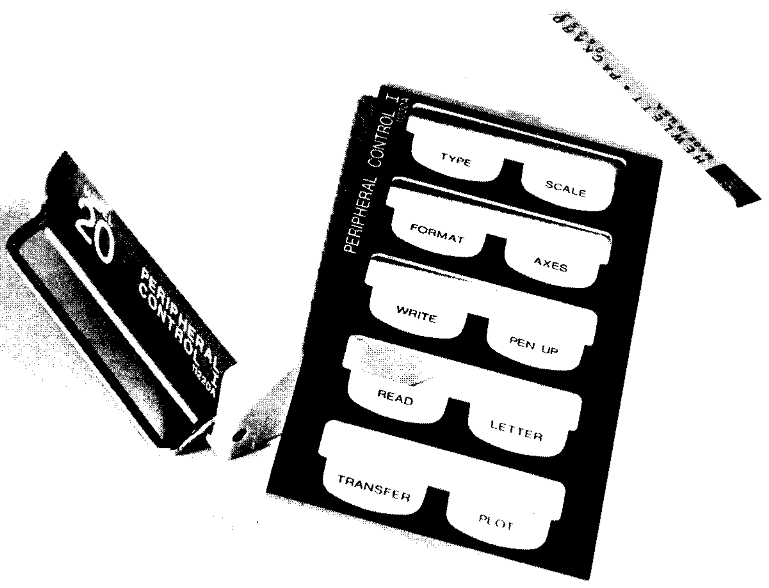
MODEL 20

HEWLETT-PACKARD 9820A CALCULATOR 11220A PERIPHERAL CONTROL I

OPERATING MANUAL



11220A PERIPHERAL CONTROL I BLOCK

HEWLETT–PACKARD CALCULATOR PRODUCTS DIVISION
P.O. Box 301, Loveland, Colorado 80537, Tel. (303) 667-5000
Rue du Bois-du-Lan 7, CH-1217 Meyrin 2, Geneva, Tel. (022) 41 54 00

TABLE OF CONTENTS

The contents of each successive chapter are listed at the beginning of that chapter.

This manual is intended to contain all operating information for the calculator peripheral devices which are controlled with the Peripheral Control I Block. The chapter index to the right lists the devices that are now described in this manual; you may use the index to quickly find operating information on each peripheral device.

As you add other suitable peripherals to your Model 20 System, each will be supplied with operating information that should be added to this manual.

The instructions in this manual assume that the reader is familiar with operation of the basic Model 20 Calculator, as described in the Model 20 Operating and Programming Manual.

NOTES

Chapter 1

GENERAL INFORMATION

INTRODUCTION TO PC I

The -hp- Model ll220A Peripheral Control I Block (the PC I Block) consists of a read-only-memory (ROM Block) and a keyboard overlay. The PC I Block enables the Model 9820A Calculator to control and send or receive data from many -hp-9800 Series calculator peripherals. Also, the PC I Block enables the Model 20 to control many other devices when they are properly interfaced to the calculator. Chapter 2 contains general interfacing and operating information which should help you determine if your specific device can be operated in a Model 20 System.

SUPPLIED EQUIPMENT

The items supplied with the PC I Block are listed below.

Table 1-1. Equipment Supplied

DESCRIPTION	QUANTITY	-hp- PART NUMBER
Key Overlay	1	7120–1687
Operating Manual	2	09820–90027
Supplements to the Model 20 Electrical Inspection Booklet:		
Supplement B	1	09820–90052
Supplement C	1	09820–90054
Supplement D	1	09820–90056
Supplement E	1	09820–90058

INITIAL INSPECTION

The PC I Block and the equipment listed in Table 1-1 were carefully inspected before they were shipped to you. Please verify that all the equipment listed is present, and inspect the ROM block for physical damage.

To check operation of the PC I Block, see the Model 20 Electrical Inspection Booklet, which is supplied with your calculator.

If any damage or electrical malfunction is found, contact the nearest -hp- Sales and Service Office; office locations are listed at the back of this manual.

INSTALLATION PROCEDURE

As with other ROM Blocks, the PC I Block may be installed in any of the ROM Slots on top of the Model 20. Also, the PC I Block defines the keyblock directly in front of where it is installed.

Even though a ROM Block can be installed in any of the slots, programs recorded on magnetic cards or tape cassettes dictate that any required block be in a specific slot – namely, the slot that the block was in when the program was recorded. Before loading any recorded program, always check the instructions which accompany the program to determine which ROM's should be installed in which slots.

It is recommended that the PC I Block be installed in ROM Slot #3, since most programs published by -hp- will specify that the PC I Block be in that slot.

To install the PC I Block:

1. Switch the calculator OFF – if you leave the calculator ON, the installed block will not be 'accepted' until MEMORY ERASE is pressed.
2. Position the block over the desired slot (ROM Slot #3 is recommended), such that the 'PERIPHERAL CONTROL' label is readable from the front of the calculator (see Figure 1-1). Push the block straight down until it is firmly seated.
3. Install the key overlay by inserting the tab at the tip of the overlay into the locking slot at the top of the keyblock; then press the overlay down over the keys.
4. Switch the calculator ON.

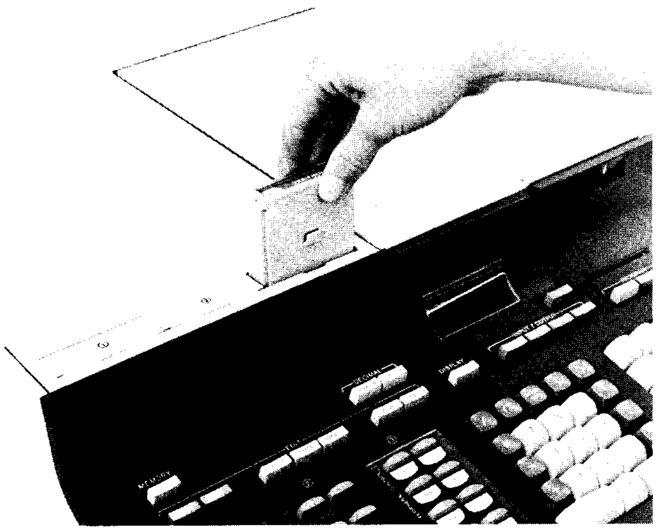


Figure 1-1. PC I Block Installation

ROM BLOCK MEMORY USAGE

When the PC I Block is installed, it requires the use of 22 words of User Read-Write Memory; this loss is indicated by the loss of 6 R-registers, as indicated at the end of any program listing.

Chapter 2

GENERAL PERIPHERAL CONTROL OPERATIONS

TABLE OF CONTENTS

NOTES

Chapter 2

GENERAL PERIPHERAL CONTROL OPERATIONS

INTRODUCTION

This chapter describes the general peripheral control operations which are available when the PC I Block is installed. The remainder of this manual contains specific operating instructions for controlling 9800 Series peripherals with the PC I Block.

Interfacing a peripheral device with the Model 20 requires both hardware and software; this chapter is primarily concerned with describing the software (i.e., types of peripheral control operations and how to perform them) available with the block. A brief discussion of the Model 20 input-output (I/O) scheme is also provided.

INTERFACING THE MODEL 20

The general I/O scheme for interfacing devices to the Model 20 is shown in Figure 2-1. The four I/O connectors on the calculator are connected in a 'party-line' fashion, thus permitting four* peripheral devices to be connected to the calculator at one time.

*Use of the Model 9868A I/O Expander permits up to thirteen devices to be connected to the calculator at one time.

Each device must be connected to the calculator through an appropriate interface card and cable. This card provides necessary electrical interface (signal conditioning, buffering, etc.) between the device and the calculator. 9800 Series peripherals are supplied with all required interfacing hardware. Also, interface kits are available for other interfacing applications.

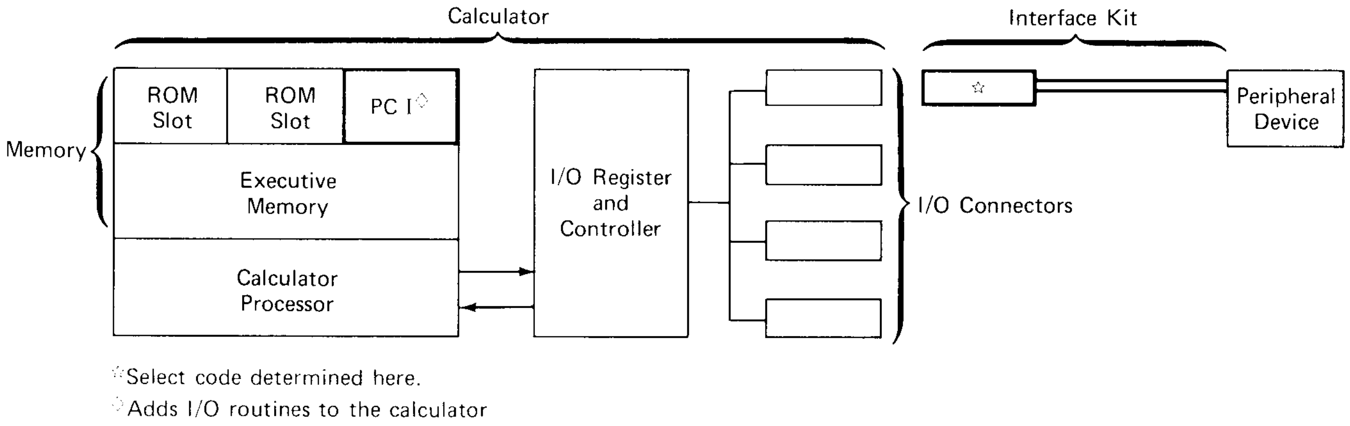


Figure 2-1. Model 20 I/O Scheme

INTERFACING THE MODEL 20

INTERFACE KITS

Interface kits are available to permit many devices which are not 9800 Series peripherals to be interfaced with the Model 20. The following is a brief description of two such interface kits.

The -hp- 11202A TTL I/O Interface enables the calculator to input and output information using standard ASCII codes. For example, the I/O Interface can be used to connect the -hp- Model 2748A Tape Reader or the -hp- Model 2895A Tape Punch to the Model 20 Calculator.

The -hp- 11203A BCD Input Interface enables the calculator to send a data request (sample) signal, and receive data from, a wide variety of devices which can output information in a 'binary coded decimal' (BCD) format (e.g., digital voltmeters, electronic counters, etc.).

Each of these interface kits is supplied with detailed 'hardware' related information (electrical specifications, recommended interface circuits, etc.) which should enable you to either interface your device directly to the Model 20 or to build the necessary additional hardware required in order to interface your device. With the proper use of one of these interface kits, the operations to be described may be utilized to control and to send or receive data to or from your peripheral device.

The nearest -hp- Sales and Service Office can furnish you with data sheets which list complete specifications for all available interface kits.

PERIPHERAL SELECT CODE

Since all peripheral devices are connected in a party-line fashion, each device must have a unique 'address' so that the calculator can specify which device should respond to each operation. This address (or select code) consists of a one or two-digit number and is determined by the interface card. In general, the select code is a fixed number for 9800 Series peripherals (e.g., the typewriter's select code is 1 5), whereas each interface kit contains a switch which permits the user to set any one of nine select codes. Each I/O operation must specify the correct select code, thereby causing the correct interface card to respond to the operation, while all other cards ignore it.

SELECT CODE SPECIFICATION

The select code specification can be in the form of an integer number or the contents of a data register. In addition, a WRITE statement can contain a select code specification in the form of an expression (NOTE 20 will appear if a READ or TRANSFER statement containing an expression to specify the select code is encountered).

INPUT-OUTPUT CODE

The PC I Block enables the Model 20 to send and receive data and to send coded commands in the form of standard ASCII* codes. The table of ASCII equivalent codes is on Page 2-7. In general, the calculator can send (write) data or commands in the form of user-selected ASCII codes, but it can receive (read) only numerical ASCII codes (data)**.

For both input and output operations, all information exchange between the calculator and a peripheral device is handled on a 'full handshake' basis, with information being transferred, character-by-character, in an 8-bit parallel, character-series fashion. Thus, for data or message output, the calculator sends one 8-bit character at a time; if another character is to be sent, the calculator will wait.

There is no provision for system 'interrupt' operation when using the PC I Block (i.e., when the peripheral device can initiate or call for an input or output operation). The calculator must be in complete control of each peripheral device while the device is involved in I/O operations. As mentioned earlier, if the device is not ready, the calculator will wait. However, the calculator can be taken out of the 'wait' status by pressing STOP.

*Americal Standard Code for Information Interchange.

**The 11203A BCD Input Interface actually converts the BCD coded data to ASCII coded data which is usable by the calculator.

WRITE STATEMENTS

The WRITE statement is a general purpose means to output information to an external device. The select code specified in the WRITE statement determines which external device receives the specified information.

The WRITE Syntax:

WRT ⟨select code⟩, ⟨parameter$_1$⟩, ⟨parameter$_2$⟩, ...

or

WRT ⟨select code⟩, ⟨list⟩

Each parameter can consist of a register name or a string of keys (see the table on Page 2-7) which are enclosed in quotes. The information in each parameter is output under the 'default format,' if it is in effect, or, under the format specified by the previous FORMAT statement. The default format and FORMAT statements are described later in this chapter.

DELIMITERS

A delimiter is a character that is used to separate one item from another item inside the list or to terminate the list.

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

READ STATEMENTS

READ statements enable the calculator to receive numerical data from an external device. The device which sends the data is specified by the select code in the READ statement.

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 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.

The READ Syntax:

RED ⟨select code⟩, ⟨register name$_1$⟩, ⟨register name$_2$⟩, ...

or

RED ⟨select code⟩, ⟨list⟩

EFFECTS OF DELIMITERS

A continuous string of numeric and certain other characters, occurring between two commas, a comma and a space, or two spaces, is a data item whose value corresponds to an element in a list of a READ statement. 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, after the CR LF has been encountered, there remain more elements in the list, they will be assigned values as they are read; the READ statement has not been terminated. However, 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.

READ STATEMENTS

FORM OF DATA ITEMS

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 an any single constant which is keyed in from the keyboard. See the NORMAL MODE section of Chapter 6 for general examples of the WRITE statement.

There is an exception to the way the delimiters operate when within a data item: 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.

For example, any of the following forms will cause the number '1234.' to be read:

```
1.234E3
1.234Eƀ3
1.234E+3
1.234+3
1.234+03
```

TRANSFER STATEMENTS

TRANSFER statements are used to transfer data, or other ASCII information, from one device to another device.

The TRANSFER Syntax:

TFR ⟨select code$_1$⟩ ; ⟨select code$_2$⟩

or

'Transfer from device$_1$ to device$_2$'

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 (see the table on Page 2-7). The EOM character is the only delimiter which the calculator responds to during a transfer operation.

DEFAULT FORMATTING

The default format is automatically set whenever the calculator is turned ON, MEMORY ERASE is pressed, or an END instruction is executed. The default format causes information to be output in a series of 18 character fields. As each item is output, it appears right-justified in one field. After four numeric items have been output (literals can be output, but they are not counted) a carriage-return line-feed (CR LF) is given, and the next item (if it exists) is output. When the list of the output statement is exhausted a CR LF is also given. (If the end-of-list CR LF coincides with a CR LF for grouping by fours, then only one CR LF is given, however.)

If a literal is longer than 18 characters, then as many fields as necessary are combined in order to make room for the literal. The literal then appears right-justified in the resulting expanded field.

The form in which numeric parameters will appear is determined by the current settings established by the FIXED or FLOAT instructions. A number that is too large to be output under a current fixed-point specification is output under the previous floating-point specification.

The default format remains in effect until another format is established with a FORMAT statement (described later).

See Chapter 4 for examples of typewriter output under the default format.

FORMAT STATEMENTS

FORMAT statements are used to precisely control the form in which output information appears.

The FORMAT Syntax:

FMT ⟨spec$_1$⟩ [, ⟨spec$_2$⟩ , ⟨spec$_3$⟩ , ...]

FORMAT statements consist of conversion specifications and editing specifications; each of these specifications is described later in this section.

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. If a FORMAT statement has not been previously encountered, the 'default format' is utilized.

FORMAT statements are not executed when encountered; instead, they are executed in conjunction with other statements that output information (TYP, WRT). 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.

NOTE 22 results if the FORMAT statement cannot be implemented with the output statement; this includes syntax errors, since FORMAT statements are not syntax checked when they are written.

Examples of FORMAT statements used with TYPE statements are shown in Chapter 4.

CONVERSION SPECIFICATIONS

Conversion specifications are used to determine the form in which a numerical parameter will appear. 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).

The syntax:

⟨integer constant⟩ FXD or FLT ⟨integer constant⟩ . ⟨integer constant⟩

or

r FXD or FLT 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.

A conversion specification like FXD 8.2 calls for outputting a fixed-point number with two digits to the right of the decimal point. The number is to appear (right-justified) in an eight-character field. (FXD is the mnemonic for the FIXED N key, not the result of pressing the F, X, and D keys, and FLT is the mnemonic for the FLOAT N key.) If d is 0, the decimal point is not output.

A number output under any conversion specification is always correctly rounded according to the number of decimal places specified.

Some guidelines should be observed in selecting w and d. Signs, decimal points, and exponents are part of the number and must fit in the field width specified by w. 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 or 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.

FORMAT STATEMENTS

CONVERSION SPECIFICATIONS (cont'd)

Here are the precise descriptions of when dollar signs occur:

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 $10 \geqslant N+d > w-1$ 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
```

EDITING SPECIFICATIONS

Editing specifications are used to control the placement of the output numerical data and to output headings and to send control characters to an external device.

There are four types of editing specifications:

The Syntax:

⟨integer constant⟩X or r X (spaces)

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

The Syntax:

⟨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.

The Syntax:

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.

The Syntax:

⟨literal⟩ or "message or ASCII characters"

When this type of editing specification is encountered, the ASCII characters corresponding to ⟨literal⟩ are output (the quote marks are not output). See Table 2-2 for a list of ASCII characters that can be output.

This editing specification must not be confused with a literal that appears in the list of a WRITE or TYPE statement. The two uses of a literal do not have the same properties. The ability to put a literal in the list of a WRITE or TYPE statement was included in the Model 20's language as a convenience in typing headings and labels on the typewriter. Such items are typed in black ribbon and in upper case letters only.

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).

KEYBOARD EXECUTION OF OUTPUT STATEMENTS

To execute an output statement from the keyboard (when using a different FORMAT statement than the one that is currently the 'last encountered FORMAT statement') the desired FORMAT statement must be in the same line as the output statement. Also, any subsequent output statement executed from the keyboard must also be preceded, in the same line, by the desired FORMAT statement. Failure to follow this rule will cause the subsequent statement to be output in the default format, and NOTE 22 will result.

FORMAT STATEMENTS

Table 2-2. 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	A	S	S	9	9	#	≠	EOT	$ **
B	B	T	T	+	+	!	STOP	RU	& **
C	C	U	U	–	–	@	GO TO	BELL	' **
D	D	V	V	*	*	$	$	NULL	SPACE **
E	E	W	W	/	/	%	%	VT	+ **
F	F	X	X	\	√	&	&	CR	– **
G	G	Y	Y	↑	ENTER EXP	'	'	LF	* **
H	H	Z	Z	(	(	?	?	ACK	√ **
I	I	0	0	)	)	WRU	LOAD	ESC	ENTER EXP **
J	J	1	1	←	→	FE	*	LEM	7 **
K	K	2	2	:	R( )	HT	*	DC4	4 **
L	L	3	3	;	;	S0	*	ERR	5 **
M	M	4	4	,	,	S1	*	SYNC	6 **
N	N	5	5	.	•	DC0	NORMAL	FF	, **
O	O	6	6	>	>	DC1	TRACE	EOM	≠ **
P	P	7	7	<	≤	DC2	FIXED N	SOM	STOP **
Q	Q	8	8	=	=	DC3	FLOAT N	RUB OUT	→ **
R	R					SPACE (ƀ)	SPACE		

* These are the blank keys (left-hand keyblock) and are shown in the same order as on the keyboard.

** Indicates shifted key.

NOTES

Chapter 3

PLOTTER CONTROL

TABLE OF CONTENTS

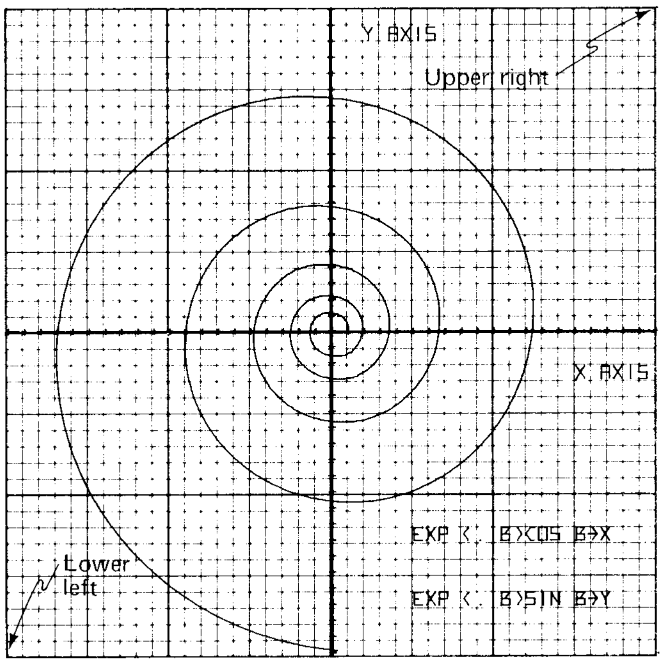


Figure 3-1. A Spiral, Plotter with the Graph Limit Controls Set for Four Inches by Four Inches.

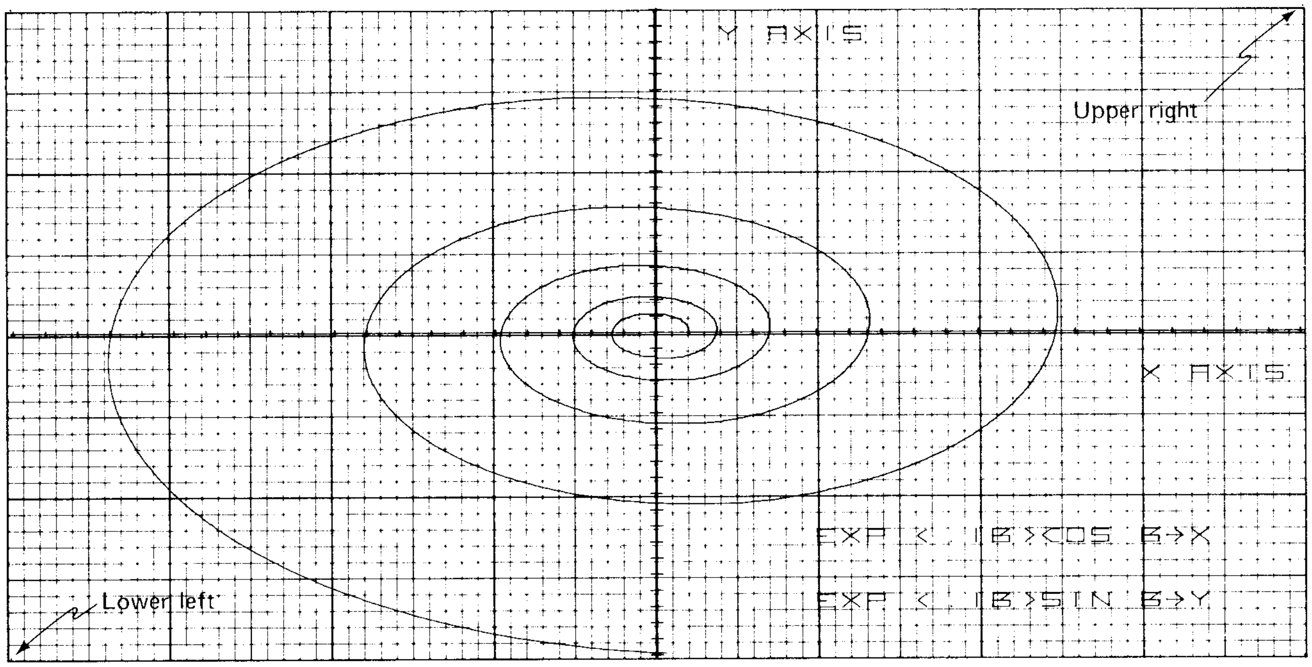


Figure 3-2. The Same Spiral, Plotter with the Graph Limit Controls Set for Four Inches by Eight Inches.

Chapter 3

PLOTTER CONTROL

INTRODUCTION

When the PC I block is installed, the Model 9862A Plotter can be used to create graphs or other images of data being processed by the calculator. Plotting solid and dotted lines, lettering in four directions, and automatically drawing axes are among the things that are easily done. All plotting and specification of coordinates are done in terms of the actual problem number range, rather than in terms of an absolute range based on the plotter's drive mechanism. The size of the lettering is independent of the numbers involved in a plot, but is related to the physical size of the plot. Nine heights and nine widths are available for lettering. The final physical size of the finished plot is unrelated to the calculator operations used to create it; front panel controls on the plotter are used to determine the size of the plot.

FRONT PANEL CONTROLS

LINE AND CHART HOLD

The LINE pushbutton is the power switch for the plotter; press it to apply power, and press it again to remove power; the white LINE lamp lights whenever the plotter is ON.

Pressing CHART HOLD activates the electro-static paper hold-down mechanism. Pressing CHART HOLD again deactivates it. The plotter will not plot or letter, and the pen holder and arm will move freely in all directions when CHART HOLD is deactivated.

LOADING PAPER

To load paper, release CHART HOLD and manually move the pen arm all the way to one side of the plotter. Lay a sheet of paper on the plotting surface and smooth out any irregularities in the paper (you may also wish to ensure that the paper is squarely against the ridge at the bottom of the plotting surface); then activate CHART HOLD.

GRAPH LIMITS

The graph limit controls are used to determine the physical size of the plot.

LOWER LEFT and the two knobs to its left are used to determine the physical location of the lower left-hand corner of the plotting area.

UPPER RIGHT and the two knobs to its right are used to determine the physical location of the upper right-hand corner of the plotting area. Together, the upper right-hand corner and the lower left-hand corner determine the size of the plotting area. Also, altering the lower left-hand setting will translate the upper right-hand setting by the same direction and amount.

To specify the lower left-hand corner of the plotting area, press LOWER LEFT; the pen will move (without touching the paper) to the lower left-hand corner of the plotting area. This point can be set anywhere within the lower left-hand quarter of the plotting surface (platen) by adjusting the two knobs associated with LOWER LEFT. (If desired, PEN DOWN and PEN UP can be used to mark or determine the exact point on the paper which is the lower left-hand corner of the plotting area.) Once the lower left-hand corner has been set, the upper right-hand corner is set in the same general way by pressing UPPER RIGHT and adjusting the two knobs associated with it. Once the plotting area has been determined, it can be relocated by moving the position of the lower left-hand corner – the upper right-hand corner will 'track' the change.

Figures 3-1 and 3-2 show the effect of adjusting the graph limit controls. In each case, the program was the same but the size of the plotting area was changed by adjusting the graph limit controls.

SCALE STATEMENTS

SCALE statements are used to specify the problem variable range, so that subsequent commands to move the pen to various coordinates will have meaning. A SCALE statement must be executed before any plotting can occur. Once a SCALE statement is in effect, it stays in effect until the memory is erased, or until another SCALE statement is executed.

A SCALE statement supplies the calculator with the following information: the maximum and minimum values of the x (horizontal direction) variable, and the maximum and minimum values of the y (vertical direction) variable (see Figure 3-3).

When a SCALE statement is executed, the calculator automatically assumes that the maximums and minimums correspond to the edges of the plotting area and computes the internal scale factors*.

*The plotter divides the plotting area into 10,000 divisions in the x direction and 10,000 divisions in the y direction. With the scaling information, the calculator converts problem coordinates into plotter coordinates (based on the 10,000 divisions) and sends the absolute coordinates to the plotter.

The SCALE syntax:

SCL ⟨value⟩ ; ⟨value⟩ ; ⟨value⟩ ; ⟨value⟩

or

SCL x_{min} ; x_{max} ; y_{min} ; y_{max}

Figures 3-4 and 3-5 show the result of plotting the same relationship with two different SCALE statements. In each case the relationship:

$$y = (\sin x)/x$$

was plotted from -4π radians to $+4\pi$ radians. In Figure 3-4, the SCALE statement set x_{min} and x_{max} to -4π, respectively, while in Figure 3-5, they were set to -8π and $+8\pi$, respectively.

Notice that even though the shapes of the graph and the location of the lettering changed, the size of the lettering did not change. This is because lettering size is specified in terms of a percentage of the size of the plotting area, and not in terms of the problem variable range.

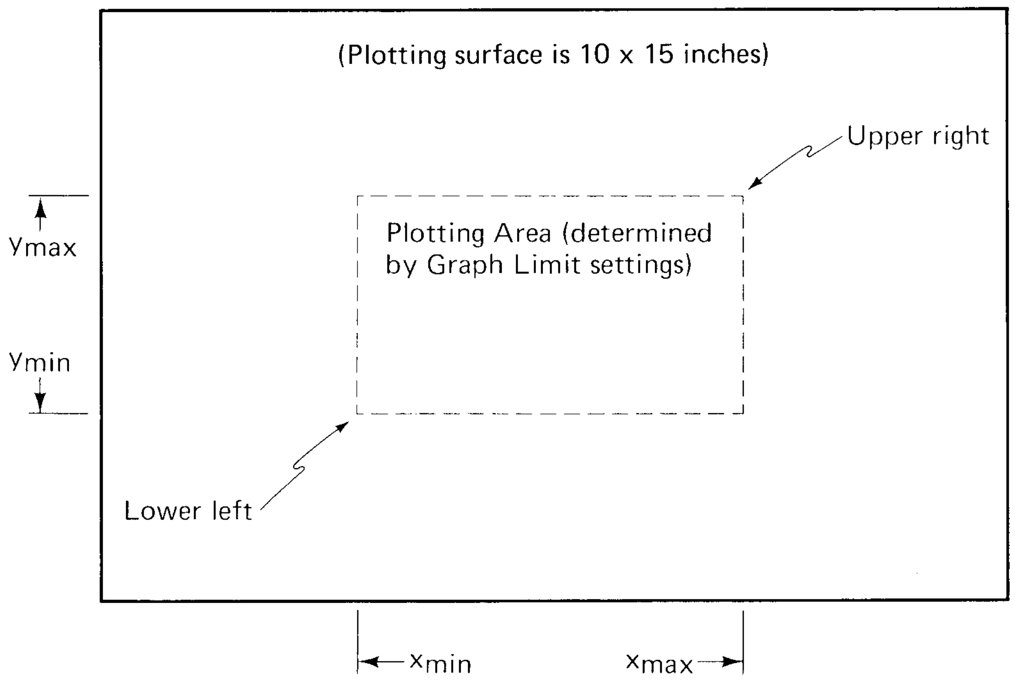

(Plotting surface is 10 x 15 inches)
Upper right
Plotting Area (determined by Graph Limit settings)
Ymax
Ymin
Lower left
Xmin
Xmax

Figure 3-3. The Plotting Surface and the Plotting Area.

SCALE STATEMENTS

```
SCL -4π,4π,-.5,1.5
```

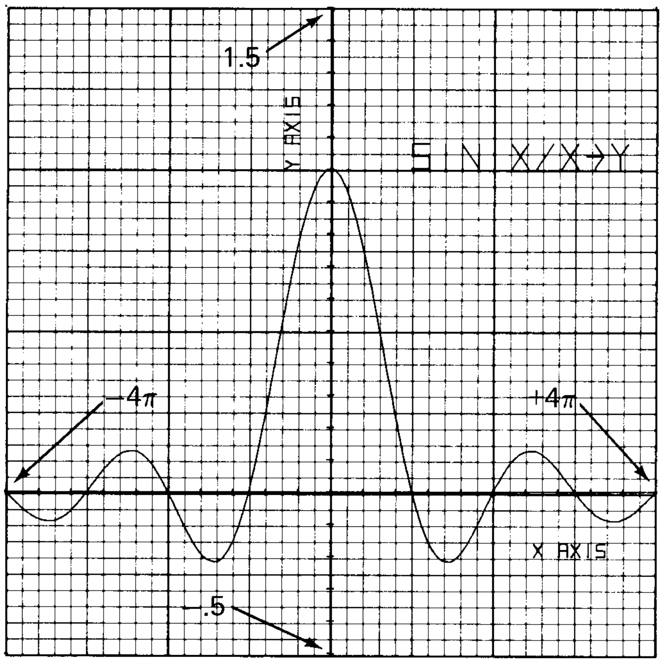


Figure 3-4. Plotting from −4π to +4π, while scaled from −4π to +4π.

```
SCL -8π,8π,-.5,1.5
```

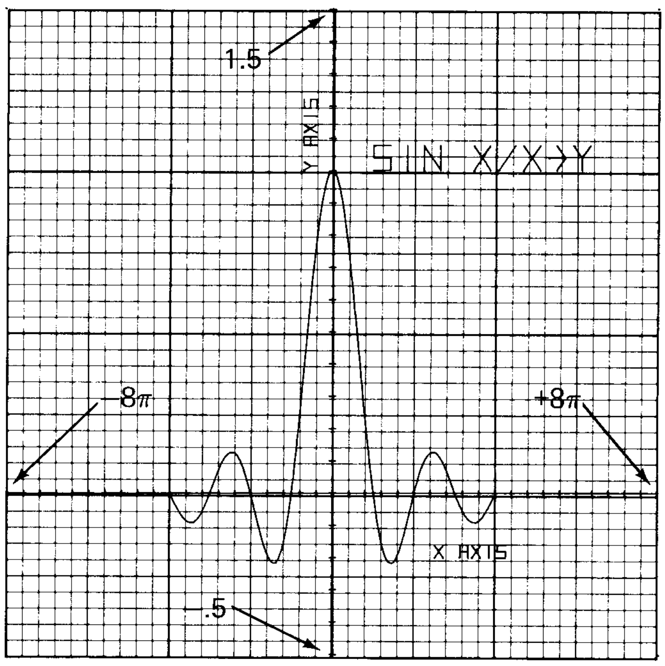


Figure 3-5. Plotting from -4π to +4π, while scaled from −8π to +8π.

PLOT STATEMENTS

PLOTTING LINES

The Syntax:

PLT ⟨value⟩ , ⟨value⟩

or

PLT $x_{\text{coordinate}}$, $y_{\text{coordinate}}$

This syntax is used to plot graphs. If the pen is raised, it will move to the coordinates specified, and then drop. If the pen is already down, it will move in a straight line from its present position to the one specified by the coordinates, drawing a straight line in the process.

Graphs of mathematical relationships, and other uses of this syntax involving curved lines, are plotted by moving the pen in small increments. The series of short, straight line segments produced is often indistinguishable from the actual curve.

Figure 3-6 is a plot of the relationship:

$$y = \frac{x^2 - 25}{x - 6}$$

The relationship was plotted from x = -10 to x =30, in steps of .21 (that particular value was chosen to accentuate the discontinuity -- Δx could have been .2 or .1, or, something else).

First, x is set to -10. Then a series of PLOT statements are executed, while incrementing the value of x until it is high enough. Two methods of generating the plot in Figure 3-6 are shown on the following page.

PLOT STATEMENTS

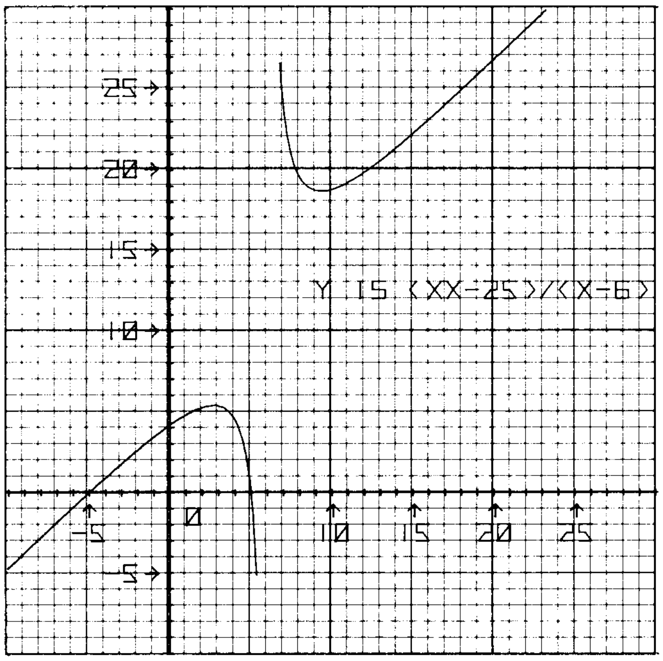


Figure 3-6. Plotting a Graph.

```
:PLT X,(XX-25)/(X-6);
                    JMP X+1→X>30⊢
```

x coordinate

y coordinate

increment x and when x gets large enough, abort this line and go to the next line; otherwise plot again.

Note that the use of the X and Y registers to represent $X_{coordinate}$ and $Y_{coordinate}$ is not necessary: any other registers or suitable expressions could have been used.

When the coordinates of a point to be plotted are outside the plotting area, the pen is lifted before it moves to the edge of the plotting area. The ERROR light on the plotter will not light, nor will the plotting process be interrupted. The pen will return to the paper and continue plotting if the coordinates again fall within the plotting area (see Figure 3-6).

```
:-10→X⊢
```

set x to -10

```
:(XX-25)/(X-6)→Y⊢
```

find the y coordinate

```
:PLT X,Y⊢
```

plot the next point

```
:IF X≤30;X+.1→X;GTO -2⊢
```

if x is not large enough, increment x, then go back and find new y and plot.

or

```
:-10→X⊢
```

set x to -10

PLOTTING LETTERS

The Syntax:

PLT "⟨literal⟩"

or

PLT message or label

This syntax causes the plotter to draw the characters contained in ⟨literal⟩. The characters that can be drawn, and their corresponding keys are shown in Figure 3-7. Note that the equals sign and the asterisk cannot be drawn and that the '*' key causes the arrow to be drawn.

The size, direction (horizontal, vertical, upside-down, etc.), and starting location of the lettering are determined with a LETTER statement. The PLOT statement only specified what, not how, the characters are to be drawn. A LETTER statement must be executed before any characters are drawn.

PLOT STATEMENTS

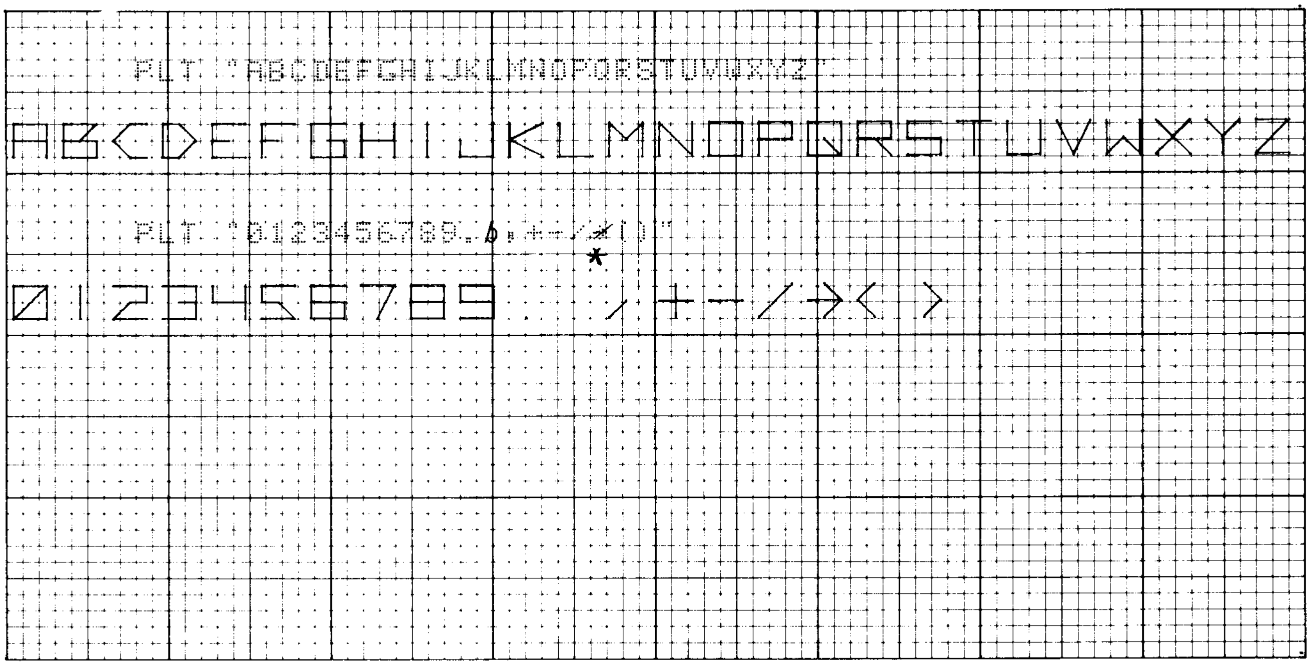


Figure 3-7. Plotter Characters and Their Corresponding Keys.

The starting location that is identified with the LETTER statement corresponds to the lower left-hand corner* of a rectangle enclosing the left-most character to be plotted. From there the lettering is done until every character in ⟨literal⟩ has been drawn. If the edge of the plotting area is reached before this process is complete, the remaining characters are drawn as straight-line segments upon the associated boundary of the plotting area; the pen does not lift under these circumstances.

PLOTTING NUMBERS

The Syntax:

PLT ⟨value⟩

or

(Draw the numeric value)

*That is, when viewing the character 'right-side up.' If the character were drawn up-side down, the starting location is the upper right-hand corner of the enclosing rectangle.

This syntax is used to draw the signs and digits of a number identified by ⟨value⟩. The size, direction, and starting location of the plotting is determined by a previously executed LETTER statement.

However, the form of the number (fixed point or floating point, and the number of digits to the right of the decimal point) is controlled by the current FIXED and FLOAT setting. In this respect the plotter behaves like the built-in printer, especially in the matter of revision to floating point when the number to be lettered is too large to fit in the given fixed point specification.

Most labelling of the axes in the example plots of this chapter were done with PLOT ⟨value⟩ statements, in conjunction with a LETTER statement that moves the starting position to successive places along the axes.

PEN STATEMENTS

The PEN Syntax:

```
. . . . ; PEN ; . . . .
```

or

(Lift the pen)

This syntax is used to lift the pen; however, it does not change the position of the pen on the plotting area. PEN statements are used when plotting discrete points, drawing dotted lines, and when drawing one graph over another - so that the line between the end of the first graph and the beginning of the second graph is not drawn. Figures 3-8 and 3-9 illustrate all three of these operations.

In each of these program segments, 'counter' is a register used to count the number of increments that are plotted while plotting a section of dotted line. When enough solid line is drawn, the pen is lifted, and x is incremented by an amount large enough to produce an appropriate blank space between solid line segments.

Plotting dashed lines with alternating long and short solid line segments is done in the same general way, except that 'enough' is alternately larger and smaller.

The basic scheme for dotted line plotting is as follows.

```
: xmin → X ⊢
: 0 → counter ⊢
: f(x) → Y ⊢
: PLT  X, Y ⊢
: IF xmax ≤ X ; GTO  +5 ⊢
: X + Δx → X ⊢
: counter + 1 → counter ⊢
: IF counter = enough ; PEN  ; 0 → counter ;
                              X + sizable shift → X ⊢
: GTO  -6 ⊢
```

or

```
: xmin → X ; 0 → counter ⊢
: PLT  X, f(x) ; IF xmax > X ; X + Δx → X ;
                counter + 1 → counter ; GTO  -0 ;
                IF counter = enough ;
         PEN  ; 0 → counter ; X + sizable shift → X ⊢
```

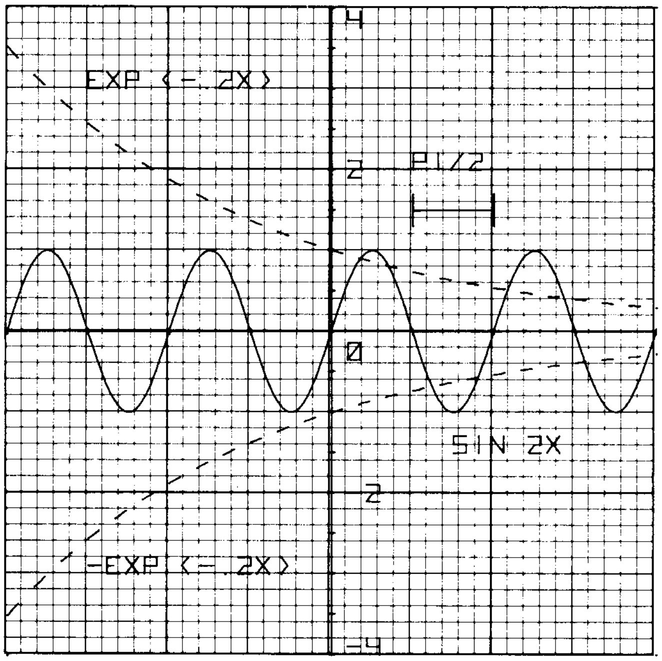


Figure 3-8. Superimposed Graphs With Dotted Lines.

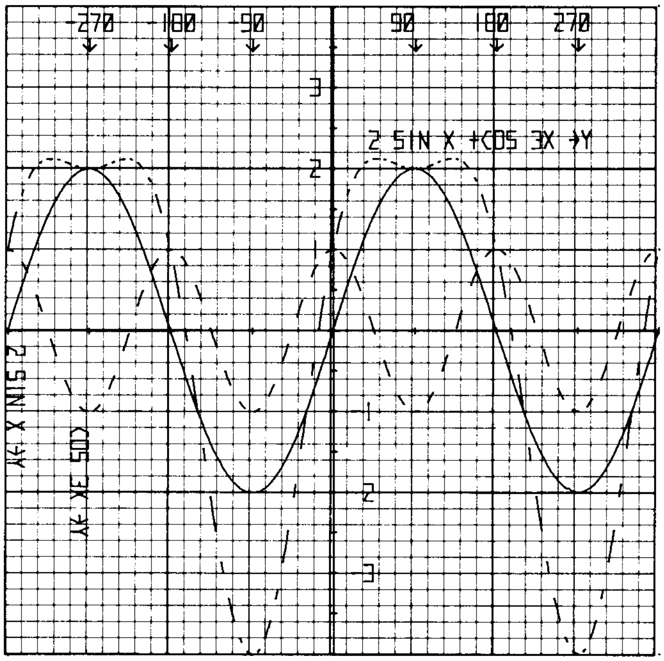


Figure 3-9. Superimposed Graphs With Two Kinds of Dotted Lines.

LETTER STATEMENTS

LETTER statements are the means to specify the starting point, size, and direction of lettering to be performed by a subsequent PLOT statement. The LETTER statement itself does not cause any actual lettering.

The LETTER Syntax:

LTR ⟨value⟩ , ⟨value⟩ [, ⟨3 digit constant⟩]

or

LTR xlocation , ylocation , HWD

Where: H = height W = width D = direction

The two ⟨value⟩'s specify the coordinates at which the lettering is to begin. When a LETTER statement is encountered, the pen will lift and then move to the coordinates. The pen will remain raised until some subsequent activity requires it to contact the paper (after lettering with a PLOT statement the pen is also raised).

The ⟨3 digit constant⟩ is simply a series of three single digits; no commas, decimal points, or items other than digits are allowed. Each digit controls some aspect of the appearance of any lettering performed while that particular HWD is in effect (see Figures 3-10 and 3-11).

HWD is optional; once specified, it may be omitted from any subsequent LETTER statements if that HWD is to remain unchanged.

H and W may range from 1 to 9, inclusive. H and W represent a percentage of the size of the plotting area along their respective directions. For instance, if H and W are both 3, then characters are drawn square (block letters), provided that the graph limit controls are set to define a square plotting area.

Exact sizes of the lettering are determined as follows:

$.0064H(y_{max} - y_{min})$ = height

$.0064W(x_{max} - x_{min})$ = width

$.0096W(x_{max} - x_{min})$ = overall character to character width

or

$.0032W(x_{max} - x_{min})$ = space between characters

D may range from 1 to 4, inclusive, and determines the direction in which the lettering will be performed (see Figure 3-11).

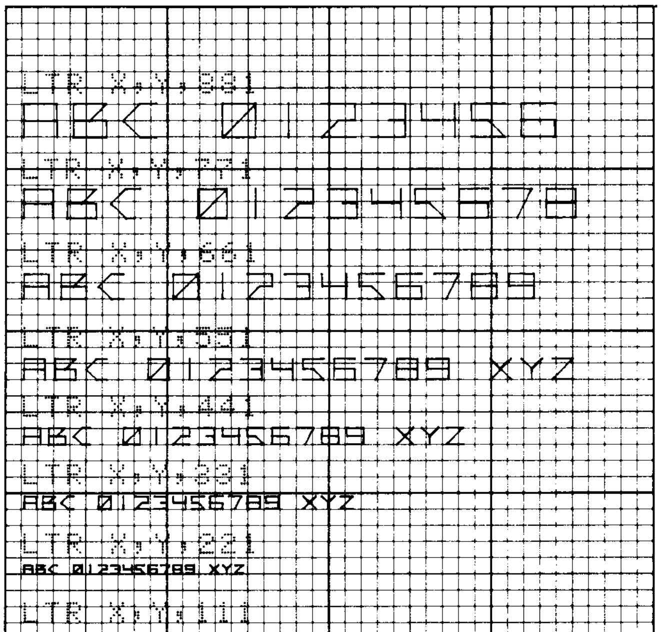


Figure 3-10. The Various Lettering Sizes.

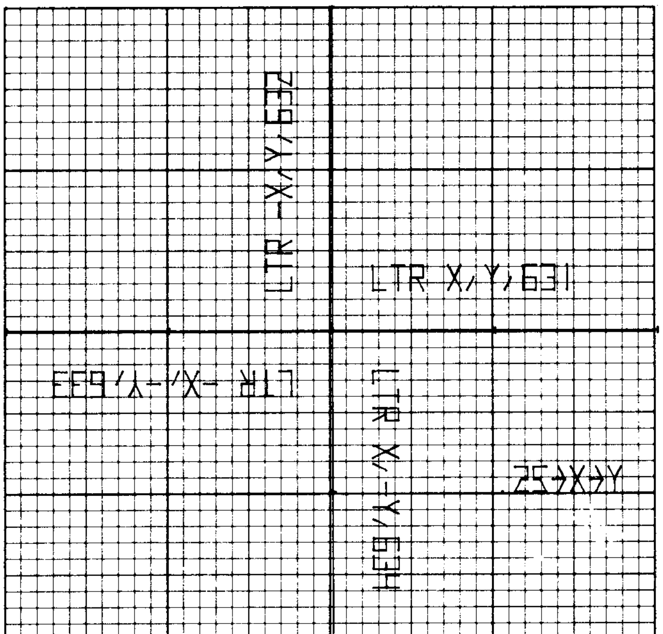


Figure 3-11. Lettering in the Four Differential Directions.

AXES STATEMENTS

AXES statements are used to automatically draw axes. The intersection of the axes can be anywhere in the plotting area, regardless of how the area is scaled. An additional feature is the optional automatic addition of tic marks to the axes.

The AXES Syntax:

AXE ⟨value⟩ , ⟨value⟩ [,⟨value⟩ , ⟨value⟩]

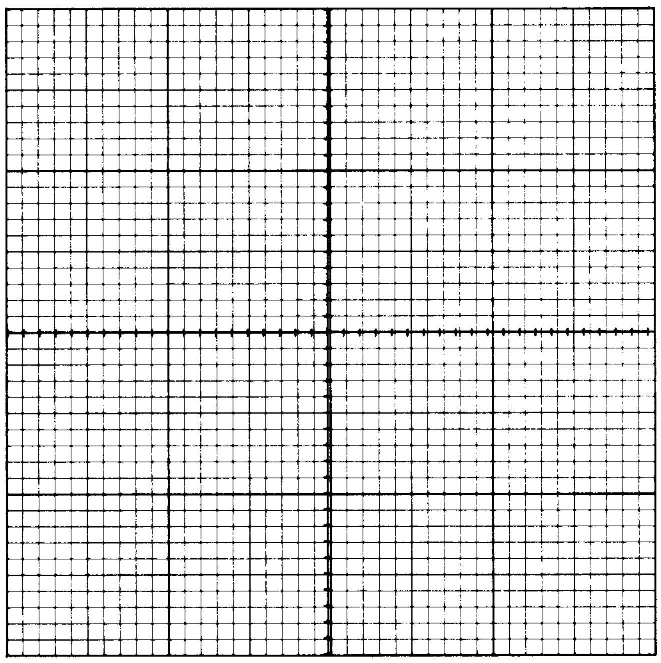
or

AXE $x_{location}$, $y_{location}$, x_{tic} , y_{tic}

$x_{location}$ and $y_{location}$ specify the coordinates at which the axes are to intersect. Generally, those coordinates are (0,0), (the origin). If x_{tic} and y_{tic} are omitted, both axes are drawn as straight lines. If either value is zero, tic marks are omitted on the respective axis. Also, if either value is small enough to require more than 10,000 tic's per axis, or if either value is so large that no tic's will 'fit' on the axis, tic marks for the respective axis are omitted. NOTE 21 occurs if $x_{location}$ or $y_{location}$ cause the intersection of the axes to fall outside the plotting area.

Axes with Tic Marks Intersecting at the Origin.

```
SCL -2,2,-20,20
AXE 0,0,.1,1
```



Axes Intersecting at the Origin.

```
SCL -1,3,-10,30
AXE 0,0,.1,1
```

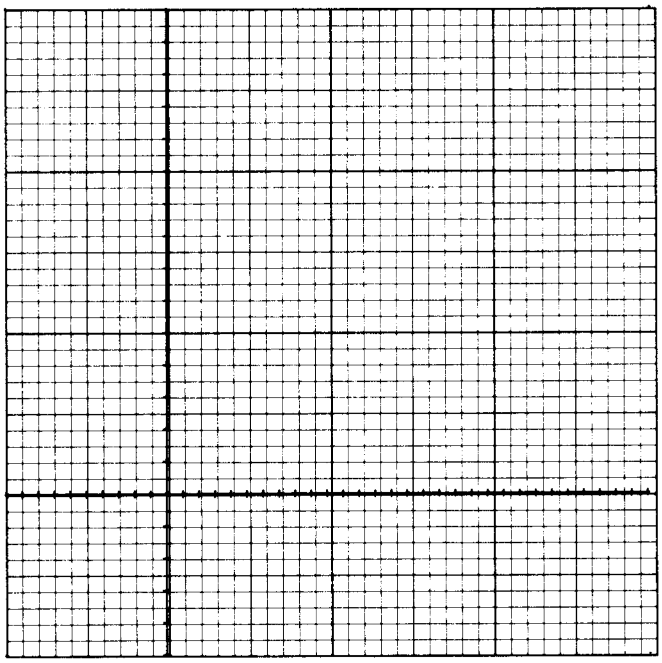
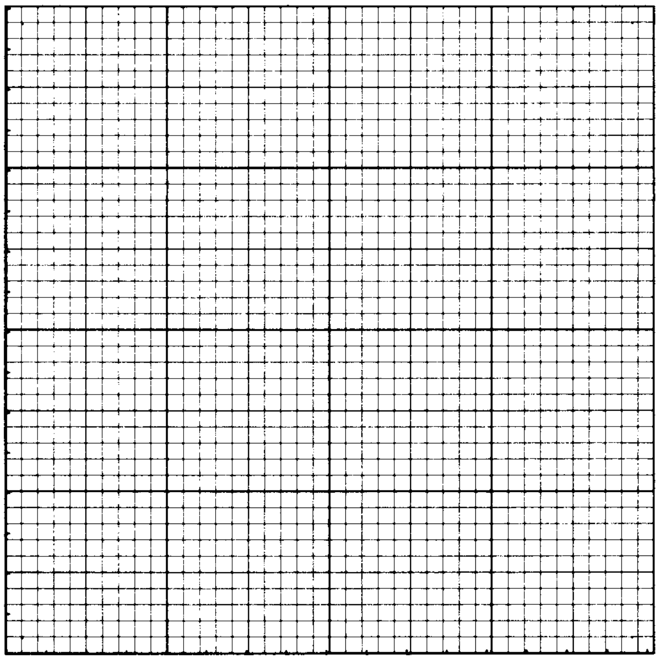


Figure 3-12. Results of Various AXES Statements.

AXES STATEMENTS

Axes Against the Edge of the Plotting Area.

```
SCL -2,2,-2,2
AXE -2,-2,.25,.2
```



Tic Marks suppressed on one Axis.

```
SCL -200,200,-100,300
AXE 0,0,25,0
```

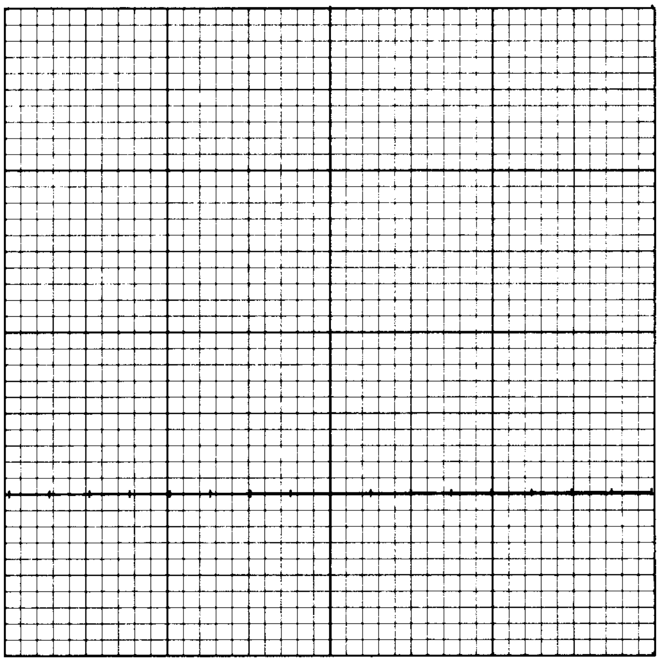


Figure 3-12. Results of Various AXES Statements. (Cont'd)

Tic Mark Error Correction

There is an inherent error in the PC I Block routine which computes where tic marks are to be drawn. This error becomes apparent when many tic marks are drawn on a given axis. The percent of error can be expressed as follows:

% of full scale error = (number of tic marks).01

To draw precise tics, a correction factor must be added to the ⟨tic⟩ parameters in each AXES statement as shown below.

AXE x_{loc} , y_{loc} , x_{tic} (1.01), y_{tic} (1.01)

For example, if the following SCALE statement is in effect

```
SCL 0,100,0,100 ⊢
```

and 100 tic marks are to be drawn on each axis, the following AXES statement will cause precise tic marks to be drawn.

```
AXE 50,50,1(1.01),1(1.01)⊢
```

If the correction factors were not included in this example, the error would actually cause 101 tic marks to be drawn on each axis.

NOTES

Chapter 4

TYPEWRITER CONTROL

TABLE OF CONTENTS

NOTES

Chapter 4

TYPEWRITER CONTROL

INTRODUCTION

The MODEL 9861A Typewriter may be controlled by using TYPE statements. TYPE statements can be used with or without FORMAT statements; the reader should be familiar with the use of FORMAT statements and 'default formatting,' as described in Chapter 2.

If you suspect the performance of the typewriter or the typewriter control operations available with the PC I, see Supplement B of the Model 20 Electrical Inspection Booklet.

TYPE STATEMENTS

The TYPE Syntax:

```
TYP ⟨list⟩

                    or

TYP ⟨parameter1⟩ ; ⟨parameter2⟩ ; . . . .
```

The elements of ⟨list⟩ can be anything that could appear in a PRINT statement. A carriage-return, line-feed (CR LF) operation is automatically given when the list is exhausted (unless a FORMAT statement dictates otherwise). Also, the typing format is in black and in upper-case characters, unless a FORMAT statement specified otherwise.

The Syntax:

```
TYP ; . . .
```

causes a CR LF when used with default formatting, and initiates a scan of the last previously encountered FORMAT statement when used under the control of a FORMAT statement. This syntax can be used to type information contained in FORMAT statements (editing specifications). For example, consider the syntax:

```
FMT ⟨edit spec1⟩[ ; ⟨edit spec2⟩ ; . . . .]
                                ; TYP ; . . . .
```

In this case, the TYPE statement is merely used to implement the FORMAT statement.

TYPING WITH THE DEFAULT FORMAT

Under the default format, the typing format consists of four columns of 18 spaces each. The four columns are adjacent to each other, and the left edge of the left-most column is set wherever the carriage is positioned when the TYPE statement is encountered. (Usually, this location is over the left margin, because of a previous CR LF.)

When the type statement is encountered, the first parameter is typed (right-justified) in the left column; the next parameter is typed in the next column; etc. After each group of four numerical parameters are typed, a CR LF is automatically given.

A literal parameter is typed in the same way as previously described; however, the literal parameter is not counted towards the limit of four numeric parameters on each typewritten line (see Example C on page 4-3). If a literal is longer than 18 characters, two or more adjacent 18-character columns are combined and treated as a single column. The literal is then right-justified in the larger column.

(continued)

TYPING WITH THE DEFAULT FORMAT

The typed appearance of a numeric parameter is controlled by the current settings established by the FIXED N and FLOAT N keys (N refers to the number of digits to be printed after the decimal point). In a fixed-point setting, no decimal point is typed if N is zero. Also, if a number is too large to be typed under the current fixed point setting, the number is typed under the previous floating point setting.

ESTABLISHING THE DEFAULT FORMAT

The default format is automatically set when the calculator is turned on or when MEMORY ERASE is pressed. The default format will remain set until another format is established with a FORMAT statement.

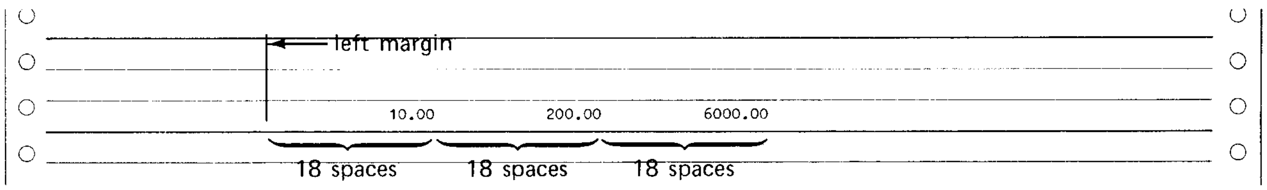
The default format can be reestablished from the keyboard by executing an END statement.

EXAMPLE A:

Press ERASE, then load and run the following program.

```
0:
FXD 2⊢
1:
10→A;200→B;6000→
C⊢
2:
TYP A;B;C⊢
3:
END ⊢
```

Result:



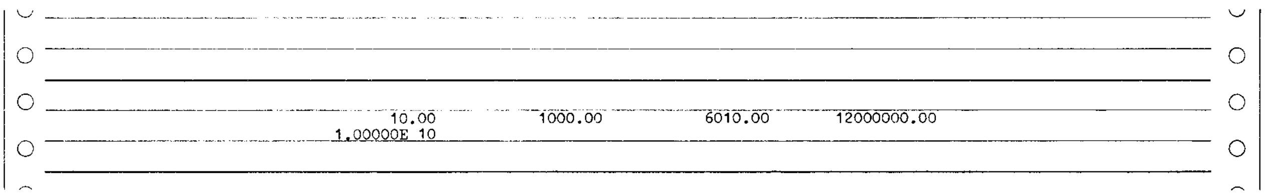
```
             10.00             200.00            6000.00
```

EXAMPLE B:

Execute the line:

```
TYP A;5B;A+C;ABC;A/1E-9⊢
```

Result:



```
             10.00            1000.00            6010.00        12000000.00
      1.00000E 10
```

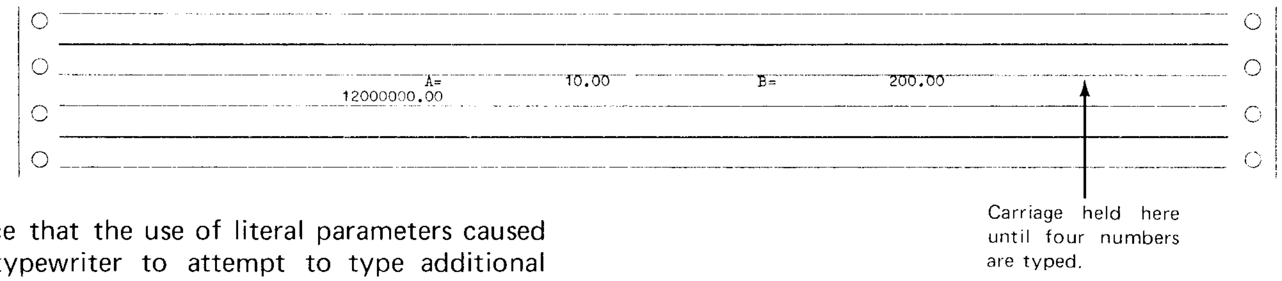
TYPING WITH THE DEFAULT FORMAT

EXAMPLE C:

Execute the line:

```
TYP "A=",A,"B=",B,"C=",C,"AB=",AB,"ABC=",ABC⊢
```

Result:



```
            A=         10.00          B=          200.00
  12000000.00
```

Notice that the use of literal parameters caused the typewriter to attempt to type additional parameters on the same line; when four numeric parameters were 'typed,' a CR LF was given and the next parameter was typed.

TYPING WITH FORMAT STATEMENTS

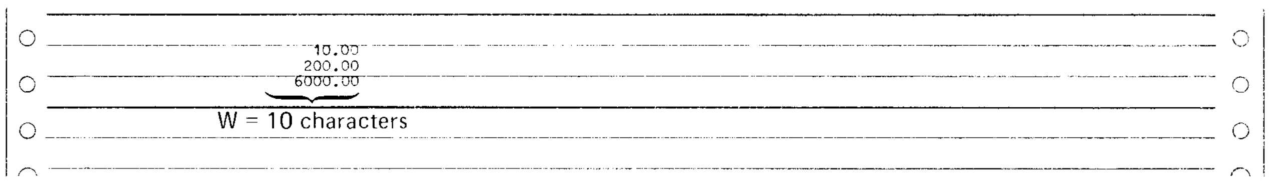
Typing with FORMAT statements allows more complete and flexible control of the typewriter than is otherwise possible. Typing is under the control of a FORMAT statement when a TYPE statement is encountered and there has been a previously encountered FORMAT statement. Under these conditions, a one-to-one correspondence is formed between the list of parameters in the TYPE statement and the list of specifications in the FORMAT statement. As the parameters are typed, each specification determines the format of the corresponding parameter.

EXAMPLE A:

Execute the line:

```
FMT FXD 10.2;TYP A,B,C⊢
```

Result:



```
     10.00
    200.00
   6000.00
```

Since there was only one conversion specification, a CR LF was given after each parameter was typed; thus, the use of one specification results in a one-column typeout.

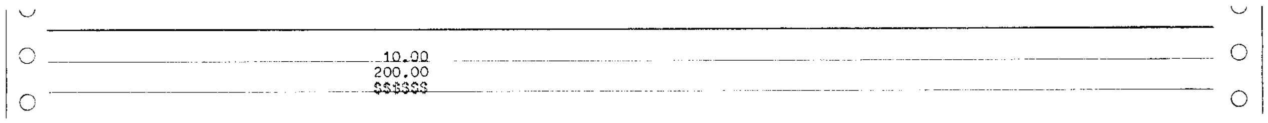
TYPING WITH FORMAT STATEMENTS

EXAMPLE B:

Execute the line:

```
FMT FXD 6.2;TYP A,B,C⊢
```

Result:



```
10.00
200.00
$$$$$$
```

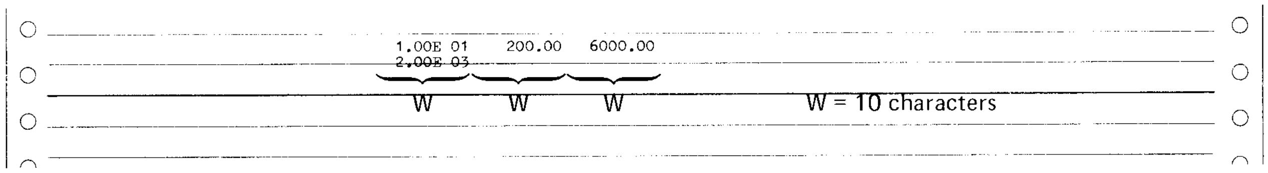
Since the field width specified was too small to allow parameter C to be typed, '$' characters were typed.

EXAMPLE C:

Execute the line:

```
FMT FLT 10.2,2FXD 10.2;TYP A,B,C,AB⊢
```

Result:



```
1.00E 01    200.00    6000.00
2.00E 03
```

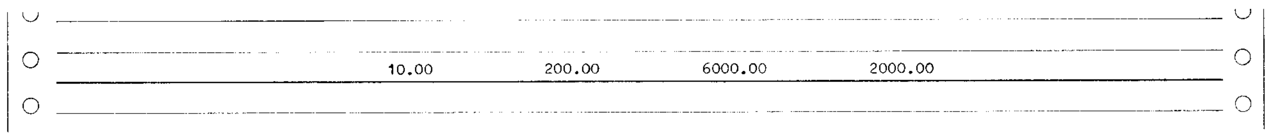
After each specification was used in the FORMAT statement, a CR LF was given and the next parameter was typed by using the first specification again. Thus, the sequence of instructions in a FORMAT statement determines the exact typing format.

EXAMPLE D:

Execute these lines from the keyboard:

```
FMT FLT 2.0⊢
TYP A,B,C,AB⊢
```

Result:



```
10.00        200.00        6000.00        2000.00
```

Display: `NOTE 22`

Since the FORMAT statement was not included in the same line as the TYPE statement, the parameters were typed in the default format and NOTE 22 appeared to remind the operator of the error.

Also, even if both statements were executed in one line, the FLT 2.0 parameter would be out of range (see Page 2-6), and $ characters would be typed for each parameter.

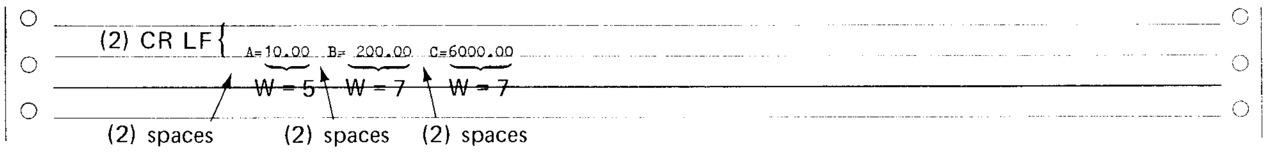
TYPING WITH FORMAT STATEMENTS

EXAMPLE E:

Load and run the following program.

```
0:
10→A;200→B;6000→
C⊢
1:
FMT 2/,2X,"A=",
FXD 5.2,2X,"B=",
FXD 7.2,2X,"C=",
FXD 7.2⊢
2:
TYP A,B,C⊢
3:
END ⊢
```

Result:



```
A=10.00  B= 200.00  C=6000.00
```

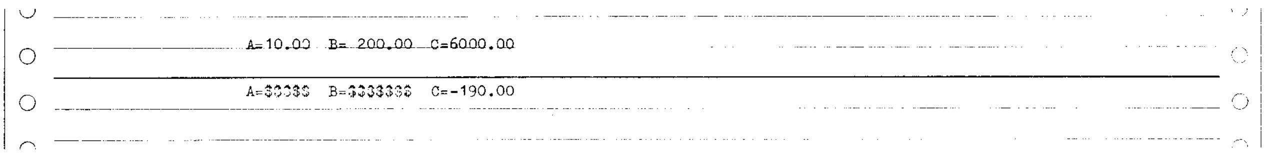
Notice the use of editing specifications to position the typewriter carriage and label the typed parameters.

EXAMPLE F:

Now modify line 2 (to cause more parameters to be typed) as shown below, and run the modified program.

```
2:
TYP A,B,C,A+B,ABC,A-B⊢
```

Result:



```
A=10.00  B= 200.00  C=6000.00
A=$$$$$  B=$$$$$$$  C=-190.00
```

Notice that after enough parameters are typed to exhaust the list of specifications in the FORMAT statement, a CR LF is given and the FORMAT statement is reused when typing the remaining parameters.

TYPING WITH FORMAT STATEMENTS

EXAMPLE G:

Load and run the following program (this program requires the use of the Math Block).

```
0:
TBL 1;SFC 14;0→A
⊢
1:
0→X⊢
2:
FMT /,6X,"DEGREE
S",6X,"RADIANS",
10X,"SIN",12X,"C
OS",10X,"TAN",/⊢
3:
TYP ⊢
4:
FMT FXD 10.0,5X,
FXD 10.2,5X,FXD
10.3,5X,FXD 10.3
,5X,FLT 10.3⊢
5:
TYP X,πX/180,
SIN X,COS X,TAN
X;A+1→A;IF A=5;0
→A;TYP ⊢
6:
IF 360>X;X+10→X;
GTO -1⊢
7:
FMT 10/;TYP ⊢
8:
END ⊢
```

Result:

```
      DEGREES       RADIANS          SIN            COS          TAN

            0          0.00        0.000          1.000    0.000E 00
           10           .17         .174           .985    1.763E-01
           20           .35         .342           .940    3.640E-01
           30           .52         .500           .866    5.774E-01
           40           .70         .643           .766    8.391E-01

           50           .87         .766           .643    1.192E 00
           60          1.05         .866           .500    1.732E 00
           70          1.22         .940           .342    2.747E 00
           80          1.40         .985           .174    5.671E 00
           90          1.57        1.000          0.000    9.999E 99

          100          1.75         .985          -.174   -5.671E 00
          110          1.92         .940          -.342   -2.747E 00
          120          2.09         .866          -.500   -1.732E 00
          130          2.27         .766          -.643   -1.192E 00
          140          2.44         .643          -.766   -8.391E-01

          150          2.62         .500          -.866   -5.774E-01
          160          2.79         .342          -.940   -3.640E-01
          170          2.97         .174          -.985   -1.763E-01
          180          3.14        0.000         -1.000    0.000E 00
          190          3.32        -.174          -.985    1.763E-01

          200          3.49        -.342          -.940    3.640E-01
          210          3.67        -.500          -.866    5.774E-01
          220          3.84        -.643          -.766    8.391E-01
          230          4.01        -.766          -.643    1.192E 00
          240          4.19        -.866          -.500    1.732E 00

          250          4.36        -.940          -.342    2.747E 00
          260          4.54        -.985          -.174    5.671E 00
          270          4.71       -1.000          0.000    9.999E 99
          280          4.89        -.985           .174   -5.671E 00
          290          5.06        -.940           .342   -2.747E 00

          300          5.24        -.866           .500   -1.732E 00
          310          5.41        -.766           .643   -1.192E 00
          320          5.59        -.643           .766   -8.391E-01
          330          5.76        -.500           .866   -5.774E-01
          340          5.93        -.342           .940   -3.640E-01

          350          6.11        -.174           .985   -1.763E-01
          360          6.28        0.000          1.000    0.000E 00
```

Notice that lines 2 and 3 cause the headings to be typed; lines 4 and 5 cause each block of parameters to be typed; and line 7 merely causes the carriage to CR LF 10 times before the program is completed.

TYPING WITH FORMAT STATEMENTS

CARRIAGE AND RIBBON CONTROL WITH FORMAT STATEMENTS

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. Figure 4-1 shows the typewriter functions that can be controlled by using the corresponding key as an editing specification. The typewriter control keys available when the calculator keyboard is 'shifted' are colored light-brown.

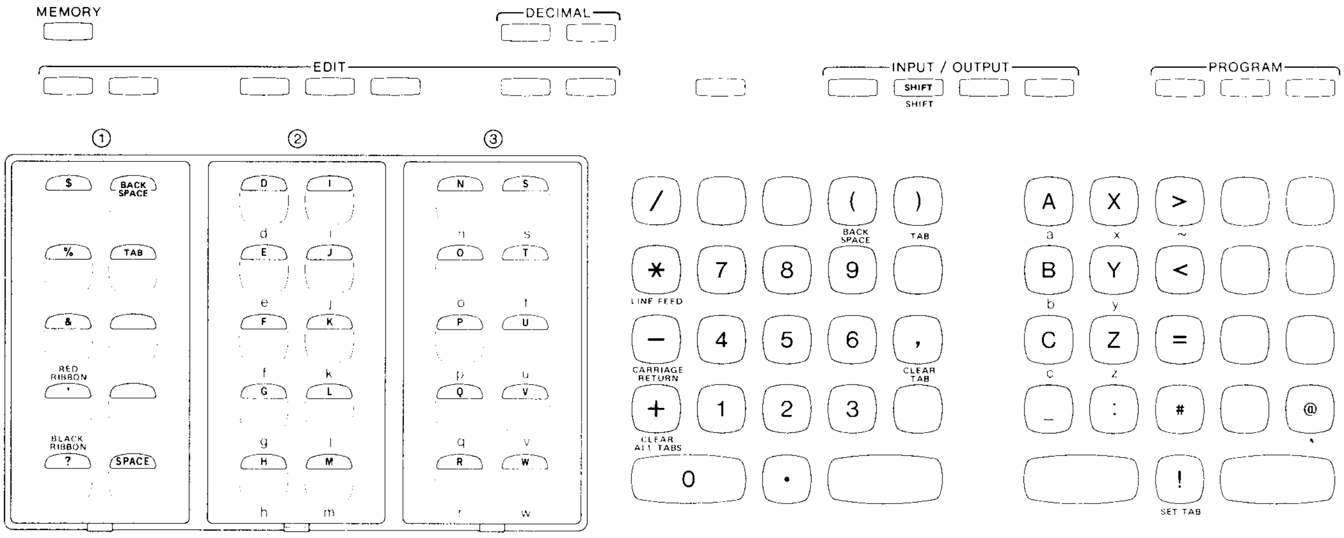


Figure 4-1. Typewriter Control Keys.

EXAMPLES

The following are typical methods of controlling discrete typewriter operations (refer to the keyboard in Figure 4-1).

1. To **clear all tabs**, execute the line:

```
FMT "⊼-",150X,"⊼+";TYP ⊢
    (shift)      (shift)
```

Result: •edit. $spec_1$ causes a carriage return
•edit. $spec_2$ causes 150 spaces
•edit. $spec_3$ causes a 'clear all tabs' operation

2. To **set a tab**, execute the line:

```
FMT 15X,"⊼!",Z;TYP ⊢
       (set tab)
```

Result: •edit. $spec_1$ causes 15 spaces
•edit. $spec_2$ causes a tab to be set
•edit. $spec_3$ suppresses the automatic CR/LF

3. To **clear the second tab**, execute the line:

```
FMT 2"⊼)","⊼,",Z;TYP ⊢
     (tab)
```

Result: •edit. $spec_1$ causes two 'tab' operations
•edit. $spec_2$ causes one tab to be cleared
•edit. $spec_3$ suppressed the automatic CR/LF

4. To **find the third tab**, execute the line:

```
FMT 3"⊼)",Z;TYP ⊢
     (tab)
```

Result: •edit. $spec_1$ causes three tab operations
•edit. $spec_2$ suppresses the automatic CR/LF

5. To do five **line feeds** without a carriage return, execute the line:

```
FMT 5"⊼*↑",Z;TYP ⊢
        (enter exp)
```

TYPING WITH FORMAT STATEMENTS

Result: •edit. $spec_1$ causes 5 carriage return operations
•edit. $spec_2$ suppresses the automatic CR/LF

If the line feed key is not followed by another key, the specified number of line feed operations will not be performed. In this case, the SQUARE ROOT ~~ENTER EXP~~ key was used since it does not program a typewriter operation.

6. To do a **carriage return** without a line feed, execute the line:

```
FMT "Ƶ-",Z;TYP ⊢
```

Result: •edit. $spec_1$ causes a carriage return operation
•edit. $spec_2$ suppresses the automatic CR/LF

7. To do five **back space** operations, execute the line:

```
FMT 5"Ɨ",Z;TYP ⊢
```

(back space)

Result: •edit. $spec_1$ causes 5 back space operations
•edit. $spec_2$ suppresses the automatic CR/LF

Chapter 5

DIGITIZER CONTROL

TABLE OF CONTENTS

NOTES

Chapter 5

DIGITIZER CONTROL

INTRODUCTION

The Model 9864A Digitizer is controlled by using the READ and WRITE statements, which are described in Chapter 2. The instructions in this chapter assume that you are familiar with all general digitizer operations, as described in the Digitizer Peripheral Manual.

If you suspect the performance of the digitizer, refer to Supplement E of the Model 20 Electrical Inspection Booklet. This supplement is supplied with the PC I Block.

THE DATA REQUEST STATEMENT

The DATA REQUEST Syntax:

RED ⟨select code⟩, ⟨register name_X⟩, ⟨register name_Y⟩

Where: X coordinate data → reg._X
Y coordinate data → reg._Y

If the digitizer is in the continuous mode when the DATA REQUEST statement is encountered, a data sample is immediately accepted and stored in the specified registers. However, if the digitizer is not in the continuous mode, the calculator will wait until either $\boxed{S}$ or $\boxed{C}$ (on the cursor) is pressed before accepting a data sample.

> **NOTE**
> The DATA REQUEST statement can be executed only from within a program; if execution is attempted from the keyboard, NOTE 11 will result.

THE STOP KEY

If STOP is pressed while the calculator is executing a DATA REQUEST statement; the operation will be terminated and the program will be halted. Also, pressing STOP while the digitizer is sending a data sample will deactivate the continuous mode.

MAXIMUM SAMPLE RATE

The maximum rate at which the calculator (when using PC I) can request and accept data samples from the digitizer is approximately 7 samples per second. Since the sample rate may be considerably slower due to program execution time, the operator must take care to move the cursor slowly; in order to obtain the maximum possible sample density. The effects of sample rate and sample density are discussed in the Digitizer Peripheral Manual.

EXAMPLE:

The following program instructs the calculator to continuously request and print data samples.

digitizer select code specified here

```
0:
RED 9,X,Y⊢
1:
PRT "Y=",Y;PRT "
X=",X;SPC 2⊢
2:
GTO 0⊢
3:
END ⊢
```

To run the program:

1. Load the program into your calculator (be sure your digitizer's select code is specified correctly in line 0).
2. Attach the Sample Data Overlay supplied with your digitizer to the digitizer platen (see Figure 5-1).

(continued)

THE DATA REQUEST STATEMENT

3. Set the origin (press $\boxed{O}$ on the cursor) approximately over point U on the overlay. Press END, RUN PROGRAM. The calculator is now waiting for a data sample from the digitizer.
4. To take continuous samples, press $\boxed{C}$. Slowly slide the cursor across the digitizing area (the corners of the digitizing area are indicated by the black dots on the platen).
5. To stop the sampling, but not halt the program, press $\boxed{C}$. Now press $\boxed{S}$ several times; the digitizer supplies one data sample each time $\boxed{S}$ is pressed.

   To stop the program, press STOP.

THE 'BEEP' STATEMENT

The BEEP Syntax:

```
        WRT ⟨select code⟩
```

The BEEP statement causes the digitizer to sound an audible tone, which lasts about one-tenth of a second. A series of these statements, when separated by DISPLAY statements, will produce a pattern of 'beeps', which can be used to signal the operator during program operation.

DOCUMENT ALIGNMENT

The following program can be used to align a document on the digitizing surface. The general procedure used to align a document is described in the Digitizer Peripheral Manual.

```
0:
RED 9,X,Y⊢
1:
IF X>.1;DSP X;
GTO 0⊢
2:
IF X≤-.1;DSP X;
GTO 0⊢
3:
IF X=0;WRT 9;
DSP X;WRT 9;GTO
0⊢
4:
WRT 9;DSP X;GTO
0⊢
5:
END ⊢
```

To run the program:

1. Load the program into your calculator (be sure your digitizer's select code is specified correctly in lines 0, 3, and 4).
2. Attach the Sample Data Overlay (or another document which is to be digitized) to the digitizing surface, as shown in Figure 5-1.
3. Place the cursor cross-hairs over point I (the upper left-hand corner of the document) and press $\boxed{O}$.
4. On the calculator: press END, RUN PROGRAM.
5. Slide the cursor to point III (the lower left-hand corner of the document) and position the cross-hairs exactly over point III; press $\boxed{C}$. (If point III is is currently within ±.1 inch of the X axis as established over point I, the digitizer will be 'beeping' slowly. Whenever point III is positioned exactly over the X axis, the digitizer will 'beep' more rapidly.) Slowly move the cursor and the overlay (together) either right or left until the display (display

DOCUMENT ALIGNMENT

equals .00) and the audible signal indicate alignment.

6. Without moving the overlay, tape the remaining three corners of the overlay (or document) to the platen. If necessary, retape the first corner.

7. Verify that the X axis of the document is in precise alignment with the platen by noting the display when the cross-hairs are positioned alternately over points I and III; return to step 3 if the X axis is not precisely aligned.

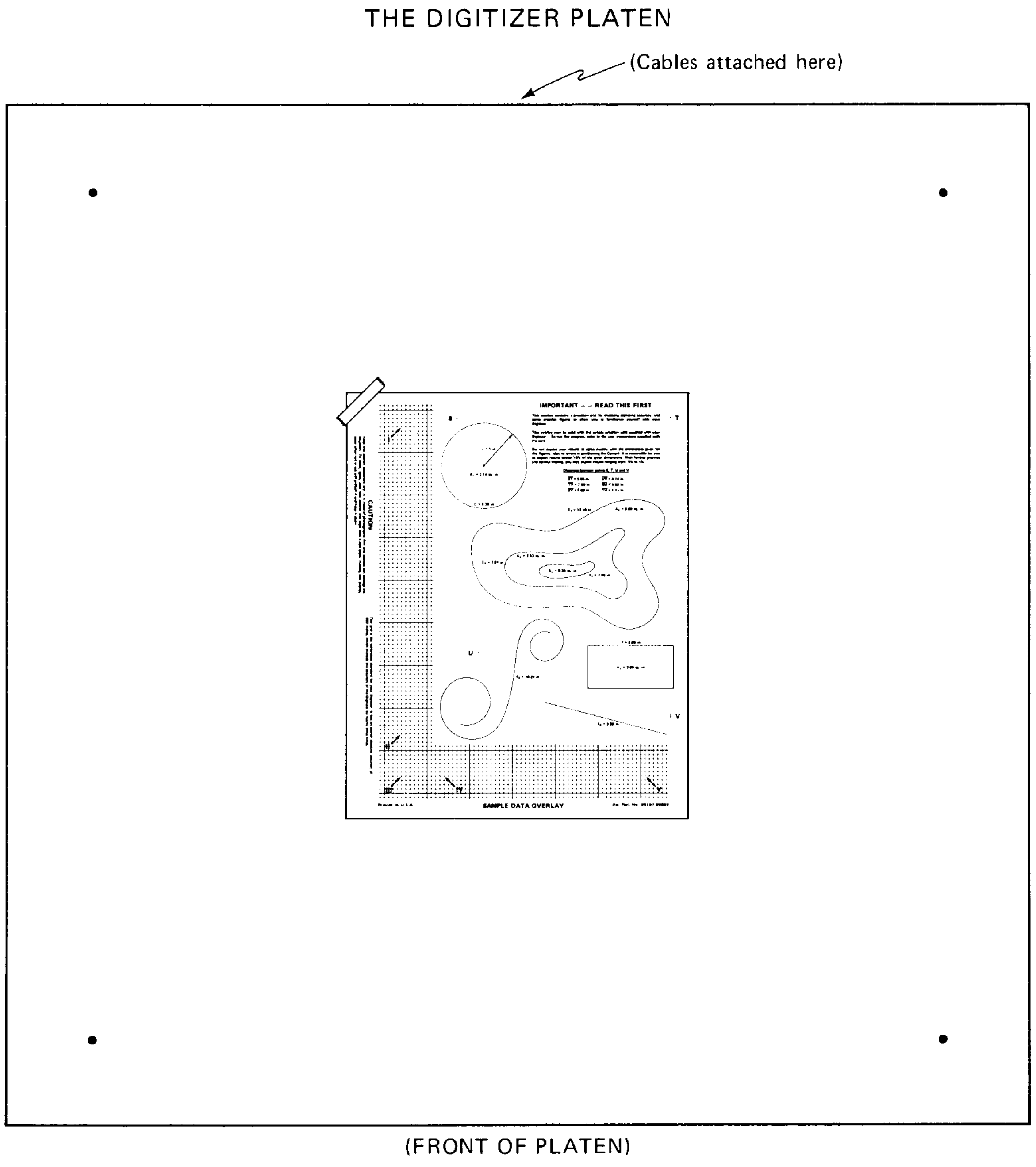


Figure 5-1. Document Alignment Procedure

NOTES

Chapter 6

TAPE READER CONTROL

TABLE OF CONTENTS

NOTES

Chapter 6

TAPE READER CONTROL

INTRODUCTION

When using the PC I Block, the Model 9863A Tape Reader can input numerical data* to the calculator. The tape reader is controlled by using the READ statement; however, the tape reader mode of operation determines the effect that data delimiters have on tape reader operation.

The instructions in this chapter assume that you are familiar with all tape reader controls, as described in the manual supplied with the tape reader (please disregard any reference to operation with the Model 10 Calculator).

The READ DATA Syntax:

RED ⟨select code⟩ ; ⟨register name$_1$⟩ ; ⟨register name$_2$⟩ ; . . . . .

or

RED ⟨select code⟩ ; ⟨list⟩

When a READ statement is executed, the first data item read is stored in the first register in the ⟨list⟩; the second data item is stored in the second register; etc. The tape reader halts when all registers in the list are filled, when an EOM character is read (see 'NORMAL MODE OPERATION'), or when the end of the tape is detected.

* The data must only be in the form of ASCII codes unless the tape reader has an optional program board. All readable ASCII characters are listed in the manual supplied with the tape reader. Also, the tape reader cannot input programs as it can with the Model 10 Calculator.

> NOTE
>
> If the end of the tape is detected while a READ statement is being executed, the program will be held at that statement (the display remains blank) until STOP, RUN PROGRAM is pressed.

As described in Chapter 2, a READ statement cannot be executed from the keyboard; an attempt to do so will cause NOTE 11 to appear.

TAPE READER SELECT CODE

When shipped from the factory, the tape reader is set to respond to select code 7. For instructions on how to set the select code to another number, see the manual supplied with the tape reader.

> NOTE
>
> Do not set the tape reader to respond to either select code 0 or more than one select code at the same time.

THE STOP INSTRUCTION

If STOP is pressed while the tape reader is reading, the operation will be immediately halted, however, FLAG 13 might be set and the current data item will probably not be correctly sent to the calculator.

NORMAL MODE OPERATION

When the Mode switch is set to NORMAL, the tape reader is fully controlled by the calculator (i.e., the tape reader program board is deactivated). When in the normal mode, the tape reader accepts the following ASCII characters for numerical data entry: digits, decimal point, +, –, and 'E' (enter exponent).

DATA ENTRY DELIMITERS

Except for the ASCII characters to be described, all characters not accepted for data entry (see the preceding paragraph) are recognized as general data delimiters. The tape reader ignores all general delimiters which precede a data item. However, after reading data, reading a general delimiter causes the tape reader to send the previously read data to the calculator. Then, the calculator either resumes execution of the READ statement or, if the ⟨list⟩ is completed, begins execution of the next program statement.

Following is a description of special delimiters usable during normal mode operation.

NORMAL MODE OPERATION

COMMA

A comma which follows a data item is treated as a general delimiter. However, a comma following any general delimiter which follows a data item (i.e., a second delimiter) will cause FLAG 13 to be set. Also, a comma which immediately precedes a data item will cause FLAG 13 to be set.

EXAMPLE A:

List of ASCII characters* on tape:

```
1ƀ200.5ƀ3ƀ4ƀ,C=5,A=6,
```

Load and run this program:

```
0:
FXD 2;0→A⊢
1:
RED 7,A⊢
2:
PRT A;SPC 2;STP
⊢
3:
END ⊢
```

Result:

```
1.00
```

Running the program successive times will cause successive data items to be read, stored, and printed. Notice that since a general delimiter and a comma follow data item '4', running the program after '4' is read will cause FLAG 13 to be is set but a data item will not be read, since the read operation will be stopped after the comma is recognized. Also, notice that all general delimiters which precede a data item are ignored.

EXAMPLE B:

Modify and run the preceding program (see the following listing). Use the same tape as shown in Example A.

```
0:
FXD 2;0→A→B→R2⊢
1:
RED 7,A,B,R2⊢
2:
PRT A,B,R2;SPC 2
;STP ⊢
3:
END ⊢
```

Result:

```
  1.00
200.50
  3.00
```

Press RUN PROGRAM again.

Result:

```
4.00
0.00
5.00
```

(FLAG 13 is set, but data is not read; thus, register B is left unchanged.)

Notice that a number of data items corresponding to the number of registers specified in the READ statement are read each time the statement is executed (except when delimiters are placed to prevent a data entry).

SLASH (/) AND LINE FEED (LF)

The tape reader ignores all data items which are enclosed in a field which begins with a slash (/) and ends with a line feed (LF). These characters perform the same operation as the Begin Deletion and End Deletion operations described in 'DATA MODE OPERATION'.

* The ƀ symbol represents a space character.

NORMAL MODE OPERATION

EXAMPLE C:

List of ASCII characters on tape:

1 2 . 8 ƀ 1 3 . 3 5 ƀ / 2 0 0 . 5 ƀ (LF) ƀ 2 5 E 5 ƀ

Run the program used in Example B.

Result:

```
     12.80
     13.35
2500000.00
```

END OF MESSAGE (EOM)

The 'EOM' ASCII character causes the tape reader to halt and a 'RUN PROGRAM' instruction to be executed.

ENTER EXPONENT (E)

The 'E' ASCII character, when followed by any of the following forms, causes the preceding data item to be raised to the power of 10 indicated:

1. ⟨data item⟩ E ⟨one or two digits⟩
2. ⟨data item⟩ E ⟨a + or − and one or two digits⟩
3. ⟨data item⟩ E ⟨a general delimiter and one or two digits⟩

The following list of data items and resulting numeric entries show the effect of the form used to specify exponentiation.

EXAMPLE D:

1. Each data item:
   E45
   1.0E45
   E+45
   Eƀ45

   Result:
```
1.0E 45
```

2. The data items ('E' is omitted):
   123+4
   123−4

   Result:
```
1.23E 06
1.23E-02
```

3. Each data item:
   10E456
   10E45.6
   10Eƀ456

   Result:
```
NOTE 11
```

In the last example, NOTE 11 appears because the exponent data consists of more than two digits or two digits and one other character.

DATA MODE OPERATION

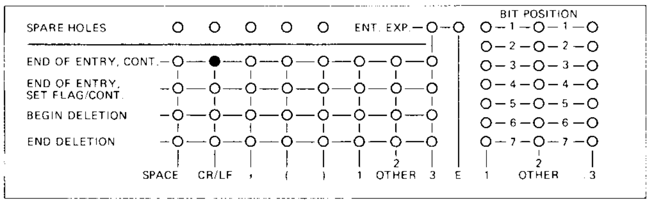
When in the DATA mode, the tape reader responds only to the ASCII characters which are specified as delimiters on the program board and accepts the following ASCII characters for data entry: digits, decimal point, +, –, and 'E' (enter exponent).

The following operations can be preset when the tape reader is in the data mode:

1. END OF ENTRY, CONTINUE: Upon reading a specified delimiter, the tape reader sends the previously read data item to the calculator. Then, the calculator either resumes execution of the READ statement or, if the ⟨list⟩ is complete, begins execution of the next program statement.
2. END OF ENTRY, SET FLAG, CONTINUE: The tape reader sends a 'SET FLAG 13' instruction to the calculator but does not make a data entry.
3. ENTER EXPONENT: Upon reading a specified delimiter, the following data (n) is entered into the calculator as 10 to the power n (10^n). For example, data written on the tape as 1.23E4 would be interpreted as 1.23E 04, and 123E4 would be interpreted as 1.23E 06.
4. BEGIN/END DELETION: Upon reading a specified delimiter, the tape reader will not transfer subsequent data to the calculator until the prespecified 'END DELETION' delimiter is read on the tape.

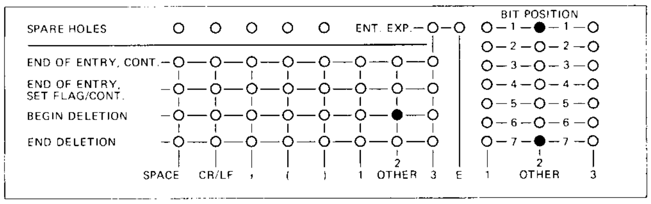
SELECTING DELIMITERS

In general, a pin inserted in one of the delimiter columns (space, CR/LF, comma, left bracket, or right bracket) will initiate the operation defined in the corresponding row when the delimiter is read. For example, if CR/LF is required to initiate END OF ENTRY, CONTINUE, the program board should be plugged as shown below.



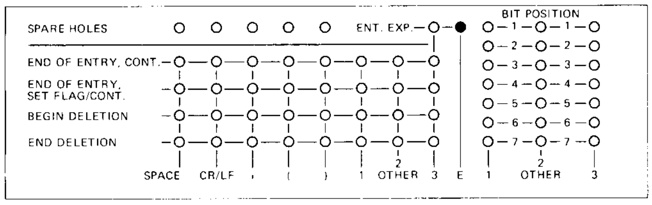
Note that only one pin can be inserted in any column (i.e., one delimiter may not initiate more than one operation).

The columns OTHER 1, OTHER 2 and OTHER 3 allow any ASCII character to be specified as a delimiter. A pin in one of these columns initiates an operation when the character set on the corresponding BIT POSITION column is read. For example, if the letter 'A' is required to initiate BEGIN DELETION (through the use of OTHER 2), pins should be installed in the program board as shown below.

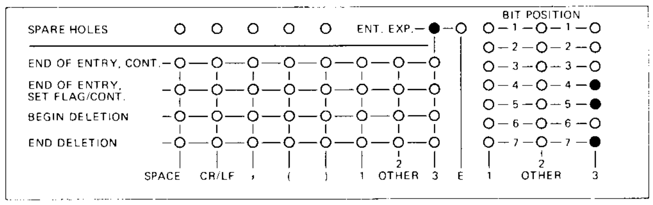


Notice that the program pins are inserted in a BIT POSITION hole when the ASCII code gives the corresponding bit as '1' (see the table of codes in the manual supplied with the tape reader).

If the ENTER EXPONENT operation is required, then either the letter 'E' can be specified as the delimiter (see the program board as shown below),



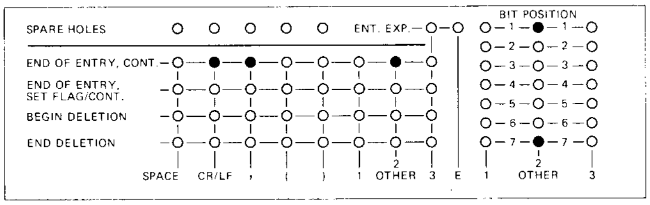
or a delimiter defined by the OTHER 3 position can be specified (as in the following example where 'X' is specified).



Note that the use of OTHER 3 to enter the exponent precludes its use for any other operation. Use of the letter 'E', however, permits OTHER 3 to initiate another operation.

DATA MODE OPERATION

More than one delimiter may be preset to initiate the same operation. For example, END OF ENTRY, CONTINUE will be initiated by CR/LF, or a comma, or the letter 'A' when the program board is set as shown below.



FIRST AND SECOND DELIMITERS

The tape reader always responds to the first delimiter which follows the data item. In addition, the tape reader will respond to the 'second' (different) delimiter which follows a data item.

The following examples show typical uses for a second delimiter.

EXAMPLE A:

List of data items on tape:

12.2,ƀ1005,,,A20.85,

↖(second delimiter)

Preset comma and CR/LF as 'END OF ENTRY, CONT' delimiters and preset the letter 'A' as 'END OF ENTRY, SET FLAG/CONT' delimiter.

Then load and run the program:

```
0:
FXD 2;0→A→B→C→R8
⊢
1:
RED 7,A,B,C,R8⊢
2:
PRT A,B,C,R8;
SPC 2;STP ⊢
3:
END ⊢
```

Result:

```
  12.20
1005.00
   0.00
  20.85
```

Notice that the tape reader responded to the second delimiter, even though the first delimiter was repeated.

EXAMPLE B:

List of data items on tape:

25.2,A$\boxed{\text{CR/LF}}$,,5E06,

(comma, CR/LF, and 'A' are still preset as delimiters.)

Run the preceding program.

Result:

```
 25.20
  0.00
506.00
```

Notice that the third and subsequent delimiters in a series are ignored.

EXAMPLE C:

List of data items on tape:

B=25.2,C=(15.2(9,X=25E5,Y=100,

Preset the comma as the 'END OF ENTRY, CONT' delimiter and 'E' as 'ENTER EXPONENT' delimiter. Then run the preceding program.

Result:

```
     25.20
     15.29
2500000.00
    100.00
```

Notice that all characters not preset as delimiters are ignored.

NOTES

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: $1 \leqslant$ s.c. $\leqslant 15$.) 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 (see Keyboard Execution of Output Statements on Page 2-6).

FOLDOUT ➡

APPENDIX

KEY MNEMONICS

TYPEWRITER CONTROL

KEY	MNEMONIC☆
TYPE	TYP*b*

GENERAL PERIPHERAL CONTROL

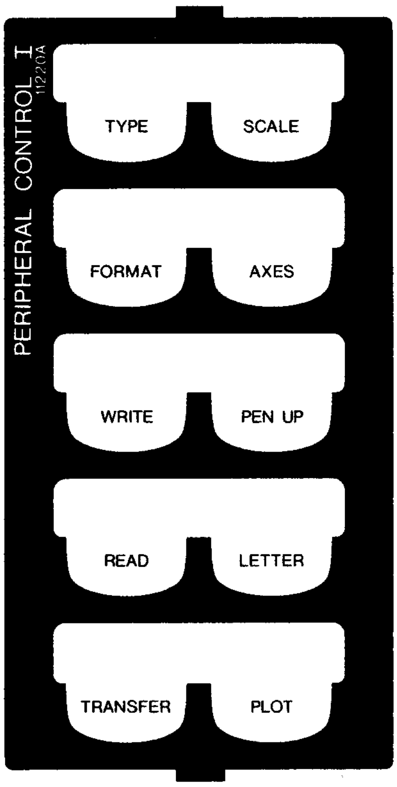
KEY	MNEMONIC☆
FORMAT	FMT*b*
WRITE	WRT*b*
READ	RED*b*
TRANSFER	TFR*b*

PLOTTER CONTROL

KEY	MNEMONIC☆
SCALE	SCL*b*
AXES	AXE*b*
PEN UP	PEN*b*
LETTER	LTR*b*
PLOT	PLT*b*

☆ *b* indicates a blank space.

KEY OVERLAY



APPENDIX

KEY MNEMONICS

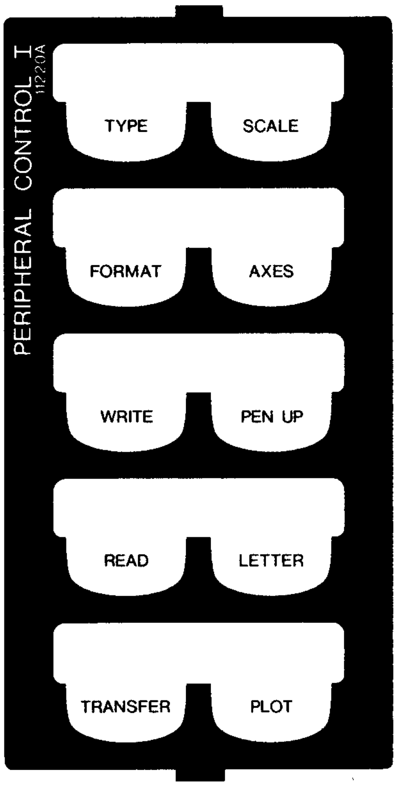
TYPEWRITER CONTROL	
KEY	MNEMONIC☆
TYPE	TYP*b*

GENERAL PERIPHERAL CONTROL	
KEY	MNEMONIC☆
FORMAT	FMT*b*
WRITE	WRT*b*
READ	RED*b*
TRANSFER	TFR*b*

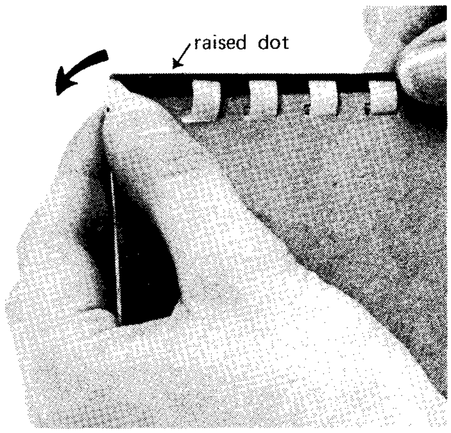
PLOTTER CONTROL	
KEY	MNEMONIC☆
SCALE	SCL*b*
AXES	AXE*b*
PEN UP	PEN*b*
LETTER	LTR*b*
PLOT	PLT*b*

☆ *b* indicates a blank space.

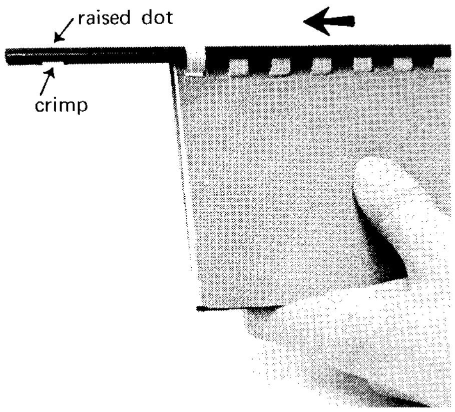
KEY OVERLAY



DISASSEMBLING THE BINDING

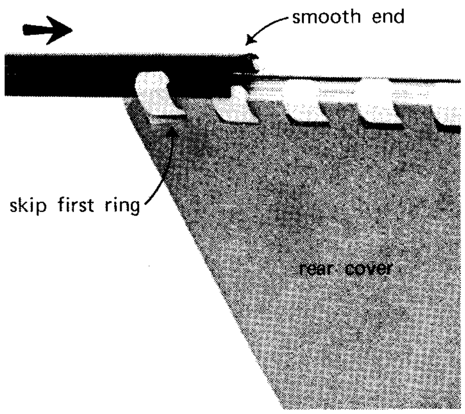


1 Unhook the end ring.

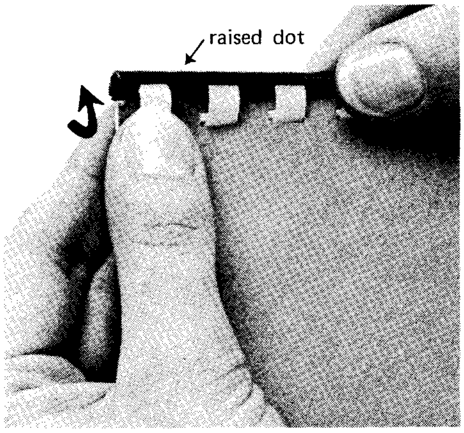


2 Slide the binding apart.

ASSEMBLING THE BINDING



3 Engage the binding, but skip the first ring.



4 Snap the ring into the groove.