



**HEWLETT-PACKARD 9820A CALCULATOR
SYSTEM REFERENCE**

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

GENERAL OPERATION
DISPLAY

IMPLIED → Z

WRITING
EDITING LINES

END STATEMENTS

STORING / EDITING
RUNNING PROGRAMS

ENTER STATEMENTS

TRACE MODE OPERATION

FLAGS

BASIC ERROR NOTES

BRANCHING WITH IF

MEMORY STRUCTURE

BASIC SYNTAX LISTS

DATA STORAGE
DESIGNATING REGISTERS

DO / SYNTAX LIST

HIERARCHY
PARENTHESES

DO ERROR NOTES

MATHEMATICAL FAUX PAS

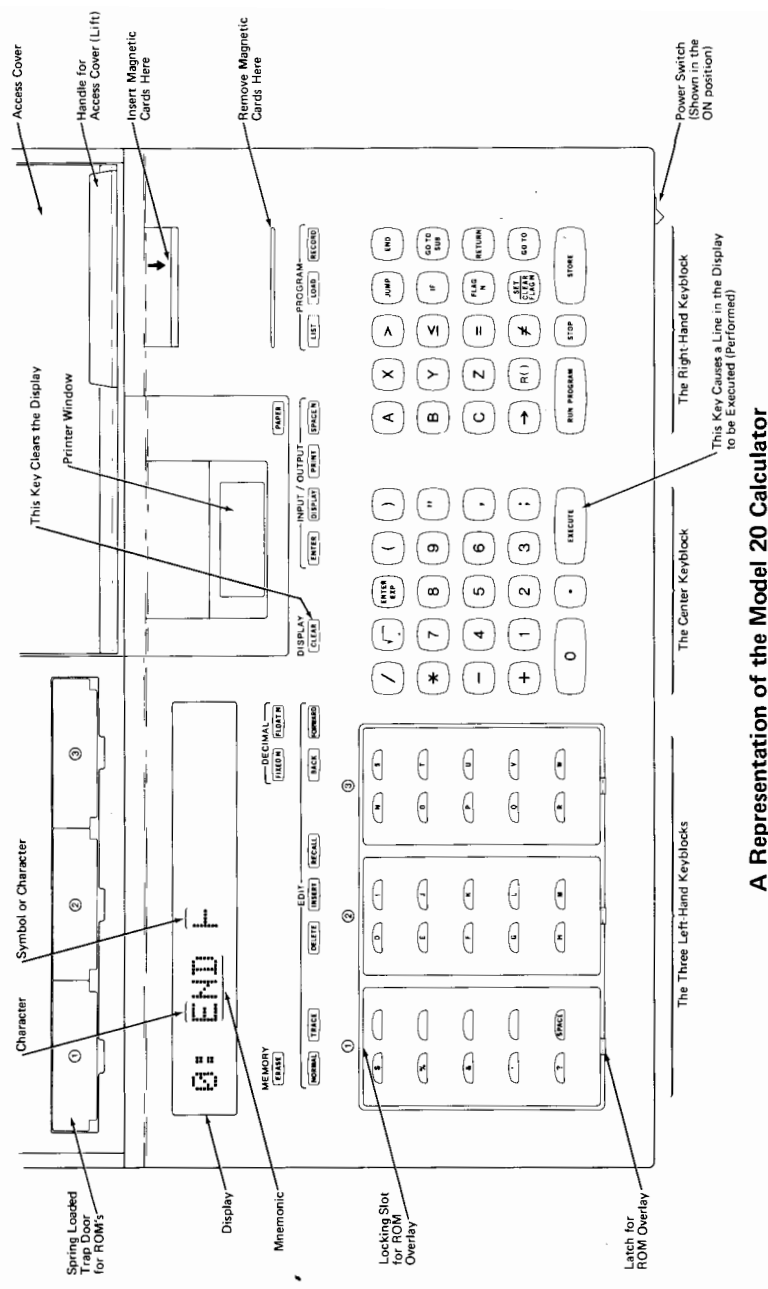
DOF APPENDIX

MATH ROM

DOF APPENDIX

USING THE PRINTER

USING THE
MAG CARD READER



A Representation of the Model 20 Calculator

PREFACE

The purpose of this booklet is to serve as a quick reference for the operation of the Model 20 Calculator. Not every subject is covered, nor are the explanations always complete. It is assumed that the user is somewhat familiar with the Calculator, and that, for the most part, this booklet needs only to refresh his memory.



GENERAL OPERATION

1. First tell the calculator what you want it to do (write a line).
2. Then tell it to do it (press EXECUTE, STORE, or RUN PROGRAM).

Lines are written by pressing valid sequences of keys. A keying mistake can usually be easily corrected by an editing operation.

Once the display contains the line that will perform the desired activity, press EXECUTE and the calculator will perform the operations indicated by the line. The answer (if there is one) will appear in the display. Pressing STORE instead will cause the calculator to "remember" that line as a part of a program. A line number is automatically assigned to identify that line.

Pressing RUN PROGRAM causes the calculator to execute the line in the display, and then to begin running the program in that has been stored in the memory. A program is run by automatically executing the lines of the program in one-after-the-other fashion.

PRESS:

corresponding display not shown

PRESS: 5 ((4 ((3 + 2)) + 5) / - √ 2 5

DISPLAY: 5(4(3+2)+5)/-√25

PRESS: EXECUTE

DISPLAY: -25.00

PRESS: GO TO 0 EXECUTE

corresponding display not shown

DISPLAY: T

"Lazy T" indicates that no numerical answer is associated with the line just executed.

PRESS: X ((A + B)) → Y

DISPLAY: X(A+B)→Y

PRESS: STORE

DISPLAY: 0: X(A+E) ÷ Y↑

Indicates the line number that identifies the line just stored. Note that it is added automatically.

The lazy T is also called the end-of-line symbol.

THE DISPLAY

The length of a line is not limited to what will fit in the display. Think of the display as a window that shows the last 13 to 16 characters worth of information that has been keyed in.

PRESS: **FIXED** 2 ; (3 * 4 + 4 * 4) → A ;

PRINT " " **Y** **SPACE** = " " , A

Notice how the display shifts as the keys after the arrow are pressed.

This is not visible — but is still present in the calculator.

This much is visible

DISPLAY: FXD 2: r(3*3+4*4) ÷ A: PRt "HYP =": A

PRESS: **EXECUTE**

RESULT: HYP = 5.00

HYP = 5.00

The form of numerical answers that appear in the display (or are printed) is controlled by the **FIXED N** and **FLOAT N** keys.

The **FIXED N** key is used to specify displays of numbers with the decimal point located in its 'true' location.

FIXED N (n) EXECUTE Where n represents a digit from 0 to 9.

The **FLOAT N** key specifies floating point displays, i.e., digit-point-digit times a power of ten.

FLOAT N (n) EXECUTE Where n represents a digit from 0 to 9.

In each case n is used to determine the number of digits to be shown to the right of the displayed decimal point. If a number is too large to be shown in fixed point, it will be shown in floating point instead.

WRITING LINES

The *line* is the fundamental unit of communication with the calculator. Lines are composed of *statements*; if there are more than one statement in a line, they must be separated with semicolons.

statements
|
FXD 2:PRT "18/3(1+2)=";PRT 18/3(1+2)
line

Generally, lines are executed from left to right by statements.

To write a line, press a series of keys that will produce a display that will accomplish the desired activity. It often happens that the needed activity can be accomplished by any of several different lines. Choose the one you are most comfortable with, provided it is not too long.

Every line is subject to restrictions on its length. These restrictions have to do with a process called compilation, and are enforced by the machine itself as each line is written. If a line is made too long, NOTE 09 appears in the display. When this happens, you must either try to salvage the line by shortening it with editing techniques*, or, you must rewrite the line entirely, in a shortened form. To do this, press CLEAR and re-write the line.

The maximum line length is about 68 keystrokes, although it may be as short as 35, depending upon the nature of the line. A typical maximum length is 50 keystrokes.

*The Operating and Programming Manual strongly advises against attempting to salvage the line, even though it can often be done. The reason is because under the proper circumstances, the compiler can add spurious characters onto the end of the line, which cannot be removed except by clearing and re-writing the entire line.

MNEMONICS AND SYMBOLS FOR THE MODEL 20 MAINFRAME

KEY	IN QUOTE FIELD?		KEY	IN QUOTE FIELD?		KEY	IN QUOTE FIELD?		KEY	IN QUOTE FIELD?	
	NO	YES		NO	YES		NO	YES		NO	YES
(1)	1	1	(R1)	R	:	(LOAD)	LOAD	NOTE 2	(S)	NOTE 3	S
(2)	2	2	(=)	=	=	(RECORD)	RECB	NOTE 2	(T)	NOTE 3	T
(3)	3	3	(≠)	≠	≠	(A)	A	A	(U)	NOTE 3	U
(4)	4	4	(>)	>	>	(B)	B	B	(V)	NOTE 3	V
(5)	5	5	(≤)	≤	≤	(C)	C	C	(W)	NOTE 3	W
(6)	6	6	(GO TO SUB)	GTOB	ā	(X)	X	X	(3)	NOTE 3	3
(7)	7	7	(RETURN)	GSBB	NOTE 2	(Y)	Y	Y	(%)	NOTE 3	%
(8)	8	8	(STOP)	RETB	NOTE 2	(Z)	Z	Z	(&)	NOTE 3	&
(9)	9	9	(END)	STPB	!	(D)	NOTE 3	D	(?)	NOTE 3	?
(0)	0	0	(JUMP)	ENDB	NOTE 2	(E)	NOTE 3	E	(b)	NOTE 3	b
(.)	.	.		JMPB	NOTE 2	(F)	NOTE 3	F		NOTE 3	

ENTER ESP	E	↑	"	IFb	NOTE 2	G	NOTE 3	G	These keys are the Mechanics Group and have no Mnemonics or Symbols.
+	+	+	FLAG N	FLGb	NOTE 2	H	NOTE 3	H	EXECUTE
-	-	-	SET CLEAR FLAG	SFGb	NOTE 2	I	NOTE 3	I	STORE
*	*	*	SET CLEAR FLAG	CFGb	NOTE 2	J	NOTE 3	J	RUN PROGRAM
/	/	/	FINDb	FXDb	NOTE 2	K	NOTE 3	K	CLEAR
√	√	√	FORMAT	FLTb	NOTE 2	L	NOTE 3	L	ERASE
(((ENTER	ENTb	NOTE 2	M	NOTE 3	M	BACK
)))	DISPLAY	DSPb	NOTE 2	N	NOTE 3	N	FORWARD
:	:	:	PRINT	PRTb	NOTE 2	O	NOTE 3	O	DELETE
;	;	;	SPACEb	SPCb	NOTE 2	P	NOTE 3	P	INSERT
"	"	NOTE 1	NORMAL	NORb	NOTE 2	Q	NOTE 3	Q	RECALL
→	→	→	TRACE	TRCb	NOTE 2	R	NOTE 3	R	LIST

1. The " character never occurs inside a quote field; it is used exclusively to begin or terminate a quote field.

2. This key produces one character with an arbitrary pattern. Sometimes the pattern will vary according to which plugin ROM's are installed.

3. If this key is used outside of a quote field, NOTE 1 1 will result unless the key is defined by a plugin ROM, in which case the mnemonic or symbol is determined by the ROM.

4. The character b denotes a blank space.

Most of the keys on the keyboard place a *mnemonic* or *symbol* in the display. See the table. However, some keys cause immediate activity:

ERASE	CLEAR
DELETE	LIST
INSERT	EXECUTE
RECALL	RUN PROGRAM
BACK	STORE
FORWARD	

These keys are easily recognized; they are yellow.

In addition, the keys STOP, NORMAL, TRACE, FIXED N and FLOAT N may or may not cause immediate activity, depending upon how they are used.

EDITING LINES

If a line (or its replica) is present in the display, the line can be modified in many different ways.

1 - EXTENSIVE CHANGES ARE REQUIRED:

Press CLEAR and re-write the line. This is often the easiest method if extensive alteration is required.

2 - WHILE WRITING A LINE, THE LAST KEY PRESSED WAS INCORRECT:

Press BACK once, then the correct key, and continue with your activities. This procedure works even if the incorrect key causes a NOTE to appear in the display.

3 - TO COMPLETELY CHANGE A LINE FROM A CERTAIN POINT ONWARDS:

Press BACK as many times as required to blank out the segment to be changed. Then continue writing the rest of the line.

IF r(AA-BB)>E-6!GTO "OVER"

PRESS: [BACK] [BACK] [BACK] [BACK]

IF r(AA-BB)>E-6!GTO "OVER"

PRESS: [L] [0] [P] [1] ["]

IF r(AA-BB)>E-6!GTO "LOOP1"

4 - TO REPLACE A SEGMENT OF A LINE WITH ANOTHER SEGMENT HAVING THE SAME NUMBER OF KEYSTROKES:

Press BACK as many times as required to blank out the segment to be replaced. Then press the correct keys to write the new segment. Then press FORWARD as many times as required to return to the end of the line.

IF r(AA-BB)>E-6!GTO "LOOP1"

PRESS: 10 times

IF R(AA-BB)>E-6:GTO "LOOP1"

PRESS: and then 9 times

IF R(AA-BB)>E-8:GTO "LOOP1"

5 - TO REMOVE A SEGMENT FROM A LINE:

Press BACK until the right-most item of the segment to be removed is also the right-most item visible in the display. Then press DELETE once for each item to be removed. Then press FORWARD to return to the end of the line.

A+ . 1→A:B+1→B:C+10→C

PRESS: 6 times

A+ . 1→A:B+1→B:C+10→C

PRESS: 6 times

A+ . 1→A:C+10→C



PRESS: **FORWARD** 6 times

A+1+R+C+10+C

6 - TO ADD A SEGMENT TO THE INTERIOR OF A LINE:

Press **BACK** until the right-most item in the display is the item that is to immediately precede the material to be added. Then press **INSERT**, followed by the correct keys to write the new segment. Then press **FORWARD** to return to the end of the line.

If you make a mistake, correct it using any of the methods mentioned. However, that will cancel the insert mode. Be sure to press **INSERT** before continuing after making the correction.

B+1+B:GTO -2

PRESS: **BACK** 3 times

B+1+B:GTO -2

PRESS: **INSERT** **PRINT** **"** **B** **=** **"** **,** **B** **:** **SPACE** **2** **:**

B+1+B:PRINT "B=";B:SPC 2:GTO -2

PRESS:  3 times



7 - TO REPLACE A SEGMENT WITH ONE OF A DIFFERENT LENGTH (IN KEYSTROKES):

Remove the old segment (case 5) and add the new one in its place (case 6).

STORING PROGRAMS

To store a program in the Calculator:

1. Set the program line counter to the beginning line number. Use END EXECUTE, (for line 0) or GO TO (n) EXECUTE.
2. Key in the first line. If you make an error, correct it using any of the editing techniques.
3. Press STORE. A *replica* of the line will appear in the display.
4. Write the next line of the program, and press STORE, etc.

As each line is stored, it is replaced in the display by its replica. The replica shows the line number that was assigned, as well as any changes made by the compiler, such as adding or removing parentheses. The replica can be inspected with BACK and FORWARD, just as any other line.

Line numbers are the basic means of distinguishing between lines, and are automatically assigned by the Calculator as each line is stored. Line numbering starts at zero, and counts upward by one.

Programming and data storage use the same memory. The more programming stored, the less memory is available for data storage.

To store a simple counting program:

PRESS:

PRESS:

0→A#FXD 1

PRESS:

↙ This is a replica of that

0: 0→A#FXD 1

↙ line number

↙ Indicates that line has been compiled

PRESS:

(A+.1)→A#DSP A#JMP 0

PRESS:

```
1: R+.1+R;DSP R;JMP 0F
```

Notice that the unnecessary parentheses have been removed.

Once a program has been stored, it can be listed on the printer. Set the program line counter to the number of the line the listing is to begin with, (use GO TO (n), or, END), and press LIST.

```
0: 0+R;FXD 1F  
1: R+.1+R;DSP R;  
JMP 0F  
R423
```

line numbers

This is the number of R registers left available — R0 through R422.

EDITING PROGRAMS

The fundamental cases that occur while modifying programs are:

1 - ADDING LINES ONTO THE END OF AN EXISTING PROGRAM:

Use GO TO (n) EXECUTE to set the program line counter to the line number that the first additional line is to have. Then write and store each line.

2 - REPLACING A LINE WITH ANOTHER LINE (LENGTH DOESN'T MATTER):

Use GO TO (n) EXECUTE to set the program line counter to the number of the line that is to be changed. Then write the new line and press STORE. The relative lengths of the old and new lines do not matter.

3 - TO REMOVE A LINE FROM A PROGRAM:

Use GO TO (n) EXECUTE to set the program line counter to the number of the line to be removed. Then press RECALL DELETE. (RECALL brings a replica of the doomed line into the display. DELETE removes it. Then the display will show a replica of the next line in the program.)

There is now one less line in the program, and therefore one less line number. No break in numbering of line numbers is permitted. To prevent this, the Calculator decreases by one the line number of each line past the one that was removed. That is why the replica that appears in the display, after RECALL DELETE is pressed, has the same line number as had the line that was removed.

4 - TO ADD A LINE TO THE INTERIOR OF A PROGRAM:

Use GO TO (n) EXECUTE to set the program line counter to the number that the new line is to have. Then write the line and press INSERT STORE. A replica of the new line will appear in the display, and the line numbers of all succeeding lines will be increased by one.

This process leaves the program line counter set to a count one higher than it started at. Thus, if another line is to be inserted after the first one, simply write it and again press INSERT STORE.

5 - TO ALTER AN EXISTING LINE:

Use GO TO (n) EXECUTE to set the program line counter to the number of the line to be altered. Then press RECALL

to bring a replica of the line into view. (The display will contain only a replica of the line, the actual line is still stored in the program memory, even if you press CLEAR.)

Now modify the line, using any of the techniques for altering a line written from the keyboard. (The only thing you must not do, however, is press DELETE while `├` is visible in the display.)

To replace the original line with the modified replica:

- a. Press FORWARD until the `├` at the end of the line reappears, or, if the `├` has been overwritten with other items, complete the line and press STORE.

or

- b. With the end of the line in view, press STORE.

In either case, it does not matter if the original line and the modified replica are the same length.

RUNNING PROGRAMS

To run a program that begins at line 0, press END RUN PROGRAM.

To run a program that begins at some other line number, first press END EXECUTE, and then use GO TO (n) RUN PROGRAM.

In each case, END performs the following initialization:

- a. Clears all flags.
- b. Prepares for high speed branching.
- c. Removes all nesting information for subroutines.

Starting a program without first executing an END statement is not recommended.

RUN PROGRAM first executes any existing line in the display prior to beginning the program. This is why it is not necessary to press END EXECUTE RUN PROGRAM to start a program that begins at line 0.

To halt a program, press STOP once.

TRACE MODE OPERATION

While the Calculator is in the Trace mode, it prints a record of its activities. Lines that are executed during the Trace mode are printed along with the answer. Lines that are stored during the Trace mode are printed along with their line numbers.

To establish the Trace mode, execute (or program) the line:

```
[....;]TRC 1
```

↖ end of line

Other statements may or may not be present.

To cancel Trace mode operation, execute (or program) the line:

```
[....;]NOR 1
```

ERRORS

If you press an illogical sequence of keys while writing a line (a *syntax error*), or, an error occurs while the Calculator is executing the line or running a program (an *error during execution*), a *NOTE* will appear in the display. The number of the NOTE identifies the nature of the problem.



An error during execution does not immediately interrupt the execution of the line. The offending operation is either simply not performed, or an alternate operation is performed in its place. The NOTE will appear in the display when the line is eventually terminated.

Only one NOTE can appear in the display at one time. If a statement contains two or more distinct errors, it is difficult to predict which one will be indicated by the NOTE.

If a line has two or more statements, each of which contain errors, the NOTE that appears in the display will be the one for the last statement which had an error.

Diagnostic Notes of the Model 20 Mainframe

INDICATION	MEANING
NOTE 01	(NOTE 01 results from general syntax errors, however, certain specific syntax errors have their own notes.) a. Missing ; between statements (usually results in one of the other conditions causing NOTE 01). b. Attempt to begin an expression with improper character, e.g., #, /,), →, =, ≠, etc. c. Two instructions side by side. d. Missing operand (resulting in two side by side operators), or, an argument of a function is missing.

- f. Numeric constant contains a comma, extra decimal point, or quote mark, etc.
- g. Exponent contains an improper character, e.g., decimal point, parenthesis, variable, * or /, etc.
- h. Exponent contains no digit.
- i. Function immediately followed by any operator except +.
- j. Empty parentheses, i.e., [].
- k. R-
- l. Variable on the right of * is missing or not completely specified.
- m. * immediately followed by -, *, /, %, etc.
- n. Missing quote mark.
- o. GTO or GSB followed by anything except a literal or a signed or unsigned series of digits. Decimal points, parenthesis, variables names, etc., are not allowed.
- p. Key is not permitted during the Enter Mode.

(continued)

Diagnostic Notes of the Model 20 Mainframe (cont'd)

INDICATION	MEANING
NOTE 02	a. Instruction followed by a parameter of the wrong type or illogical value. b. Instruction is missing a necessary parameter. c. Taking the square root of a (value) less than zero.
NOTE 03	Extra (or missing).
NOTE 04	Extra) or missing (.
NOTE 05	a. R followed by a (value) whose integer part is either greater than the number of available R registers, or is negative. b. Attempt to designate a flag other than one of flags 0 through 15.
NOTE 06	a. Attempt to store a number into a non-existent or non-available R register. b. Attempt to enter a number whose exponent has an absolute value greater than 99.

NOTE 07	A RET has not been preceded by a matching GSB.
NOTE 08	A GTO or GSB is followed by a parameter that specifies a line number that is either: <ul style="list-style-type: none"> i. Negative ii. Greater than one higher than the highest existing line number. iii. Not matched by a label in the program.
NOTE 09	<ul style="list-style-type: none"> a. Line is too long to be executed or stored. Do not attempt to salvage the line. Press CLEAR and write a shortened version. b. Subroutines nested too deeply.
NOTE 10	A computation has resulted in an intermediate or final result that has exceeded the range of the Calculator. NOTE 10 will occur only if flag 14 is not set.
NOTE 11	<ul style="list-style-type: none"> a. Pressing one of the half-keys while an associated ROM is not installed, unless the key is being used as part of a literal. b. Attempt to execute an ENTER statement from the keyboard. c. Attempt to nest ENTER statements.

(continued)

Diagnostic Notes of the Model 20 Mainframe (cont'd)

INDICATION	MEANING
NOTE 12	<p>a. Attempt to store a line when there is insufficient memory to accommodate it.</p> <ul style="list-style-type: none">i. Current program is simply too long.ii. An old (and probably long) program is still in memory, and while the new program is being manually keyed in, the remnant of the old one is still being carried — using up memory. To salvage the situation, press CLEAR, END, STORE, DELETE, and continue storing the program. <p>b. Information on magnetic cards requires more memory than is available.</p> <p>c. No GTO or GSB preceding LOD when loading information under the control of a program.</p> <p>d. GTO or GSB preceding LOD has parameter of illogical value.</p>
NOTE 13	Attempt to record a protected side of a magnetic card.
NOTE 14	Magnetic card reader operation is not completed; press EXECUTE and insert the next side.

NOTE 15	Configuration code error — the plug-in ROM's are not arranged in a suitable manner for a program being loaded from a magnetic card.
NOTE 16	Attempted printer operation while the printer is out of paper.

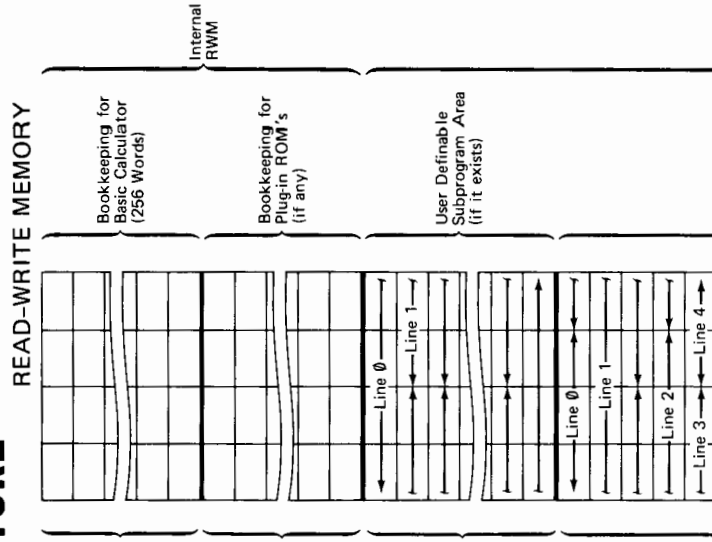


MEMORY STRUCTURE

This is the Calculator's own 'personal' memory. Its size never changes.

This is 'personal' memory for any plug-in ROM's that may be installed. Its size varies according to which ROM's are installed.

This is the area where User Definable Subprograms are located. If there aren't any, no space is used by this area.



PROGRAM STORAGE

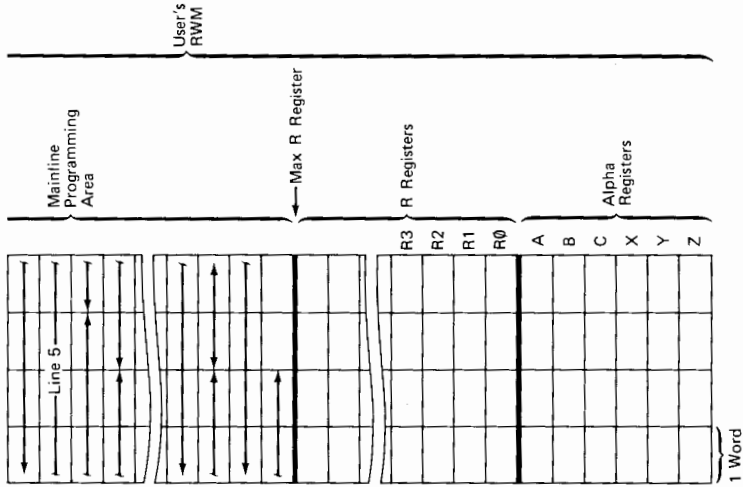
This is the area where ordinary programs are stored. The size of this area is determined by the amount of programming stored in it; the size is adjusted as lines are added, removed, or altered.

This boundary moves up and down, according to the amount of programming.

These are the R registers. The number of these is determined by the size of this area; that is, however much is left over from all the other areas.

These are the six alpha registers that are always available for use.

DATA STORAGE



DATA STORAGE WITH →

The Assignment instructions (→) tells the Calculator to store the numeric value on the left into the register named on the right.

0 → A Stores zero in register A.

10 → R5 → R8 Stores 10 in registers R5 and R8.

A + B → X + Y → R6 Stores the sum of A and B in X, and the sum of A+B+Y in R6.
If stored as part of a program, parentheses will appear as follows:

(A+B→X) + Y→R6

DESIGNATING REGISTERS

Each alpha register is designated by the key having the same name; A, B, C, X, Y, or Z.

R registers are designated with the R() key. The mnemonic is simply R, however; the parentheses are on the key to distinguish it from .

Follow R() with an item whose numeric value is the same as the number of the register you wish to designate: R5 designates register 5. Parentheses must be placed around the item following R if its value is to be computed: R(A+B) designates the register whose number is the sum of the numbers currently stored in A and B.

RR2 designates the register whose number is the number stored in R2. RR...R(....) is also permitted.

Remember:

- a. R register designation starts with R0.

- b. If the numeric value is negative, or too large, a NOTE will result.
- c. The maximum value that should follow R is actually one less than the number printed on program listings; that is the number of available R registers — but their “names” start at R0.
- d. If the value following R is not strictly an integer, the digits to the right of the decimal point are ignored.

HIERARCHY

The mathematical operations are subject to a *hierarchy*. This means that some operations will be performed ahead of others.

- | | |
|--------|--|
| First: | <ul style="list-style-type: none"> Functions (both ‘supplied’ and ‘definable’) Exponentiation (raising a number to a power) Unary Minus (negation) Implied Multiply (AB instead of A*B) Explicit Multiplication, and Division Addition, Subtraction, and Unary Plus Relational Operators (=, ≠, etc.) |
| Last: | |

function exponentiation
 $\underbrace{\text{LOG } 10^2}_2$ $\underbrace{\text{LOG } 10^2}_2$ equals $\text{LOG } 10^2$ \uparrow 2 which is one.
 $\underbrace{\text{LOG } 10^2}_2$ $\underbrace{\text{LOG } 10^2}_2$ equals $\text{LOG } (10^2)$ which is two.
 both are functions

The example illustrates the precedence of functions over exponentiation, and also that operations on the same level of the hierarchy are performed from left to right.

Remember: `SIN -30` and `Γ -A` are syntax errors. Use `SIN (-30)` and `Γ (-A)` instead.

PARENTHESES

Parentheses can be used to *group* quantities together, just as in ordinary mathematical notation:

`A(B+C)`, and `Γ(AA+BB)`

Parentheses can be *nested*, e.g., `2((A+B)/C+(X+Y)/Z)`

Parentheses can be nested as deep as line length will permit.

Parentheses must be used:

- a. To enclose the argument of a function if it is more than a single quantity: `Γ(A+B)`
- b. To enclose the argument of a function if the argument is preceded by a minus sign: `EXP (-A)`

Parentheses will be added to the replica of a stored line of a program if the line contains a multiple assignment statement. The parentheses will be added around the 'interior' assignments:

`A+B+C+X+Y+Z+R10` is stored as `((A+B+C)+X+Y)+Z+R10`

The line: $10(A+B+C)+X+Y$
stores the sum of A and B in C, and stores the additional sum of ten times the previous sum and X into Y.

MATHEMATICAL FAUX PAS

If an attempt is made to enter a number that is too big, or too small, or if an attempt is made to perform a prohibited operation, a NOTE will result.

Under these circumstances the machine will halt the execution of a program, unless *FLAG 14* has been set. If *FLAG 14* is set, the machine will assume some number as the answer and proceed. Either way, *FLAG 15* is automatically set to indicate that there was a non-valid arithmetic occurrence.

The largest absolute magnitude the machine can deal with is:

9.999999999999999 E99

The smallest non-zero absolute magnitude the machine can deal with is:

1.00000000000 E-99



SUMMARY OF PROHIBITED OPERATIONS

PROHIBITED OPERATIONS	RESULT	
	FLG 14 = 0	FLG 14 = 1
(.....)/0	NOTE 10	±9.999999999999999 E99*
$\sqrt{[a<0]}$	NOTE 02	$\sqrt{ a }$
ln 0	NOTE 10	-9.999999999999999 E99
log 0	NOTE 10	-9.999999999999999 E99
ln [a<0]	NOTE 02	ln a **
log [a<0]	NOTE 02	log a **
(-a) ^b b not an integer	NOTE 02	unspecified†
tan 90°, tan 270°, etc.	NOTE 10	+9.999999999999999 E99‡
sin ⁻¹ [x>1]	NOTE 02	unspecified
sin ⁻¹ [x<-1]	NOTE 02	unspecified

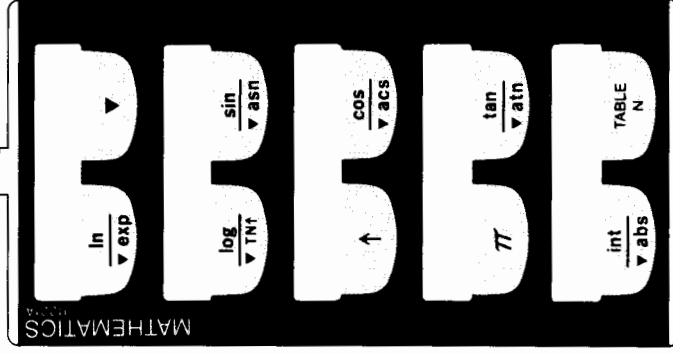
* has the sign of the numerator.

** not guaranteed.








† it cannot be guaranteed, but the result is usually +a.

‡ for the first 5 unit circles, above that it is unspecified.

THE MATH ROM














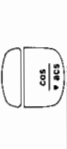
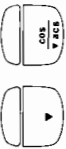
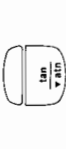
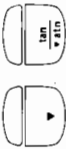
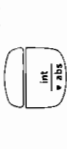




Math ROM Operations

Function Type	To Calculate these functions the syntax is;	to obtain the mnemonic . . . (b represents blank space)	. . . press these keys
Logarithms and Inverse Logarithms	natural log x ln x log _e x (e = 2.7182818284B)	(quantity) or (expression)) LN	LNb	
	e^{x-1} x log _e x antilog _e x	(quantity) or (expression)) EXP	EXPb	
	common log x log ₁₀ x	(quantity) or (expression)) LOG	LOGb	
	10 ^x log ₁₀ x antilog ₁₀ x	(quantity) or (expression)) TNT	TNTb	
Circular and	Select Circular Units	TBL (quantity) quantity = 1 – DEGREES SET 2 – RADIANS SET 3 – GRADS SET	TBLb	
	Sine x sin x	(quantity) or (expression)) SIN	SINb	
	sin ⁻¹ x arc sin x	(quantity) or (expression)) ASN	ASNb	

Inverse Circular Functions	Cosine x cos x	COS	<quantity> or <expression>	COS <i>b</i>	
	cos ⁻¹ x arc cos x	ACS	<quantity> or <expression>	ACS <i>b</i>	
	Tangent x tan x	TAN	<quantity> or <expression>	TAN <i>b</i>	
	tan ⁻¹ x arc tan x	ATN	<quantity> or <expression>	ATN <i>b</i>	
	Integer x int x	INT	<quantity> or <expression>	INT <i>b</i>	
	Absolute value of x x	ABS	<quantity> or <expression>	ABS <i>b</i>	
Miscellaneous Functions	Exponentiation "raised to the power of"	<quantity> or <expression>	<quantity> ↑ or <expression>	↑	
	pi π 3.14159265360		<constant>	π	
Initialize Program	Clear Data Storage and Flags	TBL	<quantity> quantity = 4 – clear all available R registers 5 – clear all alphabetic registers 6 – clear all flags	TBL <i>b</i>	

Math ROM Operations (Cont'd)

... press these keys	Typical Statements	Range of argument (x)	Remarks
	LN 6.25 LN A; LN (X-1); -5LN (4(X-1));	X > 0	
 	EXP .4; EXP B; EXP (-1); 3((EXP A-EXP (-A))/2)+X; Similar to LN	-225.65 < X < 227.95 X > 0	
 	Similar to EXP	X < 10 ⁹⁹	
 	TBL 1; TBL 2; TBL 3; TBL 0; (displays current units)		Once set, units remain set until deliberately changed. At turn-on 'DEGREES SET' is automatically assumed.
 	SIN 30; SIN B; SIN (-45); A+2((1-COS B) ² +(SIN B) ²)+C;	X < 1 x 10 ¹¹	
 	ASN .707; ASN (-.3);	X < 1	Calculates principle value only: $\theta = \sin^{-1} x$; $-90^\circ \leq \theta \leq +90^\circ$

	Similar to SIN	$ X \leq 1 \times 10^{11}$	
	Similar to ASN	$ X \leq 1$	Calculates principle value only: $\theta = \cos^{-1} x; 0^\circ \leq \theta \leq +180^\circ$
	Similar to SIN	$ X \leq 1 \times 10^{11}$	
	Similar to ASN	$ X < 10^9$	Calculates principle value only: $\theta = \tan^{-1} x; -90^\circ \leq \theta \leq +90^\circ$
	INT 4.6; INT A; INT (-7.2); INT (A/B+1.03);		Eliminates fractional part of value; does not affect sign or integer value.
	ABS (-4); ABS A; ABS (A/B+1.03)		Sets value positive without otherwise changing the value.
	5↑2; 5↑(-2); (A+B)↑(X/Y-3);	For A↑B if A < 0, then B must be an integer	-2 ↑ 4 is equal to -16; (-2) ↑ 4 is equal to +16.
	2πA; 180B/π+C;		
	TBL 4; clears all available R-registers. TBL 5; clears registers A, B, C, X, Y and Z. TBL 6; Clears flags 0 through 15		See 'Select Circular Units' for TBL 0, 1, 2, 3. TBL 7, 8, 9 not used.

USING THE PRINTER

To cause something to be printed, use the PRINT key:

PRINT <thing> [; <thing> ;]

The things in the brackets are optional.

'<thing>' can be:

- a. A number
- b. A register name
- c. An arithmetic combination of numbers and registers
- d. An item in quotes

Assume the following line has been executed:

```
10+A; 15+B; 20+C
```

Then the following print statements would result in the printouts shown.

```
FXD 2;PRT A
```

```
10.00
```

```
FLT 4;PRT A,B,C
```

```
1.0000E 01  
1.5000E 01  
2.0000E 01
```


FXD 0:PRT "A+B=";A+B

A+B=

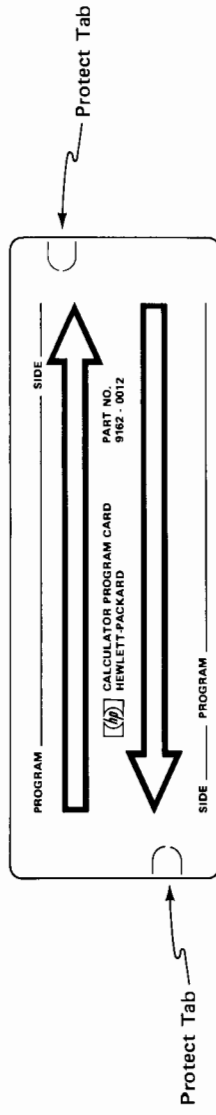
25

The PAPER button is a mechanical method to advance the printer paper, and cannot be programmed. Do not press PAPER while the printer is printing — it will cause NOTE 16.

To advance the paper under control of the Calculator, use the SPACE N key:

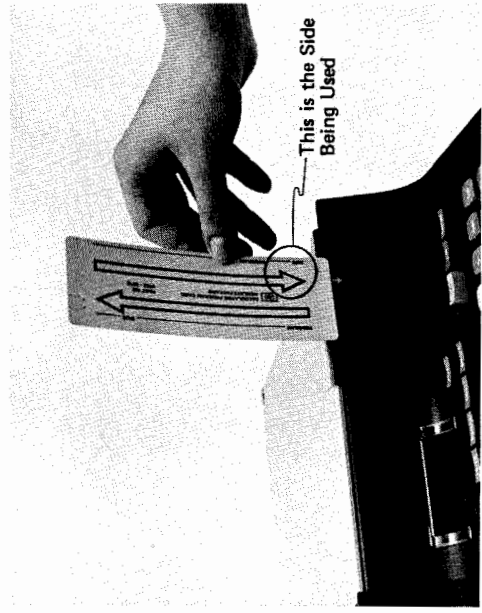
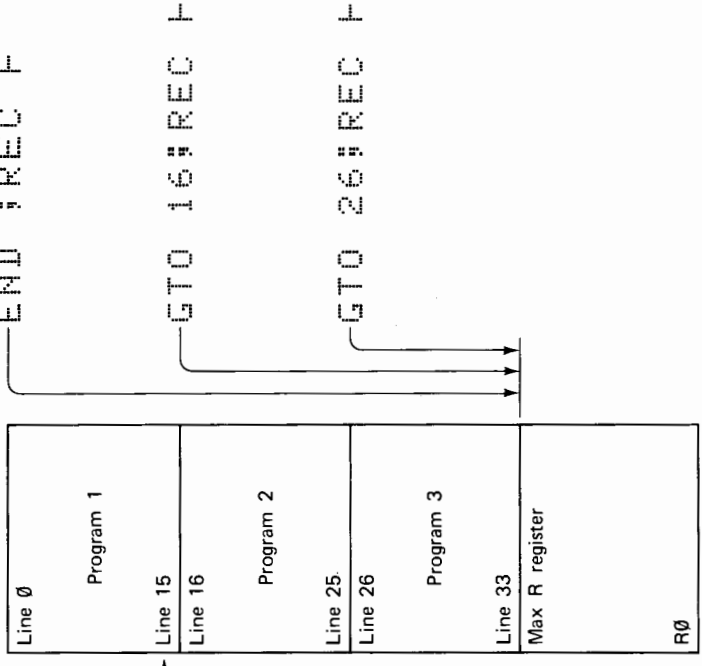
SPACE (n) $0 \leq n \leq 15$; n is the number of lines the paper is to advance.

USING THE MAGNETIC CARD READER



A 6-inch Magnetic Card

END statements in any of these programs have absolutely no effect on the recording process.



Inserting Magnetic Cards

Recording Programs

1 - RECORDING MAINLINE PROGRAMS:

Use GO TO (n) EXECUTE to set the program line counter to the line number that the recording is to begin with. Then press RECORD EXECUTE. Then insert a side of a magnetic card. If NOTE 14 appears in the display, at least one more side is needed to complete the recording process. Press EXECUTE and insert another side.

Once a side has been recorded, it can be protected from being re-recorded by snapping out the protect tab for that side.



2 - RECORDING "PRIVATE" PROGRAMS:

Use the same procedure as for case 1, but with this exception: Instead of pressing RECORD EXECUTE, press RECORD "SE" EXECUTE.

This will make the recording on the card secure, so that if the card is used to load the program, it cannot be inspected, modified, or listed. Data cannot be made secure.

3 - RECORDING DATA:

To record the contents of R0 through some specified register, R(n), execute the line:

```
REC "DA",R(n)
```

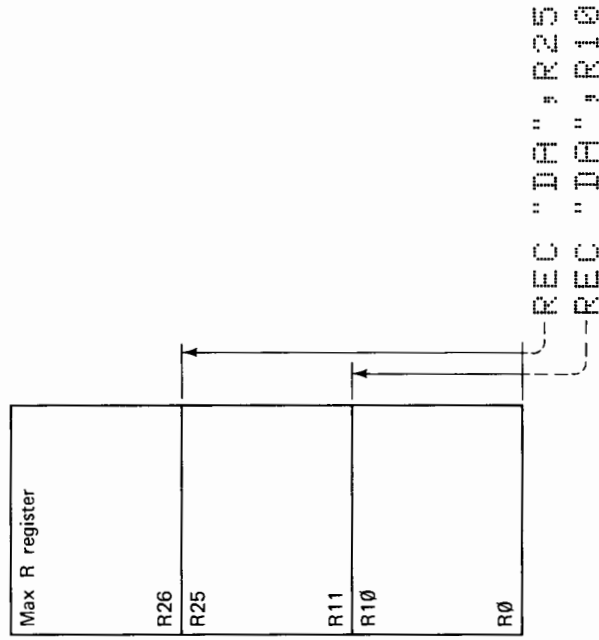
4 - LOADING PROGRAMS (UNDER KEYBOARD CONTROL):

Use GO TO (n) EXECUTE to set the program line counter to the first line number that the program to be loaded will have. Then press LOAD EXECUTE, and insert the first side.

If NOTE 14 appears, more sides are needed to finish loading the program. Press EXECUTE and insert the next side.

If NOTE 12 appears, there is insufficient memory to hold the information recorded on the card.

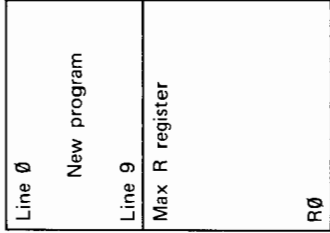
If NOTE 15 appears, it means that the configuration of the plug-in ROM's is somehow different now than when the program was recorded. NOTE 15 is not fatal, but you proceed at your own risk.



Recording Data

GTO 0;LOD F

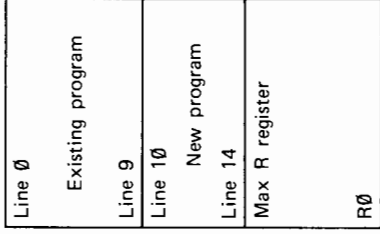
10 line program



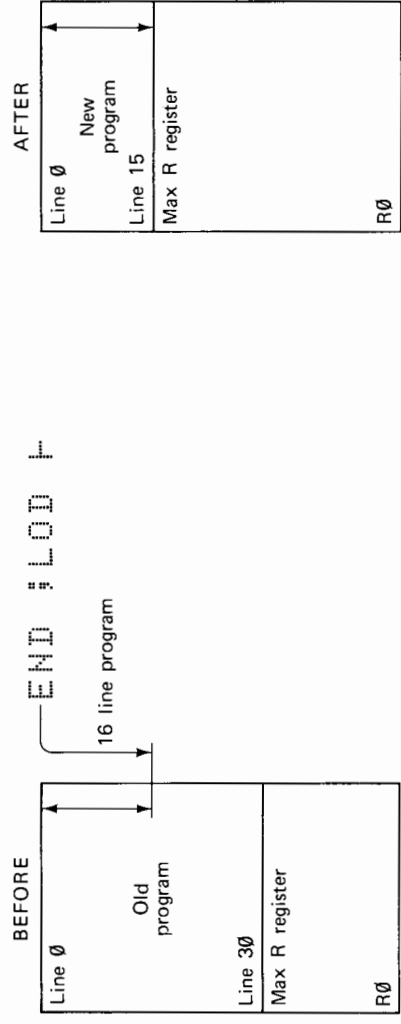
Loading a Single Program

GTO 10;LOD F

5 line program



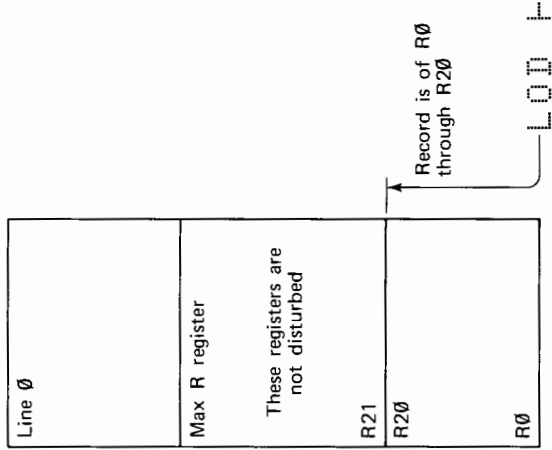
Stacking Programs



Loading a New Program over an Old Program

5 - LOADING DATA (UNDER KEYBOARD CONTROL):

To load data, simply press LOAD EXECUTE and insert the first side. The data is automatically loaded back into the same registers as from which it was recorded.



Loading Data

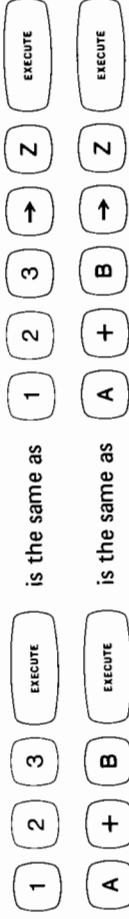
6 - LOADING DATA AND PROGRAMS (UNDER PROGRAM CONTROL):

These cases are similar to the cases where the control is from the keyboard, but there are some ramifications. See the Operating and Programming Manual.

A six-inch magnetic card will record approximately 45 registers of data per side, or about 360 keystrokes of programming per side. A ten and one half inch card will record approximately twice these amounts.

IMPLIED STORE INTO Z

If someone simply keys a number or mathematical expression into the display, and then presses EXECUTE, without having included an assignment of that number or expression into a register, it will automatically be assigned (stored into) Z.



Also, if a line contains such a statement. The $\rightarrow Z$ will be added if the line is stored as a line in a program.

Because of the implied store into Z, the Z register is not recommended for storing data during calculations performed from the keyboard, except in certain situations. For instance, suppose you wished to add a series of numbers: n_1, n_2, n_3, \dots

First, set Z to zero by executing the line ' $\rightarrow 0$ '. Then, add the numbers by executing the following lines:

- $\langle n_1 \rangle + Z$
- $\langle n_2 \rangle + Z$
- $\langle n_3 \rangle + Z$

Because of the implied store into Z, this is what is actually happening:

$$\begin{array}{ll} 0 \rightarrow Z & \\ n_1 + Z \rightarrow Z & n_1 + 0 \rightarrow Z \\ n_2 + Z \rightarrow Z & n_2 + n_1 \rightarrow Z \\ n_3 + Z \rightarrow Z & n_3 + (n_1 + n_2) \rightarrow Z \\ \vdots & \vdots \\ \vdots & \vdots \end{array}$$

END STATEMENTS

END statements are used in establishing certain initial conditions prior to running a program, and also as a means to terminate a program.

A number of things happen when an END statement is executed or stored. They are:

- a. Executing an END statement sets the program line counter to zero.
- b. If an END statement is encountered during the execution of a program, the program is halted, and any nesting information is lost.
- c. All flags are cleared when an END statement is executed.
- d. When a line containing an END statement is stored, or, such a stored line is modified through the use of the editing ability, the existing subsequent lines in that programming area are deleted.

Because storing or modifying a line containing an END statement deletes the lines of programming having higher line numbers, END statements are seldom used in the interior of a program. They are usually used only in the highest numbered line to be stored.



It is recommended that an END statement be executed before starting a program, because this accomplishes needed initialization, such as guaranteeing that no flags are set except for the ones deliberately set by the user. It also ensures that the program will be run with high speed branching.

ENTER STATEMENTS

Enter statements are a means to suspend the execution of a program for the purpose of assigning values to a list of registers. When the ENTER statement is encountered, the Calculator will 'ask' for the value to be assigned to the first register in the list. The value is supplied to the Calculator by keying in a constant, or an expression, representing the value. Then RUN PROGRAM is pressed to cause the assignment to take place. If there are more registers in the list, the Calculator will ask for the next value, otherwise the execution of the program is resumed.

```
0: FXD 2F
1: ENT A,B,C
2: PRT "A=";A;"B=";
3: "C=";C;SPC 8F
4: STO 1F
END F
```

NOTE

ENTER statements cannot be executed from the keyboard. They can be used only in the context of a program.

END RUN PROGRAM

A

1 0 RUN PROGRAM

B

2 0 RUN PROGRAM

C

1 0 + 2 0

RUN PROGRAM



```
A= 10.00
B= 20.00
C= 30.00
```

A

:

When an ENTER statement is encountered, the display will show (on the left) the name of the first or next register in the list, or, the text of the literal that is immediately to the left of that register. Only the left-most 16 characters of a long literal can be displayed.

```
0: FXD 2+
1: ENT "WHICH R REG
   ISTER";R1;"WHAT
   VALUE ?";RR1F
```

```
2: PRT RRI, "IS STOR  
ED IN REG", R1;  
SFC 8F  
3: GTO 0F  
4: END F
```

FLAGS

There are 16 flags, named flag 0 through flag 15.

Flags are designated by the mnemonic `FLG` followed by a value which determines the particular flag intended:

`FLG 15` `FLG A`

There are instructions to set and clear the flags:

`SFG 14` sets flag 14
`CFG 6` clears flag 6

The mnemonic `CFG` is obtained by pressing the `SET/CLEAR FLAG N` key twice in succession.

Three of the flags are associated with definite meanings. If flag 14 is set, certain arithmetic faux pas, such as division by zero, will be ignored. Flag 15 is automatically set whenever such faux pas occur. Flag 13 is used in conjunction with the ENTER statement.

Flag 13 is a useful indicator that the last datum has been entered. Consider the following program for summing a series of numbers:

```
0: FXD 0:0+BT
1: ENT "NEXT NUMBER
   ":AF
2: IF FLG 13=0:PRT
   A:A+B:B:GTO 1F
3: PRT "TOTAL":B:
   SPC 8:END F
```

Line 2 is a conditional branch based on flag 13. Flag 13 is normally zero, allowing the new value of A to be printed and added to the previous total. After the last number has been entered, however, RUN PROGRAM is pressed without making an entry. This sets flag 13, which allows the IF statement of line 2 to branch to line 3. Then the result is printed and the program terminated.

BRANCH WITH IF STATEMENTS

The general form of the IF statement is:

$$\text{IF } \left[\begin{array}{c} \text{thing} \\ \text{thing} \\ \text{thing} \\ \text{thing} \\ \text{thing} \end{array} \right] \left\{ \begin{array}{c} \text{one of} \\ \text{thing} \\ \text{thing} \\ \text{thing} \\ \text{thing} \end{array} \right\} \text{ thing ;}$$

where the 'thing' parts of the statement can be values, register names, arithmetic expressions or flags. Typical 'if' statements might be:

```
IF R=3; ..... F
IF R14 > (4R+2)/6; ..... F
```

When the calculator makes the check, it substitutes the appropriate value for any register name or for any arithmetic expression, just as it does in other types of statement. If the condition is true (YES), the rest of the current line is executed; if false (NO), the rest of the line is ignored and the next line is executed.

The most common use of the 'if' is to make conditional branches (that is, to conditionally execute a 'go to' statement). Consider this line, taken from an imaginary program:

```
10: IF 4 > Y; Y+1 → Y; GTO 7 F
```


As long as Y is less than 4, the condition is true; the program then adds 1 to Y and branches back to line 7. The program will continue to 'loop', executing lines 7, 8, 9 and 10 until the test is made when Y has a value of 4; now the condition is no longer true so the program goes on to line 11. In this example, line 10 also acts as a counter and counts the number of times the branch (GTO 7) will be made; if Y is initially equal to zero, then the branch will be made four times (so that lines 7, 8 and 9 will be executed five times).

SYNTAXES

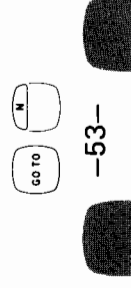
In this list of syntaxes, the symbols (. . .) and [. . .] are conventions used in illustrating properties of a syntax (their meanings are given below); they are not part of the syntax, per se, as there are no keys that correspond to them. They are shown simply as an aid in understanding what goes where in the syntax.

The symbols (. . .) denote that whatever is inside them is merely the name of general kind of thing that is to go there, and not the actual thing itself.

For example,

GTO (N)

means GO TO followed by some integer number, and not:



The symbols [...] denote that their contents are optional; that it is syntactically permissible if that element is absent.

Thus, $FXD\langle n \rangle$ or $FXD ; \dots$ is permissible.

For brevity and the sake of simplicity, there are many differences between the syntaxes listed here and in the Operating and Programming Manual. There differences are generally minor, and the presentation here adds to the ease of understanding. If any questions do arise, however, the Operating and Programming Manual gives the most technically complete description of each syntax.

$FXD\langle n \rangle$ Where $0 \leq n \leq 9$ is the number of desired places to the right of the decimal point.
 $FLT\langle n \rangle$

If $\langle n \rangle$ is not strictly an integer, its digits to the right of its decimal point are ignored.

NOTE 02 will result if the integer part of $\langle n \rangle$ is less than zero, or greater than nine.

The instructions FXD and FLT, when used without a parameter, specify their respective forms; the previous parameter for that form is automatically assumed.

$DSP\langle thing_1 \rangle ; \langle thing_2 \rangle [; \dots]$

Parameters other than literals are displayed: on the right side of the display; as their respective numeric values; and according to the current fixed or floating point specification.

Literals are displayed as the text of the literal, and on the left side of the display. If the literal is longer than 16 characters, only the left-most 16 will be displayed.

PRT <thing₁>[; <thing₂>[;]]

Parameters other than literals are printed: on the right side of the paper; as their respective numerical values; and according to the current fixed or floating point specification.

Literals are printed as the text of the literal, and on the left side of the paper. If the literal is longer than 16 characters, only the left-most 16 will be printed.

SPC <n>

Where $0 \leq n \leq 15$ is the number of rows the printer paper is to be advanced.

If <n> is not strictly an integer, the digits to the right of the decimal point are ignored.

NOTE 02 will result if the integer part of <n> is less than 0, or greater than 15.

SPC by itself is equivalent to SPC 1, but is a keystroke shorter.

ENT <register name>[; <register name>[;]]

or

ENT "<literal>" ; <register name>[; "<literal>" ; <register name> [;]]

When an ENTER statement is encountered, the display will show (on the left) the name of the first or next register in the list, or, the text of the literal that is immediately to the left of that register. Only the left-most 16 characters of a long literal can be displayed.

The value is supplied to the Calculator by keying in a constant, or an expression, representing the value. Then RUN PROGRAM is pressed to cause the assignment to take place. If there are more variables in the list, the Calculator will ask for the next value, otherwise the execution of the program is resumed.

NOTE

ENTER statements cannot be executed from the keyboard. They can be used only in the context of a program.

During the execution of an ENTER statement, but before it is satisfied, the Calculator is in the *Enter Mode*. During this mode, keys that would alter the program line counter are treated as syntax errors, or are ignored.

To abort an ENTER statement, and remove the Calculator from the Enter Mode, use the following procedure:

1. If necessary, press CLEAR to rid the display of any NOTE's.
2. Press STOP until STP appears in the display.
3. Then press CLEAR.

ENTER statements cannot be nested; that is you may not do anything that requires another ENTER statement when the Calculator is already in the Enter Mode.

While in the Enter mode, no value is assigned to a register if:

- a. RUN PROGRAM is pressed prior to specifying a value, or,
- b. A value is specified, but EXECUTE or CLEAR is pressed, followed by RUN PROGRAM.

In each case, the register in question retains its original value, and Flag 13 is set.

GTO <n>

Where n is a line number.

This statement is used to branch from a line to the beginning of the line whose line number is n.

NOTE 01 will result immediately from an attempt to make n a register, or, if n has a decimal point.

GTO + <n>
GTO - <n>

These statements are used to branch from a line, whose line number is, say, m, to the beginning of the line whose line number is m+n, or m-n, depending upon the sign preceding <n>.

GTO +0 and GTO -0 each cause a branch back to the beginning of the line which contains them.

GTO "<literal>"

This statement is used to branch from a line to the beginning of the line having a label matching <literal>.

The branch occurs in the form of a search beginning at line 0. The first line encountered having a label matching <literal> is taken as the destination of the branch. If either is longer than four characters, only the right-most four characters are considered in the search.

—58—

```
GSSB <n>
GSSB + <n>
GSSB - <n>
GSSB " <literal> "
```

GO TO SUB statements are used to branch to local subroutines. The line at which the subroutine starts is identified by any of the methods used with GO TO statements.

Insofar as each of these statements causes a branch from one line to another, each behaves in the same way as its corresponding GO TO statement.

However, one additional thing happens when a branch is made with GSSB from, say, line number n . The Calculator automatically assumes that, when the subroutine is completed, the execution of the main program is to resume at the beginning of the line numbered $n+1$. There is no way to alter this automatic assumption.

```
<line number>[...;]RET | ← 2 — end of line
```

RETURN statements are used to terminate the subroutines and branch back to the previous program. The destination of the branch will always be the line following the one which branched to the subroutine.

RET must never occur except as the last statement in a line.

(line number:)[...;]JMP (value)←→end of line

This statement is used to branch from a line, whose line number is, say, m , to the beginning of the line whose line number is $m+n$, or $m-n$, depending upon whether (value) is positive or negative, respectively.

If (value) does not represent strictly an integer, the digits to the right of the decimal point are ignored.

JMP (zero) causes the line containing it to be repeated.

NOTE

JUMP statement cannot be executed from the keyboard.

SFC (n)

Where $0 \leq n \leq 15$ denotes the desired flag.

This statement is used to set a flag (make its numeric value be 1).

If (n) is not strictly an integer, the digits to the right of the decimal point are ignored.

If, while the Calculator is running a program, SET/CLEAR FLAG N is pressed, the machine will detect this and automatically assume that flag 0 is to be set. Flag 0 will be set without interrupting the program.

--60--

CFG $\langle n \rangle$

Where $0 \leq n \leq 15$ denotes the desired flag.

This statement is used to clear a flag (make its numeric value be 0).

If $\langle n \rangle$ is not strictly an integer, the digits to the right of the decimal point are ignored.

The mnemonic CFG is produced by pressing the SET/CLEAR FLAG N key twice in succession. Once the mnemonic CFG is present in the display, it cannot be separated into its original components of 'SFG SFG' through the use of BACK or DELETE; CFG is treated as is any other mnemonic.

FLG $\langle n \rangle$

Where $0 \leq n \leq 15$ denotes the desired flag.

This syntax is used to identify a particular flag in situations other than setting or clearing that flag.

If $\langle n \rangle$ is not strictly an integer, the digits to the right of the decimal point are ignored.

$\langle n \rangle$ [. . .] IF $\langle \text{condition} \rangle$ $\{ \langle \text{statement} \rangle$ [. . .]
 $\langle n+1 \rangle$ $\langle \text{line} \rangle$

If the condition following IF is true, then the remainder of the line is executed. Should the condition be false, however, the execution of the line is terminated at that point, and the *next* line is executed.

(r:) [...]STOP ← end of line

STOP statements are used to interrupt or terminate the execution of a program.

(r:) [...]END ← end of line

END statements are used in establishing certain initial conditions prior to running a program, and also as a means to terminate a program.

- a. Executing an END statement sets the program line counter to zero.
- b. If an END statement is encountered during the execution of a program, the program is halted, and any nesting information is lost.
- c. All flags are cleared when an END statement is executed.
- d. When a line containing an END statement is stored, or, such a stored line is modified through the use of the editing ability, the existing subsequent lines in that programming area are deleted.

[...]NOR ←

This statement establishes the Trace Mode.

[...]TRC ←

This statement cancels the Trace Mode.

SYNTAX REFERENCE FOR PC 1

SCALE $\langle X_{min} \rangle$ * $\langle X_{max} \rangle$ * $\langle Y_{min} \rangle$ * $\langle Y_{max} \rangle$

This assigns problem oriented numerical values to the extremes of the plotting area specified by the graph limit controls. An initial SCALE statement must precede any plotting operations.

PLT $\langle X \text{ coordinate} \rangle$ * $\langle Y \text{ coordinate} \rangle$

This syntax moves the pen to the points specified by the coordinates. If the pen is already up, it stays up until the point is reached, and then it drops. If the pen is down, it stays down as it moves. If the point is outside the plotting area, the pen raises and moves to an extremity of the plotting area. It will not halt the program, and the pen will return to the paper as soon as the coordinates are again within range.

Either $\langle \text{coordinate} \rangle$ can be a constant, register name, or an expression.

PLT $\langle \text{numerical quantity} \rangle$

This syntax causes the plotter to write the numerical value of $\langle \text{numerical quantity} \rangle$. The starting location, size, and direction of the lettering is controlled by the last encountered LETTER statement. The form of the number that is to be written is controlled by the current settings of the FIXED N and FLOAT N keys.

PLT " $\langle \text{message} \rangle$ "

This syntax causes the plotter to draw the characters specified in $\langle \text{message} \rangle$. The starting location, size, and direction are specified with a previously encountered LETTER statement.

ABCDEFGHIJKLMNORSTUVWXYZ
ABCDEFGHIJKLMNORSTUVWXYZ

0123456789. ; + - + ()

Ø | 23456789 . / + - / → < >

PEN

This statement raises the pen from the paper.

LTR (X location) ; (Y location) [; (HWD)]

This syntax raises the pen, and then moves it to the specified location. (HWD) is a three digit constant that specifies the size and direction of any subsequent lettering performed by PLOT statements.

H specifies letter height, and can range from 1 to 9.

W specifies letter width, and can range from 1 to 9.

D specifies orientation of the lettering in the four cardinal directions; it can range from 1 to 4.

MOVE (X location) (Y location) [(X tic) (Y tic)]

This statement causes axes to be drawn which will intersect at the specified location. If (X tic) and (Y tic) are supplied, there will be tic marks every such distance along each axis.



READ (SC) [(register name) [(register name). . . .]]

NOTE

READ statements cannot be executed from the keyboard; they can only be executed in a program. An attempt to execute a READ statement from the keyboard will cause NOTE 11 to appear.

READ statements with PC I do not reference a FORMAT statement. Instead, incoming numbers are separated with delimiters, and the numbers themselves are permitted to assume a wide variety of forms.

Leading spaces in a data item are ignored. Two consecutive commas indicate that no data item is supplied for the corresponding element in the list; the current value of that list element will remain unchanged and Flag 13 will be set. An initial comma causes the first element to be skipped.

A slash causes the calculator to ignore all following characters until a CR LF has been encountered. If a CR LF is encountered (and it does not correspond to a preceding slash) the READ statement is terminated, the values of any further list elements in the READ statement remain unchanged, and Flag 13 is set.

A data item must be composed of only the following characters: the digits 0 through 9; the plus and minus signs; the decimal point; certain spaces; and an 'E' character. All other characters will be treated as data item delimiters. The data item itself can assume the same form as any single constant which is keyed in from the keyboard.

When reading a floating point number, the space which follows 'E' is not interpreted as a data item delimiter, but as a plus sign on the exponent of the number being read.

```
WRITE(S) '(thing)' ; (thing) ; . . . .]
```

Each <thing> can consist of a register name or a string of keys which are enclosed in quotes. Each <thing> is outputted under the 'default format', if it is in effect, or, under the format specified by the previous FORMAT statement.

The space (b) and the CR LF are delimiters that occur during the execution of each WRITE statement. The space (b) is used to separate items within the list, and the CR LF is used to terminate the list.*

The default format is four consecutive fields of 18 characters each. Each <thing> appears right-justified in a field. The form of numerical <thing>'s is determined by the current settings of the FIXED N and FLOAT N keys.

* The machine does not automatically insert spaces into the list; they get there through the conversion specification field width being wider than absolutely necessary.

```
TYPE (thing) '(thing) ; . . . .]
```

This statement is treated in *exactly the same way as*

```
WRITE 15 ; . . . .
```

is treated; 15 is the select code of the Typewriter.

This statement is used to supply data to the Typewriter. A (thing) can be a series of keys in a quote field, but this is restricted to typing only upper case letters in black. Carriage and ribbon control can only be achieved with FORMAT statements.

FORMAT (spec₁) [# (spec₂) # (spec₃) #]

When a FORMAT statement is encountered, its location in memory is noted and the calculator continues with program execution. Then, when an output operation (a WRITE or TYPE statement) is encountered, the calculator references the last encountered FORMAT statement while executing the output statement.

There is a correspondence between the elements of a FORMAT statement and the parameters in the list of the statement that actually specifies what is to be output. In general, the correspondence is that the first specification in the FORMAT statement controls the form (appearance) of the first parameter that is output; then the second specification controls the form of the next parameter; etc. If the end of the list of parameters is reached before all of the specifications have been used, a CR LF is given and the output operation is terminated. However, if the end of the FORMAT statement is reached before the list of the output statement is exhausted, a CR LF is given and execution of the output statement is resumed by repeating each specification of the FORMAT statement.

A *conversion* specification determines whether the number is output in fixed point or floating point, the number of digits to the right of the decimal point, and the field width in which the number appears (it will be right-justified).

r F X D w . d or r F L T w . d

Where: r is the number of consecutive times the specification is to be used (if r is 1 it may be omitted).
w is the field width in which the number is to appear.
d (0-9, inclusive) is the number of digits to appear to the right of the decimal point.

For fixed-point outputs, w should be greater than or equal to $d+3$; for floating-point outputs, w should be greater than or equal to $d+7$.

In general, if a fixed-point specification cannot be met, either because w is not large enough to because the number is simply too large, an attempt is made to output in floating-point (using the same w and d). If the calculator cannot output the number into the field width available, the field is filled with dollar signs.

In floating-point: whenever $w < d+7$ and the number is negative, or
 whenever $w < d+6$ and the number is positive.

In fixed-point: whenever $10 \geq N+d > w-1$
 where N is the number of digits to the left of the decimal point.

The conversion specification 3FXD 10.3 is identical to the series of specifications:

FXD 10.3:FXD 10.3:FXD 10.3

There are four types of *editing* specifications:

 <integer constant>X or rX (spaces)

When this specification is encountered it causes 'r' number of spaces to be output. If r is 1 it may be omitted.

 <integer constant> / or r/(CR/LF's)

When this specification is encountered it causes 'r' number of CR LF's to be output. If r is 1 it may be omitted.

.....Z[.....]

When this specification is encountered the automatic CR LF at the end of the FORMAT statement or at the end of the list of an output statement is suppressed until another FORMAT statement is encountered or until the default format is re-established.

(literal) or "message or ASCII characters"

When this type of editing specification is encountered, the ASCII characters corresponding to (literal) are outputted (the quote marks are not outputted).

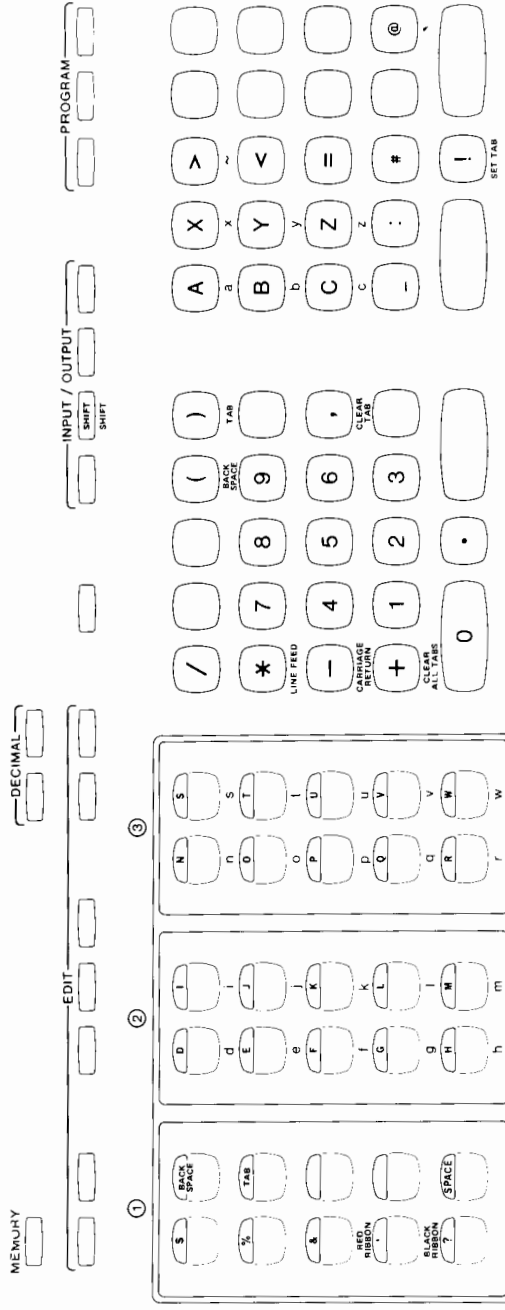
Since the Model 20 can output more ASCII characters than there are keys on the keyboard, the PC I Block provides the calculator with a 'shifted' keyboard. The term 'shifted' does not refer to the shifting of a typewriter or teletype keyboard, and should be considered local to the calculator.

At the start of any literal, the calculator's keyboard is in the unshifted mode; it is placed in the shifted mode by pressing the DISPLAY key. This also places the symbol " " in the display; however, this does not cause a character to be output, as it is merely an instruction to the calculator to shift the keyboard. The calculator's keyboard can be returned to the unshifted mode while in the middle of a literal by placing another " " in the literal (the unshifted mode is automatically set at the end of a literal).

ASCII Output Characters Available with the PC I Block

ASCII Character	Model 20 Key	ASCII Character	Model 20 Key	ASCII Character	Model 20 Key	ASCII Character	Model 20 Key	ASCII Character	Model 20 Key
A		S		9		#		EOT	
B		T		+		!		RU	
C		U		-		@		BELL	
D		V		*		\$		NULL	
E		W		/		%		VT	
F		X		\		&		CR	
G		Y		↑		'		LF	
H		Z		(?		ACK	
I		0)		WRV		ESC	











Typewriter functions such as upper and lower case, red or black ribbon, tab activities, etc., can be controlled by using one or more keys as an editing specification. As each key is encountered when executing the TYPE statement, the specified typewriter operation is performed.



Typewriter Control Keys

T F R (from SC this) (to SC that)

After encountering a TRANSFER statement, the calculator simultaneously receives and transmits the string of characters sent by the transmitting device. TRANSFER statements may be terminated by pressing STOP or by receiving an ASCII 'EOM' (end of message) character. The EOM character is the only delimiter which the calculator responds to during a transfer operation.

TYPEWRITER CONTROL		PLOTTER CONTROL	
KEY	MNEMONIC [☆]	KEY	MNEMONIC [☆]
	TYP <i>b</i>		SCL <i>b</i>
GENERAL PERIPHERAL CONTROL			AXE <i>b</i>
KEY	MNEMONIC [☆]		PEN <i>b</i>
	FMT <i>b</i>		LTR <i>b</i>
	WRT <i>b</i>		PLT <i>b</i>
	RED <i>b</i>		
	TFR <i>b</i>		

[☆] *b* indicates a blank space.

PC I DIAGNOSTIC NOTES

Most of the execution and syntax errors associated with the PC I Block are similar in form to those of the basic calculator and cause the same 'notes' to appear. The PC I Block adds NOTE 20, NOTE 21 and NOTE 22. The most likely errors (not all possible errors) are listed below.

INDICATION	MEANING
NOTE 02	a. SCALE statement does not contain four parameters. b. PLOT statement does not contain one or two parameters. c. AXES statement does not contain two or four parameters. d. LETTER statement does not contain two or three parameters.
NOTE 11	READ statement execution attempted from the keyboard.
NOTE 20	(In general, the select code specified is not in the range: $1 \leq \text{s.c.} \leq 15$.) a. Select code parameter within a READ statement is not a constant (integer number) or a variable (contents of a register). b. Select code parameter within a TRANSFER statement is not a constant (integer number).
NOTE 21	Parameters within an AXES statement will cause intersection of axes outside of the plotting area.

NOTE 22

(In general, the FORMAT statement contains a syntax error.)

- a. No conversion specification for a corresponding output parameter in a WRITE or TYPE statement.
- b. Conversion specification does not contain one of the following parameter types:
 FXD or FLT or "literal"
- c. Conversion specification does not contain field width (w) parameter or decimal point (d) parameter.
- d. A parameter is not followed by one of the following:
 : or ; or †
- e. A WRITE or TYPE statement was executed from the keyboard after a similar statement and a FORMAT statement were executed from the keyboard.

UDF BLOCK DIAGNOSTIC NOTES

Most of the execution and syntax errors associated with the UDF Block are similar in form to those of the basic calculator and cause the same 'notes' to appear. The UDF Block adds only one new note - NOTE 24.

Following is a brief description of the most likely errors (not all possible errors are listed). The symbol \vdash implies either the STORE or EXECUTE instructions.

INDICATION	MEANING
NOTE 01	<ol style="list-style-type: none"><li data-bbox="706 388 738 1102">1. 'CALL' statement not the last statement on the line.<li data-bbox="747 136 779 1102">2. Key other than F★ or IEX follows SCR.<li data-bbox="787 136 852 1102">3. Key other than 'comma', 'semicolon' or \vdash following F★ in a 'SCRATCH' statement.<li data-bbox="860 136 909 1102">4. Any other key in an 'IEX' statement (other than SCR IEX - to cancel the 'Immediate Execute Mode').<li data-bbox="917 136 982 1102">5. CLL not the first key in a statement (also use of CLL during a halt for an 'ENTER' statement).<li data-bbox="990 136 1177 1102">6. Any key following CLL except F★ or a label enclosed in quote marks. (A 'label in quote marks' is the same as the line label in the mainline area; it is not the same as the literal that is the first statement in a subprogram - in a 'CALL' statement, the former has to be keyed character by character while the latter appears as soon as the key F★ is pressed.)

	<p>7. Key other than 'semicolon' or † after 'DEFINE F★'.</p> <p>8. More than one F★ used in a 'GO TO' or 'GO TO SUB' statement.</p>
NOTE 02	<p>1. Executing a 'goes to F' statement when the function was started by means of a 'GO TO' statement.</p> <p>2. Executing a P-number as a value in an expression when the subprogram was called incorrectly (e.g., by a 'GO TO' statement), or the P-number was included in a procedure subprogram.</p>
NOTE 05	Attempt to use a P-number whose parameter is zero or less than zero [P0, P(-1), etc.].
NOTE 06	Attempt to store into a 'local register' (Pn) when the subprogram was called incorrectly (e.g., by a 'GO TO' statement) or when the P-number is used in a procedure subprogram.
NOTE 07	Executing a RET or an END (only in the definable area) without a matching GSB, CLL, or function subprogram call.
NOTE 09	Too many CALL statements in the line.
NOTE 11	Attempt to nest ENTER statements.
NOTE 24	Attempting to execute a 'SCRATCH F★' during program execution.

TAPE CASSETTE DIAGNOSTIC NOTES









Most of the execution and syntax errors associated with the cassette control block are similar in form to those of the basic calculator and cause the same 'notes' to appear. The cassette control block adds NOTE 20 and NOTES 25 through 31. The most likely errors (not all possible errors) are listed below.

INDICATION	MEANING
NOTE 02	Parameter Error a. Missing parameter in a RECORD FILE, LOAD FILE, FIND FILE, or MARK TAPE statement. b. Parameter ₁ of a RECORD FILE statement is greater than parameter ₂ (except when clearing a file). c. The parameter is missing in a SET SELECT CODE statement.
NOTE 12	The available calculator memory is too small to store the specified file contents.
NOTE 20	The select code specified is not within the range of from 0 to 15.
NOTE 25	The specified Special Program is not stored in memory.
NOTE 26	There are no Special Programs stored in memory.



NOTE 27	The cassette memory door is open or ajar.
NOTE 28	The tape cassette is protected (i.e., the protect tabs are removed).
NOTE 29	The specified file is too small to contain the data or program lines.
NOTE 30	The cassette memory detected an error when loading data or program lines from the specified file.
NOTE 31	The file is empty or contains information that cannot be loaded into the Model 20 (e.g., a Model 10 program).

TAPE CASSETTE BLOCK KEYS

CASSETTE CONTROL

KEY	MNEMONIC*	KEY	MNEMONIC*
	<i>b</i>		<i>b</i>
	<i>b</i>		<i>b</i>
	<i>b</i>		
	<i>b</i>		

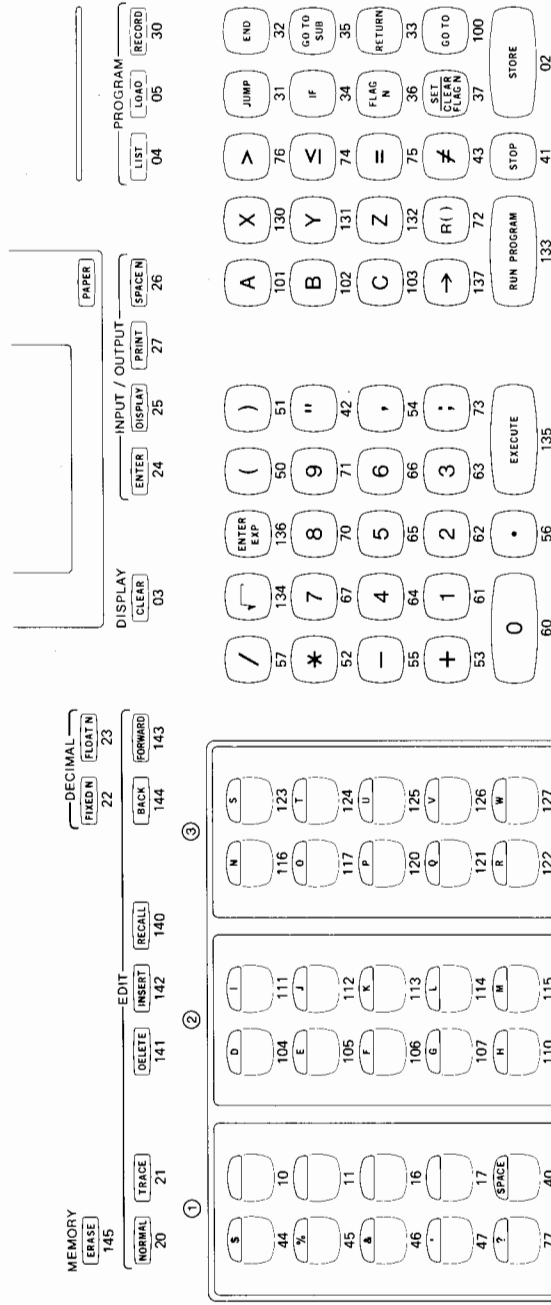
SPECIAL PROGRAMS**

KEY	MNEMONIC*	KEY	MNEMONIC*
	<i>b</i>		<i>b</i>

* *b* indicates a blank space.

**These keys are used with the Special Programs supplied by -hp-. Instructions on use of these keys accompany each Special Program.

MODEL 20 KEY CODES



NOTES

1. The numbers under the keys are keycodes.
2. PAPER is mechanical, and does not have a keycode.
3. The key codes for the half-keys are always the same — regardless of which ROM's are installed.



PART NO. 09820-90002
MICROFICHE NO. 09820-99002

PRINTED IN U.S.A.
NOV 1972