

AND
OR
NOT

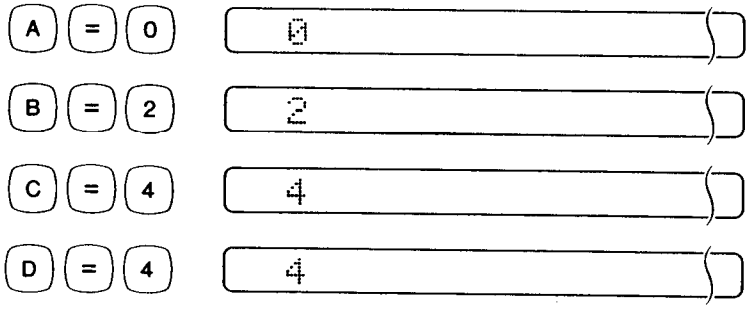
- AND compares two expressions. If both expressions are true, the result is true. If one or both of the expressions are false, the result is false.
- OR compares two expressions. If one or both of the expressions is true, the result is true. If neither expression is true, the result is false.
- NOT changes the logical value of an expression. If the expression is true, NOT changes it to false. If the expression is false, NOT changes it to true.

As stated earlier, if the result is true, a 1 is returned; if the result is false, a Ø is returned.

The expressions that the logical operators compare can be either relational or non-relational:

- If the expression is relational (like A<B), its true or false designation is determined by the particular relational value.
- If the expression is non-relational (like A), it is true if its arithmetic value is any value other than zero; it is false if its arithmetic value equals zero.

Here are some examples; first let's assign values to the variables A, B, C, and D. Execute:



Now execute these logical expressions:

A<B AND C=D

Since both relational expressions, A<B and C=D, are true, the result is true and a 1 is displayed.

A AND C=D

The expression, A, is false since its arithmetic value equals zero. The expression, C=D, is true. But since AND requires that both expressions be true to return a true result, the result is false and a Ø is displayed.

A OR B

The arithmetic value of A is zero (false) while the arithmetic value of B is two (true). Since at least one of the expressions is true, a 1 is displayed.

A OR C–D

Both arithmetic expressions have a value of Ø (false). So a Ø is displayed.

NOT A

Since A is false, NOT A is true and a 1 is displayed.

NOT B OR NOT C

Since the arithmetic values of B and C are both non-zero, they are both true. Therefore, NOT B and NOT C are both false; so a Ø is displayed.

The returned results, either Ø or 1, can be used in calculations. Using the variables, A, B, C and D again, let's evaluate S in the equation shown below.

S = ((B<C) + (NOT D=A)) * 12

24

The result of the true relation (B<C) is first added to the result of the true relation (NOT D=A). In other words, 1 + 1 = 2. This result is then multiplied by 12 for a product of 24.

Here are two methods you can use to change the logical value of a variable, say W:

W = NOT W W = (W = Ø)

In the second example, when W = Ø, the relation (W = Ø) is true, so the new value assigned to W is 1. Conversely, when W ≠ Ø, the relation (W = Ø) is false and a zero is returned.

TOTAL MATHEMATICAL HIERARCHY

The hierarchy presented below includes the relational and logical operators discussed in the previous section.

Functions	highest precedence
↑	
NOT	
* /	
+ –	
Relational Operators	
AND	
OR	lowest precedence

As mentioned before, for operations at the same level in the hierarchy, the order of execution is from left to right.

All operations enclosed within parentheses are performed first, thus altering the normal order of execution.

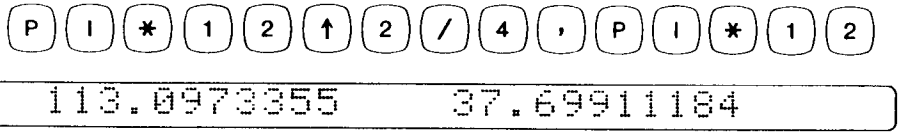
ADDITIONAL FEATURES

In the calculator mode, a few additional features are available.

SIMULTANEOUS CALCULATIONS

It is possible to execute several expressions simultaneously if the expressions are separated either by commas or semicolons.

For instance, if the diameter of a circle is 12 units (d=12), both the area ($\pi d^2/4$) and the circumference (πd) can be solved simultaneously by executing:



More than two expressions can be solved simultaneously; however, since only 32 of the allowable 80 characters can be displayed at once, it may be necessary to hold ←, the left arrow key, down to view all the results.

The only difference between separating the expressions by commas or semicolons is that semicolons generally cause the results to be 'packed' together while commas cause the results to be 'spread' apart.

PRINT, WRITE

The PRINT statement and the WRITE statement are two important program statements.† But they can also be used in the calculator mode, to have the results of calculations printed.

Execute the following example:

```
PRINT 222*11, 528*8
```

The printout will be:

```
2442            4224
```

An advantage to using the WRITE statement is that you can select the device you want to output on. The primary printer has select code 15 assigned to it. But if you have a secondary printer, it would have another select code, say, 8. Then either printer could be specified in the WRITE statment as follows:

```
WRITE (15,*) 222*11, 528*8
```

or

```
WRITE (8,*) 222*11, 528*8
```

† See page 3-4 and 3-21.

Chapter 3

PROGRAMMING & PROGRAMMING STATEMENTS

The programming language, BASIC, is used in the Model 30. In this chapter, the BASIC statements are individually discussed.

The following information is for the experienced BASIC user:

- These statements can be used in the same way to which you are accustomed:

LET	REM	DEF FN	IF
PRINT	STOP	DIM	FOR . . NEXT
INPUT	END	COM	READ . . DATA
GOTO			GOSUB . . RETURN

  In some cases, these statements have additional features and minor variations from other BASIC languages. But, essentially, they can be used just as you would expect.

- These statements, available on the Model 30, are not common to many BASIC languages:

'implied'LET	WAIT	STANDARD	GOTO & GOSUB with OF
DISP	RAD	FIXED N	Multiple-Line Functions
WRITE	DEG	FLOAT N	Programmable Tape Commands
FORMAT	GRAD		

- These statements have additional features, not common in many BASIC languages:

STOP	DIM	READ . . DATA with RESTORE
END	COM	

Before discussing each statement, a quick introduction to writing and running a program will be presented.

PROGRAM WRITING

A program is a set of instructions organized to accomplish certain tasks. It is organized by lines (statements) with each statement preceded by a unique line number.

Line numbers must appear in the program in ascending order for 'bookkeeping' purposes. However, in the Model 30, you can type program lines in any order because lines are automatically sorted as they are entered. Line numbers 1 through 9999 are allowed.

Program lines can be a maximum of 80 characters in length. In general, any spacing between characters that you use is totally arbitrary. For the calculator inserts appropriate spacing into each line that is entered into memory to make the line easily readable upon review. Only in quote fields and REM statements (discussed later), will the spacing necessarily remain exactly as it was input.

PROGRAM WRITING
(continued)

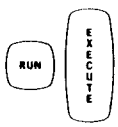
After each line is written, it is entered into calculator memory when the END OF LINE key is pressed.

Normal program execution proceeds from the lowest-numbered line to the highest-numbered line. However, the order of execution can be altered by some of the statements discussed later.

Flowcharting techniques are often valuable aids to program writing. They are discussed at the end of this chapter where an example program is flowcharted.

PROGRAM EXECUTION

If a program has been correctly keyed in and is the lowest-numbered program in memory, it can be executed by pressing:



If the program is operating properly, it will perform the required tasks and then halt. However, you can halt a program that is running if you press:

However, if the program is waiting at an INPUT statement (discussed on page 3-8), press – STOP EXECUTE – instead.

In Chapter 4, all the programming related commands are discussed.

The rest of this chapter discusses each BASIC statement.

At the end of this chapter, the syntax requirements of all the BASIC statements are presented. These syntaxes are intended primarily for the advanced programmer.

ASSIGNMENT

In many BASIC languages, this statement is called the LET statement. But in the Model 30, the mnemonic, LET, is optional; so it is referred to simply as an assignment or 'implied' LET statement.

This statement can assign a value to both simple and array variables.† The value assigned can be another variable, a constant, or an expression.

An equals sign must be used to separate the variable (to the left of the equals sign) from its assigned value (to the right of the equals sign).

Examples

```
15 G=32.172
25 T=2
35 D=G*T↑2
45 X=Y=Z=0
55 X=X+1
65 U[2]=1
75 A=G
85 LET A=G
```

Lines 15 & 25: The simple variables, G and T, are assigned constant values.

Line 35: The expression, G*T↑2, determines the value of D. The variables, G and T, must have assigned values for the statement to be executed.

Line 45: In one statement several variables can be assigned the same value. This is a useful technique for initializing variables in the beginning of a program.

Line 55: The variable, X, is set equal to its old value plus 1. This type of statement can function as a 'counter' in a program to determine how many times certain operations are performed.

Line 65: The array variable, U(2), is assigned the value of 1. Since the array variable is less than U(11), it need not be dimensioned (see page 3-36 for dimensioning rules).

Lines 75 & 85: The variable, A, is assigned the value of the variable, G. Both statements are equivalent. The word, LET, is optional and can be used in any of the statements discussed.

† If an array variable is used, it must be properly dimensioned; see page 3-36. When array variables are input, parentheses are used, as in A(3). However, these parentheses are converted to brackets as in A[3] when the line is entered into memory.

PRINT

The PRINT statement can cause various outputs on the printer:

- Text can be printed in any form.
- The values of variables, expressions, and constants can be printed.
- Printer lines can be skipped to separate outputs.

Any text that is to be printed must be enclosed by quotation marks; e.g., "ABCDE". The information within the quotation marks is often referred to as a quote field.

Several variables, expressions, and constants can be included in the same PRINT statement as long as they are separated by commas or semicolons. The difference between commas and semicolons is shown in the examples.

Text can also appear in a PRINT statement with variables, expressions, and constants. Aside from the quotation marks, no additional punctuation is necessary when separating text from other information. If neither the comma nor the semicolon is used, the semicolon is assumed.

Examples

```
 11 X=3
 15 Y=4
 17 PRINT "X IS EQUAL TO";X
 27 PRINT
 37 PRINT X"SQUARED =";X↑2;
 47 PRINT Y"SQUARED =";Y↑2
 57 PRINT 1,2;X,Y;5
 67 PRINT 1;2;X;Y;5
 77 PRINT "THE SQUARE ROOT OF";X"SQUARED PLUS";Y"SQUARED EQUALS";SQR(X↑2+Y↑2)
```

Lines 13 & 15: Variables must have assigned values before being used in a PRINT statement.

Line 17: The quote field is printed exactly as seen. Spaces are not ignored in quote fields as they are in other places. The semicolon between the quote field and the variable, X, is not needed since a semicolon would have been assumed anyway. The variable, X, must have an assigned value or an error occurs when the program is run. With X=3 when the program is run, the printout is:

```
X IS EQUAL TO 3
```

Line 27: This statement tells the printer to skip a line before doing any more printouts. Additional printouts can then begin on the following line.

Lines 37 & 47: A variable, a quote field, and an expression are all designated in each of these PRINT statements. Ending the first of two PRINT statements with a semicolon (as in line 37) causes both printouts to appear on the same line. Here is the printout:

```
3     SQUARED = 9      4     SQUARED = 16
```

A comma could be used instead of the semicolon. But then the printout would be:

```
 3          SQUARED = 9                    4          SQUARED = 16
```

Without any punctuation at the end of line 37, the printout would be:

```
 3          SQUARED = 9
 4          SQUARED = 16
```

Lines 57 & 67: Both constants and variables are printed. The only difference between the two statements is that in line 57 the different items are separated by commas, whereas in line 67 they are separated by semicolons. Here are the two printouts:

```
Line 57   1                    2                    3                    4                    5
Line 67   1          2         3         4          5
```

When only commas are used (as in line 57), successive printout fields are 15 character spaces apart and as many as five values can always be printed on the same line.

However, when semicolons are used, as in line 67, the values are 'packed' together in the printout. As many as 12 values can be output on the same line (with a field width of 6 character spaces per value) if semicolons are used. But if any value requires more than 4 character spaces (remember, one space is always allocated to the sign), the number of available fields per line is reduced.

Line 77: A long line of text is combined with variables and an expression. The printout is:

```
THE SQUARE ROOT OF 3     SQUARED PLUS 4     SQUARED EQUALS 5
```

The maximum printout per line using the PRINT statement is 72 characters.

Each value output by a PRINT statement is left justified within its respective field (with the left-most character space reserved for the sign).

The same rules that apply to outputs of keyboard calculations if the standard format is used (see page 2-1) also apply to outputs from the PRINT statement.

If your printer has a lower-case character set:

> Lower-case characters can be output as text within a quote field if they are keyed in with the SHIFT key held down. Text in WRITE and FORMAT statements (discussed beginning on page 3-21) performs in the same way.

The PRINT statement can only access the printer with select code 15. Be certain that your primary printer is set at this select code. (Most printers are shipped to the customer with select code 15 set.)

DISP

The DISP (Display) statement operates like a PRINT statement except that the outputs appear on the display rather than on the printer.

Three advantages of using the DISP statement are:

- If you don't have a printer, you can still run programs.
- In many programming applications, various operations are performed over and over; in these cases, the DISP statement is a good tool for viewing the changing values of variables – without having to use printer paper and by taking less time than it takes to print the information.
- It's extremely useful in labeling INPUT statements (see page 3-8).

Two advantages of using the PRINT statement are:

- Permanent records of the outputs can be obtained.
- For output lines greater than 32 characters in length, not all the information can appear on one printed line. (The DISP statement could still be used, however, if the program ends with the display still visible; then ←, the left arrow key, could be held down to view the end of the display.)

Examples

```
10 B=5
20 DISP "THE VALUE OF B IS"B
30 DISP B↑2-3
40 DISP -1111,2222;3333
```

Lines 20, 30, & 40: These statements merely show that the same operations that are allowed in a PRINT statement are also allowed in a DISP statement. The spacing between fields would be the same with either PRINT or DISP. Here are the three displays:

```
THE VALUE OF B IS 5
```

```
 22
```

```
-1111          2222          3333
```

NOTE

These lines are not meant to be run as one program. If they were, lines 20 and 30 would appear on the display for just an instant. If, in a program, you wish to have two or more successive displays, the WAIT statement (see page 3-17) can be used to prolong each display.

If the last visible output in a program results from a DISP statement, the display remains even after the program is completed.

STOP, END

Either the STOP or END statement can be used to terminate program execution. The statements can be located in any portion of the program.

The only difference between the two statements is as follows:

- When STOP is encountered, the program halts and the current position of the program line counter† is retained.
- When END is encountered, the program halts and the program line counter reverts to the lowest-numbered statement in memory.

The CONT (Continue) key is discussed on page 4-4. When the keys – CONT EXECUTE – are pressed, the calculator will begin execution at the position of the program line counter. So if a STOP statement is encountered, the program halts. The values of variables can be checked. Then the program can be continued, as though it had never been halted, by pressing: CONT EXECUTE.

Examples

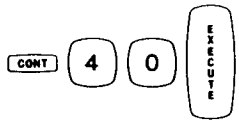
```
10 P=12
20 DISP P
30 STOP
```

(press: CONT EXECUTE)

```
40 DISP P↑2
50 END
```

Lines 10 through 50: If this program is run, lines 10 through 30 are executed; the program halts after the STOP statement with the program line counter positioned at line 40. If the keys – CONT EXECUTE – are then pressed, lines 40 and 50 are executed. The program halts after the END statement with the program line counter repositioned at line 10.

If line 30, in the previous example, had been an END statement, line 40 still could have been accessed by pressing:



But to have the program halt during, say, a FOR . . . NEXT loop (see page 3-12) and then to be able to continue as though it had never been halted, the STOP statement must be used.

The two keys, STOP and END, cannot be used as program statements; both statements *can* be programmed only by typing them in, letter by letter.

† The 'program line counter' is an internal device used by the calculator to determine the order of program execution.

INPUT

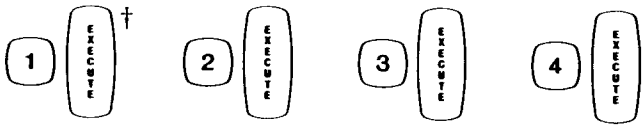
The INPUT statement allows variables to be assigned values from the keyboard during program execution.

When the INPUT statement is accessed, a '?' appears on the display. A value can then be input for each of the variables designated in the INPUT statement. For instance, in the statement:

```
10 INPUT A,B,C2,D[3]
```

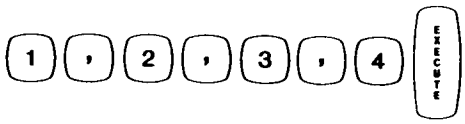
One value must be assigned to each of the four variables.

Values can be assigned individually or in groups. For instance, the values, 1, 2, 3, and 4, can be assigned as:



†

or as:



In the first case, the '?' will reappear on the display after each input until all four values are input.

In the second case, all four values are input together. The values can be separated either by commas or by semicolons.

In both cases the values, 1, 2, 3, and 4, are assigned to A, B, C2, and D(3), respectively.

Examples

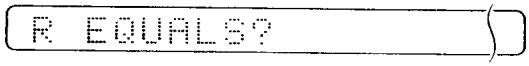
```
12 INPUT D,D1,D2
22 INPUT D[1],D[2]
```

```
32 DISP "R EQUALS";
42 INPUT R
```

Line 12: The simple variables, D, D1, and D2, are to be assigned values. When more than one variable is designated in an INPUT statement, the variables must be separated by commas.

Line 22: The array variables, D(1) and D(2), are to be assigned values. Since the array variables are less than D(11), they need not be dimensioned (see dimensioning rules on page 3-36).

Lines 32 & 42: By ending line 32 with a semicolon, the text, R EQUALS, will be displayed together with the next display in the program, which is the '?' from line 42. So the display will be:



† Either the EXECUTE or the END OF LINE key can be pressed to assign values to variables in the INPUT statement.

Then the variable, R, in line 42 can be assigned a value.

When a program encounters an INPUT statement, a '?' appears on the display and the program waits for values to be keyed in. If you prefer to terminate the program or edit a Special Function key at this point, first press the keys: STOP EXECUTE or END.

The INPUT statement can be bypassed completely by using the CONT command, as discussed on page 4-4.

IF

The IF statement contains an expression and another line number. First, the expression is logically evaluated. If it is evaluated as 'true', the program continues execution at the specified line number. But if the expression is evaluated as 'false', the program continues execution in its normal sequence with the statement following IF. (Logical evaluation is discussed in Chapter 2, beginning on page 2-19.)

Examples

```
                    1 INPUT X,Y

 22 IF X=3 THEN 142          404 IF X AND Y AND X+Y THEN 424
 32 IF INT(X)=3 THEN 222     414 STOP
 42 STOP                     424 PRINT X,Y
  .                          434 IF X+Y <= 10 THEN 464
  .                          444 DISP Y-X
  .                          454 STOP
313 IF Y<X↑2 THEN 333       464 DISP X+Y
323 IF Y THEN 363           474 STOP
333 STOP
  .
  .
  .
```

Line 1: For all of these examples, let's assume that X=3.01 and Y=23.

Lines 22 through 42: The IF statement in line 22 is evaluated as 'false' since 3.01 is not equal to 3. So the following statement, line 32, is accessed. This IF statement is then evaluated. Since the integer value of 3.01 equals 3, the statement is evaluated as 'true'. So the program continues execution at line 222.

Lines 313 through 333: Since Y is greater than X^2, the IF statement in line 313 is evaluated as 'false'. So instead of accessing line 333, the program continues execution in its normal sequence with line 323. Line 323 is logically evaluated as 'true' since Y has a non-zero value. So the program continues execution at line 363.

Lines 404 through 474: Since both X and Y have non-zero values, line 404 is evaluated as 'true' and program execution is transferred to line 424 where the values for X and Y are printed. Line 434 checks to see if X+Y is less than or equal to 10. The evaluation is 'false' so program execution continues at line 444. The value of Y−X is computed and displayed. Then the program stops at line 454.

Avoid unnecessary punctuation; for instance:

```
22 IF X=3,THEN 142
```

The comma is not allowed here. If it's inadvertently keyed in, ERROR 24 occurs.

GOTO

The GOTO statement is used to alter the normal sequence of program execution. The program continues execution at the line number specified in the GOTO statement.

Examples

#1

```
 11 A=1
 21 IF A↑2<1000 THEN 51
 31 DISP A;A↑2
 41 STOP
 51 A=A+1
 61 GOTO 21
```

#2

```
 105 Z=0
 110 X=1
 115 Z=X+Z
 120 DISP X,Z/X
 125 X=X+1
 130 GOTO 115
```

Example No. 1: This program 'loops' (repeats part of itself) several times until the IF statement, line 21, is no longer true; that is, until A↑2≥1000. If A↑2<1000, the variable, A, is incremented by 1 in line 51. Then the GOTO statement, line 61, is executed causing the program to loop back to line 21.

Example No. 2: The GOTO statement in this program causes the program to loop between lines 115 and 130. This is a 'closed loop'; that is, there is no program statement in the loop that can cause the loop to be exited. So to halt this program, press the STOP key.

Line numbers in GOTO statements must be constants; statements like, 15 GOTO P, are not allowed. Furthermore, the line number specified must be in memory when the program is run or else an error will occur.

REM

The REM (Remark) statement is merely a note to the programmer and is not executed by the program. However, the statement does appear on a program listing.

Any series of characters can follow REM, the only restriction being the maximum line length of 80 characters.

Examples

```
10 REM   YOU CAN SAY
20 REM--ANYTHING YOU WANT
30 REM:TNAW UOY YAW YNA
40 REM..IN A $<#=@* REM STATEMENT
```

FOR . . . NEXT

The FOR and NEXT statements allow for the controlled repetition of a group of statements within a program.

The FOR and NEXT statements form a loop with the statements between them in a program. The FOR statement defines the number of times the loop is to be performed. For example:

```
10 FOR I=1 TO 5
 .
 .
 .
100 NEXT I
110
 .
 .
 .
200 END
```

This FOR . . . NEXT loop would be executed five times: when I=1, 2, 3, 4, and 5. Each time the NEXT statement is executed, the value of I is incremented by one. If I is less than or equal to 5, the loop is executed again. But when the value of I passes the final value, that is, when I=6, the statement following the NEXT statement (in this case, line 110) is accessed.

The advantages of using FOR . . . NEXT rather than IF can be shown in the following simple example where the numbers 1 through 100 are to be displayed in succession:

Using IF

```
10 N=1
20 IF N>100 THEN 60
30 DISP N
40 N=N+1
50 GOTO 20
60 END
```

Using FOR . . . NEXT

```
10 FOR N=1 TO 100
20 DISP N
30 NEXT N
40 END
```

In the example the program is easier to key in, takes up considerably less calculator memory, and executes faster if the FOR. . .NEXT loop is used.

Examples

#1

```
11 Z=0
21 FOR P=90 TO 100
31 Z=Z+P
41 NEXT P
51 PRINT "THE TOTAL IS"Z
61 END
```

#2

```
1 Z=0
11 INPUT M,N
21 FOR P=M TO N
31 Z=Z+P
41 NEXT P
51 PRINT "THE TOTAL IS"Z
61 END
```

#3

```
19 FOR A=1 TO 12
29 PRINT A;
39 NEXT A
49 PRINT A
59 END
```

#4

```
9 Z=0
19 INPUT V
29 FOR A=1 TO V
39 Z=Z+A
49 NEXT A
59 PRINT "THE AVERAGE IS"Z/(A-1)
```

or

```
59 PRINT "THE AVERAGE IS"Z/V
69 END
```

Example No. 1: The initial value of the variable assigned in the FOR . . . NEXT loop need not be 1; in this case, P is assigned the value, 90. This example takes the summation of the integers, 90 through 100, and prints the total.

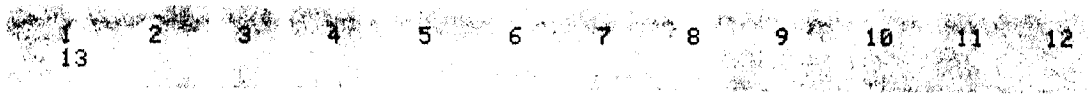
Example No. 2: This example is like example no. 1, except that the user can vary the numbers to be totaled. The variable, P, in line 21 is assigned the variable value, M, for its first value and is assigned the variable value, N, for its final value. M and N are both assigned values in the INPUT statement, line 11.

- If M=3 and N=5, the summation of 3, 4, and 5 would be taken.
- If M=3.1, and N=5.1, the summation of 3.1, 4.1, and 5.1 would be taken.
- If M=3.1 and N=5, the summation of 3.1 and 4.1 would be taken.

In each case the value of P increments by 1 after each loop. In the last case, the loop would be performed only twice, when P=3.1 and 4.1; for when P increments to 5.1, the value of N, which is 5, is exceeded and line 51, which follows the NEXT statement, is accessed.

By the way, if M is set equal to a value greater than N, the loop is immediately bypassed and the total printed in line 51 would be Ø.

Example No. 3: An aspect of the FOR . . . NEXT loop that is often overlooked is emphasized in this example. After each loop is performed, the NEXT statement, line 39, increments the value of A by 1. It then compares the incremented value to the final value indicated in line 19. If this incremented value is less than or equal to the final value, another loop is performed starting at line 29; but if the incremented value is greater than the final value, (that is when A=13) the loop is no longer accessed and line 49 is executed. In this line the value of A is printed. Although the final loop is performed when A=12, the last incremented value for A is 13 and the calculator memory retains this as the value of A. So the total printout would be:



In all cases, the final value retained for the variable in the FOR . . . NEXT loop is greater than the final value in the FOR statement. In some programs, this could be a minor problem. The next example shows how to compensate for it.

Example No. 4: This program calculates the average of the inputs (either version of line number 59 could be used).

- In the first case, the summation (Z) is divided by the number of inputs (A−1). A−1 is used to compensate for the final incremented value of A, as discussed in the previous example.
- In the second case, Z is divided by V to get the average. Since V retains its initial value, no compensation factor is needed.

(continued)

FOR . . . NEXT
(continued)

#5

```
1 FOR G=1 TO 100
2 IF G↑2 >= 500 THEN 5
3 NEXT G
4 STOP
5 PRINT G;G↑2
6 END
```

#6

```
12 FOR I=1 TO 4
22 A[I]=I↑2
32 PRINT A[I];
42 NEXT I
52 PRINT
62 PRINT A[1];A[2];A[3];A[4]
72 END
```

Example No. 5: This program shows that a FOR . . . NEXT loop can be exited before the final variable value is reached. When this program halts, the printout is:

```
 23   529
```

Example No. 6: This program shows how the FOR . . . NEXT loop can be used to assign values to arrays. In this example, the array variables, A(1) through A(4) are assigned values. The printout is:

```
 1    4    9    16
 1    4    9    16
```

FOR . . . NEXT loops can be nested; that is, they can be located inside one another. However, one loop cannot overlap another. For instance:

Correct

```
10 FOR E=1 TO 15
20 FOR Z=3 TO 12
 .
 .
 .
80 NEXT Z
90 NEXT E
```

Incorrect

```
10 FOR E=1 TO 15
20 FOR Z=3 TO 12
 .
 .
 .
80 NEXT E
90 NEXT Z
```

Notes:

- Entering a FOR . . . NEXT loop at any place other than the FOR statement (with statements like GOTO or IF), can cause unpredictable results and should, therefore, be avoided.
- Each FOR statement can have only one associated NEXT statement.

The initial and final variable values in the FOR statement can be other variables or expressions. Once the FOR . . . NEXT loop is set up, the values of these variables can change without affecting the number of times the loop is repeated.

In the example below, the initial value of A is 1 and B is 6. The variables A and B can be used within the loop for other purposes, but the loop itself is repeated only six times.

```
10 A=1
20 B=6
30 FOR I=A TO B
40 A=278
50 B=999
60 PRINT I;A;B
70 NEXT I
80 END
```

FOR . . . NEXT with STEP

It's not necessary that the variable in the FOR . . . NEXT loop be incremented by 1 each time the loop is executed. For instance:

```
10 FOR I=0 TO 12 STEP 3
 .
 .
 .
50 NEXT I
```

By adding STEP to the FOR statement, the variable will be incremented by the number following STEP (in this case, 3). So this loop will be executed five times: when I=Ø, 3, 6, 9, and 12. But if the program read:

```
10 FOR I=0 TO 11 STEP 3
 .
 .
 .
50 NEXT I
```

The loop would be executed only four times: when I=Ø, 3, 6, and 9. As soon as the incremented value passes the final value, the loop is exited.

Examples

#1

```
12 FOR X=1 TO 4
or
12 FOR X=1 TO 4 STEP 1
22 PRINT X↑2-X;
32 NEXT X
42 END
```

Example No. 1: Either version of line 12 could be used with identical results. In either case, when the program loops, the value of X is incremented by 1. Therefore, if you want the variable to increment by 1, it is unnecessary to use the STEP feature.

#2

```
15 FOR C=1 TO 6 STEP 2
25 DISP C;
35 NEXT C
45 END
```

Example No. 2: This loop is executed three times: when C=1, 3, and 5. The final display is:

```
1        3        5
```

(continued)

FOR . . . NEXT with STEP
(continued)

```
#3
17 FOR T=4 TO -8 STEP -4
27 PRINT T;T↑2
37 NEXT T
47 END
```

Example No. 3: This program shows that FOR . . . NEXT loops can be incremented by negative values, in this case, –4. This loop is executed four times: when T=4, Ø, –4, and –8. The printout is:

```
 4      16
 Ø      Ø
-4      16
-8      64
```

```
#4
4 DEG
14 FOR S=-30 TO -90 STEP -15
24 PRINT SIN(S)
34 NEXT S
44 END
```

Example No. 4: Line 4 allows angles to be calculated in degrees. The FOR . . . NEXT loop is incremented negatively as it was in example No. 3. The loop is incremented five times: when S=–30, –45, –60, –75, and –90. The printout is:

```
-Ø.5ØØØØØØØØ
-Ø.7Ø71Ø6781
-Ø.866Ø254Ø4
-Ø.9659258Ø6
-1
```

The step size need not be an integer value; e.g., STEP 2.5 is allowed.

Furthermore, the initial variable value, the final variable value, and the step size can all be expressions; e.g.,

```
10 FOR I=A/6 TO A/2 STEP Ø.1*A
```

WAIT

The WAIT statement introduces a timed delay between two program statements. It's particularly useful for delaying consecutive printouts or prolonging displays.

The WAIT statement causes the program to pause the specified number of milliseconds (1000 milliseconds = 1 second). The delay can be set to vary between approximately Ø and 33 seconds.

Examples

#1

```
10 Y=0
20 X=2↑Y
30 DISP "2 ↑"Y"="X
40 WAIT 3000
50 Y=Y+1
60 GOTO 20
```

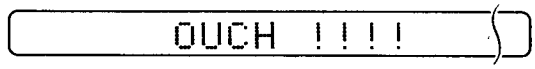
#2

```
10 DISP "DON'T TOUCH ME!"
20 WAIT 32000
30 DISP "      OUCH !!!!"
40 WAIT 4000
50 GOTO 10
```

Example No. 1: In this program, the value of Y is incremented by 1; and for each value of Y, 2^Y is calculated and displayed. By putting in the statement, WAIT 3000, about a three second delay occurs between consecutive displays.

Example No. 2: The quote field in line 10 is displayed for about 32 seconds, and then the quote field in line 30 is displayed for about 4 seconds, etc. The WAIT statement has one additional feature. Most of the Model 30 keys, if pressed, will cause the current WAIT statement to be terminated and program execution to be resumed. (Of course, if the STOP key is pressed, the program halts.) So if the calculator is displaying:

and, say, an alphabetic key is pressed, the calculator would immediately display:



The maximum delay that can be specified in a WAIT statement is about 33 seconds. Any WAIT statement with a number greater than 32767 will wait the maximum amount of time.

The specified delay in a WAIT statement can be a constant, a variable, or an expression.

READ ... DATA

The READ and DATA statements combine to assign values to variables. The READ statement determines the variable (for instance, 10 READ Y), while the DATA statement determines the value to be assigned to the variable (for instance, 90 DATA 12.4).

More than one variable can be specified in a READ statement (for instance, 10 READ X,Y,Z) and several values can be specified in a DATA statement (for instance, 90 DATA 5.1, 7.6, 9.3, 4, 16, 9.2).

The calculator uses an internal mechanism, called a 'pointer', to locate the data element that is to be read. The leftmost element of the lowest-numbered DATA statement is read first. After this element is read, the data pointer repositions itself one element to the right, and continues to do so each time another data element is read.

After the last element in a DATA statement is read, the data pointer locates the next higher-numbered DATA statement and repositions itself at the first element in that statement. But if there are no higher-numbered DATA statements, the data pointer remains at the end of the previous DATA statement; and any effort to read additional data will result in an error message.

The location of the DATA statement in a program is immaterial unless there is more than one DATA statement; if so, just be sure they are in the order you want.

Examples

#1

```
10 READ A,B,C
 .
 .
 .
40 READ A1,B1
 .
 .
 .
70 READ P
 .
 .
 .
200 DATA 4,5,6,7.31,2.69,0
```

#2

```
12 FOR I=1 TO 5
22 READ X
32 PRINT X"SQUARED ="X↑2
42 NEXT I
52 DATA 24,8.3,17,19,3.2
62 END
```

#3

```
7 DATA 3,4,8,15,7,24,47,1104
17 FOR J=1 TO 4
27 READ X,Y
37 PRINT X;Y;SQR(X↑2+Y↑2)
47 NEXT J
57 END
```

#4

```
18 READ N
28 FOR P=1 TO N
38 READ D,D1
48 PRINT D↑2-D1
58 NEXT P
68 DATA 3
78 DATA 9,1,8,4,7,9
88 END
```

Example No. 1: This example shows that several READ statements can apply to the same DATA statement. The data pointer just moves to the right each time another data element is read. The first variable is assigned the first data value, the second variable is assigned the second value, and so on.

Data can appear in more than one DATA statement. Thus, the following representations would all be equivalent:

```
200 DATA 4,5,6,7.31,2.69,0
```

or

```
200 DATA 4,5,6
201 DATA 7.31,2.69
202 DATA 0
```

or

```
200 DATA 4,5
201 DATA 6,7.31,2.69,0
```

Example No. 2: READ statements are often used in FOR . . . NEXT loops. Each time this loop is executed, a new value for X is read and a new line is printed. The printout is:

```
24   SQUARED = 576
8.3  SQUARED = 68.89
17   SQUARED = 289
19   SQUARED = 361
3.2  SQUARED = 10.24
```

Example No. 3: The DATA statement can appear anywhere in the program. In this case, it is the first statement. In this program, the FOR . . . NEXT loop is executed four times, and each time, new values are assigned to X and Y. The printout is:

```
3     4     5
8     15    17
7     24    25
47    1104      1105
```

Example No. 4: In this program one READ statement is outside the FOR . . . NEXT loop (thus being executed only once) and one is inside the loop (thus being executed three times). The printout is:

```
80
60
40
```

The DATA statement is accessed only by a READ statement; otherwise, the program ignores it.

Once the READ statement variables are assigned values from a DATA statement, the statement following READ is executed.

READ . . . DATA with RESTORE

Data elements can be read more than once if the RESTORE statement is used.

There are two variations of the RESTORE statement; for instance:

```
80 RESTORE  or  90 RESTORE 150
```

- If the first RESTORE statement, 80, is encountered during program execution, the data pointer reverts to the first data element in the lowest-numbered DATA statement.
- If the second RESTORE statement, 90, is encountered during program execution, the data pointer reverts to the first data element in the DATA statement specified (in this case, line 150).

Examples

#1

```
10 FOR I=1 TO 5
20 READ A
30 PRINT A"SQUARED ="A↑2
40 NEXT I
50 RESTORE
60 PRINT
70 FOR J=1 TO 3
80 READ B
90 PRINT B"CUBED ="B↑3
100 NEXT J
110 DATA 4,9,12,8,27
120 END
```

#2

```
10 READ N
20 FOR I=1 TO N
30 READ A
40 PRINT A"SQUARED ="A↑2
50 NEXT I
60 RESTORE 130
70 PRINT
80 FOR J=1 TO 3
90 READ B
100 PRINT B"CUBED ="B↑3
110 NEXT J
120 DATA 5
130 DATA 4,9,12,8,27
140 END
```

Example No. 1: In this program, immediately before the RESTORE statement is accessed, the data pointer is located after the last data element in line 110. The RESTORE statement resets the data pointer to the first element in line 110 so that the data can be reread.

Example No. 2: In this program, there are two DATA statements (line 120 and line 130). Immediately before the RESTORE statement is accessed, the data pointer is located after the last data element in line 130. This RESTORE statement directs the data pointer to the first element in line 130. If the line number, 130, had not been specified, the data pointer would have been reset to line 120.

In both examples, the printout is:

```
 4    SQUARED = 16
 9    SQUARED = 81
 12   SQUARED = 144
 8    SQUARED = 64
 27   SQUARED = 729

 4    CUBED = 64
 9    CUBED = 729
 12   CUBED = 1728
```

WRITE

This section shows how the WRITE statement can be used like a PRINT statement although it is generally used with the FORMAT statement (discussed on page 3-22).

When used without the FORMAT statement, the WRITE statement works like a PRINT statement with one major exception: the desired output device can be specified in a WRITE statement whereas the PRINT statement always assumes the primary printer (select code 15).

The WRITE statement specifies the output device by select code. The primary printer is always select code 15. If you have a secondary printer, it might be set at select code 8. Either of these select codes (or the select code of another compatible output device) could then be specified.

To write the constants, 1, 2, 3, on the primary printer, the statement could be:

```
10 WRITE (15,*)1,2,3
```

If this statement is executed, the printout is identical to the following PRINT statement:

```
10 PRINT 1,2,3
```

Within the parentheses, the select code† is specified, followed by the '*'. The '*' indicates that the WRITE statement is being used without a corresponding FORMAT statement and that all the rules of the PRINT statement are to be followed when outputting the constants, expressions, etc.

Examples

#1

```
10 X=4
20 WRITE (8,*)X"SQUARED ="X↑2
30 END
```

#2

```
12 INPUT S,Z
22 IF S#8 AND S#15 THEN 12
32 WRITE (S,*)Z"SQUARED ="Z↑2
42 END
```

Example No. 1: If a secondary printer with select code 8 is connected to the calculator, the printout would be:

```
 4  SQUARED = 16
```

Example No. 2: In this program, if both the primary printer (select code 15) and the secondary printer (select code 8) are connected to the calculator, either one can be accessed. The value input for the variable, S, determines the select code. So if 15 is input for S and 6 is input for Z, the printout on the primary printer would be:

```
 6  SQUARED = 36
```

† The select code can be a variable.

FORMAT

The FORMAT statement gives output specifications to the WRITE statement that references it. The formatting of numbers and the spacing between successive items is easily controlled with the FORMAT and WRITE statements. Also, symbols not available with the PRINT statement can be output by using FORMAT and WRITE.

The WRITE statement references a FORMAT statement as follows:

```
10 WRITE (15,200)A
```

where, in this example, 15 refers to the select code of the output device, and 200 refers to the line number of the corresponding FORMAT statement.

Like the PRINT statement, the WRITE statement can output:

- Text (in a quote field)
- Values for variables
- Values for expressions
- Constants.

Text is output the same in either statement; the information inside the quotation marks is output character for character.

If expressions, variables, and constants are specified in a WRITE statement, their values can be output according to either of the following FORMAT statement specifications:

- Fw.d where F indicates fixed-point format;
  w indicates total field width; must be an integer constant;
  d indicates the number of digits to the right of the decimal point; must be an integer constant.

- Ew.d where E indicates exponential format (often called floating-point or scientific notation);
  w indicates total field width; must be an integer constant;
  d indicates the number of digits to the right of the decimal point; must be an integer constant.

The other available FORMAT statement specifications are:

- X which indicates a blank character space;
- / which indicates a carriage return–line feed for the printer;
- "text" which indicates a quote field (also allowed in the WRITE statement);
- B 'Binary' format which allows symbols that are otherwise not obtainable to be output by the Model 30.

Any combination of the specifications can appear in the same FORMAT statement, but different items must be separated by commas.

Any of the specifications can be duplicated a specific number of times if a repeat factor is specified; for instance:

```
10 FORMAT 2F6.2,4X,F6.2
```

The first fixed-point field would appear twice, followed by four character spaces, and then another fixed-point field.

The FORMAT statement, like the DATA statement, can appear anywhere in the program. It is never executed by the program until it is referenced by a WRITE statement.

Examples

```
12 FORMAT F6.1,F10.2,F2.0
20 WRITE (15,12)7.7,-92.6

33 FORMAT E8.1,E12.2
40 WRITE (15,33)1234,2.6E+95
```

Lines 12 and 20: The three numbers are output according to the three fixed-point formats. The printout is:

```
ΔΔΔ7.7ΔΔΔΔ-92.00Δ6
  1      2     3
```

Fw.d The field widths, w, for the three printouts are 6, 10, and 2, respectively. In the printout, the three field widths have been labeled 1, 2, and 3. In the first two formats, the number of digits to the right of the decimal point is 1 and 2, respectively. In the third case, since d=Ø, the decimal point is suppressed.

Notice all three printouts are right justified within their respective fields. In each case, there is a space available for the sign (although in 1 and 3, since the sign is '+', the sign is suppressed). Also, in every case (except for d=Ø), the decimal point takes up one character space. If the field width does not allow for both the sign and the decimal point, the value will not be output (see lines 90 and 99 on the following page).

Lines 33 and 40: The two numbers are output according to the two exponential formats. The printout is:

```
Δ1.2E+03ΔΔΔΔ2.60E+95
   1          2
```

Ew.d The field widths, w, for the two printouts are 8 and 12, respectively. The number of digits immediately to the right of the decimal point, d, is 1 and 2, respectively.

The 'E' format follows the same general rules as the 'F' format: printouts are right justified; the sign (or assumed sign) and the decimal point take up one character space each. But in 'E' format, the field width must be at least four greater than in 'F' format because the last four digits are needed to indicate the power of 10 to which the number should be raised.

(continued)

In these examples the 'Δ' will be used to indicate a blank space in the printout.

FORMAT
(continued)

```
55 FORMAT F6.2,E10.2
60 WRITE (15,55)4,97,8,43,6

90 WRITE (15,99)12.2,1.8
99 FORMAT F4.1,E6.0
```

Lines 55 and 60: The 'F' and 'E' formats are both used here. The printout is as follows:

```
 4.00 9.70E+01
 8.00 4.30E+01
 6.00
```

After the first two values are printed according to the specified formats, the printer performs a carriage return—line feed (resets itself to the beginning of the next line). Then two more values are printed according to the same two specified formats, etc.

Lines 90 and 99: The WRITE statement is located before the FORMAT statement; it makes no difference to the calculator. For a value of 12.2 to be printed as it appears would require an 'F' format with field width of 5 (to allow for the implied sign). Since the specified field width of 4 is not large enough, the calculator outputs dollar signs, $$$$, in the entire field to indicate the lack of space. The second value, 1.8, is rounded up to 2 since the second format calls for zero decimal places. (Incidentally, had this specified field width been one less, dollar signs would also have been displayed here.) This printout is:

```
$$$$ 2E+00
 1    2
```

Formatting Rules:

- Using Fw.d If d > 0, then the minimum field width allowable is w=d+3. If d=0, the minimum field width allowable is w=2. One of the character spaces is always allocated for the sign (although only the minus sign is displayed). Another space is needed for the digit preceding the decimal point. Additional spaces are required if more than one digit precedes the decimal point. For instance, F4.1 is sufficient for the number, 2.6; but F5.1 is needed for the number, 12.6.

- Using Ew.d If d > 0, then the minimum field width allowable is w=d+7. If d=0, the minimum allowable field width is w=6. Of course, the field width can be greater to increase the number of blank character spaces.

The following six examples show additional formatting techniques.

#1
```
10 FORMAT F5.0,2X,"TROMBONES"
12 WRITE (15,10)76
14 END
```

#2
```
10 FORMAT F5.0
12 WRITE (15,10)76,"TROMBONES"
14 END
```

#3
```
10 FORMAT F5.0,/,"TROMBONES"
12 WRITE (15,10)76
14 END
```

#4
```
10 FORMAT F5.0,/
12 WRITE (15,10)76,"TROMBONES"
14 END
```

#5

```
10 FORMAT F5.0," TROMBONES"
12 WRITE (15,10)76," OR FLUTES"
14 END
```

#6

```
10 FORMAT F5.0," TROMBONES",F6.2
12 WRITE (15,10)76," OR FLUTES"
14 END
```

Example No. 1: The number, 76, is printed according to the F5.0 specifications; it is followed by two character spaces (2X) and the word, TROMBONES. The printout is:

```
   76  TROMBONES
```

Example No. 2: The number, 76, is again printed according to the F5.0 specifications. The FORMAT statement then checks for additional format specifications. Since nothing follows the F5.0, the printer performs a carriage return–line feed. Then the WRITE statement is checked for additional specifications. Since there is a quote field following the 76, the quote field is output. The printout is:

```
   76
TROMBONES
```

Example No. 3: This output is the same as in example no. 2. This time, however, the '/' specifies the carriage return–line feed. As mentioned, the printout is:

```
   76
TROMBONES
```

Example No. 4: In this case, the printer performs the carriage return–line feed twice: first when the '/' is accessed; and then because nothing else follows the '/'. The printout is:

```
   76

TROMBONES
```

Example No. 5: This example shows that after the value, 76, is output according to the F5.0 specifications, the text in the FORMAT statement is output before the text in the WRITE statement; and since nothing follows the text in the FORMAT statement, a carriage return–line feed is performed before the text in the WRITE statement is output. The printout is:

```
   76 TROMBONES
 OR FLUTES
```

Example No. 6: This example illustrates a technique that often is useful. Occasionally it would be beneficial to suppress the carriage return–line feed before the WRITE statement is completed. A 'dummy' format specification can be used to accomplish this; that is, an 'F' or 'E' specification in the FORMAT statement without a corresponding value in the WRITE statement. The printout is:

```
   76 TROMBONES OR FLUTES
```

(continued)

FORMAT
(continued)

Three more examples are included here: The first example includes all the concepts discussed earlier, and the last two examples show some applications of FORMAT B.

```
#7
10 INPUT X,Y,Z
20 FORMAT "AVERAGE",5X,"MEAN",5X,"MEDIAN",/,F5.1,6X,F5.1,5X,F5.1
30 WRITE (15,20)X,Y,Z
40 END
```

```
#8
10 FORMAT B                          or    10 FORMAT 3B
20 WRITE (15,10)34,91,93                   20 WRITE (15,10)34,91,93
30 END                                     30 END
```

```
#9
10 WRITE (15,20)34,91,93,34
20 FORMAT 11X,"1   2",/,B,"LOOK AT IT'S",B,"ITS",B," SIZE,",B,
                                                    "EXCLAIMED MARY!"
30 DISP "WHICH IS CORRECT, 1 OR 2";
40 INPUT X
50 PRINT
60 IF X=2 THEN 90
70 PRINT X"IS THE INCORRECT RESPONSE"
80 STOP
90 PRINT X"IS THE CORRECT RESPONSE"
100 END
```

Example No. 7: Suppose X, Y, and Z were input as 9.9, 10.2, and 10.3, respectively. When the FORMAT statement is accessed, the headers for the three numbers are output with a character spacing of five between words. Then the printer performs a carriage return–line feed and prints the three numbers according to the specified formats. The spacing between the 'F' fields (6X and 5X) is needed so that the numbers will be positioned under the appropriate headings. (Generally it takes a few attempts before the spacing works out correctly.) The printout is:

```
AVERAGE      MEAN      MEDIAN
  9.9        10.2       10.3
```

Example No. 8: FORMAT B is used to output symbols not otherwise available on the Model 30. 'B' is indicated in the FORMAT statement and the corresponding number in the WRITE statement is interpreted as a code. (This code is actually the decimal equivalent of an octal number understood by the calculator.) The symbol you will get for each number input is shown in Table F-2, Appendix F. Most of these symbols are already easily accessible on the Model 30 without using FORMAT B; but a few of them cannot be obtained unless FORMAT B is used. For instance, in the first printout:

```
"
[
]
```

Using FORMAT 3B, the printout is:

```
"[ ]
```

None of these symbols can be output without using FORMAT B. Quotation marks (") can be input to represent quote fields but cannot be output within quote fields. As can be seen by this printout, the codes for these symbols are 34, 91, and 93, respectively.

Example No. 9: In this example, practical uses for some of these symbols are shown. Everything in the FORMAT statement is output as usual. When the first 'B' is encountered, the first number in the WRITE statement is taken as the code for the symbol to be output there. When the second 'B' is encountered, the second number is the code, etc.

The printout from the WRITE and FORMAT statements is:

```
                1   2
"LOOK AT IT'S[ITS] SIZE,"EXCLAIMED MARY!
```

This grammatical question has two choices, 1 or 2. If '2' is keyed in when the INPUT statement is encountered, the final printout is:

```
2     IS THE CORRECT RESPONSE
```

Unlike the PRINT statement, in which the maximum printout per line is 72 characters, with WRITE and FORMAT the maximum printout per line is restricted only by the character length of your printer.

A semicolon or comma at the end of a WRITE statement suppresses the carriage return—line feed if the corresponding FORMAT statement specifications have not been completed.

#10

```
10 WRITE (15,20)
20 FORMAT 80"*"
30 END
```

#11

```
10 FOR I=1 TO 8
20 WRITE (15,50)I;
30 NEXT I
40 PRINT
50 FORMAT 8F10.2
60 END
```

Example No. 10: The FORMAT statement is used in this example to output 80 asterisks on one line. It is referenced by a WRITE statement which does not include a list of variables.

Example No. 11: The FOR ... NEXT loop repeats eight times. The semi-colon in the WRITE statement suppresses a line feed each time. Notice that since the FORMAT statement specifies a field width of 10 character spaces per item, an 80-character line is effectively used. Line 40 is required to dump the buffer after all eight numbers have been generated.

PRINT with TAB

The TAB command is most often used with the PRINT statement but it can likewise be used in DISP and WRITE† statements. With the TAB command, outputs can be located at a specified character position. Character positions Ø through 71 can be designated. Notice it is the absolute character position that is specified — not the character spacing. If a character position greater than 71 is specified, a carriage return–line feed is output.

The TAB command can also specify a variable character position.

Two reminders:

- If TAB is used in a DISP statement, anything past TAB31 will not be visible on the display.
- TAB is ignored if the character position is already past the specified TAB; for example, 10 PRINT 12, TAB6, 24. Since the punctuation after the 12 is a comma, the value of 12 is automatically contained in a 15 character field. And so, TAB6 is irrelevant.

Examples

```
#1
10 INPUT X,Y,Z
20 PRINT "AVERAGE"TAB20"MEAN"TAB40"MEDIAN"
30 PRINT X,TAB20;Y,TAB40;Z
40 END

#2
10 DEG
20 X=0
30 PRINT TAB(35+25*SINX);X;"DEG"
40 X=X+30
50 GOTO 30
```

Example No. 1: The PRINT statement in line 20 sets up the headers for the variables, X, Y, and Z. The first heading, AVERAGE, starts at character position Ø; the heading, MEAN, starts at character position 20; and the heading, MEDIAN, starts at character position 40. Then in line 30, the variables are printed under the three headings. Suppose X, Y, and Z were input as 11.23, 11, and 11.4, respectively. The printout would be:

```
AVERAGE              MEAN                 MEDIAN
 11.23                11                   11.4
```

Example No. 2: In this example, an application of a variable TAB is shown — TAB (35+25*SINX). The TAB origin is at character position 35. The curve for the sine of X fluctuates to either side of the origin — to the right for positive sine values and to the left for negative sine values. (The sine of X is multiplied by a factor of 25 in this

† The WRITE statement will accept the TAB command only when WRITE operates like a PRINT statement; e.g., 10 WRITE (15,*)2, TAB 30, 4.

example to enable the TAB to make larger increments.) This program halts whenever you press the STOP key. The printout is:

```
                          0     DEG
                                    30    DEG
                                         60    DEG
                                              90    DEG
                                         120   DEG
                                    150   DEG
                          180   DEG
               210   DEG
    240   DEG
270   DEG
    300   DEG
               330   DEG
                          360   DEG
                                    390   DEG
                                         420   DEG
                                              450   DEG
                                         480   DEG
                                    510   DEG
                          540   DEG
               570   DEG
    600   DEG
630   DEG
    660   DEG
               690   DEG
                          720   DEG
                                    750   DEG
                                         780   DEG
                                              810   DEG
                                         840   DEG
                                    870   DEG
                          900   DEG
               930   DEG
    960   DEG
990   DEG
    1020     DEG
               1050     DEG
                          1080     DEG
                                    1110     DEG
```

GOSUB ... RETURN

The GOSUB statement transfers program execution to the line number specified by GOSUB. Eventually, a RETURN statement is accessed, which transfers program execution to the statement following the previous GOSUB statement.

The GOSUB and RETURN statements eliminate the need to repeat frequently used groups of statements in a program.

Examples

#1

```
10 INPUT A
 .
 .
 .
60 GOSUB 1000
70 PRINT A
 .
 .
 .
200 GOSUB 1000
210 PRINT A,N
 .
 .
 .
990 STOP
1000 N=0
 .
 .
 .
1200 RETURN
1210 END
```

#2

```
10 INPUT A,N
 .
 .
60 GOSUB 1000
70 PRINT A,N
 .
 .
900 STOP
 .
 .
1000 IF A*N<150 THEN 1020
1010 GOSUB 2000
1020 A=A*5
 .
 .
1200 RETURN
 .
 .
1990 STOP
2000 A=(A*N)/5
 .
 .
2200 RETURN
2210 END
```

The order of execution by line number in the two examples is:

Example #1	Example #2	
	when A*N<150	when A*N≥150
10–60	10–60	10–60
1000–1200	1000	1000–1010
70–200	1020–1200	2000–2200
1000–1200	70–900	1020–1200
210–990		70–900

In both examples, before each subroutine (lines 1000–1200 and lines 2000–2200) there is a STOP statement. A statement like this is a good precautionary measure to ensure that a subroutine is not inadvertently accessed; if it is, the error – improper RETURN – will probably occur.

Example 2 (when A*N≥150) shows that subroutines can be nested; that is, a second subroutine can be executed before the RETURN in the first subroutine is accessed.

Each nested subroutine requires one additional word of calculator memory during program execution. Therefore, the number of nested subroutines is restricted only by the size of the calculator memory.

GOTO & GOSUB with OF

The GOTO and GOSUB statements were discussed earlier (pages 3-10 and 3-30, respectively). In both statements the program reverts to the line number specified. But, in this section, two additional features of the statements, called the computed GOTO and the computed GOSUB, are discussed.

Both statements contain an expression followed by the word 'OF'; to complete the statement, one or more line numbers are specified.

The expression is evaluated and its rounded integer value is determined. This integer value then acts as a pointer to determine which line number, from those specified, is to be accessed. The first line number specified corresponds to an integer value of 1, the second line number corresponds to an integer value of 2, etc. If the rounded integer value of the expression is either less than 1 or greater than the number of specified statements, then the statement following the computed GOTO or GOSUB statement is accessed. The form is always the same as seen in the following examples.

Examples

#1

```
10 INPUT X
20 GOTO X OF 300,330,360,390
30
 .
 .
 .
500 END
```

#2

```
10 INPUT X,Y
20 GOSUB ABS(X-Y) OF 200,300,400
30
 .
 .
 .
110 GOTO 10
 .
 .
 .
400 PRINT X↑2+Y↑2
 .
 .
 .
500 RETURN
```


Example No. 1: In this example, the expression to be evaluated is merely the variable, X. Values of X equal to 1, 2, 3, or 4 correspond to the four line numbers listed. If X=2, the second line number (line 330) is accessed. If X=2.5, the value of X is rounded to 3 in the computed GOTO statement, and the third line number (line 360) is accessed. (Decimal values of .5 and greater are always rounded to the next higher integer value.) If the rounded integer value of X is either less than 1 or greater than 4, the statement following the computed GOTO statement (line 30) is accessed.

Example No. 2: This example uses a computed GOSUB statement. The expression to be evaluated is: ABS(X−Y). Since three statements are specified, if the integer value of the expression is 1, 2, or 3, the first, second, or third subroutine is accessed; otherwise, the following statement, line 30, is accessed.

A relational expression can also be used in the computed GOTO or GOSUB statement, but since each relational expression returns a logical value of Ø or 1 only, add 1 to the expression. Consider this statement: 20 GOTO 1 + (A OR B) OF 40,80. When (A OR B) is false (i.e., Ø), the program branches to line 40; when true (i.e., 1), the program branches to line 80.

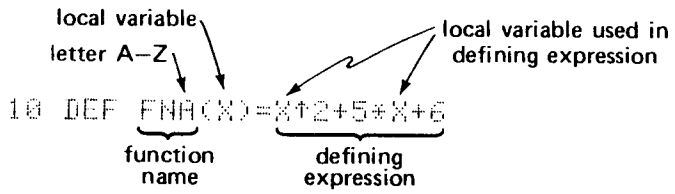
DEF FN

The DEF FN (Define Function) statement can define a function in one line (for single-line functions) or in several lines (for multiple-line functions). Single-line functions will be discussed first.

SINGLE-LINE FUNCTIONS

If an algebraic expression has to be evaluated several times in a program, it may be convenient to define the expression as a function.

A maximum of 26 defined functions, FNA through FNZ, are possible in a program. Besides a function name, a letter (A through Z), the function also needs a simple variable (referred to as a 'local'† variable). If the local variable is also included in the expression, the function can easily be evaluated for different values of the local variable.



Examples

#1
```
10 DEF FNB(X)=X↑2+X
20 PRINT FNB4;FNB5;FNB6
30 END
```

#2
```
10 INPUT Y
20 PRINT FNZ(Y)-5
30 END
40 DEF FNZ(Y)=Y↑2-Y
```

#3
```
10 INPUT X
20 DEF FNA(Y)=SQR(X↑2+Y↑2)
30 FOR I=1 TO 5
40 INPUT Y
50 PRINT FNAX
60 NEXT I
70 END
```

Example No. 1: The function, B, is defined with X as its local variable. Three values of the function are evaluated and printed: for X=4, 5, and 6. In the example, X is defined only as a local variable and is undefined as a global variable (see footnote). The printout is:

```
 20          30          42
```

Example No. 2: The function, Z, is defined with Y as its local variable. (Notice, it doesn't matter where the DEF FN statement is located in a program.) A value for the global variable, Y, is input in line 10. Then the PRINT statement in line 20 causes the function to be evaluated for that value of Y. The result is then printed. (In this example, as in the following example, the local variable is also defined as a global variable.)

† A 'local' variable is defined only in relation to the function. It has no significance in the rest of the program. Every other program variable is referred to as a 'global' variable in that, once defined, it remains defined during the entire program. A local variable and a global variable can be defined by the same symbol (see examples no. 2 and no. 3).

NOTE

In the DEF FN statement, the local variable (in this example, Y) must be separated by parentheses. But in statements where the function is called, parentheses are optional.

Example No. 3: The function, A, is defined with X as its local variable, even though the expression has two variables, X and Y, in it. If 3 is input for X, and Y is assigned the five values, 4, 5, 6, 7 and 8, the printout is:

```
 5
 5.880951895
 6.708203932
 7.615773106
 8.544003745
```

MULTIPLE-LINE FUNCTIONS

Multiple-line functions work much like single-line functions with two additional capabilities:

- More sophisticated functions can be defined since a function can be described in a series of program statements.
- The value of any variable or expression within a multiple-line function can be output.

In multiple-line functions, the line boundaries for the total expression are determined by a DEF FN statement and a RETURN† statement.

There can be more than one RETURN statement in a multiple-line function (see example no. 2). But after the first RETURN statement is accessed (for instance, RETURN X), the program returns the value of the function to the statement that called the function; e.g., 10 PRINT FNA 3.

Whenever a multiple-line function is in a program, precautions should be taken so that the function will not be inadvertently accessed. Having an END statement immediately prior to the function is, in general, the best possible precaution. Similarly, when editing a function, press STOP and END to halt program execution before you access the function (especially if the function is on a Special Function key – see page 6-3). Be sure to press END again after editing has been accomplished to exit the key mode. Failure to follow this procedure may result in illogical numbers in statements which reference line numbers (e.g., GOTO, LOAD, WRITE).

(continued)

† This statement is not to be confused with the RETURN statement at the end of a subroutine (although they do perform similarly). For multiple-line functions, RETURN is always followed by a variable or followed by an expression (e.g., 90 RETURN X).

DEF FN
(continued)

Examples

#1
```
5 FOR I=1 TO 10
10 PRINT FNA(I,2
15 NEXT I
20 END
30 DEF FNA(X)
40 Z=X↑2+2*X+1
50 Q=SQR(Z
60 RETURN Q
```

#2
```
11 INPUT K
21 WRITE (15,31)FNK(K)
31 FORMAT F8.2
41 END
51 DEF FNK(K)
61 L=K↑2-6*K+8
71 M=K↑2-8*K+15
81 IF L >= M THEN 111
91 PRINT "M =";
101 RETURN M
111 PRINT "L =";
121 RETURN L
```

#3
```
10 INPUT V
20 DISP FNB(V)
30 END
40 DEF FNB(V)
50 W=ABS(4*V-82)
60 X=(W-ABS(V))
70 Y=(W-X)↑2
80 Z=ABS(W+X-Y)
90 IF Z<100 THEN 120
100 DISP "WHEN V ="V"Z ="
110 RETURN Z
120 Z=(Z+1)↑2
130 DISP "WHEN V ="V"(Z+1)↑2 ="
140 RETURN Z
```

Example No. 1: As soon as the PRINT statement, line 10, calls on the function, A, (in the first loop, it is FNA 1) the multiple-line function, lines 30 through 60, is accessed. The required operations are performed and values for Z and Q are determined. Q is then returned as the value of the function and printed. This PRINT statement also prints the value of Z. Each time I is incremented from 1 to 10, the multiple-line function is accessed when the PRINT statement calls on it. The printout is:

```
 2            4
 3            9
 4            16
 5            25
 6            36
 7            49
 8            64
 9            81
 10           100
 11           121
```

Notice the END statement immediately prior to the multiple-line function; this keeps the function from being inadvertently accessed during normal program execution.

The sequence of variables in the PRINT statement is extremely important. Z is undefined until the multiple-line function is called upon. Since PRINT FNAI calls upon the function, an error would occur if the PRINT statement tried to output the value of Z first; e.g., 10 PRINT Z, FNAI.

Example 1 could have been written more easily. For instance, there could have been a line 45 as follows:

```
45 RETURN SQR2
```

Then both lines 50 and 60 could be eliminated.

Example No. 2: The WRITE statement, line 21, calls upon the multiple-line function, lines 51 through 121. The function returns the value of either L or M depending on the result of the IF statement, line 81. If L > = M, then L is returned as the value of the function. If L < M, then M is returned as the value of the function. Here are two possible printouts:

If K is input as 2:

```
M =     3.00
```

If K is input as 6:

```
L =     8.00
```

Example No. 3: The DISP statement, line 20, calls upon the multiple-line function, lines 40 through 140. Various calculations are performed and if Z (line 90) > = 100, the value of Z calculated in line 80 is returned as the value of the function. But if Z < 100, a new value of Z is calculated (line 120) and this value is returned. Three possible displays are:

```
WHEN V = 4     Z = 112
```

```
WHEN V = 8     (Z+1)↑2 = 841
```

```
WHEN V = 16    Z = 236
```

Example 2 has a WRITE statement that calls upon a multiple-line function. But if there were a WRITE statement inside the multiple-line function, too, then ERROR 46 would result. (In essence, this would result in two FORMAT statements being accessed at the same time, which is not allowed.)

ARRAYS

CONCEPTS

An array is often a convenient tool for labeling large groups of data within a program. An array is specified by its:

- Name – any letter from A through Z, followed by its
- Size – either one or two numbers enclosed within parentheses.

A two-dimensional array is divided into rows (horizontal) and columns (vertical) – A(I,J) – where I refers to the number of rows and J refers to the number of columns; e.g., A(2,4) could have the following data elements:

$6_{A(1,1)}$	$7_{A(1,2)}$	$5_{A(1,3)}$	$2_{A(1,4)}$
$8_{A(2,1)}$	$4_{A(2,2)}$	$9_{A(2,3)}$	$3_{A(2,4)}$

The subscripts show how each element in the array can be specified; for example: the element, 7, is in row 1, column 2; the element, 8, is in row 2, column 1; and the final element, 3, is in row 2, column 4. The subscripts for the final element also indicate the actual array size – a 2 by 4 array; hence, 8 elements total.

A one-dimensional array can have several rows but only one column. So it can be thought of as – A(I,1) – although it is generally written as – A(I); e.g., a three-element array, A(3), could have the following data elements:

$3_{A(1)}$

$9_{A(2)}$

$6_{A(3)}$

Arrays having only one row but several columns must be written as a two-dimensional array – A(1,J).

The size of an array can be specified in either a DIM or COM statement as discussed on pages 3-38 and 3-39. The array size must be specified in one of these statements if the array is either:

- One-dimensional with more than 10 elements, or
- Two-dimensional with more than 10 rows or columns.

If the array size is not specified in either of these statements, the array size is assumed to be 10 for one-dimensional arrays and 10 by 10 for two-dimensional arrays.

If an array is small, say A(2,2), and you would like to maximize memory availability, the actual array size should be specified in either a DIM or COM statement; otherwise, the calculator reserves storage space for all the elements A(1,1) through A(10,10).

Elements in an array can be referenced in several types of program statements: Assignment, PRINT, DISP, WRITE, INPUT, and READ statements. They can also be used in program expressions, say, as: PRINT 4*A(2)↑3.

PROGRAMMING DATA ELEMENTS INTO ARRAYS

To input several data elements into array form, a program should perform a 'looping' operation. The IF and GOTO statements can accomplish this looping, but it is generally easier to use the FOR . . . NEXT statements. The following examples show a few of the available techniques. (In these examples, the array sizes are small enough so that neither the DIM nor the COM statement is required.)

- One-dimensional arrays (7 elements):

#1
```
10 FOR I=1 TO 7
20 INPUT A[I]
30 NEXT I
40 END
```

#2
```
10 FOR I=1 TO 7
20 READ A[I]
30 NEXT I
40 END
50 DATA 9,8,7,6,5,4,3
```

In the first example, a data element is input from the keyboard each of the seven times the loop is executed. In the second example, a data element is read in from the DATA statement each of the seven times the loop is executed.

- Two-dimensional arrays (3 by 5 — 15 elements):

#1
```
10 FOR I=1 TO 3
20 FOR J=1 TO 5
30 READ A[I,J]
40 NEXT J
50 NEXT I
60 END
70 DATA 11,12,13,14,15
80 DATA 21,22,23,24,25
90 DATA 31,32,33,34,35
```

#2
```
10 FOR I=1 TO 3
20 FOR J=1 TO 5
30 DISP "ROW"I,"COLUMN"J;
40 INPUT A[I,J]
50 NEXT J
60 NEXT I
70 END
```

#3
```
10 FOR I=1 TO 3
20 DISP "INPUT ROW"I;
30 INPUT A[I,1],A[I,2],A[I,3],A[I,4],A[I,5]
40 NEXT I
50 END
```

Example No. 1: In this program, there are two FOR . . . NEXT statements. For each value of I, the 'J' loop is executed five times. (So, five data elements are assigned per row.) The three DATA statements correspond to the way in which the data elements are read — row by row, five elements per row. The array elements are positioned as follows:

$11_{A(1,1)}$	$12_{A(1,2)}$	$13_{A(1,3)}$	$14_{A(1,4)}$	$15_{A(1,5)}$
$21_{A(2,1)}$	$22_{A(2,2)}$	$23_{A(2,3)}$	$24_{A(2,4)}$	$25_{A(2,5)}$
$31_{A(3,1)}$	$32_{A(3,2)}$	$33_{A(3,3)}$	$34_{A(3,4)}$	$35_{A(3,5)}$

Examples No. 2 & No. 3: In these examples, the arrays are assigned values through INPUT statements. The technique used in example no. 2 is much like the one used in example no. 1, with the two FOR . . . NEXT statements. But in example no. 3, values are input using only one FOR . . . NEXT statement.

DIM

The DIM (Dimension) statement causes the calculator to reserve memory for the specified simple and array variables. If arrays are dimensioned, the actual array sizes then input can be less than or equal to the sizes specified.

When either the INIT (Initialize) key is pressed or the program is run, memory storage is allocated to the specified variables.

- Generally, four words of memory are allocated per data element (16 calculator bits per word); e.g.,

```
10 DIM A[5,4],B[5,8]
```

240 words are allocated:
80 in array A, 160 in array B.

Each data element has full-precision (12-digit) accuracy.

- But there may be times when it is necessary to conserve memory storage. So two words of memory can be allocated per data element with split-precision (6-digit) accuracy for each element; e.g.,

```
10 DIM AS[5,4],BS[5,8]
```

where 'S' indicates split-precision and 120 words are allocated:
40 in array A, 80 in array B.

These data elements cannot have values either less than −9.99999E+63 or greater than +9.99999E+63.

- Only one word of memory is allocated per data element with integer-precision accuracy for each element; e.g.,

```
10 DIM AI[5,4],BI[5,8]
```

where 'I' indicates integer-precision and 60 words are allocated:
20 in array A, 40 in array B.

These data elements are rounded to integer values; the values cannot be less than −32767 or greater than +32767.

Full-precision, split-precision, and integer-precision accuracy can be mixed in a DIM statement; e.g.,

```
10 DIM A[25],B[5,5],CS[25,2],DI[100],EI[50,2],FS[50]
```

In this example, each array is allocated 100 words of memory due to the particular specifications.

Notes:

- The location of the DIM statement within the program is arbitrary.
- A program can have more than one DIM statement.
- One-dimensional and two-dimensional arrays cannot have the same name; e.g., 10 DIM A(3), A(5,5), is not allowed.
- Once a split or integer-precision array is dimensioned, no additional reference can be given to the 'S' or 'I'; e.g.,

```
10 DIM GS[8],HI[4,3]
20 G[1]=12.4
30 H[2,1]=3E-2
```
} Allowed

```
20 GS[1]=12.4
30 HI[2,1]=3E-2
```
} Not allowed

COM

The COM (Common) statement, like the DIM statement, causes the calculator to reserve memory for the designated number of simple and array variables. † The unique features of the COM statement are as follows:

- Data can be transferred from one program to another if data, common to both programs, is specified in separate COM statements.
- Once a common storage area is reserved in memory, it remains there until either the program memory is erased or another COM statement alters its appearance.

When storing data on tape cassettes, (see STORE DATA, Chapter 5), COM has two important features:

- The values of all variables specified in COM can be stored on the same tape cassette file. (Without COM, only one array can be specified per file.)
- Since simple variables can be specified in COM, their values can also be stored on a tape cassette. (Without COM, only arrays can be specified in a STORE DATA command.)

The Model 30 immediately reserves memory storage space for the data elements specified in the COM statement when it is entered into memory. Because of this, the following two rules must be followed:

- COM must be both the first program statement entered and the lowest numbered statement in memory – if several required statements are already in memory and you need to enter a COM statement, follow these four steps: 1) store the required statements on a tape cassette, 2) erase program memory – SCRATCH EXECUTE, 3) key in the COM statement, and finally 4) load the rest of the statements back into memory from the cassette (see STORE and LOAD, chapter 5).
- COM cannot be edited once it is entered into memory; however, if you would like to change the COM statement, follow the four steps just discussed.

The COM statement may not be necessary in your program:

- When programs are brought into memory from a tape cassette, the COM statement is not necessarily needed to transfer data from one program to another. Often, it is more convenient to use the LINK command (see LINK, Chapter 5).
- The same variables cannot be specified in both a COM and a DIM statement. So if the only need for either of these statements is to define array sizes, it is probably more convenient to use the DIM statement.
- COM is generally unnecessary for programs on Special Function keys (see pages 6-4 and 6-5); the values of variables in mainline memory are also accessible by a program on a Special Function key.

Common area storage space is allocated in the same order in which the variables appear in a COM statement; elements within an array are stored row by row; e.g.,

```
1 COM C[2,3],D,A[3]
```

Storage space for this COM statement is allocated in the following order:

C(1,1), C(1,2), C(1,3), C(2,1), C(2,2), C(2,3), D, A(1), A(2), A(3)

† When string variables are used, other factors must be taken into consideration. See the 11274 String Variables ROM Operating Manual.

COM
(continued)

If the COM statements in two successive programs are identical, then the values assigned to the variables in the first program will be assigned to the same variables in the second program.

——— For Advanced Programmers ———

If, however, the COM statements in two successive programs are different, then the values assigned to the variables in the second program depend on the element positions of the values. For example, if the COM statement in the first program is:

```
1 COM A[2,3],B[4]
```

and the following program has the COM statement:

```
1 COM Z[6],A[2,2]
```

then the elements in common storage are assigned as follows:

Element Position	First Program Reference	Second Program Reference
1	A(1,1)	Z(1)
2	A(1,2)	Z(2)
3	A(1,3)	A(3)
4	A(2,1)	Z(4)
5	A(2,2)	Z(5)
6	A(2,3)	Z(6)
7	B(1)	A(1,1)
8	B(2)	A(1,2)
9	B(3)	A(2,1)
10	B(4)	A(2,2)

A reference to the variable, A(2,2), in the first program accesses the value in element position 5; whereas a reference to the variable, A(2,2), in the second program accesses the value in element position 10.

The variable, A(1,1), in the first program references the same element position as the variable, Z(1), in the second program.

Simple variables, split-precision variables, and integer-precision variables can also be used in COM statements.

However, if two successive programs have different COM statements, (to ensure that element positions line up as intended) arrays should be aligned with arrays having the same number of elements, simple variables should be aligned with simple variables, split-precision arrays should be aligned with split-precision arrays, etc.

Successive COM statements need not have the same number of elements. But a program on a Special Function key (see page 6-4) cannot have a COM statement larger than the COM statement in mainline memory.

If, say, the first program that is run specifies:

```
1 COM A[10],B[5]
```

and the second program that is run specifies:

```
1 COM A[10]
```

then the storage allocated to the B array is erased.

ADDITIONAL STATEMENTS

FIXED N, FLOAT N, STANDARD

In PRINT and DISP statements, numerical outputs appear in standard notation unless either fixed-point or floating-point notation has been specified in a prior program statement.† If standard notation is then specified in a later program statement, numerical outputs from PRINT or DISP statements will again appear in standard notation.

Examples

```
10 PRINT 3
20 FIXED 4
30 PRINT 3
40 FLOAT 5
50 PRINT 3
60 STANDARD
70 PRINT 3
80 END
```

Since PRINT statements output in standard notation unless fixed-point or floating-point has been previously programmed in, the printout from line 10 is:

```
 3
```

Since line 20 programs in a fixed '4' format, the printout from line 30 is:

```
 3.0000
```

Since line 40 programs in a float '5' format, the printout from line 50 is:

```
 3.00000E+00
```

Since line 60 causes the program to revert back to standard notation, the printout from line 70 is:

```
 3
```

(continued)

† Standard, fixed-point, and floating-point notation are discussed beginning on page 2-1.

ADDITIONAL STATEMENTS

(continued)

DEG, GRAD, RAD

In a program, angles are calculated in radians unless either degrees or grads have been specified in a prior program statement. If radians are then specified in a later program statement, angles will again be calculated in radians.

Examples

```
10 X=SIN30
20 DEG
30 Y=SIN30
40 GRAD
50 Z=SIN30
60 RAD
70 X1=SIN30
80 PRINT X,Y,Z,X1
90 END
```

In line 10, X is calculated in radians; in line 30, Y is calculated in degrees; in line 50, Z is calculated in grads; in line 70, X1 is calculated in radians. The printout for X, Y, Z, and X1 is:

```
-0.988031624     0.500000000     0.453990500     -0.988031624
```

When the calculator is initialized or another program is run, the calculator reverts both to standard notation and to radians.

PROGRAMMABLE TAPE COMMANDS

All tape cassette commands are programmable. These commands are discussed individually in Chapter 5.

FLOWCHARTING TECHNIQUES

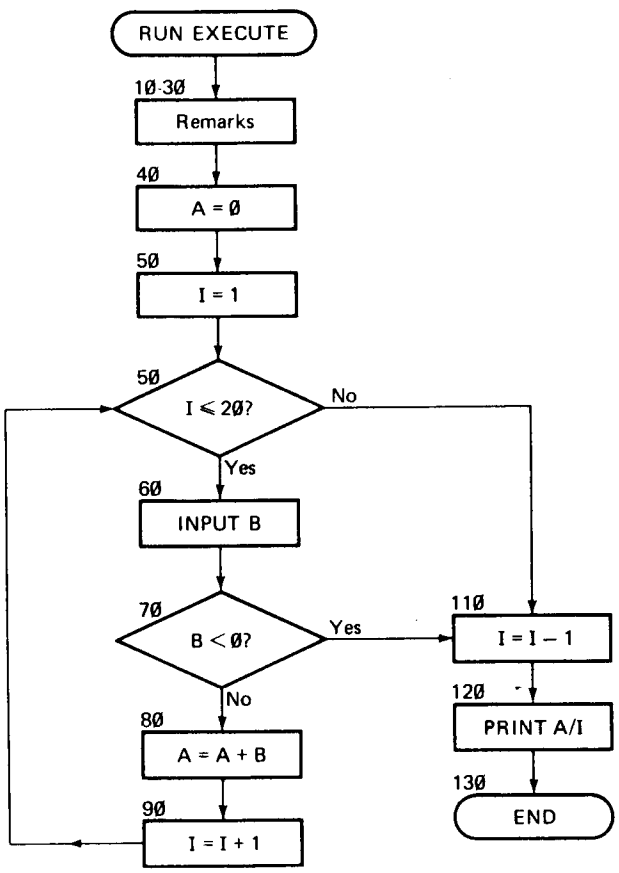
If you are a beginning programmer, you may be interested in knowing some of the techniques available that allow you to more easily advance from a conceptual stage in a programming task to a finalized stage.

Flowcharts, which logically pictorialize the solution to a programming task, are valuable programming tools. They are often drawn long before the actual program statements are written.

While flowcharting your problem, you might change or simplify your approach, see a flaw in your logic, etc. After several attempts, you should have a workable flowchart and, once you do, your programming task is greatly reduced.

Any flowchart that you draw is useful; but a few basic flowcharting conventions are described briefly here. Terminal (that is, starting or ending) activities are represented by ovals. Arrows, indicate the order of operations between the terminals. Most calculator operations are represented by rectangles. A diamond represents a decision point. If the information within the diamond is computed as 'YES', the logical flow continues in one direction; if it is computed as 'NO', the logical flow continues in another direction.

For your convenience, in the following example we have labeled each flowcharting operation with the corresponding program line number. As can be seen, once the flowchart is finalized, the program can be written relatively easily.




```
10 REM--THIS PROGRAM AVERAGES UP TO 20 POSITIVE NUMBERS.
20 REM--IF YOU HAVE FEWER THAN 20 NUMBERS TO ENTER,
30 REM--ENTER A NEGATIVE NUMBER TO END THE INPUTTING.
40 A=0
50 FOR I=1 TO 20
60 INPUT B
70 IF B<0 THEN 110
80 A=A+B
90 NEXT I
110 I=I-1
120 PRINT "THE AVERAGE OF THE"I"NUMBERS IS"A/I
130 END
```

BASIC SYNTAXES

Legend

brackets [ ]	– items enclosed within brackets are optional.
coloring	– colored items must appear as shown.
character	– a letter, a number or a symbol.
constant	– a number within the range of the calculator.
expression	– a constant [like 16.4], a variable [like B or D(6)] or an expression [like 8*A↑2 or A=6].
letter	– an alphabetic character from A through Z.
line number	– an integer from 1 through 9999.
local variable	– a simple variable defined only in relation to the function.
n	– an integer from Ø through 11.
select code	– an integer from 1 through 15.
text	– a series of characters enclosed within quotation marks.
variable	– a simple variable [like B or B7] or an array variable [like F(9)].

All programming statements must be preceded by a line number.

variable = expression or LET variable = expression
Assignment statement; assigns variable a value.

COM variable [, variable • • • , variable]
Reserves memory for the specified variables and allows data to be transferred from one program to another. Must be both the first statement entered and the lowest-numbered statement.

DATA constant [, constant • • • , constant]
Specifies data for READ statement variables.

DEF FN letter (local variable) = expression or DEF FN letter (local variable)
Defines a function in one line (first syntax) or in several lines (second syntax); in the latter case, a RETURN statement is also needed.

DIM variable [, variable • • • , variable]
Reserves memory for the specified variables when the calculator is initialized.

DISP [any combination of text and expressions]
Allows text and values to be output on the Model 30 display; follows the same rules as the PRINT statement.

END
Terminates program execution and resets the program line counter to the lowest-numbered statement in memory.

FOR simple variable = expression TO expression [STEP expression]
Executes the program lines between FOR and the corresponding NEXT statement a designated number of times; each time the loop is executed, the simple variable is incremented by 1, unless STEP is specified.

FORMAT any combination of text or other specifications
Gives output specifications to the WRITE statement that referenced it; specifications can be: Fw.d (for fixed-point format), Ew.d (for exponential format), X (for a character space), / (for a carriage return–line feed), B (for special ASCII characters); any specification can be repeated (say, 6Fw.d); specifications must be separated by commas.

GOSUB line number
Begins executing the subroutine at the specified line number; must have corresponding RETURN statement.

GOSUB expression OF line number [, line number · · · , line number]
Begins executing the subroutine at the first line number if the expression is rounded to 1, at the second line number if the expression is rounded to 2, etc.; must have corresponding RETURN statement.

GOTO line number
Transfers program execution to the specified line number.

GOTO expression OF line number [, line number · · · , line number]
Transfers program execution to the first line number if the expression is rounded to 1, to the second line number if the expression is rounded to 2, etc.

IF expression THEN line number
Expression is logically evaluated; if it is evaluated as 'true', program execution is transferred to the specified line number.

INPUT variable [, variable · · · , variable]
Allows values to be assigned to the variables, from the Model 30 keyboard, during program execution, when a '?' appears on the display.

LET variable = expression or variable = expression
Assignment statement; assigns variable a value.

NEXT simple variable
Marks the end of the corresponding FOR loop.

PRINT [any combination of text and expressions]
Allows text and values to be output on the primary printer; successive expressions must be separated either by commas (for maximum spacing between successive outputs) or by semicolons (for minimum spacing between successive outputs); if no parameters follow PRINT, the printer performs a carriage return–line feed; parameters can also include: TAB expression – so that the following parameter is output beginning at the absolute character position (from Ø through 71) specified by TAB.

READ variable [, variable · · · , variable]
Reads, from the DATA statement (beginning at the current data pointer position), values for the specified variables.

REM [any combination of characters]
Inserts non-executable remarks in a program.

RESTORE [line number]
Resets data pointer (see READ and DATA) to the first constant in the lowest-numbered DATA statement if the line number is not specified; or resets the data pointer to the first constant in the DATA statement with the specified line number.

RETURN [expression]
With no expression specified, RETURN is the subroutine exit, transferring program execution to the line following the GOSUB statement; if an expression is specified, RETURN is the multiple-line function exit, transferring the value of the function to the statement that called it (see DEF FN).

(Continued)

BASIC SYNTAXES
(Continued)

STOP
Terminates program execution but, unlike END, it retains the current position of the program line counter.

WAIT expression
Causes the calculator to halt the specified number of milliseconds; the delay can vary between Ø and 32767 milliseconds.

WRITE (select code, [*] or [line number]) [any combination of text and expressions]
With the '*' specification, WRITE is like the PRINT statement except that any printer can be specified by select code; with the line number specification, the parameters are output according to the specifications in the corresponding FORMAT statement.

[DEG] or [GRAD] or [RAD]
Specifies units for calculating angles; if not specified, RAD is assumed.

[FIXED n] or [FLOAT n] or [STANDARD]
Specifies numerical output form for PRINT and DISP statements; if not specified, STANDARD is assumed.

Chapter 4

PROGRAMMING-RELATED INFORMATION

In Chapter 3, general programming techniques were discussed along with the BASIC language statements. In this chapter, most of the topics discussed emphasize the ease of programming the Model 30. Topics include:

- Standard programming commands
- Time-saving commands
- Program-viewing and editing techniques
- Programming checks
- Calculator memory.

STANDARD PROGRAMMING COMMANDS

The following keys and commands were discussed briefly in Chapter 3:

- END OF LINE which enters program lines into memory.
- RUN / EXECUTE which runs the lowest-numbered program in memory.
- STOP which halts a program that is running unless the program is waiting at an INPUT statement.
- STOP / EXECUTE which halts a program that is waiting at an INPUT statement.
- CONT / EXECUTE which continues a program that was previously halted; if the program was halted by either a STOP command or a STOP statement, the program continues from the line number where it was halted. Otherwise, it continues at the lowest-numbered line in memory.

RUN, STOP, and CONT have additional capabilities as discussed next.

STANDARD PROGRAMMING COMMANDS
(continued)

If there are several programs in memory, any of these programs can be executed by using the RUN command. For instance:

Program #1
```
10 INPUT A,B
20 PRINT A↑2+B↑2
 .
 .
 .
 .
90 END
```

Program #2
```
100 INPUT C,D
110 PRINT C↑2-D↑2
 .
 .
 .
 .
190 END
```

Program #3
```
200 INPUT X,Y
210 PRINT SQR(X↑2+Y↑2)
 .
 .
 .
 .
290 END
```

Pressing:

[RUN] [EXECUTE] executes program #1.

Pressing:

[RUN] [2] [0] [0] [EXECUTE] executes program #3, beginning at line number 200.

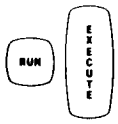
Or pressing:

[RUN] [1] [0] [0] [EXECUTE] executes program #2, beginning at line number 100.

Whenever the RUN command is executed, variables that were assigned values in the previous program become undefined (unless both programs have a COM statement). For instance:

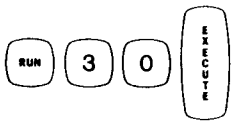
```
10 X=15
20 END
30 PRINT X
40 END
```

If you press:



Lines 10 and 20 are executed and the variable X is set equal to 15.

But then if you press:

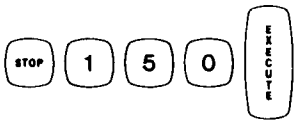


The calculator tries to execute line 30 but cannot since the variable X is undefined. So error 40 results. (The CONT command, discussed on page 4-4, could have been used instead.)

Besides being used to halt a program that is currently running (see page 4-1), the STOP command has another variation. For instance, if the following program is in memory:

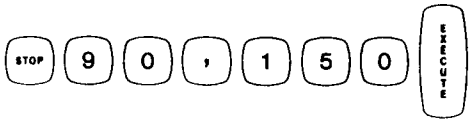
```
10 INPUT X,Y
   .
   .
   .
90 DISP X↑2+Y↑2
   .
   .
   .
150 X=X+Y
   .
   .
   .
200 GOTO 10
```

and the following keys are pressed:



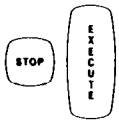
If the program is subsequently run without having any program lines edited, the program will halt at line 150 (that is, line 150 is not executed). If the program is then continued or re-run, and line 150 is again encountered, the program will once again halt there.

Either one or two line numbers can be specified in a STOP command. For instance, the following keys could be pressed:



If the previous program is then run, it will halt at line 90. If it is then continued from this point, it will subsequently halt at line 150. When either of these two lines is encountered, the program halts.

To have the program revert to normal execution when it is again run, either press:



or edit any portion of the program.

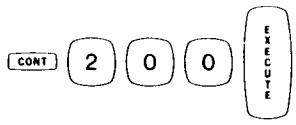
STANDARD PROGRAMMING COMMANDS
(continued)

CONT

When the keys – CONT EXECUTE – are pressed, program execution continues from the point where it was previously halted, if the program was previously halted by either a STOP command or a STOP statement. Otherwise, it continues at the lowest-numbered line in memory.

Unlike the RUN command, which erases the values of variables in memory, the CONT command does not affect variables. Like the RUN command, which can run a program beginning at a specified line number. For instance, pressing:

For instance, pressing:



would continue a program's execution at line 200 without affecting any of the variables previously defined.

The CONT command can also be used to bypass an INPUT statement. For example, suppose you want to input a value for variable Q only after you find the value of another variable, say Z, in the program. Assume you are running the following program:

```
    .
    .
    .
110 INPUT Q
120 PRINT
    .
    .
    .
1200 DISP Z;
1250 GOTO 110
    .
    .
    .
```

By pressing CONT 120 EXECUTE, when '?' appears on the display, the program continues running without actually entering a value for Q.

Similarly, the CONT command can be used in this manner to bypass *selected* variables in an INPUT statement. If the INPUT statement asks for values for Q, R and S, for example, you can enter a number for Q and R and then execute the CONT command to bypass variable S. Be sure the variables you may wish to bypass are the last ones included in the INPUT statement, since CONT immediately branches to, and executes, the specified line.

TIME-SAVING COMMANDS

The following two commands simplify program line numbering.

AUTO#

Each program line must be preceded by a line number. If several program lines need to be keyed in, the AUTO# (Automatic Line Numbering) command can be used to simplify the task. By pressing:

AUTO# EXECUTE The first line number, 10, immediately appears on the display. After this line is keyed in, successive line numbers will be: 20, 30, 40 . . .; that is, a spacing of 10 between consecutive line numbers.

The AUTO# command can specify the beginning line number to appear on the display. If it does, it can also specify the spacing between line numbers. The first number specified is the beginning line number; the second number specified is the spacing. If no spacing is specified, a spacing of 10 is always assumed. Here are two examples:

AUTO# 3 0 EXECUTE The line numbering would be: 30, 40, 50, 60, . . .

AUTO# 5 0 , 5 EXECUTE The line numbering would be: 50, 55, 60, 65, . . .

During automatic line numbering, after one line is entered into memory, the next line number appears on the display. If the line numbering sequence is somehow altered (say, the CLEAR key is pressed), automatic line numbering ceases. In a case such as this, to reinstate automatic line numbering at a specific line number, just use the techniques previously discussed. During automatic line numbering, the DELETE LINE key can be pressed instead of CLEAR. DELETE LINE performs the same function as CLEAR without eliminating automatic line numbering (see DELETE LINE, page 4-8).

REN

Occasionally, it is necessary to change the spacing between line numbers to allow for the insertion of additional program lines. The REN (Renumber) command provides this capability. With this command, all program lines in memory are renumbered. Furthermore, program lines referenced in statements (e.g., GOTO 20) are appropriately renumbered. Here are some examples of the REN command:

If these are the program lines in memory:

1, 2, 5, 6, 7, 8, 111, 112, 113, 114

and you press:

R E N EXECUTE The program lines are renumbered. The lowest line number is 10 with a spacing of 10 between successive line numbers. The ten lines in memory are now numbered:

10, 20, 30, 40, . . . 100

As in the AUTO# command, both the lowest line number and the spacing between consecutive line numbers can be specified. The first number specified is the beginning line number; the second number specified is the spacing. If no spacing is specified, a spacing of 10 is always assumed. Here are two examples:

R E N 3 EXECUTE The new line numbering would be: 3, 13, 23, 33, . . .

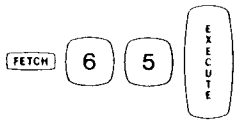
R E N 5 0 , 2 5 EXECUTE The new line numbering would be: 50, 75, 100, 125, . . .

PROGRAM VIEWING

The commands discussed in this section allow the programmer to view individual program statements or to view a series of program statements.

FETCH

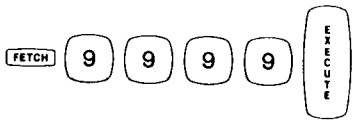
Any program line in memory can be brought to the display if the FETCH command is used. If line 65 is in memory, you can display it by pressing:



Any line can be specified in this manner.

If the line number specified by the FETCH command is not in memory, then the next higher-numbered line is displayed; if there are no lines numbered higher than the one specified, then the highest-numbered line in memory is displayed.

The highest-numbered line in memory is always displayed if you press:



The lowest-numbered line in memory is always displayed if you press:

↓ ↑

Every program line in memory can be viewed by using these keys.

Each time ↓ is pressed, the next higher-numbered program line is displayed. Each time ↑ is pressed, the next lower-numbered program line is displayed.

When either of these keys is pressed, the first line number displayed depends on the last line number that was accessed. For example, if the following lines are in memory:

10, 20, 30, 40, 50

and the FETCH command is used to access line 30:

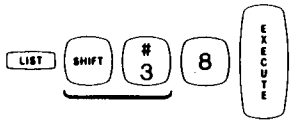
> Then if ↓ is pressed, line 40 is displayed; whereas had ↑ been pressed, line 20 would have been displayed.

But if a program has just been run:

> Pressing ↓, displays the lowest-numbered line in memory; whereas pressing ↑, displays the highest-numbered line in memory.

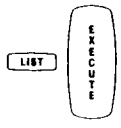
If either of these keys is held down for about two seconds, the operation of the key is rapidly repeated.

With the LIST command, program lines can be output on any compatible printer or paper tape device. Merely specify the select code of the appropriate device; for example:



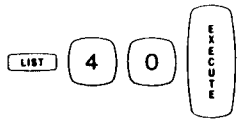
All the program lines currently in memory would be output in line number order on the device with select code #8. The '#' (number sign) is necessary in this case.

However, if you want to output on the primary printer (select code #15), just press:



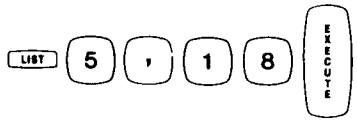
Select code #15 is assumed if no select code is specified. So, in this case, all program lines are output on the primary printer.

Program lines can be selectively output by using the LIST command. For example, if lines 1 through 100 are in memory and you press:



All program lines, numbered 40 and higher, are printed.

Or if you press:



Program lines 5 through 18 are printed.

If only one line number is specified, as in the first case, only the lower boundary of the listing is specified. But if two line numbers are specified (separated by commas), as in the second case, both the lower and the upper boundaries of the listing are specified.

A number always appears on the display after a LIST command is executed. This number is the available calculator memory. The calculator memory is discussed beginning on page 4-12.

PROGRAM EDITING

Programs can be easily modified if the keys and commands, discussed in this section, are used.

BACK FORWARD INSERT

These three keys can be used in the programming mode in the same way that they are used in the calculator mode (see page 2-10). Any time a program line appears in the display, it can be modified by using these keys. After the line is appropriately corrected, press the END OF LINE key to put the corrected version of the line into memory.

> **NOTE**
>
> It makes no difference where the blinking indicator is positioned when the END OF LINE key is pressed. Regardless of the position, the entire program line is entered into memory.

RECALL

This key also can be used in the programming mode in the same way that it is used in the calculator mode. If you try to enter a program line into memory and an error message appears on the display, press RECALL. The line you just keyed in is returned to the display. Appropriate corrections can then be made and the line can be entered into memory by pressing the END OF LINE key.

DELETE LINE

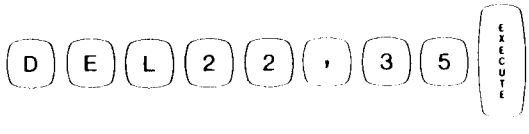
During keyboard operations, if either the CLEAR key or the DELETE LINE key is pressed, the display is erased. The difference between the two keys is evident if a program line that is stored in memory is displayed. With the symbol, ⊢, displayed at the end of the program line, the DELETE LINE key can be pressed both to clear the display and to erase the program line from memory; pressing the CLEAR key clears the display only.

Whenever you are displaying a line previously entered into memory, (e.g., 10 A=6⊢) to erase it from memory, just press the DELETE LINE key.

As mentioned with the AUTO# command, pressing the DELETE LINE key during automatic line numbering operations does not affect the automatic line numbering; it merely erases the rest of the display. However, pressing the CLEAR key erases the entire display, including the automatic line numbering.

DEL

The DEL (Delete) command can erase any number of consecutive lines from memory. If lines 1 through 100 are in memory, lines 22 through 35 can be deleted by pressing:



but if, instead, you press:

D E L 2 2 EXECUTE All program lines, beginning at line 22, are deleted.

To delete just one program line, say, line 82, you can press:

D E L 8 2 , 8 2 EXECUTE

But there is an easier way to delete one line without using the DEL command. Just press:

8 2 END OF LINE and line 82 is deleted. Any time a line number is keyed in, followed immediately by END OF LINE, the specified line is erased from memory.

The SCRATCH key was discussed on page 2-14. The SCRATCH commands that are most useful in programming are as follows:

SCRATCH EXECUTE All variables and program lines are erased from mainline memory. (Information on Special Function keys, see Chapter 6, is not affected.)

SCRATCH A EXECUTE All of memory is erased as in turning the power off then on again. Before a large program is entered into memory, this command ensures that the maximum amount of memory is available.

PROGRAMMING CHECKS

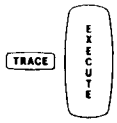
The techniques discussed in this section allow the programmer to make various checks during program execution to ensure that the program is operating correctly.

TRACE NORMAL

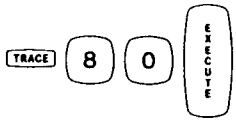
The TRACE command is used to follow the order of statement execution in a program. If the TRACE key is pressed during program execution, the program line numbers are printed in the order in which they are accessed; if the NORMAL key is then pressed while the program is still running, the TRACE command is nullified. So during program execution, both TRACE and NORMAL are 'immediate execute' keys.

If the program is not currently running, the TRACE command has the following capabilities.

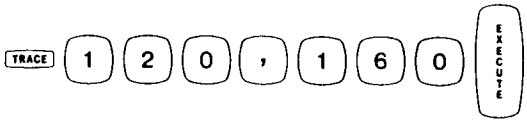
- It can be set to trace the order of statement execution when the program is again run. Just press:



- It can be set to trace from a specific line number, if the lowest line number to be traced is specified. For instance, with the following command, all lines numbered 80 and higher will be traced when the program is again run:



- It can be set to trace groups of line numbers, if both the lowest and the highest line numbers to be traced are specified. For instance, with the following command, all lines numbered 120 through 160 will be traced when the program is again run:



While the program is running, any of these TRACE commands are immediately nullified if the NORMAL key is pressed. But if the program has been halted with TRACE still in effect, TRACE is nullified if you press:

STEP

If a program is not running, it can be continued one line at a time if the STEP command is used. STEP is always an 'immediate execute' key. As soon as it is pressed, the line designated by the internal program line counter is executed, then the program halts again. When the program halts, the current position of the program line counter is displayed; that is, if STEP is pressed again, the line specified in the display will be executed.

For example, suppose the following program is run:

```
10 Y=SINPI/2
20 DISP Y
 .
 .
 .
60 STOP
70 DEG
80 Z=SIN90
90 DISP Y,Z
 .
 .
 .
200 END
```

This program halts after line 60, the STOP statement, is executed. So the program line counter is positioned at line 70. If [STEP] is then pressed, only line 70 is executed. Each time [STEP] is pressed, only the line designated by the program line counter is executed. An entire program can be run this way. (The next section, Checking a Halted Program, shows why it can be desirable to execute a program step-by-step.)

For Advanced Programmers

In the previous program, after line 200 is executed, the program line counter reverts to line 10. If the STEP command (or the CONT command) is then used to continue program execution at line 10, the program does not go through an initializing phase; that is, variables retain their current values, operating modes (like calculating angles in radians) are not reset, etc. With the RUN command, this initializing phase is automatic. So, in this example, if STEP is used after line 200 is executed, the angle, Y, is calculated in degrees since line 70 previously set this mode.

CHECKING A HALTED PROGRAM

Various operations can be performed on a halted program:

- Values of variables can be checked merely by keying in the variable name, followed by EXECUTE.
- Values of variables can be assigned or changed if commands like A=7 are executed.
- Many program statements can be keyed in as executable commands; that is, without line numbers and followed by the EXECUTE key:

  PRINT statements
WRITE statements
DISP statements } to immediately output specified information.

  GOTO statements
GOTO with OF statements } to reset the program line counter to a specified line.

  RESTORE statement – to reset the data pointer to the first data element.
- Any keyboard command can be executed: degrees, grads, radians, fixed n, float n, standard, etc. Also, arithmetic operations can be performed.

(continued)

PROGRAMMING CHECKS
(continued)

If the halted program is continued with either the CONT or the STEP command, any of the previously mentioned operations that affect the program execution remain intact. For instance: values of variables that were changed retain their new values; a GOTO command causes the program to continue at the specified line number; a DEG command causes the program to calculate angles in degrees; etc.

If, however, the halted program is restarted with a RUN command, then the program is initialized and the calculator reverts to all standard operating modes.

Furthermore, if any program lines are inserted, deleted, or edited while the program is halted, the program line counter is repositioned at the lowest numbered line in memory.

CALCULATOR MEMORY

In the Model 30, the total amount of user memory is expressed in 'words'. (For the computer oriented, there are 16 bits — two bytes — per word.) The basic calculator (with no memory options) has 1760 words of available memory.

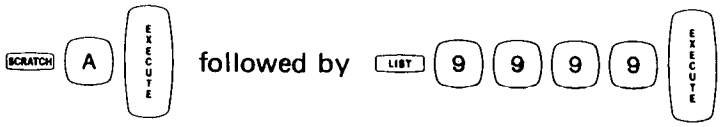
Program lines take up a variable number of words with as few as three words allocated in some lines, such as:

```
50 STOP          120 PRINT          800 END
```

Methods for determining the available memory are discussed next.

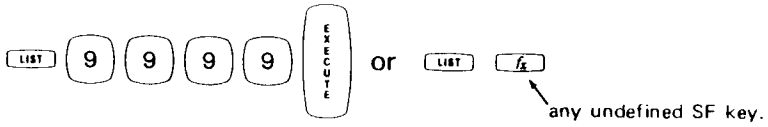
LIST

As mentioned earlier in this chapter, the LIST command always displays the available user memory after it performs a program listing. To determine the maximum amount of user memory, it is necessary first to erase everything in memory (thus making all of memory available) and then to perform a LIST command. So if you press:



the number that appears on the display is the total calculator memory. (Any memory options that have been installed are also included in this number.)

If information is in memory and you want to determine how much memory is still available, it is generally easiest to press:



The number that immediately appears on the display is the available memory. Whenever the number specified in the LIST command is greater than the largest line number currently in memory, no program lines are listed and the available memory is immediately displayed. So by using LIST 9999, you need not be concerned about the line numbers currently in memory. For no line number greater than 9999 can be specified.

For Advanced Programmers

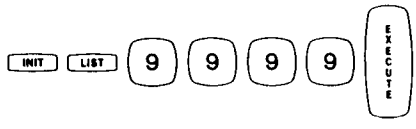
The LIST command gives totally different results when executed both before and after a program is run:

- Prior to running a program
  - without a COM statement, the file size needed to store the program on tape is determined by subtracting the LIST 9999 result from the total user memory. (Tape cassettes are discussed in Chapter 5.)
  - with a COM statement, the file size determined by the previous method minus the memory reserved for variables in the COM statement is the true file size needed to store the program. (Remember, each full-precision element in common has four words of memory allocated to it.)
- After running a program, when LIST 9999 is again executed, the available memory being displayed, subtracted from the total user memory, determines the approximate† amount of memory needed to run this program.

INIT

The INIT (Initialize) key resets the calculator to the state it was in immediately prior to program execution unless the program has a COM or DIM statement in it. Furthermore, program variables, not defined in a COM statement, are erased when the INIT key is pressed.

If a program is executed and a LIST 9999 command is then executed, the approximate amount of memory needed to run the program is determined by subtracting the number on the display from the total calculator memory. If this program has neither a COM nor a DIM statement in it, the file size needed to store the program on tape can then be determined if you press:



and subtract the number on the display from the total calculator memory.

Other uses of the INIT key are discussed in the following section.

† This is usually very close to the amount of memory actually needed. The difference exists because, during program execution, some memory is temporarily needed to run different parts of the program.

ADDITIONAL COMMANDS

The three commands discussed in this section provide the Model 30 with additional programming capabilities. Their uses are intended primarily for the advanced programmer.

INIT

The INIT (Initialize) key was discussed in the previous section as a tool for determining program memory. Other uses for this key are discussed here.

When INIT is pressed, it performs identically to the RUN command except that the program is not executed:

- All variable values, aside from those specified in a COM statement, are erased.
- All normal programming modes are reset — angles are calculated in radians, outputs appear in standard notation.
- Memory is reserved for all elements in a DIM statement. (Memory for the elements in a COM statement is immediately reserved when COM is entered into memory.)
- The memory reserved for DIM and COM statement elements can be accessed. (Although memory is reserved for COM before the INIT key is pressed, this portion of memory cannot be accessed until either the INIT or RUN command is given.)

Since the INIT command performs this way, it offers the following additional feature when it is pressed:

> The programmer has the option of inputting values for some or all of his program variables before the program is run. After the required data [say, A(1)=6. . .A(12)=8, etc.] is keyed in, the program can be executed by pressing:
>
> CONT EXECUTE (Remember the RUN command would erase the values just assigned to the variables.)

NOTE

The INIT command is unnecessary if only simple variables are being input and if the program has neither a COM nor a DIM statement.

PTAPE

The PTAPE command allows the calculator to read in program statements from an input device, ASCII† character by ASCII character. Compatible devices include the -hp- 9863A Tape Reader, the -hp- 9869A Card Reader, the -hp- 2748A Optical Tape Reader, and the -hp- 11205A Serial Interface (with teletype).

Just key in either PTAPE# or PTA# followed by the appropriate select code for the input device. Then press the EXECUTE key.

If the device with the specified select code is not connected to the Model 30, the calculator waits until it is attached to complete the command. During this time the display will be blank. You can, of course, regain immediate control of the calculator by pressing the STOP key.

† ASCII (American Standard Code for Information Interchange) is understood by the Model 30.

During the implementation of this command, each line being loaded into memory has its syntax checked; if a line is in error, it will be rejected – thus, only those lines with correct syntax are loaded into the calculator. To obtain a record of the rejected lines, it is necessary to put the calculator in the print-all mode prior to executing the PTAPE command; in print-all mode, all rejected lines are printed.

> **NOTE**
>
> To punch information onto paper tape, use the LIST command as discussed earlier.

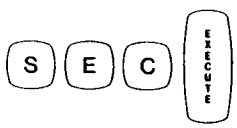
SEC

The SEC (Secure) command has the capability of concealing program lines from potential users; that is, your program could be given to another person who could load it into the calculator from a cassette file and then run it – however, he would not be able to view particular program lines nor could he store the program on any other cassette file. (Tape cassettes are discussed in the next chapter.)

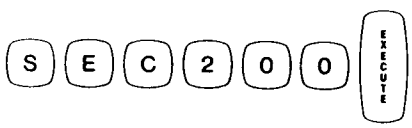
Any attempt to display or list a secured program line results in the line number appearing, followed by an asterisk (*).

The SEC command has the following three variations:

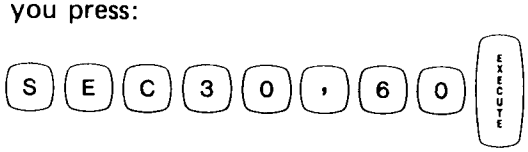
- All program lines in memory can be secured if you press:



- All program lines beginning at a specified line number can be secured. For example, all program lines beginning at line 200 are secured if you press:



- Program lines can be selectively secured. For example, program lines 30 through 60 are secured if you press:



Since a program can have both secured and unsecured program lines, on a program listing only the secured lines appear with an asterisk following the line number.

When any part of a program is initially secured, the program can still be reproduced onto as many cassette files as necessary. However, once the program is erased from memory, (even though it can be loaded back into memory) no portion of it can be reproduced onto any other files – not even an unsecured portion.

> **NOTE**
>
> When any program lines are secured, the entire calculator is in a 'secured mode'. Therefore, after the secured program is stored away, the user should erase all of memory before inputting other programs – thus, avoiding 'secured program' errors.

Table 5-1. Typical Cassette Storage Capacities

File Size (words)	Maximum* No. of Files	Typical No. of Files	Typical Cassette Capacity (words)
4	714	925	3700
25	457	600	15000
50	320	425	21250
75	246	335	25125
100	200	270	27000
200	114	155	31000
300	80	105	31500
400	61	80	32000
500	50	64	32000
600	42	53	31800
700	36	45	31500
800	32	40	32000
900	28	35	31500
1000	25	32	32000
1500	17	21	31500
2000	13	16	32000
2500	10	12	30000
3000	8	10	30000
3500	7	9	31500
4000	6	8	32000
4500	5	7	31500
5000	5	6	30000
5500	4	5	27500
6000	4	5	30000
6500	4	4	26000
7000	3	4	28000
7500	3	4	30000
8000	3	3	24000

CAUTION
THE CASSETTE MEMORY SERVICE WARRANTY DOES NOT COVER DAMAGE WHICH RESULTS FROM THE USE OF A TAPE CASSETTE NOT SUPPLIED BY HP.

*Maximum number of files should be used to ensure compatability of one cassette in any 9830A Calculator. This number is derived by multiplying the file length by 1.5 and adding 50 words. The sum of this result for all files on the cassette cannot exceed 40,000 words.

Chapter 5

USING A TAPE CASSETTE

The tape transport, built into the Model 30, provides the calculator with considerable flexibility as described in this chapter. Cassette commands are described beginning on page 5-4.

THE TAPE CASSETTE

The -hp- tape cassette used with the Model 30 is a precision unit, containing 300 feet of digital-quality, magnetic recording tape. These and other important characteristics make this tape cassette ideally suited for use with the calculator.

SPECIFICATIONS

Search Speed: Approximately 130 ft/min (search is bi-directional).

Data Storage: See Table 5-1.

OTHER CASSETTES

Although many other manufacturers' tape cassettes will initially work with the Model 30, many of these products will not make reliable recordings. Also, since the use of some tape cassettes will actually damage the tape transport, you should be cautious about using other cassettes.

The tape cassette must be a high-quality digital recording tape with transparent leader at both ends. The cassette case must be white or a light color. For additional information on requirments for a tape cassette, contact your nearest HP Sales and Service Office listed in the back of this book.

INSERTING TAPE CASSETTES

To open the transport door, flick the switch on the upper right-hand corner of the calculator keyboard. Then insert the tape cassette by sliding it through the guide posts on the transport door; be sure the cassette is right-side up with the FRONT label facing you. Finally, push the door closed.

Whenever you want to remove the cassette from the transport, press the key REWIND first. This fully rewinds the tape to clear-leader, thus shielding the recorded portion of the tape against dirt or damage.

PROTECTING CASSETTES

The information recorded on a tape cassette can be protected (that is, further recording is not allowed) if you remove both tabs that are on top of the cassette – see Figure 5-1.

If only the left tab is removed, the tape cassette is still essentially protected. But, in this case, it would still be possible for someone to insert the tape cassette backwards into the tape transport and record over previously recorded information.

(Continued)

THE TAPE CASSETTE
(Continued)

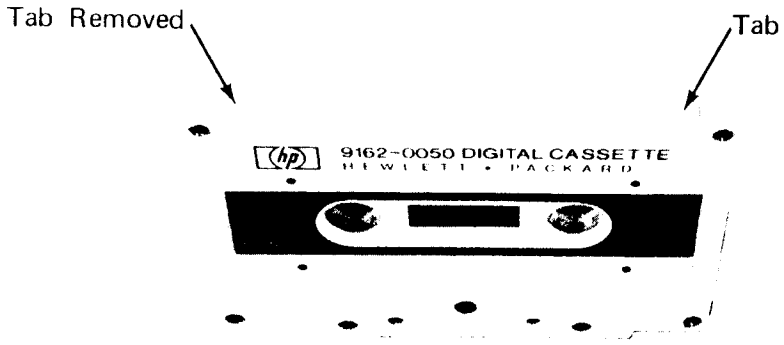


Figure 5-1. HP Tape Cassette

STORING CASSETTES

Since magnetic tapes are easily damaged, be sure to put each tape cassette in its plastic case when you have finished using it. Also, to keep the tape clean, be sure it is fully rewound (on clear-leader) before you remove it from the tape transport.

As with most magnetic tape products, the information recorded in the tape cassette can be altered or destroyed if it is exposed to a strong magnetic field, such as one produced by a bulk tape eraser, a toy magnet, or a metal detection device (like the ones used in many airports). If you keep your tape cassettes in a metal container, such as a card index box, they will be protected from most magnetic fields.

CLEANING THE TAPE HEAD

To ensure the reliability of tape cassette operations, it is recommended that the tape head be cleaned after every eight hours of cassette operations. Furthermore, it is always a good idea to clean the tape head before making important cassette recordings.

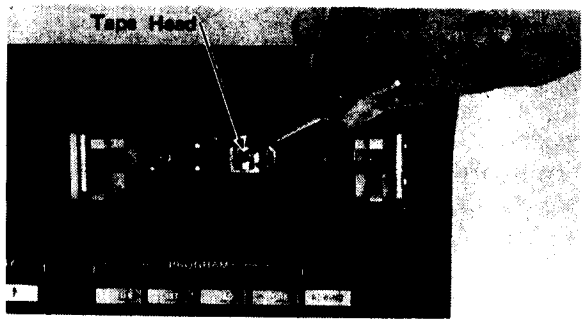


Figure 5-2. Cleaning the Tape Head

The tape head is easily cleaned as follows:

1. Open the transport door and remove the tape cassette.
2. Clean the tape head (see Figure 5-2) with a cotton applicator that has been dampened with head cleaning solution. Just wipe the top of the tape head a few times with the cotton applicator. Remove any other dust that has accumulated in the vicinity of the tape head.
3. Close the transport door. It's a good practice to keep the transport door closed whenever possible to prevent excess dust from accumulating in the transport.

In conclusion, special care must be taken to ensure reliable tape cassette operations. Always:

- keep the transport door closed whenever possible to prevent dust from accumulating in the transport.
- clean the tape head after every eight hours of cassette operations and before making important cassette recordings.
- make duplicate (spare) tapes of important programs or data.
- either remove the cassette from the transport or verify that the tape is positioned on clear-leader when switching the calculator ON.
- protect the tape cassette from scratches, dust and magnetic fields, like those associated with high-voltage electrical equipment. This includes the rear panels of input and output calculator devices.

TAPE FILE STRUCTURE

The tape is organized by files, which are established by the MARK command (MARK is discussed beginning on page 5-6). The first file on a tape is file Ø; subsequent files are identified as file 1, file 2, etc. Both the number of files and the lengths of the files (in words) can be designated in a MARK command.

The tape structure is as follows:

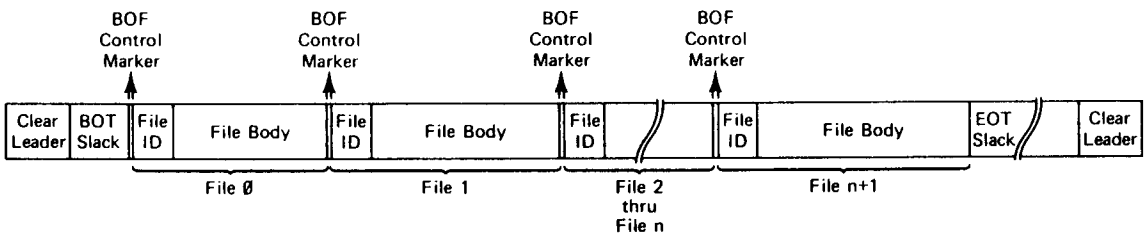


Figure 5-3. Tape Structure

Clear-Leader: Both the beginning and end of the tape have a clear-leader so that the calculator will know when the end of the tape is reached.

BOT Slack: The beginning of tape (BOT) has a small slack (unused) area to ensure that file Ø is read correctly during cassette operations.

BOF Control Marker: The beginning of each file (BOF) has a control marker. This control marker separates consecutive files.

File ID: The file identifier contains information like the file number, the file type (program, data, etc.), the absolute file size (in words), the current file size (number of words currently in use), etc. The 13 words used in the file ID are additional to, and separate from, the words specified by the user in the MARK command.

File Body: The file body is the portion of the file used to retain information. The absolute file body size is determined by the MARK command specifications.

EOT Slack: The end of tape (EOT) has an unused area the size of which depends on the number of files marked and the length of each file.

CASSETTE COMMANDS

The cassette commands are briefly described below:

- MARK – establishes files with specified lengths.
- STORE – reproduces, onto tape, programs that are in memory.
- LOAD – reproduces, into memory, programs that are on tape.
- LINK – works like LOAD but, in addition, it retains variables currently in memory.
- MERGE – inserts, between program lines currently in memory, program lines that are on tape.
- FIND – locates a specific tape file.
- REWIND – rewinds the tape to clear-leader.
- STORE DATA – reproduces, onto tape, data that is in memory.
- LOAD DATA – reproduces, into memory, data that is on tape.
- STORE KEY – reproduces, onto tape, information that is on Special Function keys (see Chapter 6).
- LOAD KEY – reproduces, into the Special Function keys, information that is on tape.
- LOAD BIN – reproduces, into memory, specially written programs that are on tape.
- TLIST – prints information about each file.

These commands are discussed individually beginning on page 5-6).

PROGRAMMABILITY

All tape commands are programmable as well as being keyboard executable. The same syntax is used in either case. Of course, program statements are preceded by a line number and are entered into memory if the END OF LINE key is pressed; whereas keyboard commands are executed if the EXECUTE key is pressed.

Three cassette commands have dedicated keys .that can be used when executing the commands; i.e., [STORE], [LOAD], [REWIND]. However, if any of these commands are to be used in a program, the commands must be keyed in, letter by letter.

SYNTAX

The terminology shown below appears in the command syntaxes:

array name	– the letter used to define the array (A through Z);
file	– the number of the tape file;†
1st line number	– the first line number designated;
2nd line number	– the second line number designated (it can appear only if the first line number is designated);
length	– the length of the designated files;†
no. of files	– the number of files designated.†

†Can be a constant, a variable, or an expression.

The following conventions are also used:

brackets [ ] — to indicate optional items;

coloring — to indicate that the colored item must appear as shown.

Here's an example using the complex version of the STORE command. Once you understand this syntax, you should have no trouble with any of the syntaxes.

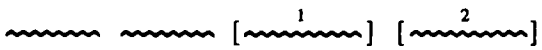
Syntax:

STORE file[, 1st line number[, 2nd line number]]

Let's look at the syntax, step-by-step, from left to right:

1. The word, STORE, is necessary to identify the command.
2. The 'file' number is needed to specify the file that will record the program.
3. The '1st line number' is optional; but if it is specified, then a comma is needed to separate the 'file' from the '1st line number'.
4. The '2nd line number' is optional, too; but for it to be specified, the '1st line number' must also be specified; if both are specified, then a comma is needed to separate the two items.

Since the brackets are nested in this command, for the most deeply nested item to be specified, all other items must be specified, too. If, however, the brackets had appeared as:



then this dependency between items would not exist; that is, the information within the second bracket could be specified without having the other bracketed information specified.

The syntax requirements for each command are shown individually in the tape command discussions and are shown all together on page 5-25.

MARK

The MARK command is used to establish tape files. Both the number of files to be established and the lengths of the files (in words) are specified in this command.

Syntax:

MARK no. of files , length

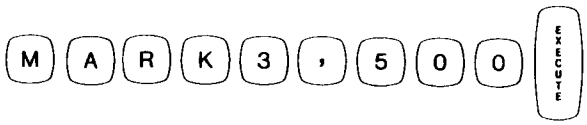
When the MARK command is executed, files are marked beginning at the tape's current location. If the tape is rewound, the marking begins at file Ø; but if the tape is located at, say, file 3, then the marking begins at file 3.

The tape can be located at any file if the FIND command (see page 5-16) is used.

MARKING NEW TAPES

To mark a new tape, first be sure that the tape is rewound by pressing the key, REWIND. Then mark the number of files you want with the lengths that you want.

If you want, say, three files marked, each with 500 word lengths, press:



In about 30 seconds the symbol, ⊢, appears on the display, indicating that the operation is completed. File Ø, file 1, and file 2 have now been marked. Each of these files can record information up to 500 words in length.

An extra file, file 3, is also marked. There are two reasons for the extra file:

- The BOF (beginning of file) control marker (see Figure 5-3, page 5-3) for each file serves as an EOF (end of file) control marker for the previous file.
- In a FIND command, the extra file can be accessed; so new files can be marked beginning at this point. (Had the extra file not been marked, another MARK command would have had to re-mark file 2.)

The extra file should not be used to record information, however, since any information in this file will be erased when more files are marked.

A new tape can also be marked with several files of varying lengths if successive MARK commands are used. First, be sure that the tape is rewound. Then to mark, say, two files with 300 word lengths, four files with 1000 word lengths, and one file with a 2500 word length, execute the following:

1. MARK 2,300: In about 30 seconds file Ø, file 1, and the extra file (file 2) are marked. Then if you execute:
2. MARK 4,1000: In about 60 seconds file 2, file 3, file 4, file 5, and the extra file (file 6) are marked. This MARK command takes considerably longer to execute since much more tape is marked. Then if you execute:
3. MARK 1,2500:† In about 70 seconds file 6 and the extra file (file 7) are marked.

† Of course, the absolute size of a recording is limited by the memory size of the calculator; so, if the memory size of your calculator is 1760 words, it would be pointless to mark a file with a length of 2500 words.

The pictorial below illustrates these MARK commands:

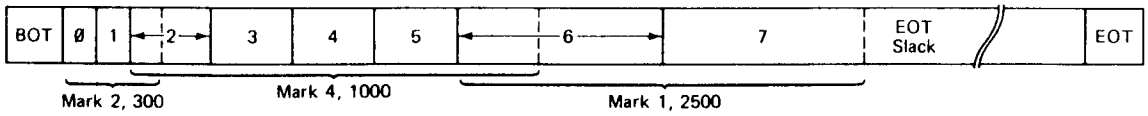


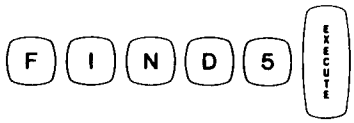
Figure 5-4. Marking Successive Files

Notice that the extra file marked by each command is temporary, as indicated by the dotted file line. The extra file is replaced by a new file each time another MARK command is executed. In this case, file 7 is the remaining extra file. Additional MARK commands can be executed from this point.

MARKING A USED TAPE

The MARK command is the same for both new and used tapes. But for a used tape, it is necessary to position the tape at the file where the marking is to begin. This is accomplished by using the FIND command (discussed on page 5-16).

If the tape is still marked as shown on Figure 5-4 in the previous section, file 5 is located if you press:



You can now mark files beginning at this point. If, however, you wish to retain any recorded information that is on file 5 or file 6, then a FIND 7 command is preferable. This locates the tape at the extra file, file 7; new files can be marked at this point without destroying any previously recorded information (unless you have made the mistake of recording on the extra file).

NOTE

Whenever marking a used tape, begin marking at a point from which you do not need any of the previously recorded information. For the contents of all previously used files from this point onward will be either destroyed or altered (even if only a small portion of the tape is re-marked).

If you are uncertain of the number of marked files, the lengths of the files, the information in the files, etc., use the TLIST command as discussed on page 5-23, to identify your tape files.

STORE

The STORE command is used to record, onto tape, programs that are in memory. The contents of memory are not altered by this command. If you want to record all program lines that are in mainline memory†, the syntax is simply:

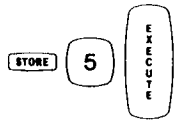
STORE [file]

The file specification is optional; for if no file is specified, file Ø is assumed. Since file Ø is the most easily accessible file, it is often convenient to use file Ø as a 'scratch-pad' file.‡

So if you press:

All program lines in memory are recorded in file Ø.

Or if you press:



All program lines in memory are recorded in file 5.

The LOAD, LINK, and MERGE commands (discussed later) also assume file Ø if no file is specified.

For more sophisticated applications, the syntax is:

STORE file [, 1st line number [, 2nd line number]]

1st line number: The first line number specified is the lowest-numbered line in memory that you want recorded. All higher-numbered lines are also recorded unless the 2nd line number is also specified.

2nd line number: The second line number specified is the highest-numbered line in memory that you want recorded. All lines between the specified line numbers are also recorded.

If the cassette is not protected (as discussed on page 5-1), a program can be stored in any file that has a length greater than or equal to the size of the program – in words.

When a program is stored into a tape file, everything previously in that file is erased.

Examples

For the following examples, assume lines 10, 20, 30, . . . 100 are in memory. If so the following commands could be executed:

STORE 3: All program lines are recorded in file 3.

† The term, 'mainline memory', is used here to emphasize that programs on Special Function keys are not stored unless you are in the KEY mode (see page 6-7).

‡ A 'scratch-pad' file is used to retain temporary information.

STORE 5, 40: All program lines, beginning at line 40, are recorded in file 5. (To specify the 1st line number, you must also specify the file.)

STORE Ø, 40: All program lines, beginning at line 40, are recorded in file Ø. (Even though file Ø is used here, it must be designated since the 1st line number is designated.)

STORE 2, 20, 80: All program lines, from line 20 through line 80, are recorded in file 2. (To specify the 2nd line number, you must also specify the 1st line number.)

STORE X+3, 60: All program lines, beginning at line 60, are recorded in the file indicated by the expression, X+3. So if X=4, the program lines are recorded in file 7.

NOTE

Do not confuse the STORE key with the LOAD key. If, by mistake, you execute a LOAD command instead of STORE, you could erase the program in memory. Likewise, if, by mistake, you execute a STORE command instead of LOAD, you could erase the program on tape.

LOAD

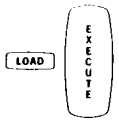
The LOAD command takes programs that are stored on tape and reproduces them into the calculator memory.

To replace the information in memory by the program lines in a specific file, you can use the following syntax:

LOAD [file]

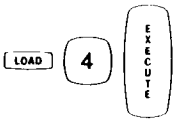
The file specification is optional; if no file is specified, file Ø is assumed.

So if you press:



The program lines from file Ø are loaded into memory. Information previously in memory is erased.

Or if you press:



The program lines from file 4 are loaded into memory. Information previously in memory is erased.

For more sophisticated applications, the syntax is:

LOAD file [, 1st line number [, 2nd line number]]

Whenever the LOAD command is executed, all program lines in the specified tape file are reproduced into memory. All program lines previously in memory are erased unless the 1st line number is specified.

Whenever the 1st line number is specified, the reproduced program lines are renumbered with the beginning line number corresponding to the specified 1st line number. (The spacing between consecutive line numbers remains the same; furthermore, statements, like GO TO 30,† are appropriately renumbered.) Program lines previously in memory, with line numbers lower than the 1st line number, are retained; all other lines previously in memory are erased.

In the calculator mode (after the program is loaded into memory):

- If the 2nd line number is not specified, the calculator halts.
- If the 2nd line number is specified, program execution begins at this line number.

In the programming mode (in this case, specifically after the LOAD command is executed during program execution):

- If the 2nd line number is not specified, program execution is 'restarted' either with:
  - the program line immediately following the LOAD command in the original program, or with

† A GOTO statement in a tape file must refer to a line in the file for the renumbering to be successful; if the GOTO statement on tape refers to a program line currently in memory, an error will occur when the renumbering is attempted.

  - the first line of the loaded program; that is, if there were no lines after the LOAD command in the original program, or if the lines were destroyed by the LOAD command.

- If the 2nd line number is specified, program execution is 'restarted'† with this line number.

Examples

For the following examples, assume that lines 5, 15, 25, . . . 95 are originally in memory and that lines 10, 20, 30, . . . 100 are in file 4.

LOAD 4: The program lines from file 4 replace all the program lines previously in memory.

LOAD 4, 10: All program lines from file 4 are loaded into memory. The first line loaded into memory is assigned line number 10 and the other lines are appropriately renumbered (however, in this example, the first line loaded into memory was already line number 10 – so no renumbering is necessary). Only the program lines previously in memory, with line numbers lower than 10, are retained; so in this example, line 5 is retained. Therefore, the lines in memory would be: 5, 10, 20, 30, . . . 100.

LOAD 4, 30: This is like the previous example except that, in this case, the renumbering is necessary since '30' does not correspond to the first line loaded into memory. After this command is executed, the lines in memory are: 5, 15, 25, 30, 40, 50, . . . 120. Notice that lines 5, 15, and 25 are retained.

LOAD 4, 30, 5: This is like the previous example except that, in this case, after the command is executed, program execution begins at line 5.

The following examples use LOAD in the programming mode:

#1

in memory	in file 7
5 X=3	10 A=5
15 Y=4	20 B=6
.	.
.	.
.	.
95 LOAD 7	100 END

When line 95 is executed, the program in file 7 replaces all program lines previously in memory. The new program then begins execution at line 10.

#2

in memory	in file 2
5 COM X,Y	10 COM X,Y
15 X=3	20 Z=SQR(X↑2+Y↑2)
25 Y=4	30 PRINT "Z="Z
.	.
.	.
.	.
95 LOAD 2	100 END

This is like the previous example, except that both programs have a COM statement. So, even though the original program is replaced by the program in file 2, variables that are 'common' to both programs (via the COM statement) retain their current values.

† The term, 'restarted', is used here because, during program execution, the LOAD command functions as though – RUN EXECUTE – had been pressed: variables (except for those defined by COM) become undefined, operating modes like STANDARD and RAD (radians) are reset, etc.

LOAD
(Continued)

#3

in memory

```
5 COM X,Y
15 X=3
25 Y=4
 .
 .
 .
95 LOAD 8,10
```

in file 8

```
10 Z=X+Y
20 DISP Z
 .
 .
 .
100 END
```

after LOAD command

```
5 COM X,Y
10 Z=X+Y
20 DISP Z
 .
 .
 .
100 END
```

When line 95 is executed, the program from file 8 is loaded into memory beginning at line 10. The COM statement, line 5 in the original program, is retained since this line is numbered lower than line 10. Since the COM statement is retained, the values of the variables, X and Y, are also retained.

#4

in memory

```
5 D=6
15 E=7
25 LOAD D-2,50
35 D=3
45 E=E+1
55 P=1
 .
 .
 .
95 END
```

in file 4

```
10 M=5
20 N=6
 .
 .
 .
100 END
```

after LOAD command

```
5 D=6
15 E=7
 .
 .
 .
45 E=E+1
50 M=5
60 N=6

140 END
```

When line 25 is executed, the entire program in file 4 is loaded into memory and renumbered beginning at line 50. All lines previously in memory that are numbered lower than line 50 are retained. Program execution is then restarted at line 35 since this line immediately follows the LOAD command. [In this example, however, error 40 (undefined variable) would occur in line 45 since the variable (E) no longer has the value assigned to it in line 15; the LINK command, described in the following section, could have been used here instead of LOAD.]

#5

in memory

```
5 Q=7
15 R=8
25 S=Q↑3-R
 .
 .
 .
95 LOAD 6,20,5
```

in file 6

```
10 S=Q↑3+R
 .
 .
 .
90 GOTO 10
100 END
```

after LOAD command

```
5 Q=7
15 R=8
20 S=Q↑3+R
 .
 .
 .
100 GOTO 20
110 END
```

When line 95 is executed, the entire program in file 6 is loaded into memory and renumbered beginning at line 20. [Notice that the number referenced in the GOTO statement (line 90, file 6) is also appropriately renumbered.] Since lines 5 and 15 from the previous program are lower than line 20, they are retained. Program execution is then restarted at line 5.

If a COM statement is in the original program and if a LOAD command erases COM, the variables specified in COM can no longer be accessed; but a portion of memory is still

allocated to the 'common' variables. Then if another program, having COM, is loaded into memory, the 'common' variables are once again accessible.

Increasing Effective Calculator Memory

As shown in the previous examples, LOAD can be used to chain programs together, piece by piece. With a COM statement in each program segment, only the variable values needed in each program segment are retained, and the program lines in the previous segment can be erased. So a maximum of memory is always available. Therefore, it is possible to execute, a portion at a time, programs that would otherwise be too large to execute on the Model 30.

LINK

Like the LOAD command (see previous section), the LINK command takes programs that are stored on tape and reproduces them into the calculator memory.

LINK and LOAD have the following differences when executed:

- With LOAD, the values of variables not referenced in COM statements become undefined; whereas with LINK, all variable values are retained.
- With LOAD, the calculator reverts to normal operating modes, like RAD and STANDARD; whereas with LINK, the current operating modes, say, DEG and FIXED 2, are retained. (But the data pointer is reset with either LOAD or LINK — see DATA statement, page 3-18.)

The LINK syntaxes parallel the LOAD syntaxes:

LINK [file]
or
LINK file [, 1st line number [, 2nd line number]]

The specifications in the LINK syntax follow the identical rules as the specifications in the LOAD syntax.

There is no key for the LINK command, so the word, LINK, must always be typed in. Since LINK is most useful in the programming mode, the following examples show its practical uses.

Examples

#1

in memory

```
5 X=3
15 Y=4
25 FIXED 3
 .
 .
 .
95 LINK 2
```

in file 2

```
10 Z=Y↑2-X↑2
20 PRINT X,Y,Z
 .
 .
 .
100 END
```

When line 95 is executed, the program on file 2 replaces all program lines previously in memory. This program then continues execution with line 10. The variables, X and Y, defined in the original program, retain their values. Also, operating modes that were defined in the original program (like FIXED 3) are retained. Had LOAD been used instead of LINK, the variables, X and Y, would have been undefined; and the operating mode, FIXED 3, would have reverted to STANDARD.

(Continued)

LINK
(Continued)

#2

in memory

```
5 Y=-1
15 Y=Y+1
25 LINK Y,30,30
```

in file Ø

```
30 R=16
40 T=EXP(R)
 .
 .
 .
100 GOTO 15†
```

after LINK command

```
5 Y=-1
15 Y=Y+1
25 LINK Y,30,30
30 R=16
40 T=EXP(R)
 .
 .
 .
100 GOTO 15
```

When line 25 is executed, the program in file Ø (since the variable, Y, equals Ø) is linked to the existing program beginning at line 30. Program execution then continues at line 30. The last statement in this program, 100 GOTO 15, returns the program to line 15 where the variable (Y) is incremented by 1. When line 25 is again accessed, the program in file 1 will then be linked to the current program in memory.

† This statement is acceptable since renumbering does not occur in this example.

MERGE

The MERGE command takes program lines from a tape file and positions them in memory in front of the program currently there, between consecutive lines in the program currently there, or behind the program currently there.

The MERGE command always retains the program lines previously in memory.

The following simplified syntax can be used:

MERGE [file]

The file specification is optional; if no file is specified, file Ø is assumed.

Examples

In these examples assume that the following six program lines are in memory:

10, 20, 80, 90, 100, 110

- If the program lines in file 7 are 30, 40, 50, 60, and 70, then the following command could be executed:

  MERGE 7: This inserts program lines between lines 20 and 80 of the program currently in memory. Then the program lines in memory would be 10, 20, 30, . . . 110.

- If the program lines in file 8 are 1, 2, 3, . . . 9, then the following command could be executed:

  MERGE 8: This inserts program lines in front of the program currently in memory. Then the program lines in memory would be 1, 2, 3, . . . 9, 10, 20, 80, 90, 100, 110.

- If the program lines in file 9 are 115, 125, . . . 205, then the following command could be executed:

  MERGE 9: This inserts program lines behind the program currently in memory. Then the program lines in memory would be 10, 20, 80, 90, 100, 110, 115, 125, . . . 205.

- If the program lines in file 10 are 85, 95, 105, 115, the command – MERGE 10 – could not be executed since the line numbers in the two programs cannot be interwoven; any attempt to do so results in an error.

- If the program lines in file 11 are 20, 30, . . . 70, the command – MERGE 11 – could not be executed; for if any line number on the file matches a line number currently in the program (in this case, line number 20), an error occurs.

For more sophisticated applications, the syntax is:

MERGE file [, 1st line number [, 2nd line number]]

The syntax specifications are much like those for LOAD and LINK. If the 1st line number is specified, the program lines on the specified file are renumbered beginning with the 1st line number. The spacing between renumbered lines remains the same.

In the calculator mode:

- If the 2nd line number is not specified, the calculator halts after the program lines are merged.
- If the 2nd line number is specified, program execution begins at that line after the program lines are merged.

In the programming mode (that is, after the program lines are merged):

- If the 2nd line number is not specified, program execution is restarted† with the line immediately following the MERGE command.
- If the 2nd line number is specified, program execution is restarted† at that line.

Examples

For the following examples, assume that the program lines currently in memory are 70, 80, 90, 100.

- If file 4 has lines 20, 30, . . . 70, the following command can be executed.

  MERGE 4, 10, 10: By specifying the 1st line number as 10, the program lines from file 4 are renumbered so that they will fit in front of the program lines currently in memory. Then the lines in memory are 10, 20, . . . 100. Since the 2nd line number is specified as 10, program execution begins at that line.

- If file 5 has lines 1, 2, . . . 100 and you want to insert these lines between lines 70 and 80 currently in memory, it would be necessary first to renumber the lines currently in memory to allow for a 100 line insertion. The command – REN 700,200 – achieves this and renumbers the lines in memory to 700, 900, 1100, 1300. So the MERGE command could then be – MERGE 5,750. The lines in memory would then be 700, 750, 751, 752, . . . 849, 900, 1100, 1300.

---

† Restarting program execution means that although the program does not halt, the values of variables are erased and operating modes are reset.

FIND

The FIND command is used to locate a specified tape file. It has two primary functions:

- It is used with the MARK command to locate the tape file at which marking is to begin.
- It is used to locate the next tape file that is to be accessed. This can have considerable time savings in either the calculator or programming mode since other operations can still be performed while the FIND command is being executed.

The syntax is as follows:

FIND file

As soon as the FIND command begins execution, the tape searches forward until the first file ID is encountered. When it is encountered, both the direction and the number of files that the tape must travel are known. At this point, control of the calculator is regained by the user while the tape continues to search for the desired file. In the calculator mode, control is regained when the symbol '⊢' appears on the display; whereas in the programming mode, control is regained when the program continues executing. (Incidentally, in the programming mode, if any other cassette command is encountered while the FIND command is still executing, the FIND command is immediately overridden by the new cassette command.)

Do not use the FIND command when in the Advanced Programming I ROM lower case mode. See the Advanced Programming I ROM operating manual for details.

Example

```
10 DEG
 .
 .
 .
100 FIND 8
110 FOR X=0 TO 360
120 PRINT TAB(35+30*SINX)"*"
130 NEXT X
 .
 .
 .
200 LINK 8,10,10
```

When line 100, FIND 8, is executed, the tape cassette searches forward to determine its correct location. When the first file ID is found, the tape continues the search in high speed and the program continues execution at line 110. When line 200 is finally accessed, the tape is already positioned at file 8, so the LINK command can be immediately executed. (However, if file 8 had not already been located, the LINK command would have overridden the FIND command, continued the search, and performed the linking operation.)

REWIND

The REWIND command causes the cassette to rewind to clear-leader. However, any other cassette command immediately overrides the REWIND command.

All other cassette commands have their execution terminated if the STOP key is pressed. But the quickest way to terminate the REWIND command is to open the transport door.

The syntax is simply:

REWIND

In the calculator mode, the internal tape cassette is most easily rewound if you press the REWIND key. But in the programming mode, REWIND must be keyed in, letter by letter.

In the programming mode, the REWIND command can be specified after all other cassette commands to ensure that the tape is on clear-leader when the program is finished and the cassette is removed from the transport.

STORE DATA

The STORE DATA command takes data from memory and reproduces it onto tape. Data can be specified in either of two ways:

- If an array name (A through Z) is specified in the command, the value of each variable in the array is stored in a tape file.
- If no array name is specified in the command, the values of all variables defined in a COM statement are stored.† (From a COM statement, both simple and array variables can be stored.)

The syntax is:

STORE DATA file [, array]

As mentioned, if the array is not specified, the variables referenced in the COM statement are stored.

All variables stored need not have assigned values. For instance, a 20 element array — A(20) — could be stored on tape, while an element within the array, say, A(6), has no defined value.

The STORE DATA command can be executed in the calculator mode. But to do so generally requires either a COM or a DIM statement in memory and always requires that the calculator must have been previously initialized by either a RUN or an INITIALIZE command.

Here are two examples of using STORE DATA in the programming mode.

Examples

#1

```
1 COM A[20],B,B1
10 FOR I=1 TO 20
20 A[I]=I↑2-2*I
30 NEXT I
40 INPUT B,B1
50 DISP "FILE NO.";
60 INPUT Z
70 STORE  DATA Z
80 END
```

#2

```
10 DIM A[6,6],B[30]
20 FOR J=1 TO 25
30 B[J]=LOGJ
40 NEXT J
50 STORE  DATA 3,B
 .
 .
 .
100 END
```

Example No. 1: In this example the STORE DATA command, line 70, references a variable file (Z), which is assigned a value in line 60. Since no array is specified by STORE DATA, all variables referenced in the COM statement are stored on the specified file.

Example No. 2: In this example the STORE DATA command, line 50, references file 3 and array B. So all the variables in the B array are stored in file 3. Notice that although B(1) through B(30) are all stored, B(26) through B(30) remain undefined.

† For those who have the 'String Variables ROM' — string variables can be stored only in this manner; that is, by being referenced in a COM statement.

The file specified by STORE DATA must be large enough to include the entire array even if many of the array variables do not have assigned values.

> **NOTE**
>
> In general, it is a good idea to mark file sizes larger than the information that is going into the files. This is especially true when storing data.

LOAD DATA

The LOAD DATA command takes the data that was previously recorded in a file (with the STORE DATA command) and reproduces it into memory.

The syntax parallels the STORE DATA syntax:

LOAD DATA file [, array]

If an array is specified in the STORE DATA command, then the LOAD DATA command that subsequently accesses the same file must also specify an array (although the array name specified need not be the same as the one used in the STORE DATA command).

If no array is specified in the STORE DATA command, then the corresponding LOAD DATA command cannot specify an array. If STORE DATA did not specify an array, the variables from the COM statement were stored; so when the corresponding LOAD DATA statement is executed, a COM statement that parallels the previous COM statement should be in memory.

Examples

#1

storing data

```
1 COM A,B,C[2]
2 A=54
3 B=16.8
4 C[1]=9
5 C[2]=7.3
6 STORE  DATA 2
7 END
```

(then after memory is erased)
loading data

```
or 1 COM A,B,C[2]
   1 COM Q,T,V[2]
   2 LOAD  DATA 2
   .
   .
   .
   100 END
```


Line 6, STORE DATA 2, in the original program stored the values of the variables specified in the COM statement. Since an array was not specified in this STORE DATA command, the subsequent LOAD DATA command cannot specify an array either.

Notice that the COM statement used for the LOAD DATA command need not specify the same variable names as the COM statement that is used with the corresponding STORE DATA command. But simple variables should correspond to simple variables, arrays should correspond to arrays of the same size, split-precision arrays should correspond to split-precision arrays, etc.

(Continued)

LOAD DATA
(Continued)

#2

storing data

```
10 DIM A[30]
20 FOR I=1 TO 30
30 A[I]=6*SQR I
40 NEXT I
50 STORE  DATA 5,A
60 END
```

(then after memory is erased)
loading data

```
 10 DIM A[30],G[30]
or{20 LOAD  DATA 5,A
or{20 LOAD  DATA 5,G
  .
  .
  .
100 END
```

Line 50 in the original program stored the elements of array A into file 5. Since an array was specified in this STORE DATA command, the subsequent LOAD DATA command must also specify an array. The array name specified by LOAD DATA need not be the same as the array name specified by STORE DATA; however, it must be dimensioned at least as large as the stored array.

NOTE

If you have recorded data on a cassette using either an HP Model 10 or an HP Model 20 Calculator, you can load the data back into the Model 30 Calculator. Use the LOAD DATA command with the 'array' specification – and be sure that a COM or DIM statement in memory is large enough to assign variable names to all the data being entered. If the data was recorded using the Model 20, however, it will be loaded into the Model 30 in the reverse order.

STORE KEY

The STORE KEY command takes the information that is defined on all the Special Function keys (see Chapter 6), and reproduces this information onto tape.

Syntax:

STORE KEY file

Sometime later, after the information on the Special Function keys has been erased, the corresponding LOAD KEY command can be executed to put the information back on the Special Function keys.

LOAD KEY

The LOAD KEY command takes the information that was previously recorded in a file (with the STORE KEY command) and reproduces it onto the Special Function keys.

Each Special Function key can then perform the same operations that it previously did before being stored on tape; that is, a program, a function, or text previously defined on, say, f_9, is once again defined on f_9.

Syntax:

LOAD KEY file

If, say, the information on the 'BASIC' key overlay (supplied with the Model 30) is entered onto the Special Function keys as text, it can be reproduced onto tape with a STORE KEY command. Then anytime this information is needed, the corresponding LOAD KEY command can be executed.

Executing a LOAD KEY command is similar to executing a LOAD command in some respects. All variables not defined in a previous COM statement (see pages 3-39 to 3-41) are initialized. Similarly, operating modes like STANDARD and RAD are reset, and statement references for GOSUB/RETURN and FOR. . .NEXT are lost.

LOAD BIN

The LOAD BIN (Load Binary) command reproduces into the calculator memory, from tape, binary information (assembly language programs). The assembly language program may be, for instance, a system diagnostic or an I/O subroutine.

The assembly language program cannot be listed or displayed, nor can it be stored onto tape by the user.

Syntax:

LOAD BIN file

NOTE

Files on cassettes are tagged as unused files, binary files, data files, program files, or key files. If an attempt is made to load from a particular file and the specified command incorrectly identifies the file tag, an error occurs. For example, executing the command, LOAD BIN 2, when file 2 is a data file, causes an error to occur.

TLIST

Beginning at the tape's current location, the TLIST command prints the information that is contained in subsequent file ID's (see Tape Structure, Figure 5-3).

Syntax:

TLIST

The information for each file is printed on one line. There are no column headers identifying the information in each line; the assumed headers and their explanations are as follows:

File No.	File Type (Code No.)	Absolute File Size (in words)	Current File Size (in words)	Program Line Nos. (First)	(Last)	Common Area (in words)

1. File No: The first column always specifies the file number.

2. File Type: The five file types are coded as follows:

   Ø for an unused file
   1 for a binary file
   2 for a data file
   3 for a program file
   4 for a key file.

   Furthermore, if the file is secured, (see page 4-15) a '2' appears in front of the code number. For instance, the code for a secured program file is '23'. (Data files cannot be secured, however.)

3. Absolute File Size: The length (in words) that the file was marked.

4. Current File Size: The number of words currently being used.

5. First Program Line No: For program files, this column displays the lowest line number on the file. But for data files, this column has another meaning. It is coded as follows:

   Ø for a full-precision array
   1 for a split-precision array
   2 for an integer-precision array
   3 for variables stored via the COM statement.

6. Last Program Line No: For program files, this column displays the highest line number on the file. For any other file type, this column has no meaning.

7. Common Area: For program files, this column displays the number of words of memory needed for the COM specifications. For any other file type, this column has no meaning.

PERIPHERAL CASSETTES

Besides the tape transport built into the Model 30, up to nine peripheral cassette memories can be added to the Model 30.

The peripheral is referred to as the 9865A Cassette Memory. Four cassette memories can be connected directly to the calculator through the four I/O slots in the rear panel. If you have a 9868A I/O Expander, however, up to nine cassette memories can be added (this would still leave four I/O slots available for other peripherals).

If any of the peripheral cassette memories are used, all the commands previously discussed can still be used. But in order to specify which cassette is being accessed, the select code of the peripheral cassette must appear in the command immediately after the command name. For example, the following sample syntaxes can be used to specify a peripheral cassette:

```
STORE # select code [,file]
LOAD # select code , file [,1st line number[,2nd line number] ]
FIND # select code, file
REWIND # select code
STORE DATA # select code, file [,array]
```

All cassette command syntaxes can be updated in this manner to specify a peripheral cassette. The number sign '#' is always required.

Peripheral cassette select codes are identified as #1 through #9. The internal tape transport has select code #10; if no select code is specified in a tape command, select code #10 is assumed.

CASSETTE SYNTAXES

Legend

brackets [ ]	– items enclosed within brackets are optional.
coloring	– colored items must appear as shown.
array name	– the letter used to define the array (A through Z).
file	– the file number; can be a constant, a variable, or an expression.
1st line number	– the first line number designated.
2nd line number	– the second line number designated; it can appear only if the first line number is designated.
length	– the files' lengths; can be a constant, a variable, or an expression.
no. of files	– the number of files; can be a constant, a variable, or an expression.

FIND file
Locates a specific tape file.

LINK [file] or LINK file [, 1st line number [, 2nd line number]]
Works like LOAD but, in addition, it retains variables currently in memory.

LOAD [file] or LOAD file [, 1st line number [, 2nd line number]]
Reproduces, into memory, programs that are on tape via STORE.

LOAD BIN file
Reproduces, into memory, assembly language programs that are on tape.

LOAD DATA file [, array]
Reproduces, into memory, data that is on tape via STORE DATA.

LOAD KEY file
Reproduces, into the Special Function keys, information that is on tape via STORE KEY.

MARK no. of files , length
Establishes files with specified lengths.

MERGE [file] or MERGE file [, 1st line number [, 2nd line number]]
Inserts, between program lines in memory, program lines that are on tape via STORE.

REWIND
Rewinds the tape to clear-leader.

STORE [file] or STORE file [, 1st line number [, 2nd line number]]
Reproduces, onto tape, programs that are in memory.

STORE DATA file [, array]
Reproduces, onto tape, data that is in memory.

STORE KEY file
Reproduces, onto tape, information that is on the Special Function keys.

TLIST
Prints information about each file.

Chapter 6

SPECIAL FUNCTION KEYS

The Special Function keys, located in the upper left-hand portion of the keyboard, add considerable flexibility to the Model 30.

There are 10 keys, labeled f_0 through f_9, in this region. There are, however, effectively 20 accessible Special Function keys since each key can be accessed normally or with the SHIFT key held down.†

The Special Function keys can be used effectively in three ways:

- To represent text (where text can be used as a typing aid).
- To represent either single-line or multiple-line functions.
- To represent programs.

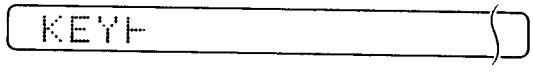
These three uses will be discussed individually after a brief introduction to entering and exiting the Special Function mode (referred to hereafter as the 'KEY mode').

ENTERING AND EXITING KEY MODE

To put information on a Special Function key, it is necessary first to enter the KEY mode by pressing:

FETCH f_x ‡

If there is no information on the key, the display will now be:



Whenever you are in the KEY mode and you press the CLEAR key, this same display will appear.

While in the KEY mode, you can input as discussed in the following sections.

To exit the KEY mode, merely press the END key. Often, the KEY mode is automatically exited when certain operations are performed. These operations are also discussed in the following sections.

We will refer to all 20 keys as f_0 through f_{19}: f_0 through f_9 in the normal mode, and f_{10} through f_{19} in the shifted mode.

The particular key specified (f_0 through f_{19}) is the one accessed by the command. Any time f_x is referred to in this text, any of the twenty keys can be specified.

KEYS AS TEXT

A Special Function key can be used as a typing aid, as illustrated in the following example:

Example

First a key, say [f6], is accessed by the method previously shown:

[FETCH] [f6]

If no information is on the key, – KEY ⊢ – appears on the display. Then text can be entered if an asterisk (*) is keyed in first. If you plan to write a program with several PRINT statements, you can key in:

[*] [P] [R] [I] [N] [T] [END OF LINE]

The END OF LINE key is necessary to complete the entry. After it is pressed, the KEY mode is automatically exited.

Even if information is already on the key, the method just discussed can be used. But, when the END OF LINE key is pressed, everything previously on the key is erased.

Once the key is defined as text, the text is immediately accessible. If you are writing a program and you want line 55 to be a PRINT statement, press:

[5] [5] [f6]

and the display is:

```
55PRINT
```

Then the rest of line 55 can be keyed in normally.

Notice, anything already on the display is retained when the typing-aid key is pressed.

Any Special Function key can be used in the manner just described.

The text that is on a Special Function key can be a command; e.g.,

[FETCH] [f4] [*] [L] [I] [S] [T] [1] [,] [3] [0] [END OF LINE]

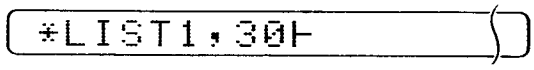
Then whenever you want to execute this command, you can press:

[f4] [EXECUTE]

However, it would be more convenient if this were an 'immediate execute' command. If an asterisk both precedes and follows the command, it is immediately executable. For example, press:

[FETCH] [f4]

and the display is:



This text can be edited to include the asterisk immediately after the text. Just press:

BACK * END OF LINE

and the line now in memory is: *LIST1,30*

Then whenever you want to execute this command, just press:

f

KEYS AS FUNCTIONS

A Special Function key can be used to store either single-line or multiple-line functions. (Single-line and multiple-line functions are discussed beginning on page 3-32.)

To put a single-line function on a key, the function must be entered as a program line, as shown in the following example:

Example

First a key, say f, is accessed normally:

FETCH f

If there are program lines already on this key, they can be easily deleted if you press:

D E L EXECUTE

(If there is text on the key, it is automatically deleted when the first program line is entered.)

A DEFFN statement can then be input. It must be the lowest-numbered program line on that key. For example, key in:

1 D E F F N A (X) = X ↑ 3 − X END OF LINE

Then to evaluate this function for a particular value of X, say 3, first press the appropriate key:†

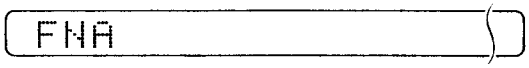
f

(continued)

† It doesn't matter whether or not you are in KEY mode when you perform this operation. You could have previously exited from KEY mode if you had pressed the END key.

KEYS AS FUNCTIONS
(continued)

The function name is immediately displayed:



Solving for FNA3 is then accomplished merely by keying in the 3 and pressing the EXECUTE key.

In this example the result, 24, would then be displayed.

After the function is evaluated, the KEY mode is immediately exited.

In the previous example, the function could also be called by a program that is being run. If the program is in mainline memory and the function, FNA, is called, the calculator first searches mainline memory to see if the function is defined there; if it is not, the calculator then searches the first line of each Special Function key (in the order in which they were defined) to see if the function is defined on a key. If the program that calls the function is on a key, the calculator first searches each line of that program to see if the function is defined there; if it is not, the calculator then searches the first line of each Special Function key (in the order in which they were defined); finally the calculator searches mainline memory. Hence, it is possible for a function to be defined in more than one place. Then if a program calls on the function, the particular function called depends on the order of the search.

Multiple-line functions can be on a key, also. As in single-line functions, just be sure that the DEFFN statement is the lowest-numbered statement on the key. By doing so, when the key is pressed, the function name appears on the display; then the function can be immediately evaluated.

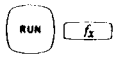
KEYS AS PROGRAMS

A Special Function key can contain an entire program. The program can be keyed in just like any other program once the specified key is accessed. Automatic line numbering can be used, the editing keys can be used, etc.

Only the COM statement is restricted. If COM is to be used, it must be previously specified in mainline memory; furthermore, the number of elements specified by COM in the KEY mode can be no greater than the number specified in mainline memory – for there is only one common area allocated in memory.

After a program is entered into a particular key, the KEY mode can be immediately exited if the END key is pressed.

A program on a specified key can be run if you press:



The KEY mode is exited if the program is terminated by an END statement.

Often it is more convenient to perform a 'continue' operation on a program in the KEY mode. This is done simply by pressing the appropriate [f].

In this case, no additional array variable storage is allocated, no variable values are erased, etc. So if just the key is pressed, the program executes as though a CONT command had been given.

The program or text on a Special Function key can be edited once the specified key is accessed. It may even be necessary to edit a key once a program has begun to run. When this happens, press STOP and END to halt program execution† and FETCH f_X to access the key to be edited. Once the editing is completed, press RUN f_X (if the program is loaded on a key) or RUN EXECUTE (if the program is loaded in mainline memory). When editing is completed, attempting to *continue* execution of a program without initializing variables (by using the RUN command, for example) may result in an error.

The following example shows a simple application for using Special Function keys in this way:

Example

If the following program is in mainline memory:

Mainline Memory

```
10 Z=0
20 INPUT N
30 FOR I=1 TO N
40 INPUT A
50 Z=A+Z
60 NEXT I
70 END
```

and the following two programs are on keys, f_5 and f_6, respectively:

f_5

```
10 PRINT "THE TOTAL IS"Z
20 END
```

f_6

```
10 PRINT "THE AVERAGE IS"Z/N
20 END
```

When the program in mainline memory is run and line 20 is accessed, the number of entries to be keyed in must be specified. Then each time the program loops, another input is accepted. After the program is completed, the user can press [f5] to determine the total of the inputs and the user can press [f6] to determine the average of the inputs.

In this example, if either of the following were pressed instead:

[RUN] [f5] or [RUN] [f6]

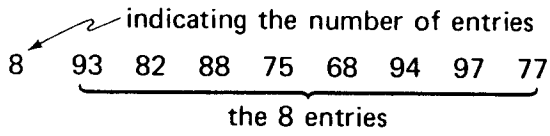
an error would occur since the RUN command erases variable values specified in a previous program, in this case, Z and N, whose values are needed for the Special Function programs.

(continued)

† STOP and END should be pressed before FETCH even if program execution appears to halt, as in the case of an INPUT statement.

KEYS AS PROGRAMS
(continued)

If the program in mainline memory is run and the following entries are executed each time a ? appears:



When the program halts:

- If you press [A], the printout is:

```
THE TOTAL IS 674
```

- If you press [A], the printout is:

```
THE AVERAGE IS 84.25
```

ADDITIONAL KEY OPERATIONS

KEYBOARD COMMANDS

If you are in KEY mode, located at a specified key that contains a program, in general, operations can be performed the same as in normal mode:

RUN EXECUTE to run the program.

FETCH 2 5 EXECUTE to display line 25.

LIST 1 5 EXECUTE to list all program lines, beginning at line 15.

Other commands, such as REN and DEL, can also be used in this manner.

To erase all information on the keys, press:

SCRATCH K EXECUTE which only erases information on keys.

SCRATCH A EXECUTE erases everything on the keys — of course, it also erases everything else in memory.

Whether or not you are in KEY mode, the following command does not erase programs or text on keys.

Of course, this command does erase all programs in mainline memory. Furthermore, it does erase all variables defined in the calculator.

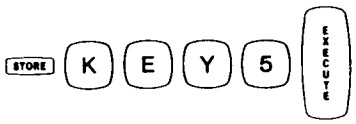
Some commands, however, can be directed at a particular key, whether it is a program, a function, or a typing-aid key.

SCRATCH f_x where only the specified key is erased.

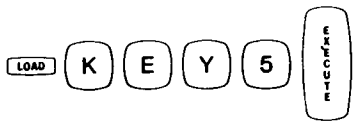
LIST f_x where everything on the specified key is listed.

CASSETTE COMMANDS

The tape cassette commands were discussed in Chapter 5. However, those commands that can be used with the Special Function keys are mentioned here, also. (In these examples, file 5 is arbitrarily selected.)



Everything on all the Special Function keys is stored in file 5.



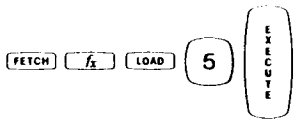
The information in file 5 is loaded back into memory in the same order in which it was extracted; that is, the information previously in each f_x will be loaded back into each f_x.

To load information into memory with a LOAD KEY command, the information must have been previously stored with a STORE KEY command.

A program on a particular key can be stored in a file, too. First, access the key with a FETCH command. Then use a STORE command to put it in a file.

The program in the specified f_x is stored into file 5. (Typing-aid keys cannot be stored in this manner.)

To load this program back on a key at some later time, press:



Once a key is accessed, other tape commands, like LINK and MERGE, can be used in this manner.

KEY OVERLAYS

Three overlays are supplied for the Special Function keys. They are easily inserted over the Special Function keys if you lock the extended tab on the right into the appropriate keyboard slot. Then press the overlay down over the keys and secure it with the left tab lock.

Two of these overlays, BASIC and MATH, have words pre-printed corresponding to specified keys. If desired, each pre-printed word can be keyed into the appropriate f_x as a typing-aid key. Then the STORE KEY command can be used to retain all the information on tape for easy accessibility. The third overlay, SPECIAL FUNCTIONS, can be labeled any way you want.

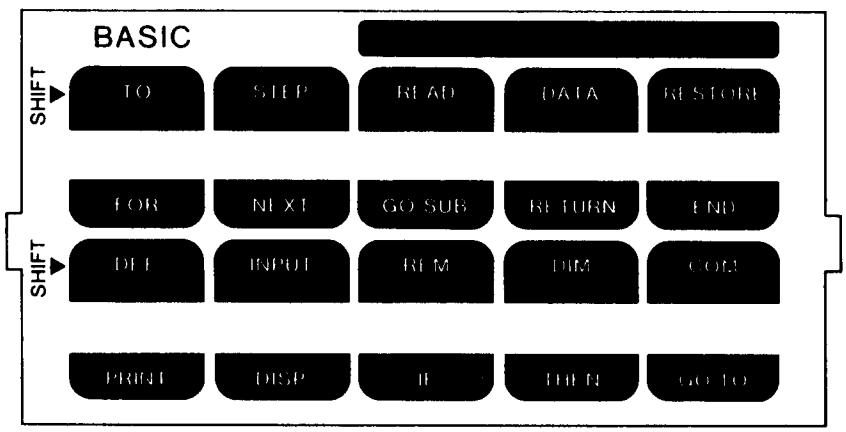


Figure 6-1. BASIC Key Overlay

APPENDIX A

CALCULATOR AND PRINTER INSTALLATION PROCEDURES

This appendix contains inspection, installation, and maintenance procedures for your calculator and printer.

THE CALCULATOR

INSPECTION PROCEDURE

The various parts of your calculator system were carefully inspected before they were shipped to you. All equipment should, therefore, be free of scratches and should operate properly. Carefully inspect the calculator, plug-in ROM's, peripheral equipment, cables, etc., for physical damage sustained in transit. Notify HP and file a claim with the carrier if there is any such damage.

Please check to ensure that you have received all of the items which you ordered and that any options specified on your order have been installed in your calculator. Decals located inside the ROM door (see Figure C-1) show the option number of any internal option installed in the calculator. Also check to ensure that all accessories are present (refer to 'Equipment Supplied' in Appendix B).

If you wish to check the operation of your system, or any part of it, refer to the System Test Instructions book, which contains the information needed to run the System Test Cassette. Before running the test, however, be certain that your calculator is properly installed.

If you have any difficulties with your system, if it is not operating properly, or if any items are missing, please contact your nearest HP Sales and Service Office; addresses are supplied at the back of this book.

POWER REQUIREMENTS

The Model 9830A Calculator has the following power requirements:

- Line Voltage: The calculator operates from nominal powerline voltages of 100, 120, 220, and 240 ac volts. The range of operation is from −10% to +5% of each nominal voltage. Two switches on the rear panel of the calculator enable any one of the four voltages to be selected (refer to page A-3, Initial Turn-on).
- Line Frequency: The calculator can be operated with any line frequency from 48 Hz to 66 Hz (nominally 50 Hz and 60 Hz).
- Power Consumption: With no peripheral equipment connected, the calculator requires a maximum of 150 voltamps.

THE CALCULATOR
(Continued)

POWER OUTLETS

There are two power outlets on the rear panel of the calculator (see Figure A-1). These are used to supply ac power to peripheral equipment. No more than a combined total of 610 voltamps must be drawn from these two outlets. The outlets are 'live' whenever the calculator is plugged in; they are not switched on or off by the LINE ON/OFF switch on the front of the calculator.

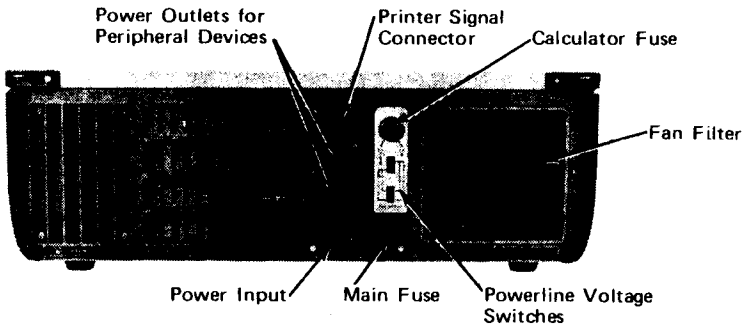


Figure A-1. The Rear Panel

GROUNDING REQUIREMENTS

To protect operating personnel, the National Electrical Manufacturers' Association (NEMA) recommends that the calculator keyboard and cabinet be grounded. The calculator is equipped with a three-conductor power cable which, when connected to an appropriate power receptacle, grounds the keyboard and cabinet of the calculator. To preserve this protection feature, do not operate the calculator from an ac power outlet with no ground connection.

FUSES

The calculator has two fuses, which are located on the rear panel (see Figure A-1).

The main fuse is a 6-amp fuse. It protects the calculator and any peripheral devices connected to the two power outlets on the rear of the calculator.

The calculator fuse is either a 2-amp fuse for 100 or 120 ac volt operation, or a 1-amp fuse for 220 or 240 ac volt operation. This fuse protects the calculator only.

WARNING

TO AVOID THE POSSIBILITY OF SERIOUS INJURY, ALWAYS DISCONNECT THE CALCULATOR FROM ITS POWER SOURCE BEFORE CHANGING A FUSE.

To change a fuse, first disconnect the power cable from the calculator. Next, press inward on the fuse-holder cap while twisting the cap in the direction indicated by the arrow on the cap. Withdraw the cap and fuse from the fuse-holder and remove the fuse from the cap. Insert the replacement fuse (either end) into the cap; then put the fuse and cap back into the fuse-holder. Press on the cap and twist it in the direction opposite to that indicated by the arrow until the cap is properly locked into place.

INITIAL TURN-ON

With the calculator disconnected from its ac power source, verify that the correct calculator fuse has been installed for the powerline voltage in your area (refer to the previous section, Fuses).

Next, ensure that the two switches on the rear panel are set for the correct powerline voltage. Figure A-1 shows the location of the switches and Figure A-2 shows the correct settings for each nominal line voltage. If it is necessary to alter the setting of either switch, insert the tip of any small instrument into the white slot on the switch. Slide the switch so that the position of the white slot corresponds to the desired voltage, as shown in Figure A-2.

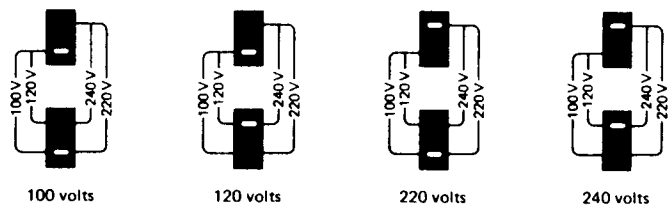


Figure A-2. Switch Settings for the Nominal Powerline Voltages

Switch the OFF/ON switch, located on the right front of the calculator, to the OFF position.

Connect the power cord to the power input connector (Figure A-1) at the rear of the calculator; plug the other end of the power cord into a suitable ac power outlet.

Switch the OFF/ON switch to the ON position; the symbol '⊢' will appear in the display indicating that the calculator is ready to operate.

> **NOTE**
>
> When switching the OFF/ON switch to the OFF position, wait at least three seconds before switching it ON again. The protection circuit capacitor associated with the power supply requires this time to discharge. If the calculator is turned ON less than three seconds after it was turned OFF, loss of calculator control may result. To regain control when this happens, switch the calculator OFF again, wait over three seconds, and then switch the calculator back ON.

CLEANING THE CALCULATOR

The calculator can be cleaned with a soft cloth dampened either in clean water or in water containing a mild detergent. Do not use an excessively wet cloth nor allow water to penetrate inside the calculator. Also, do not use any abrasive cleaners, especially on the display window.

The fan filter (Figure A-1) should normally be cleaned about every three months. To clean the filter, first turn the calculator off. Then remove the filter by prying it out with an instrument, such as a screwdriver; this is done by inserting the instrument into one of the slots on either side of the filter, and by prying the filter out from the rear panel. Clean the filter either by holding it under running water, or by washing it in warm, soapy water, followed by rinsing it in clean water. Dry the filter thoroughly. Finally, install it again by snapping one side back into place, and then the other.

If you have the 9866A Printer, follow the same cleaning procedure.

PRIMARY PRINTERS

The 'primary printer' with the Model 30 is the printer which is set to select code 15. The HP 9866A Printer (Option 30) was designed specifically to be the primary printer for the Model 30. So the procedure to connect this printer to the calculator is included in this appendix, and operating information is in Appendix F.

However, there are other printers that can be used as the primary printer. Although each of these printers has some unique requirements and features, all of them are controlled by the same statements as is the 9866A Printer. Two of these printers will be referred to in this book since they are representative of the types of printers that are available. The two printers chosen are the HP 9861A Output Typewriter and the Teletype Model 38 ASR Data Terminal (teleprinter).

Any printer other than the 9866A Printer requires an interface to connect it to the calculator. The typewriter is supplied with its own special interface. The teleprinter must be interfaced by means of the HP 11205A Serial I/O Interface; this is a general purpose interface used to connect, to the calculator, devices that meet EIA Standard RS-232-C.

Installation procedures for the typewriter and teleprinter can be found in the manuals supplied with them and in the manuals for the interfacing equipment. Operating information is included in Appendix F of this book.

> **NOTE**
> The 9861A Output Typewriter manual states that a calculator requires a ROM to control the typewriter. Despite that statement, no ROM is required to operate the typewriter as either the primary or as the secondary printer with the Model 30.

THE 9866A PRINTER

DESCRIPTION

The -hp- 9866A Printer is a high speed, thermal line-printer capable of printing up to 240 lines per minute with up to 80 characters per line. The Option 30 version of the printer connects directly to the 9830A Calculator and requires no special equipment to operate. The casing of the printer is designed to be placed either on any flat surface or on top of the calculator, as shown in the photograph on the title page of this book.

General information about the printer — accessories, power requirements, turn-on procedure, ordering printer paper, etc. — is included in the Peripheral Manual (-hp- Part No. 09866-90000) supplied with the printer. Some of the information is also included here for your convenience. However, for complete information about your printer, please refer to the Peripheral Manual.

INSTALLATION AND TURN-ON

- Power Requirements: The 9866A Printer has the same power requirements as the calculator, except that the printer has a maximum power consumption of 250 voltamps (refer to page A-1, Power Requirements).
- Grounding Requirements: Grounding for the printer is similar to grounding for the calculator. If the calculator is properly grounded, as described earlier in this appendix, use the inter-instrument power cord (-hp- Part No. 8120-1575) supplied with Option 30 printers to ensure that the printer is also properly grounded.
- Fuses: The printer uses a 3-amp fuse for 100 or 120V operation or a 1.5-amp fuse for the 220 or 240V operation. If your printer is being turned on for the first time, ensure that the correct fuse is installed (refer to page A-2, Fuses).

Before connecting the printer to the calculator, ensure that the correct fuse is installed and that the slide switches on the rear of the printer are properly set for the voltage in your area. Once the calculator switches have been properly set, the printer switches should be set to the same positions (see Figure A-2).

Set the printer LINE switch to the OFF position. Then place the printer on top of the calculator and make the power and signal connections as shown in Figure A-3. The connectors on both ends of the interface cable are keyed for proper insertion into the sockets on the printer and calculator; both connectors are identical. To connect the cable, press the connector against the socket and slowly rotate the connector until you feel it align with the socket. Then twist the knurled sleeve on the connector clockwise to lock the connector and socket together. (Twisting the knurled sleeve counter-clockwise unlocks the connector.) The calculator and printer are now ready to operate. Set the LINE switches on the calculator and printer to the ON position.

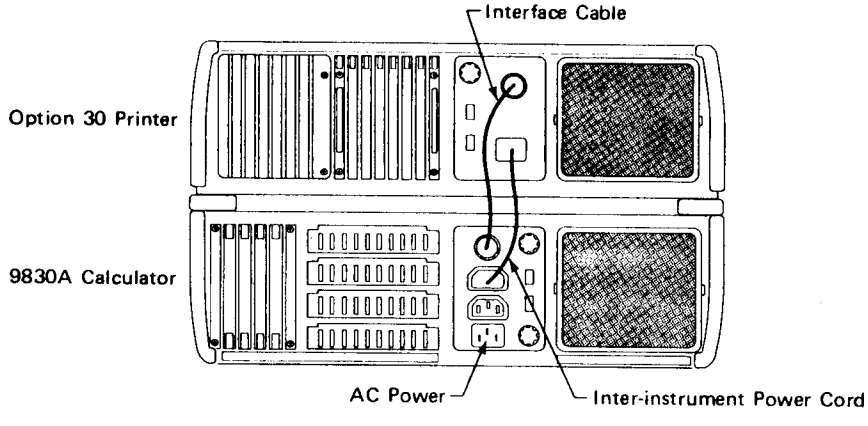


Figure A-3. Connecting the 9866A Printer

LOADING PRINTER PAPER

The LOAD light, below the PAPER key, lights any time the printer is out of paper. To load paper, lift the lid and refer to the diagram underneath it to see how paper is loaded. Place the roll, with the free end as shown, into the printer. Then press the PAPER key until the paper emerges from the front of the printer.

APPENDIX B

GENERAL OWNER'S INFORMATION

The information in this appendix should be read soon after the arrival of your calculator.

EQUIPMENT SUPPLIED

The following items are packaged with your calculator:

- The items listed in Table B-1, which are included with every Model 9830A Calculator;
- A copy of either a manual or an operating note for the options installed in your calculator (except for the memory options, see page B-2);
- Any additional items listed in any extra manuals or operating notes.

Plug-in ROM's and peripheral devices are packaged separately; each of these has its own manual or operating note and may also have extra items packaged with it.

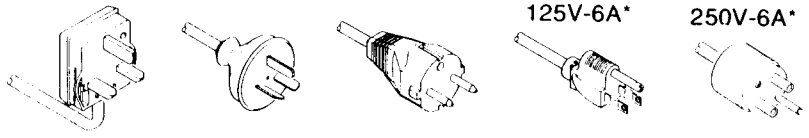
Table B-1. Standard Accessories Supplied

Item	Quantity	-hp- Part Number
Simplified Operating Instructions	1	09830-90000
Operating and Programming Manual	1	09830-90001
Quick Reference Card	1	09830-90039
Simplified Training Cassette	1	09830-90014
9830A Program Pad	1	09830-90016
9830A Math Pac	1	09830-70001
Math Pac Manual	1	09830-70000
Tape Cassette – Math Pac 1	1	09839-70000
Sheet of Math Overlays (1–6)	1	7120-3511
Sheet of Math Overlays (7–9)	1	7120-3514
Key Overlays:		
Special Functions	1	7120-3053
Basic	1	7120-3054
Math	1	7120-3055
Template Holder	2	9230-0065
System Test Cassette	1	09830-90062
System Test Instructions	1	09830-90027
Tape Cassette – Blank	2	9162-0050
Power Cord	1	One of numbers shown in Figure B-1
ROM Door Key	2	5040-7437
Dust Cover	1	4040-0978
Magnetic Head Cleaner	1	8500-1251
Cotton Applicators	10	‡
Spare Fuses – 250V Normal-blow ¼ in. dia. X 1¼ in. lg.		
1-amp	2	2110-0001
2-amp	2	2110-0002
6-amp	2	2110-0056

‡ A package of 100 cotton applicators can be ordered under part number 8520-0023

EQUIPMENT SUPPLIED
(Continued)

Power cords with different plugs are available for the calculator. Each plug, together with the part number of the power cord which has that plug, is shown in Figure B-1. Each plug has a ground connector. The cord packaged with each calculator depends upon where that calculator is to be delivered. If your calculator has the wrong power cord for your area, please contact your local -hp- Sales and Service Office.



8120–1351 8120–1369 8120–1689 8120–1348 8120-0698

*UL listed for use in the United States of America.

Figure B-1. Power Cords

SERVICE CONTRACTS

Service contracts are available for all -hp- calculators and calculator-related equipment. For further information contact any -hp- Sales and Service Office.

PROGRAM PACS

Program pacs containing programmed solutions to problems from many disciplines are available. A Math Pac is supplied with each calculator. For a complete list of pacs and for pricing information, please contact any -hp- Sales Office.

KEYBOARD MAGAZINE

'Keyboard' is a periodical magazine containing general information about Hewlett-Packard calculators and related equipment. It includes articles and programs written by calculator users; descriptions of the latest equipment and program pacs; programming tips; and many other items of general interest to calculator users.

To receive your free subscription to 'Keyboard', please complete the order form supplied with your calculator. This will ensure that your name is added to the mailing list for 'Keyboard'. (Don't forget to tear off, and keep, the warranty statement at the bottom of the order form.)

MEMORY OPTIONS

The size of the memory in the basic calculator is 1760 (16-bit, 2-byte) words. A calculator with Option 275 installed has a total memory of 3808 words, while a calculator with Option 276 installed has a total memory of 7904 words. If your calculator has the basic memory or the 3808-word memory and you wish to add 4096 words to it, please order the 11281A Additional Memory Field Kit; our service personnel will then install the additional memory.

OPTIONAL EQUIPMENT

Various optional devices are available to increase the computing power of your calculator and the versatility of your system.

These options consist of the ROM's (described in Appendix C), which are additions to the calculator, and the peripheral devices (listed in Table B-2), which are additions to the system. Interfacing equipment is available to enable you to connect your calculator to non-calculator-dedicated devices, such as teleprinters and measuring instruments.

Following is a list of peripheral equipment (dedicated to the 9800 Series Calculator System) available at the beginning of 1974. As new equipment becomes available after that date, it will be described in the 'Keyboard' magazine and in data sheets available from HP Sales and Service Offices.

Table B-2. 9800 Series Calculator Peripherals

2607A Line Printer	9865A Cassette Memory
9860A Marked Card Reader	9866A Thermal Printer
9861A Output Typewriter	9867A/B Mass Memory Drive
9862A Calculator Plotter	9868A I/O Expander
9863A Paper-Tape Reader	9869A Calculator Card Reader
9864A Digitizer	9870A Card Reader
	9880A/B Mass Memory Subsystem

CONNECTING PERIPHERAL DEVICES

Except for the 9866A Printer, which is described in Appendix A, peripheral devices connect to the calculator by means of an I/O (Input/Output) card. An I/O card can be inserted into any of the four slots at the back of the calculator (or into the 9868A I/O Expander if it is used), as shown in Figure B-2. The card is constructed so that it cannot be inserted upside-down.

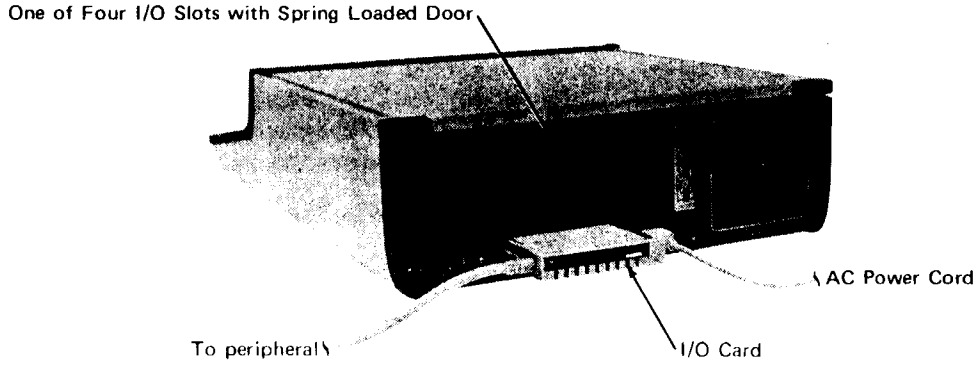


Figure B-2. Connecting an I/O Card

Before connecting an I/O card, make sure that the calculator and peripherals are switched off. Slide the card as far as it will go (about 3-3/4 inches) into any slot. Then press the card firmly into place, to ensure that it is properly connected to the calculator.

APPENDIX C

ROM OPTIONS

This appendix describes general ROM information and specifically describes the ROM's that are available with the Model 30.

GENERAL DESCRIPTION

ROM (Read-Only-Memory) differs from user memory in that information stored in a ROM is fixed. You cannot 'write' into a ROM in the same way, for instance, that you can write program lines into user memory.

Several optional ROM's are available for use with the Model 30. Each ROM consists of a block of memory (additional to the memory already built into the calculator) dedicated to performing specific tasks – controlling the plotter, enabling matrix and string variable manipulation, and so on. A list of ROM's is included in this appendix.

Most ROM's can be purchased in either one of two forms: as an accessory plug-in ROM or as an internal modification to the calculator. The calculator is capable of holding up to eight ROM's at one time; any combination of up to five plug-in ROM's and up to three internal ROM's can be installed.

The plug-in version of a ROM is a small block, about the same size as a tape cassette, which plugs into any one of the five slots behind the ROM door (see Figure C-1). You can install or remove a plug-in ROM in seconds.

The internal ROM requires a modification to the calculator, which must be made by qualified HP personnel. Whenever an internal ROM is added to a calculator, an identifying decal (showing the option number of the modification) is attached to the inside of the ROM door, so that you can readily determine which internal ROM's are installed in your calculator.

Operation of either the plug-in or the internal version of the ROM is identical once it has been installed. Which version of a ROM you choose to order is, therefore, entirely a matter of your convenience. However, please remember that no more than three internal ROM's can be installed, while up to five plug-in ROM's can be inserted at any one time.

INSTALLING A PLUG-IN ROM

The plug-in ROM's are installed in the slots behind the ROM door, as shown in Figure C-1. Unless otherwise specified in the appropriate ROM operating manual, any ROM can be plugged into any slot.

Switch the calculator off and open the door by pressing on the ribbed part of the door. If the door is locked, use the key (shown in the figure) to unlock it; the key turns about ¼ of a turn.

Slide the block, with the label 'right-side-up', into any slot. Press the block firmly into place, to ensure that it is properly connected to the calculator. Close the door.

(Continued)

INSTALLING A PLUG-IN ROM

(Continued)

The door is spring-loaded so it is not necessary to lock it, unless you prefer to do so for security, or other reasons.

Turn on the calculator, and the ROM is operational.

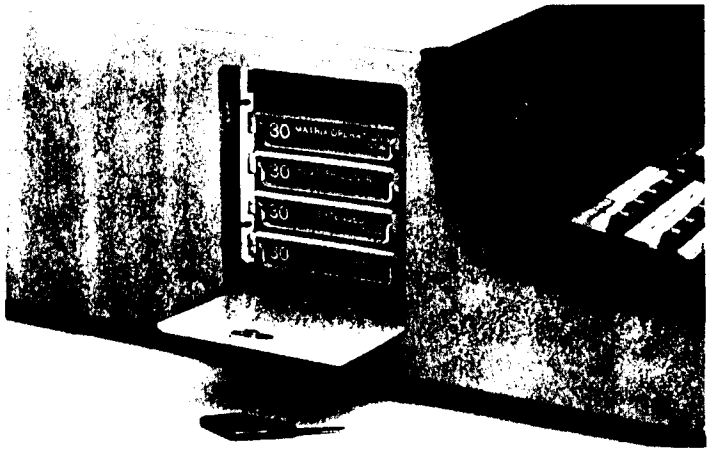


Figure C-1. Plug-In ROM's

HOW TO ORDER A ROM

To ensure that you receive the version of the ROM that you want, please order using the appropriate number, as described below. The different numbers for each ROM are given in the following section.

- To order a plug-in version of a ROM:

  Quote the accessory number. This consists of five digits followed by the letter 'B'.

  Example:

  11271B Plotter Control ROM

- To order an internal version of a ROM at the same time that you order your calculator:

  Quote the option number. This consists of the last three numerical digits of the accessory number given to the plug-in version of the ROM.

  Example:

  Option 271 Plotter Control ROM

- To order an internal version of a ROM after you have received your calculator:

  Quote the field kit number. This is the same as the accessory number except that it uses the letter 'F' (for 'field kit').

  Example:

  11271F Plotter Control ROM Field Kit

Once the field kit is installed in your calculator, it becomes an option; so the kit includes the appropriate option decal to be attached to the ROM door. In the above example the decal would be marked 'Option 271'.

THE ROM'S

The ROM's listed below are available at the beginning of 1974; as other ROM's become available, they will be described in data sheets and in the 'Keyboard' magazine.

MATRIX OPERATIONS

11270B	(plug-in)
Option 270	(internal – factory-installed)
11270F	(internal – field-installed)

This ROM extends the BASIC language to include the statements used to manipulate matrices and array data. It provides fast solutions to simultaneous equations and to business and statistical problems, as well as providing convenient ways to handle large blocks of data. The determinant operation (not normally available in BASIC) is particularly helpful to structural and electronic engineers in solving their design problems.

PLOTTER CONTROL

11271B	(plug-in)
Option 271	(internal – factory-installed)
11271F	(internal – field-installed)

This ROM enables the Model 30 to control the HP 9862A Calculator Plotter. Using your own 'problem' units (as opposed to some artificial 'plotter' units) you can very easily draw and mark axes on the plotter, and plot points or functions. You can label the axes and plotted points and you can use your plotter as a printer, formatting the printout as well as specifying character size and printing angle. In addition, you can establish a unique 'typewriter' mode which enables you to print, on the plotter, one character at a time from the calculator keyboard.

EXTENDED I/O

11272B	(plug-in)
Option 272	(internal – factory-installed)
11272F	(internal – field-installed)

This ROM enables a wide variety of peripheral devices to be controlled by the calculator. It allows a two-way transfer of information between the calculator and the peripheral devices and between peripheral devices. Data is transferred by means of standard ASCII code; however, automatic code conversion capability enables the calculator to be very easily programmed to receive and send other codes. In addition, various logic functions enable you to manipulate binary bits, thus increasing the variety of I/O operations available.

(Continued)

THE ROM'S
(Continued)

MASS MEMORY

11273B (plug-in)

This ROM, which must be used in conjunction with the rest of the 9880A/B Mass Memory, offers the advantages of a very large tape cassette. The mass memory can store entire programs as well as data. The amount of available storage space is 1.2 million words per mass memory platter. (Up to four platters can be included in the system.) In addition, short data access time makes the system extremely functional and easy to use. The average access time is less than 50 milliseconds.

STRING VARIABLES

11274B	(plug-in)
Option 274	(internal – factory-installed)
11274F	(internal – field-installed)

This ROM enables the calculator to recognize and operate on letters and words ('strings') in very much the same ways that it recognizes and operates on digits and numbers. The calculator can be programmed to understand everyday language, making your programs truly conversational in style, and, consequently more easily used by personnel with no special training.

TERMINAL

11277B	(plug-in)
Option 277	(internal – factory-installed)
11277F	(internal – field-installed)

This ROM enables the calculator to be used as the terminal in many time-share systems, and yet still retain its calculator capabilities. The calculator's own memory and text-editing ability have considerable cost-reduction advantages over other terminals. First, you can store and edit a program in the calculator and then transmit the complete program to the time-share system, thus saving on connect time. Secondly, you can store your programs in tape cassette files, thus saving on overnight and long-term storage costs. You can also send and receive program lines in languages other than BASIC, because syntax checking in the calculator can be temporarily suspended.

BATCH BASIC

11278B (plug in)

This ROM enables the calculator to accept marked or punched educational basic data processing cards which can be used for data or programming. This is especially useful when using the 9869A Card Reader. Also, all ASCII input peripheral devices can input data or programs. Programs can be 'stacked' and executed consecutively without further instructions from the calculator keyboard.

ADVANCED PROGRAMMING I

11279B (plug in)

This ROM provides the calculator with a number of convenient programming statements used to perform complex operations easily. For example, all of the information on the calculator's internal tape cassette can be duplicated onto a peripheral cassette with one command. Also, base 8 (octal) numbers can be converted to base 10 (decimal) numbers, the calculator 'beep' can be programmed, functions can be called by name, and string variables can be converted to numeric data – all with one command per operation.

ADVANCED PROGRAMMING II

11289B (plug-in)

Commercial and data base applications are speeded up when routines like SORT, SEARCH and TRANSFER are used. In addition, these routines make programming the 9830A and its peripherals much simpler. Other features of the APII ROM include error recovery (SERROR), numeric to string conversion (STRING), FLAG and BEEP. With SERROR and BEEP, programs can be designed so that anyone can operate the calculator successfully. Flags make branching easier and conserve memory space, while the STRING statement enables the calculator and its peripherals to write monetary values in any currency format.

DATA COMMUNICATIONS ROMS

Data Comm. 1 (Interface Control):

11296B	(plug-in)
Option 296	(internal-factory-installed)
11296F	(internal-field-installed)

Data Comm. 2 (Binary Synchronous):

11297B	(plug-in)

Data Comm. 3 (Interactive):

11298B	(plug-in)

The Data Communications ROMs, when used with the Data Communications Interface (11284A), offer the ability to communicate with a variety of computers.

The Data Communications Interface and the Data Communications 1 (Interface Control) ROM, provide a programmable data communications capability for the HP 9830A Calculator. By adding a second ROM, the Data Communications 2 (Binary Synchronous) ROM, the Binary Synchronous line organization protocol is available. By adding the Data Communications 3 (Interactive) ROM, the interactive capability for timesharing is provided.

APPENDIX D

MODEL 60 CARD READER

This appendix describes the use of the Model 9860A Card Reader with the Model 30 (see Figure D 1). If you have purchased this peripheral, you should read this appendix. The 9870A Card Reader replaces the 9860A Card Reader which is no longer available. This appendix is included solely for Model 60 Card Readers still in use. Call your local -hp- Sales and Service Office for information about the new Model 70 Card Reader.

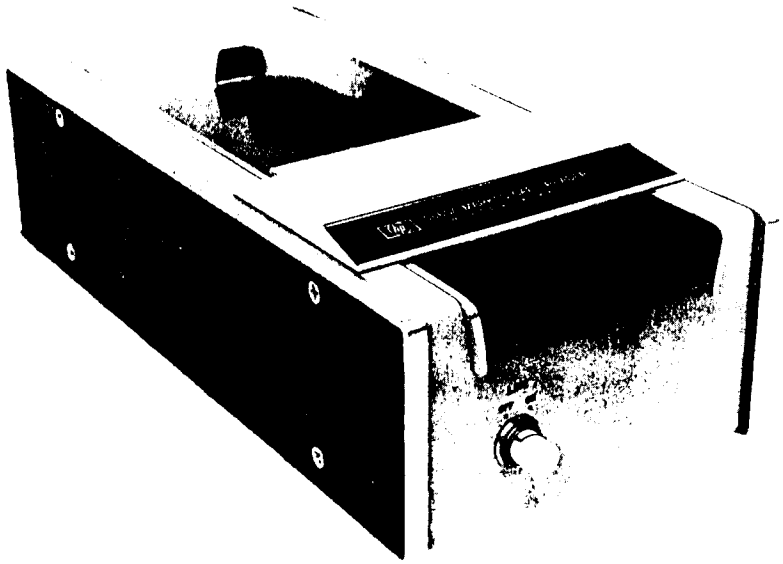


Figure D-1. 9860A Marked Card Reader

GENERAL INFORMATION

The Hewlett-Packard Model 9860A Marked Card Reader simulates a remote keyboard by allowing most operations that are available with the Model 30 Calculator. Key sequences are encoded on special cards. These cards are fed through the Model 60, which, by using an optical technique, senses the various combinations of marks on the card. There is a combination of marks to represent most keys on the Model 30 keyboard, and as a combination is detected, it is as if the associated calculator key has been pressed. A card is encoded to represent a series of keys by marking various combinations of boxes on the card with an ordinary black lead pencil.

The Model 60 does not require any special ROM to operate with the Model 30, nor are there any syntaxes associated with it.

The Card Reader Operating Manual (see Table D-1) describes the use of the card reader with the 9810A Calculator. Refer to that manual for general information about your card reader.

The accessories and equipment supplied with the Model 60 are listed in Table D-1.

(Continued)

GENERAL INFORMATION

(Continued)

Table D-1. Card Reader Equipment

Item	Quantity	-hp- Part Number
Operating Manual	2	09860-90001
Interface Cable Assembly	1	09860-61605
AC Power Cord	1	8120-1575
Spare Lamp[1]	1	09160-67901
Program and Data Card	25	9320-2085
(The following items are not used in a 9830A Calculator system.)		
Diagnostic Card	1	09860-90002
Data Card	25	9320-2088
Supplement A	1	09820-90050
Model 20 Program and Data Card	50	09320-2885

[1] Located in the instrument – see 'CHANGING THE LAMP' in the operating manual.

CARD READER OPERATION

INSPECTION

Refer to 'Inspection Procedure' in Appendix A of this manual.

INSTALLATION

CAUTION

DO NOT APPLY OPERATING POWER TO THE MODEL 9860A MARKED CARD READER UNLESS THE LINE VOLTAGE SWITCH ON THE REAR PANEL IS IN THE PROPER POSITION. OTHERWISE, DAMAGE TO THE POWER TRANSFORMER MAY RESULT.

Refer to Chapter 1 in the Card Reader Operating Manual for information concerning power and grounding requirements and installation procedures. Refer to Chapter 6 of that manual for the procedure to replace the lamp.

USING THE CARD READER

The Model 60 Card Reader acts somewhat like a remote keyboard to the calculator. A card is marked, in a simple code, with the desired key sequence and the card is passed through the reader. The calculator then behaves as if the calculator keys, corresponding to the encoded key sequence, have been pressed.

Almost any key or key sequence can be input to the calculator from the card reader. Inputs can include data, expressions, program statements, and 'immediate execute' commands (such as CLEAR, PRINT ALL, EXECUTE, etc.).

The card reader starts automatically as soon as a card is fed in. The card passes through the reader at a constant speed so that the key codes are read and input to the calculator at regular intervals. There is no system of 'interrupt' between the calculator and the card reader so the calculator must be either idle or at an INPUT statement before the card reader can be used.†

Because there is no interrupt possible, any instruction which causes immediate activity on the part of the calculator, such as END OF LINE or EXECUTE, should be the last keycode on the card. If other keycodes do follow, some of them may be missed by the calculator while it is carrying out the earlier instruction. It is possible to encode 'skips' on the card to give the calculator extra time to complete an operation before it receives the next keycode. However, this is not recommended because the number of skips required is difficult to predict (e.g., it obviously takes longer to enter an 80-character line into memory than it takes to enter a 10-character line).

Table D-2. Key Codes

CODE	KEY (or symbol)	CODE	KEY (or symbol)	CODE	KEY (or symbol)	CODE	KEY (or symbol)
0	f_0 (note 1)	40	Space Bar	100	@	140	(not used)
1	f_1	41	!	101	A	141	a (note 3)
2	f_2	42	"	102	B	142	b
3	f_3	43	#	103	C	143	c
4	f_4	44	$	104	D	144	d
5	f_5	45	%	105	E	145	e
6	f_6	46	&	106	F	146	f
7	f_7	47	'	107	G	147	g
10	f_8	50	(	110	H	150	h
11	f_9	51	)	111	I	151	i
12	LIST	52	*	112	J	152	CLEAR
13	EXECUTE	53	+	113	K	153	RESULT
14	CONT	54	,	114	L	154	l
15	STEP	55	–	115	M	155	m
16	TRACE	56	.	116	N	156	n
17	RUN	57	/	117	O	157	ENTER EXP
20	RECALL	60	0	120	P	160	PRT ALL
21	FETCH	61	1	121	Q	161	q
22	BACK	62	2	122	R	162	END OF LINE
23	FORWARD	63	3	123	S	163	DELETE LINE
24	↓ (display)	64	4	124	T	164	FIXED N
25	↑ (display)	65	5	125	U	165	FLOAT N
26	← (display)	66	6	126	V	166	SCRATCH
27	→ (display)	67	7	127	W	167	AUTO #
30	LOAD	70	8	130	X	170	x
31	STORE	71	9	131	Y	171	y
32	INIT	72	:	132	Z	172	z
33	/	73	;	133	[	173	(not used)
34	END	74	<	134	■ (note 2)	174	(not used)
35	STD	75	=	135	]	175	(not used)
36	NORMAL	76	>	136	↑	176	(not used)
37	INSERT	77	?	137	⊢ (note 2)	177	STOP

Notes: 1 – f_{10} through f_{19} not available via the card reader.

2 – Codes 134 and 137 display a symbol but serve no other function.

3 – Not all of the lower case alphabet is available via the card reader.

† TRACE and NORMAL are the only instructions which can be input via the card reader while the calculator is running a program.

CARD READER OPERATION

(Continued)

ENCODING

> **NOTE**
>
> Please read Chapter 2, The Model 60 Card, in the Card Reader Operating Manual, before reading the following material.

Figure D-2 shows a card marked with two data numbers (94.62 and −136.328E6) to be input during an INPUT statement. Notice that the sequence of keys marked is exactly the same as the sequence which would be pressed if the data numbers were to be input from the keyboard.

The card is divided into columns and rows; each row corresponds to one key. The extreme left of the card can be used to write a title and date. The next column, labeled 'STEP', is for use with the 9810A calculator (where every keystroke in a program has to have a step number); so it can be ignored. The two columns marked 'KEY' and 'CODE' are for you to write the keys to be encoded and the code number for each key. Table D-2 shows the key corresponding to each valid code number (using invalid code numbers will produce unpredictable results). The columns of rectangular boxes are used for the actual coding that is read by the card reader. The card reader scans the boxes a row at a time and inputs the corresponding key code from each row into the calculator.

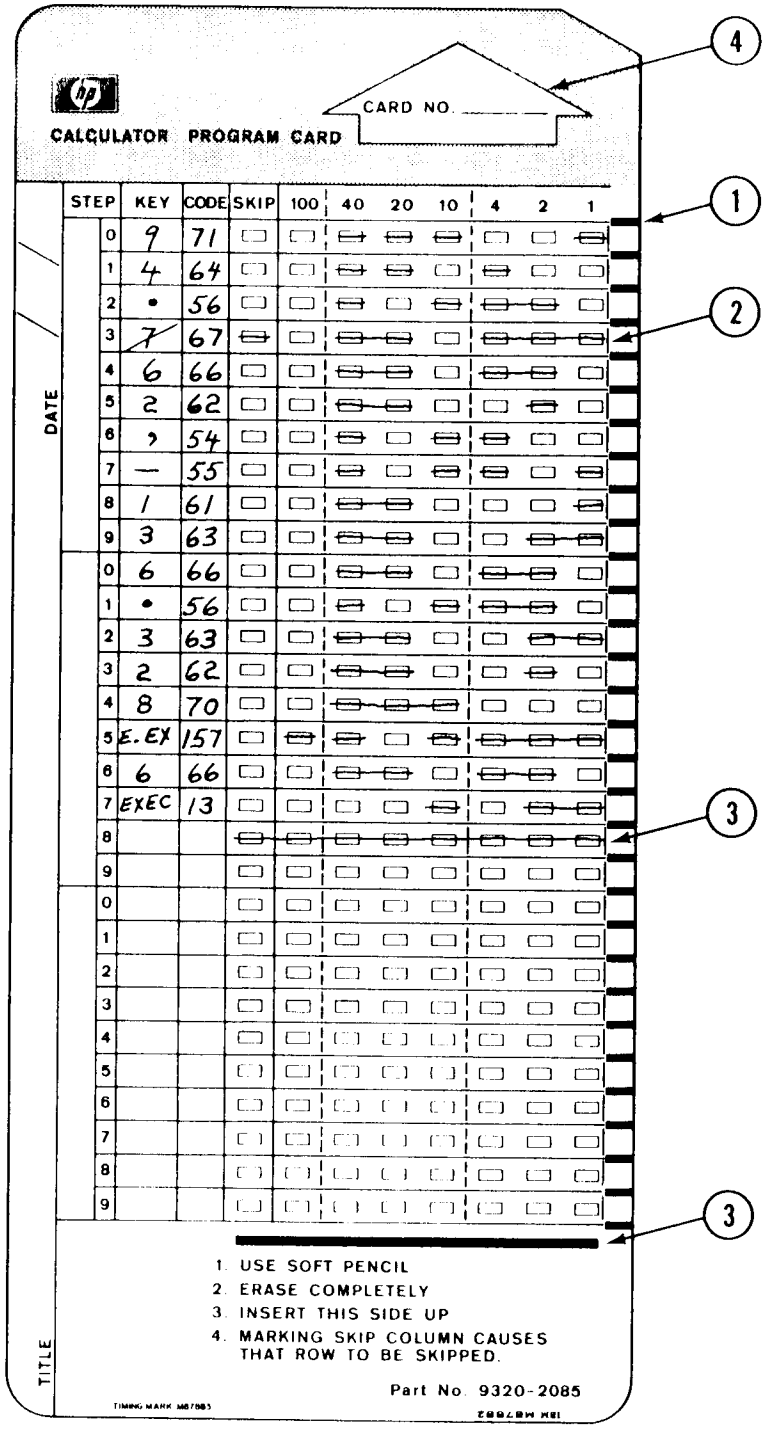
Ignoring the SKIP column for the moment, the columns marked 100, 40, 20, 10, 4, 2 and 1 are for marking the key-codes. These boxes are marked with a pencil, so that the value of the boxes marked on a row, when totaled, equals the code for the desired key. For example, the first key in the card shown in Figure D-2 is the digit 9, which has code number 71. 71 is encoded by marking boxes 40, 20, 10 and 1 (40+20+10+1=71).† Similarly, the ENTER EXP key, which has code 157, is encoded on the card by marking boxes 100, 40, 10, 4, 2, and 1. (The E key, code 105, could have been used instead of ENTER EXP.)

The SKIP column, if marked, will cause the Model 60 to ignore any boxes that are marked on that row. Marking the SKIP column can be used as an alternative to erasing; should you make an error in marking a row, you may either erase the error and correct it, or mark the SKIP column. In Figure D-2 the digit '7' was included in error, so the SKIP column was marked to prevent the card reader from reading that digit.

Near the bottom of the card, there is a preprinted mark that intersects all of the boxes. This mark, interpreted by the reader as 'SKIP 177', causes the card reader to stop reading the card. When a 'SKIP 177' is seen by the card reader, no additional information on that card is transferred to the calculator. If you do not use all of the rows on a card then you must mark a 'SKIP 177' immediately after the last row used. Otherwise, all unmarked rows will be interpreted as code 'Ø', the code for the f_0 key (and if f_0 is undefined, ERROR 10 will result).

If a key sequence is longer than can be encoded on a single card, simply encode the first part of the sequence on one card, and the second part on another. Neither card needs any special encoding. Just insert the cards, one after the other, in the obvious order.

† The computer-oriented operator will recognize the marked boxes as representing the binary equivalent of an octal number.



NOTES

① Strobe Marks.

② SKIP is marked to cover an error.

③ SKIP 177 turns the reader off. SKIP 177 is not a key-code – it is an instruction to the Model 60 only.

④ Insert card into the card reader, printed side up, in the direction indicated by the arrow.

Figure D-2. The Program Card

APPENDIX E

ERROR CODES

The error codes and messages listed in Table E-2 are also located on a slide-out card underneath the Model 30.

When an error occurs, the calculator makes a soft beeping sound and an error code appears on the display. The error message that corresponds to this code helps to pinpoint the cause of the error. Errors can be either recoverable or non-recoverable. The differences between the two are discussed next.

RECOVERABLE vs NON-RECOVERABLE ERRORS

Most errors are non-recoverable. When a non-recoverable error occurs in the calculator mode, press the RECALL key to return your input to the display. Make suitable corrections as referenced by the error message and then re-execute the line.

Recoverable errors (error 100 through 107) occur when you are working with very large or very small numbers. When a recoverable error occurs, an approximation of your result appears on the display together with the appropriate error code.

If, say, you execute the following:

```
A = 1.2E63*4E41
```

The display is:

```
ERROR 100    9.99999E+99
```

The code, ERROR 100, indicates that the positive range of the calculator has been exceeded. In this example the variable, A, is set equal to 9.99999999999E+99. (All the 9's are not displayed if you are in the standard mode.) The variable, A, can then be used in further calculations.

If either a recoverable or a non-recoverable error occurs during program execution, the program halts. If the error is recoverable, the program can be continued from that point if you press the keys – CONT EXECUTE. But if the error is non-recoverable, the program must be corrected and re-executed.

ERROR MESSAGES

Table E-2 is a duplicate of the slide-out, error-code card underneath the Model 30. Please keep in mind that the error messages only help to reference an error. In some cases, the particular error that is displayed may not pinpoint the specific error that occurred.

Errors 4 through 8 are general errors; they are often displayed before the calculator can determine a more specific error.

Table E-1, below, gives some additional explanations to particular error-code messages.

Table E-1. Additional Error-Message Explanation

Code	Message
ERROR 1	ROM configuration error; occurs if a program that requires a particular ROM is run without having the ROM installed.
ERROR 2	Memory overflow; occurs if the calculator needs more memory than is available. (Remember that during program execution, the calculator temporarily uses a small portion of user memory.)
ERROR 4	Missing line number or integer, or integer out of range; often occurs when END OF LINE is pressed instead of EXECUTE.
ERROR 5	No statement or command recognized; often occurs if END OF LINE is pressed instead of EXECUTE, or vice versa.
ERROR 40	Variable or function is undefined; most commonly occurs if the simple variable specified does not have a value.
ERROR 44	Line not found; occurs if the line referenced in a statement is not in memory.
ERROR 58	Cassette status error; occurs a) if the cassette door is open or slightly ajar; b) if the end of tape is reached without finding the specified file; c) if the cassette is protected (see Chapter 5); d) if the specified peripheral cassette memory is turned off; e) if a STOP command is given while information is being recorded on tape.
ERROR 59	Check sum error; often occurs if the tape head is dirty or if the portion of the tape accessed is damaged.
ERROR 60	Improper file size; also occurs if there is no COM statement in memory and you attempt to STORE DATA without the array specification.

A fold-out of both the error codes and the calculator keyboard is included here for your convenience.

Those error codes that are related to the ROM's are also given in the back of the ROM manuals.

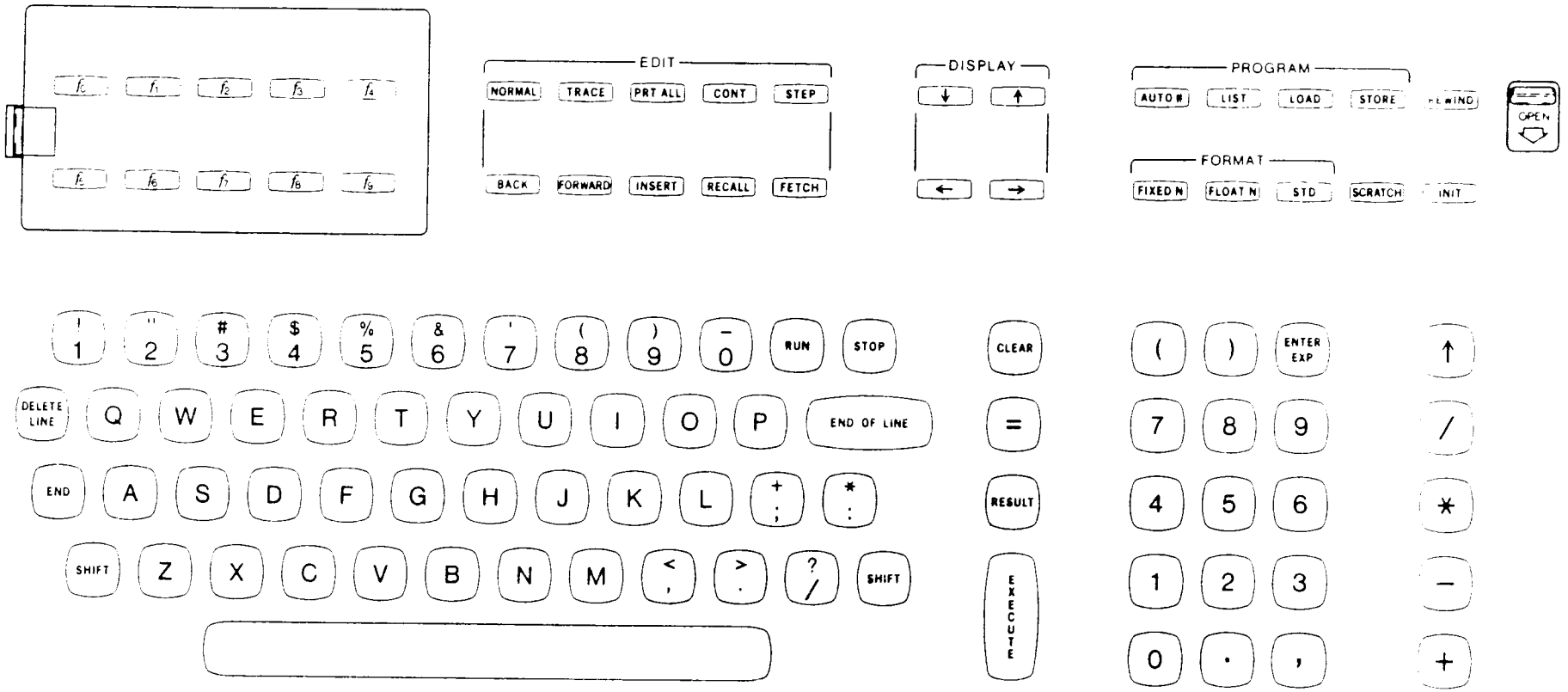


Figure E-1. Model 9830A Keyboard

Table E-2. Error Codes

9830 NON–RECOVERABLE ERRORS

1 Memory configuration error
2 Memory overflow
3 Statement is not allowed in keyboard mode
4 Missing line number or integer, or integer out of range
5 No statement or command recognized
6 Improper arithmetic expression, missing number or expression
7 Characters follow statement's logical end
8 Missing punctuation in program statement
9 Invalid command unless in KEY mode
10 User KEY is undefined
11 Exponent is out of range
12 Two decimal points in number
13 Sign given without number
14 Missing comma
15 Missing left parenthesis
16 Missing right parenthesis
17 Missing subscript
18 String not permitted
19 No opening quote or missing string variable
20 No closing quote
21 Missing or improper function identifier
22 Missing function parameter
23 Missing or incorrect DATA item
24 Improper IF . . . . THEN statement
25 Missing OF in conditional GO TO statement
26 Missing variable
27 Missing or improper FOR variable
28 Missing TO in FOR statement
29 Missing STEP or illegal characters following FOR statement
30 Missing assignment operator
31 Missing or improper assignment
32 Improper FORMAT specification
33 COM statement rules not followed
34 Improper common declaration
35 Array is doubly defined
36 Precision of variable is doubly defined
37 Inconsistent dimensions are given
38 Array has unknown dimensions
39 Dimensions are too large
40 Variable or function is undefined
41 Array or string has not been initialized
42 Subscript exceeds bounds
43 Select code exceeds bounds
44 Line not found
45 Improper statement type referenced
46 Improper statement nesting in multiline function
47 Improper RETURN
48 FOR statement has no matching NEXT or incorrect FOR nesting
49 Out of DATA
50 Last statement is not END
51 LOG or LGT of negative number
52 SQR of negative number
53 Zero to zero power
54 Non integer power of negative number
55 Cassette operation statement syntax error
56 Wrong file or file not found
57 Improper operation on SECURE program
58 Cassette status error
   a. Door open
   b. Clear leader
   c. Write not permitted
   d. Cassette power off
   e. STOP given during a write operation
59 Check sum error
60 Improper file size
61 Improper precision or data type
62 Improper file type
63 Program overlay

ROM-RELATED ERROR MESSAGES

Matrix Operations ROM

66 Matrix must be square for operation
67 Dimensions exceed DIM specifications
68 Matrix has no inverse
69 Incompatible dimensions

String Variables ROM

70 Incomplete IF statement
71 Improper string function syntax
72 Logical string length exceeded
73 Substring requested is undefined
74 Maximum string length exceeded
75 Character data found, numeric data expected
76 VAL function argument non-numeric
77 Improper characters entered in INPUT statement

Terminal I ROM

78 Improper select code or baud rate
79 Untranslated program lines in memory

Plotter Control ROM

80 SCALE statement not previously executed
81 Pen movement greater than 18.4% of plotting area height
82 Improper parameters in AXIS statement

Extended I/O ROM

83 End of data reached
84 Invalid format specification
85 Improper syntax in numeric input
86 Conversion table or code not found

Advanced Programming I ROM

87 First file on master tape is not file 0
88 Master tape file size exceeds 9830 memory
89 End of tape (clear-leader) reached

Mass Memory ROM

90 No power supplied to a component
91 File name or code exceeds allowable length
92 Improper protection code
93 Syntax not valid
94 File not found
95 Available storage space exceeded
96 File size not valid
97 File already exists
98 Improper file type or precision data
99 End of file or record marker reached

9830 RECOVERABLE ERRORS

100 Numeric overflow (assumes + or - ∞)
101 Numeric underflow (assumes 0)
102 LOG or LGT of zero (assumes -∞)
103 Division by zero (assumes + or - ∞)
104 Zero to negative power (assumes + ∞)
105 Integer variable overflow (assumes + or -32767)
106 Split variable overflow (assumes + or 9.99999E+63)
107 Split variable underflow (assumes 0)

Note The calculator approximates + and - ∞ by 9.99999999999E+99 and -9.99999999999E+99, respectively.

9830 NON–RECOVERABLE ERRORS

1 Memory configuration error
2 Memory overflow
3 Statement is not allowed in keyboard mode
4 Missing line number or integer, or integer out of range
5 No statement or command recognized
6 Improper arithmetic expression, missing number or expression
7 Characters follow statement's logical end
8 Missing punctuation in program statement
9 Invalid command unless in KEY mode
10 User KEY is undefined
11 Exponent is out of range
12 Two decimal points in number
13 Sign given without number
14 Missing comma
15 Missing left parenthesis
16 Missing right parenthesis
17 Missing subscript
18 String not permitted
19 No opening quote or missing string variable
20 No closing quote
21 Missing or improper function identifier
22 Missing function parameter
23 Missing or incorrect DATA item
24 Improper IF. . . . THEN statement
25 Missing OF in conditional GO TO statement
26 Missing variable
27 Missing or improper FOR variable
28 Missing TO in FOR statement
29 Missing STEP or illegal characters following FOR statement
30 Missing assignment operator
31 Missing or improper assignment
32 Improper FORMAT specification
33 COM statement rules not followed
34 Improper common declaration
35 Array is doubly defined
36 Precision of variable is doubly defined
37 Inconsistent dimensions are given
38 Array has unknown dimensions
39 Dimensions are too large
40 Variable or function is undefined
41 Array or string has not been initialized
42 Subscript exceeds bounds
43 Select code exceeds bounds
44 Line not found
45 Improper statement type referenced
46 Improper statement nesting in multiline function
47 Improper RETURN
48 FOR statement has no matching NEXT or incorrect FOR nesting
49 Out of DATA
50 Last statement is not END
51 LOG or LGT of negative number
52 SQR of negative number
53 Zero to zero power
54 Non integer power of negative number
55 Cassette operation statement syntax error
56 Wrong file or file not found
57 Improper operation on SECURE program
58 Cassette status error
   a. Door open
   b. Clear leader
   c. Write not permitted
   d. Cassette power off
   e. STOP given during a write operation
59 Check sum error
60 Improper file size
61 Improper precision or data type
62 Improper file type
63 Program overlay

9830 RECOVERABLE ERRORS

100 Numeric overflow (assumes + or - ∞)
101 Numeric underflow (assumes 0)
102 LOG or LGT of zero (assumes -∞)
103 Division by zero (assumes + or - ∞)
104 Zero to negative power (assumes + ∞)
106 Split variable overflow (assumes + or -9.99999E+63)
107 Split variable underflow (assumes 0)
105 Integer variable overflow (assumes + or -32767)

Note: The calculator approximates + and - ∞ by 9.99999999999E+99 and -9.99999999999E+99, respectively.

ROM-RELATED ERROR MESSAGES

Matrix Operations ROM

66 Matrix must be square for attempted operation.

67 New dimensions exceed existing DIM specifications.

68 Matrix has no inverse. The data contained in the matrix does not have a solution.

69 Incompatible dimensions.

String Variables ROM

70 Incomplete IF statement

71 Improper string function syntax.

72 Logical string length exceeded.

73 Operation is non-contiguous string. Substring requested is beyond the logical boundary for the string and is undefined.

74 Maximum string length exceeded.

75 DATA encountered during READ statement execution. Character data found; numeric data expected.

76 VAL function argument non-numeric.

77 Illegal characters entered during INPUT statement execution. Character data found, numeric data expected.

Terminal I ROM

78 The TERM command contains improper select code or baud rate specification.

79 The RUN command is non-executable because untranslated program lines are stored in the calculator memory.

Plotter Control ROM

80 Attempt to execute an AXIS, OFFSET, PLOT or IPLOT statement before executing a SCALE statement.

81 – 'Character height' specification in a LABEL statement greater than 18.4% of the height of the plotting area.

– 'Aspect ratio' in a LABEL statement specifies a character width greater than 18.4% of the height of the plotting area.

– The X or Y parameter in a CPLOT statement requires a pen movement greater than 18.4% of the height of the plotting area.

82 Attempt to execute an AXIS statement

a. with the 'start point' specified to be out of the plotting area, or

b. with the tic mark spacing too small

Extended I/O ROM

83 End of data reached or data contains more than ten blanks in a row.

84 Invalid format specification.

85 Numeric input has syntax error: multiple decimal points, more than one E, or other non-numerical input.

86 Conversion table or code not found. Check for integer initialization in DIM statement.

Advanced Programming I ROM

87 First file on master tape is not file 0; negative file count specified in DUP command; or files on master tape are not sequential.

88 File size on master tape is larger than available memory.

89 End of tape (clear-leader) reached before DUP command is completed.

Mass Memory ROM

90 Mass Memory power OFF; Controller power OFF; Mass Memory fault or drive not ready; Specified UNIT does not exist; Check word or address error; Hardware write protect (write not permitted).

91 File name or protection code greater than six characters or of zero length.

92 Protected file accessed in FILES statement; Incorrect protection code; Protection code is not given for protected file; Protection code is given for an unprotected file; File already protected.

93 Syntax not valid.

94 File not found; File number reference not valid; Record number reference not valid; Unit number not valid; File not assigned.

95 Available storage space exceeded; Availability table full; Directory full.

96 File size not valid; Null program.

97 File already exists.

98 Improper file type; Improper precision data type; Numeric overflow on data type conversion.

99 End of file marker reached; End of record marker reached.

ROM-RELATED ERROR MESSAGES

(continued)

Data Communications ROMs

79 The RUN command cannot be executed because untranslated program lines are stored in the calculator memory. If the stored program lines are valid BASIC language instruction, execute the COMP command and then execute the RUN command.

300 The Binary Synchronous ROM is not plugged into the calculator.
The TBATCH statement was executed while in Binary Synchronous mode.

301 The STOP key was pressed during the DIAL sequence.
Seven DIAL attempts were completed without an answer (this includes a busy line or no answer).

302 TBATCH was executed out of TEXT mode.

303 BIN code in a TREAD or TWRITE is not allowed in Binary Synchronous mode. Use BWRITE statement.

304 The Data Comm. 1 ROM is not plugged into the calculator.

305 An ENQ is received in response to an ENQ. ACKCLR is executed automatically.

306 ACKs are out of step during a message in the TWRITE statement. ACKCLR is executed automatically and an EOT is automatically sent.

307 TWRITE note: 15 ENQs transmitted without response, or bad data was received in response to a message block or ENQ. An acknowledgment was expected but not received. ACKCLR is automatically executed and an EOT is sent automatically.

308 TWRITE note: Received 15 NAKs to a block of text. ACKCLR is executed automatically and an EOT is transmitted automatically.

309 An EOT was received as a response after a TWRITE. This indicates one of the following:
1. Receiver has I/O problems and cannot continue current reception.
2. Receiver is unable to accept a message at this time.
ACKCLR is executed automatically.

310 TWRITE note: A NAK was received in response to an ENQ. ACKCLR is executed automatically.

311 BTIMR has expired without any line activity in the TREAD statement. ACKCLR is executed automatically.

APPENDIX F

PRINTER OPERATING PROCEDURES

This appendix discusses operation of the primary printers described in Appendix A.

PRINTER SELECT CODE

Any device connected to the calculator requires a 'select code' so that the calculator can distinguish it from all other devices. The select code on a primary printer must be set to '15'. The 9866A Printer (Option 30), the interface for the 9861A Output Typewriter, and the 11205A Serial I/O Interface used with the teleprinter, are all preset, at the factory, to select code 15. But any of the printers can be used as a secondary printer with the Model 30, too, if the select code of the supplied interface card is changed. (To use the 9866A Printer as a secondary printer, the -hp- 09866-61610 interface card must be purchased.)

THE 9866A PRINTER

Of the printers discussed in Appendix A, the 9866A Printer is easiest to operate because it has no mechanical controls, other than the PAPER key, which is used to manually advance paper. Once the printer has been connected to the calculator, loaded with paper, and turned on (see Appendix A), it is controlled by the calculator.

The statements and commands used to control the printer are fully described in the appropriate places in this book. Table F-1 lists each operation and tells you in which chapter to look for its explanation. You may also wish to refer to 'Printer Character Codes', later in this appendix; however, it is recommended that you become proficient in the use of the WRITE and FORMAT statements before doing so.

THE 9861A OUTPUT TYPEWRITER

Before the typewriter can be controlled from the calculator, its controls must be properly set. Necessary control settings on the typewriter consist of:

- The desired line-spacing;
- The margins, to suit your paper;
- Black ribbon (red or black ribbon can be set from the calculator, but only if the typewriter is set to black ribbon);
- The 'shift' key being unlocked (the calculator will automatically shift the typewriter keyboard, if it needs to, to obtain a particular character);
- The impact regulator being set for the desired copy hardness;
- Select code 15 being set.

Even though the typewriter is connected to the calculator, the keyboard on the typewriter is completely operational. So, if you wish, you can clear and set tabs, carriage-return, or perform any other 'initializing' operations; or you can type directly

(Continued)

THE 9861A OUTPUT TYPEWRITER

(Continued)

from the typewriter keyboard. All of these operations can also be controlled from the calculator.

The statements and commands used to control the typewriter (which are the same as those used to control the 9866A Printer) are fully described in the appropriate places in this book (see Table F-1). The section, 'Printer Character Codes', later in this appendix gives the codes that are needed to perform additional control functions, such as clearing and setting tabs, and changing from black to red ribbon.

The calculator automatically shifts the typewriter keyboard if it needs to do so to print a particular character. So, when you are typing characters for the typewriter on the calculator keyboard, you do not need to know whether the typewriter keyboard is to be shifted or not; but you do need to know if the calculator keyboard has to be shifted (as it does, for example, when you wish to print a lower case letter).

Example

Following is a printout, from a typewriter, and the program which produced it. The program demonstrates some of the flexibility that you have in formatting the typewriter output. This program relies heavily on the use of the character codes, which are described at the end of this appendix. Therefore it is recommended that you become proficient with the typewriter and, in particular, with the WRITE and FORMAT statements, before you spend any time analyzing this program.

```
The 9861A Typewriter can be controlled by the 9830
Calculator. The ribbon can be changed to red and
back to black.

                Physical tabs can be set,
                and the tab key can be operated.
                The following special characters
                can be typed:
                                [
                                ]
                                {
                                }
                                ~
                                "
                                `

Also you can cause the typewriter to
skip lines, backspace, and clear tabs.
```

```
10 FORMAT 15B
20 FIXED 0
30 X=9830
40 PRINT TAB60;
50 WRITE (15,10)11,7
60 PRINT "The 9861A Typewriter can be controlled by the";X
70 PRINT "Calculator. The ribbon can be ";
80 WRITE (15,10)6"changed to red "7"and"
90 PRINT "back to black."
100 PRINT
110 PRINT TAB18;
120 WRITE (15,10)1,"Physical tabs can be set,"
130 WRITE (15,10)9"and the tab key can be operated."
140 WRITE (15,10)9"The following special characters "
150 WRITE (15,10)9"can be typed:"
160 PRINT TAB18;
170 WRITE (15,10)12;
180 PRINT TAB30;
190 WRITE (15,10)1;91,10,8,93,10,8,123
200 WRITE (15,10)9,125,10,8,126,10,8,34,10,8,96
210 WRITE (15,10)11,10,10,"Also you can ";
220 PRINT "cause the typewriter to"
230 WRITE (15,10)"skip lines, backspace";
240 FOR I=1 TO 9
250 WRITE (15,10)8;95;8;
260 NEXT I
270 PRINT "          , and clear tabs."
280 END
```

Table F-1. Printer Operations

COMMAND or STATEMENT	DESCRIPTION	REFER TO CHAPTER...
PRINT	Standard printing statement	3
WRITE	Enables more flexible format for printing	3
LIST	Lists program lines	4
PRT ALL	Prints each operation	2
TRACE	Prints line number as line is executed	4
NORMAL	Cancels TRACE	4
TLIST	Lists information from tape cassette	5

THE TELETYPE 38 ASR DATA TERMINAL

Before the teleprinter can be controlled by the calculator, its controls must be properly set.

On the teleprinter:

- Select LINE operation;
- Ensure that SHIFT LOCK is unlocked.

On the interface:

- Set select code 15 (see page F-1, Printer Select Code);
- Set a baud rate of 110 – the interface is preset at the factory for a baud rate of 110, so it will probably not need adjusting† (refer to the manual for the 11205A Serial I/O Interface for complete details).

The statements and commands used to control the teleprinter (which are the same as those used to control the 9866A Printer) are fully described in the appropriate places in this book (see Table F-1). The section 'Printer Character Codes', later in this appendix, gives the codes that are needed to perform additional control functions, such as changing ribbon color, form-feeding, and executing a line feed without a carriage return (or vice versa).

The calculator automatically shifts the teleprinter keyboard if it needs to do so to print a particular character. So, when you are typing characters for the teleprinter on the calculator keyboard, you do not need to know whether or not the teleprinter keyboard has to be shifted for that character; but you do need to know if the calculator keyboard has to be shifted (as it does, for example, when you wish to print a lower case letter).

Example

Following is a printout, from a teleprinter, and the program which produced it. The program demonstrates some of the flexibility that you have in formatting the teleprinter output. This program relies heavily on the use of the character codes, which are described at the end of this appendix. Therefore, it is recommended that you become proficient with the teleprinter and, in particular, with the WRITE and FORMAT statements, before you spend any time analyzing this program.

```
The Model 38 teleprinter can be controlled by the 9830A
Calculator. Among other things, the ribbon can be changed
to red and back to black.

          The teleprinter can be made to
               tab        to any        position,
          as can be seen from this printout!

The following special characters
can be typed:
                " [ \ ]

                ` { | } ~

Also you can make the teleprinter carriage return
without executing a linefeed. This enables you to
underline.
```

† If necessary, it is very easy to adjust the baud rate, by means of a calibrated screwdriver control on the underside of the interface card. Remove the card from the interface slot at the back of the calculator, and adjust the control to the desired setting.

```
10 FORMAT F5.0
20 X=9836
30 PRINT
40 PRINT
50 WRITE (15,60)"can be controlled by the",X
60 FORMAT "The Model 38 teleprinter ",F5.0,"A"
70 PRINT "Calculator. Among other things, the ribbon ";
80 WRITE (15,10)"can be "27,51"changed"
90 WRITE (15,10)"to red and "27,52"back to black."
100 PRINT
110 PRINT TAB10"The teleprinter can be made to "
120 PRINT TAB15"tab"TAB25"to any"TAB39"position,"
130 PRINT TAB10"as can be seen from this printout!"
140 PRINT
150 PRINT "The following special characters"
160 PRINT "can be typed:"
170 PRINT TAB13;
180 WRITE (15,10)34,32,91,32,92,32,93,10
190 PRINT TAB13;
200 WRITE (15,10)96,32,123,32,124,32,125,32,126
210 WRITE (15,10)10"Also you can ";
220 PRINT "make the teleprinter carriage return"
230 WRITE (15,10)"without executing a linefeed. This ";
240 WRITE (15,10)"enables you to"13,0;
250 PRINT TAB47;
260 WRITE (15,10)95,95
270 WRITE (15,10)"underline."13,0;
280 FOR I=1 TO 10
290 WRITE (15,10)95;
300 NEXT I
310 PRINT
320 END
```

Notice in lines 80 and 90 of the program that the teleprinter requires two keys to change ribbon color:

ESC 3 (codes 27, 51) selects red ribbon;
ESC 4 (codes 27, 52) selects black ribbon.

PRINTER CHARACTER CODES

Table F 2 lists the decimal codes used in WRITE statements to reference FORMAT B statements (described in Chapter 3). The table lists (decimal) numbers Ø through 127 and gives the corresponding ASCII character (or control function) for the 9866A Printer, for the 9861A Typewriter and for the 38 ASR teleprinter. Some of these characters and control functions can be obtained only by using their decimal equivalents in WRITE (with FORMAT B) statements. However, many of them can also be obtained directly from the calculator keyboard. An empty space opposite a decimal code in the table indicates that the specific printing device does not recognize that code number and that it will simply ignore it.

(Continued)

PRINTER CHARACTER CODES

(Continued)

> **NOTE**
>
> If you are using any other printer, refer to its manual for its character set. Some manuals may give the binary or octal equivalents, rather than the decimal equivalents of the characters. In these cases you must convert the binary or octal numbers to their equivalent decimal values, because only the decimal numbers can be used in WRITE (with FORMAT B) statements.

In general, you will not need the character codes for the 9866A Printer because most of its characters, and all of its control functions, can be obtained from the calculator keyboard. When using the typewriter or the teleprinter, you will need these codes to perform additional control functions, such as changing from black ribbon to red ribbon, setting tabs (on the typewriter) and form-feeding (on the teleprinter).

The codes for all of the teleprinter keys are included. However, the teleprinter itself does not respond to some of these keys, even though, as a terminal, it may transmit them. For example, it has a backspace key (BS, code 8) but its carriage cannot backspace. (Refer to the manual supplied with the teleprinter for complete details.)

Table F-2. Printer Characters and Equivalent Decimal Codes

Decimal Code	9866A Printer	9861A Typewriter	38 ASR Teleprinter	Decimal Code	9866A Printer	9861A Typewriter	38 ASR Teleprinter
0			NUL	25			EM
1		TAB SET	SOH	26			SUB
2			STX	27			ESC
3			ETX	28			FS
4			EOT	29			GS
5			ENQ	30			RS
6		RED RIBBON	ACK	31			US
7		BLACK RIBBON	BEL	32	SPACE	SPACE	SP
8		BACKSPACE	BS	33	!	!	!
9		TAB	HT	34	"	"	"
10	RETURN & LINE FEED	LINE FEED	LF	35	#	#	#
11		CLEAR ALL TABS	VT	36	$	$	$
12		TAB CLEAR	FF	37	%	%	%
13		CARRIAGE RETURN	CR	38	&	&	&
14			SO	39	'	'	'
15			SI	40	(	(	(
16			DLE	41	)	)	)
17			DC1	42	*	*	*
18			DC2	43	+	+	+
19			DC3	44	,	,	,
20			DC4	45			
21			NAK	46	.	.	.
22			SYN	47	/	/	/
23			ETB	48	0	0	0
24			CAN	49	1	1	1

Table F-2. Printer Characters and Equivalent Decimal Codes (cont'd)

Decimal Code	9866A Printer	9861A Typewriter	38 ASR Teleprinter	Decimal Code	9866A Printer	9861A Typewriter	38 ASR Teleprinter
50	2	2	2	90	Z	Z	Z
51	3	3	3	91	[	[	[
52	4	4	4	92	\		\
53	5	5	5	93	]	]	]
54	6	6	6	94	↑	^	^
55	7	7	7	95	_	_	_
56	8	8	8	96	@	'	'
57	9	9	9	97	A	a	a
58	:	:	:	98	B	b	b
59	;	;	;	99	C	c	c
60	<	<	<	100	D	d	d
61	=	=	=	101	E	e	e
62	>	>	>	102	F	f	f
63	?	?	?	103	G	g	g
64	@	@	@	104	H	h	h
65	A	A	A	105	I	i	i
66	B	B	B	106	J	j	j
67	C	C	C	107	K	k	k
68	D	D	D	108	L	l	l
69	E	E	E	109	M	m	m
70	F	F	F	110	N	n	n
71	G	G	G	111	O	o	o
72	H	H	H	112	P	p	p
73	I	I	I	113	Q	q	q
74	J	J	J	114	R	r	r
75	K	K	K	115	S	s	s
76	L	L	L	116	T	t	t
77	M	M	M	117	U	u	u
78	N	N	N	118	V	v	v
79	O	O	O	119	W	w	w
80	P	P	P	120	X	x	x
81	Q	Q	Q	121	Y	y	y
82	R	R	R	122	Z	z	z
83	S	S	S	123	|	{	{
84	T	T	T	124	\		¦
85	U	U	U	125	|	}	}
86	V	V	V	126	↑	~	~
87	W	W	W	127			DEL
88	X	X	X				
89	Y	Y	Y				

INDEX

S

T

V

W

F

G

H

I

K