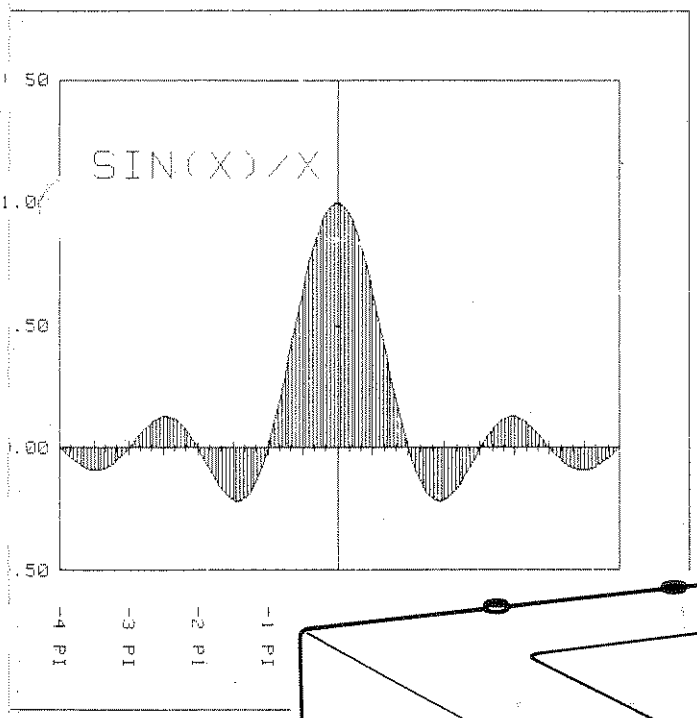
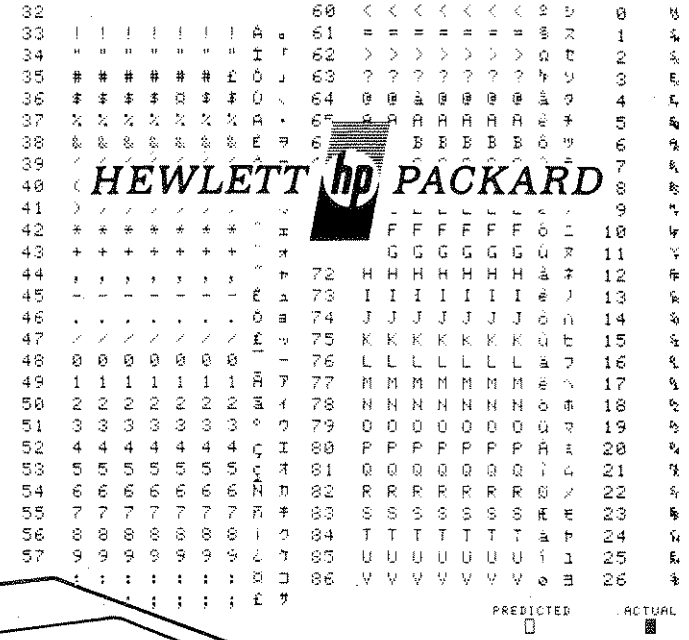
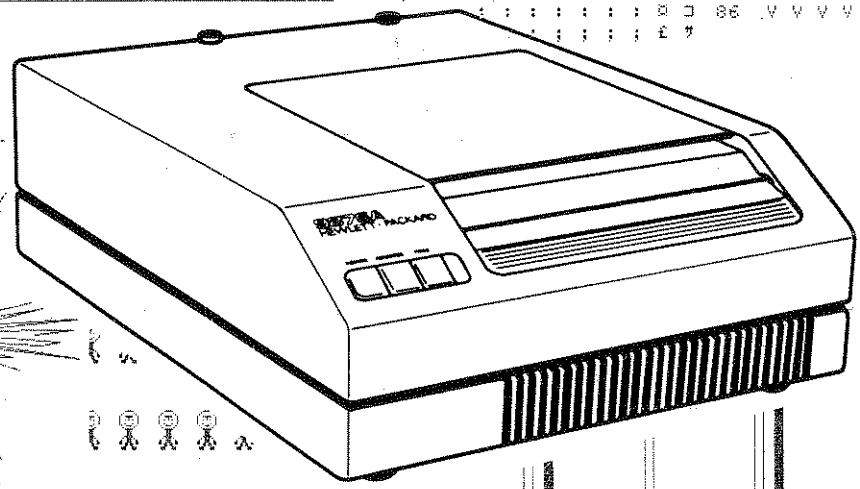
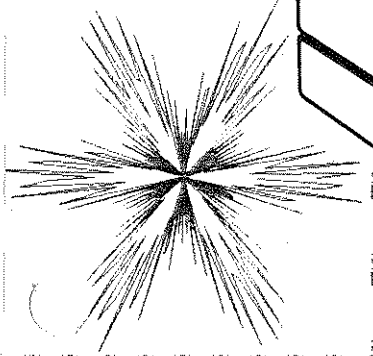
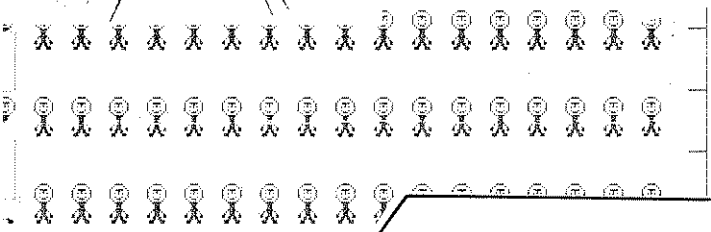
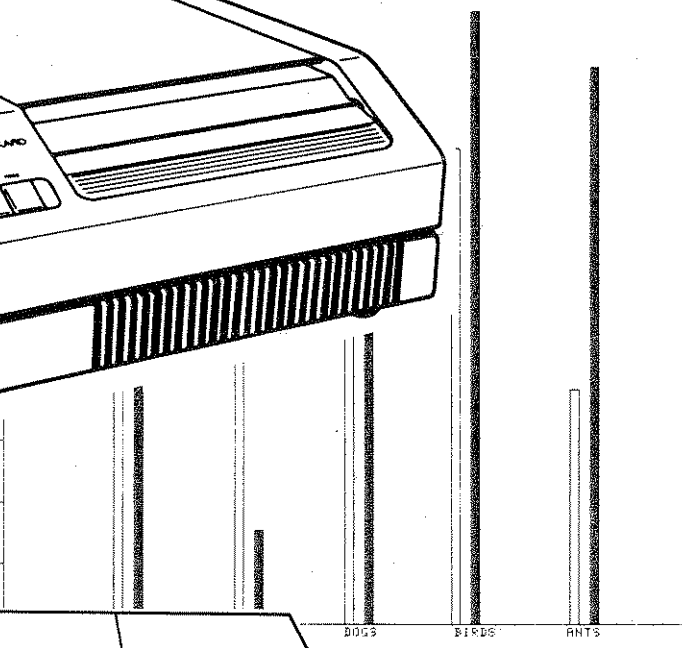


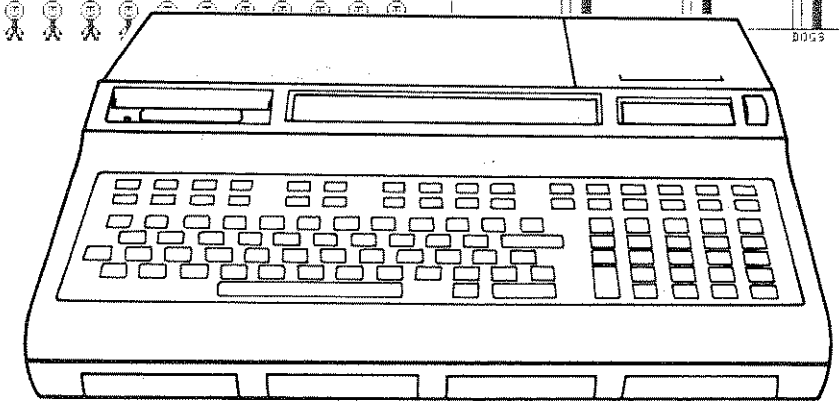












9825A

HP 9876A THERMAL GRAPHICS PRINTER

PROGRAMMING TECHNIQUES

HEWLETT hp PACKARD

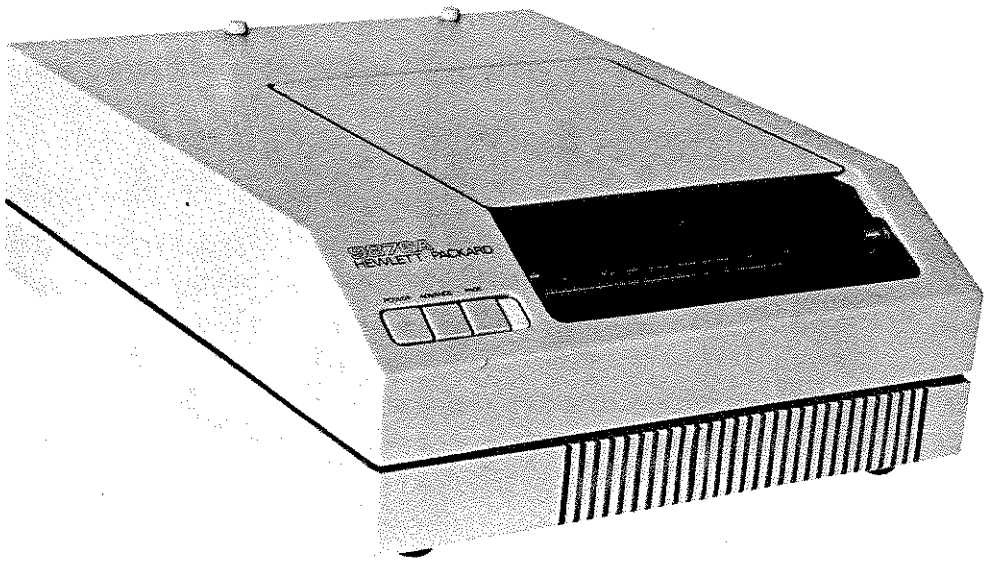
Warranty Statement

Hewlett-Packard products are warranted against defects in materials and workmanship. For Hewlett-Packard Desktop Computer Division products sold in the U.S.A. and Canada, this warranty applies for ninety (90) days from date of delivery.* Hewlett-Packard will, at its option, repair or replace equipment which proves to be defective during the warranty period. This warranty includes labor, parts, and surface travel costs, if any. Equipment returned to Hewlett-Packard for repair must be shipped freight prepaid. Repairs necessitated by misuse of the equipment, or by hardware, software, or interfacing not provided by Hewlett-Packard are not covered by this warranty.

NO OTHER WARRANTY IS EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. HEWLETT-PACKARD SHALL NOT BE LIABLE FOR CONSEQUENTIAL DAMAGES.

*For other countries, contact your local Sales and Service Office to determine warranty terms.

9876A Thermal Graphics Printer



August 1, 1979

Hewlett-Packard Desktop Computer Division
3404 East Harmony Road, Fort Collins, Colorado 80525
(For World-wide Sales and Service Offices see back of manual.)

Customer Questionnaire

There are 3 questionnaires in the back of this manual. We have included more than 1 questionnaire because we want response from those of you who use this manual only on a part time basis. If you are the sole user of this manual, we would appreciate hearing from you at different stages of familiarity with our manual.

Your answers to these questions can help us in producing better, more useful manuals. Your feedback is our only way of knowing your likes and dislikes of our product (the manuals). Please complete the questionnaires and mail them – postage is already paid in the United States.

Please feel free to contact us at any time with comments concerning the documentation of your HP products.

Thank you,

Peripheral Documentation Group

Introduction

This manual is intended to provide some insight into programming for the 9876A Thermal Graphics Printer. The first sections of the manual describe the software tools available for manipulating the printer – Production of Listings and Catalogues, Simple Printing and Plotting, Formatting, Escape Sequences, and HP-IB Operations. The remainder of the book is devoted to an analysis of the various methods of generating high resolution graphic output.

A considerable amount of attention is devoted to the topic of graphics, as it is a unique aspect of this printer and requires some new approaches to programming.

The 9876A can have any HP-IB address code from 0 to 30. For simplicity, the factory address setting of 1 is assumed in all examples throughout this manual except for specifically noted exceptions. The printer is also assumed to be connected to the system through an HP-IB card with a select code of 7. If your system requires the printer to be located at some other address or select code, or if you are not using an HP-IB system, consult Appendix A of this manual and The Printer Peripheral Manual (09876-90000).

The reader is assumed to have a working knowledge of HPL throughout this manual.

Manual Overview

Basic Printing Operations **Section I**

Simple printing operations include listings and catalogues, using WRITE and WRITE BINARY statements, and basic character plotting. This section deals with printing operations that reside entirely in the computer, as opposed to the special capabilities of the 9876A Thermal Graphics Printer.

Listings and Catalogues **Chapter 1**

Listing programs to a 9876A is discussed. Catalogues of Mass Storage Units are also discussed.

Basic Printing **Chapter 2**

The concept of a character field, and the default forms for computer output of data are introduced. How various types of data are dealt with in WRITE (`wrt`) and WRITE BINARY (`wtb`) statements are discussed.

Character Plots of F(X) with a Vertical X-axis **Chapter 3**

The necessary methods of scaling the results of the evaluation of a function are presented as well as the procedures for labeling the graph. The algorithm for generating the graph a line at a time is discussed.

Advanced Printing **Section II**

This section will cover two methods for exercising precise control over the printer. The first is formatting and involves the use of WRITE (`wrt`) and FORMAT (`fmt`) statements. The formatting functions are resident in the computer. The second technique uses Escape Sequences and Control Characters to control the printer. Format statements are used to control the flow of data, while Escape Sequence and Control Characters are imbedded in the data stream.

Table of Contents

SECTION I

Simple printing operations include listings and catalogues, using wrt and wtb statements, and basic character plotting. This section deals with printing using functions that reside entirely in the computer, as opposed to special capabilities of the 9876A Thermal Graphic Printer.

Listings and Catalogues — Chapter 1

Listing programs to a 9876A is discussed. Catalogues of disk drives are also discussed.

Basic Printing — Chapter 2

The concept of a character field, and the default forms for computer output of data are introduced and the print list is presented. How various types of data are dealt within wrt and wtb statements is discussed.

Character Plots of F(X) with a Vertical X-axis — Chapter 3

The necessary methods of scaling the results of the evaluation of a function are presented as well as the procedures for labeling the graph. The algorithm for generating the graph a line at a time is discussed.

Basic Printing Operations

CHAPTER 1

Introduction

Listings and Mass Storage catalogues are two of the most common printouts in small computer systems. These are generated using the `list` and `cat` statements as described below.

Listings

Listing to An External Printer

Executing `list #701` will direct a listing of the program currently in memory to a 9876A with factory address settings (interface card select code = 7, printer address = 1).

Partial Listings

Begining and ending lines can be added to `list #` to produce a listing of specific program lines.

Executing `list #701,5` will direct a partial listing of a program (from line 5 to the last line of the program) to a 9876A with factory address settings (interface card select code = 7, printer address = 1).

Eliminating Spacing and Checksum

The 9825 normally outputs 3 CR-LFs before and after a program listing, along with a checksum immediately after the listing. The checksum and CR-LFs can be suppressed by suffixing a decimal point and a non zero digit to the select code.

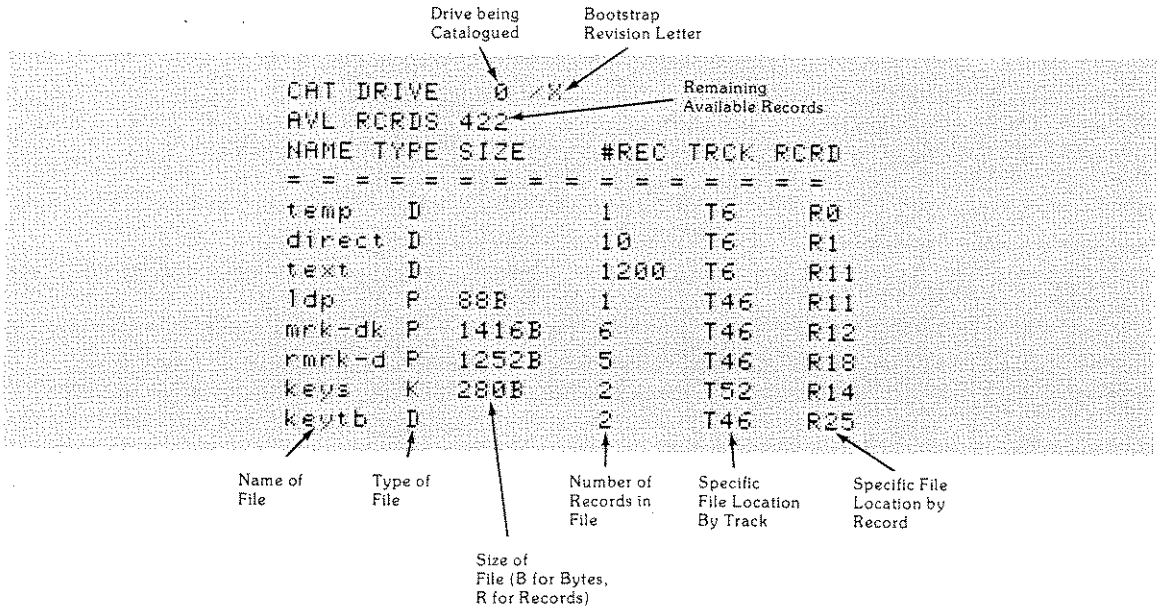
Example: `list #701.1`

Catalogues

If your system includes a 9885 Flexible Disk Drive, the `cat` statement may be used to direct catalogues of a 9885 Flexible Disk Drive to a 9876A printer if the select code of the printer is suffixed to the statement.

Executing `cat 701`

will produce a printout similar to this one:



CHAPTER 2

Introduction

The most basic forms of printing are done with unformatted write (wrt) and write binary (wtb) statements.

The Write Statement

The write statement (wrt) outputs the characters, signs, and decimal point of each item in its list to the addressed peripheral. Each item in the list can be a numeric expression, text, or a string name (when the String ROM is in use). Unless a format statement is in effect, the free-field format is used.

The Free-Field Output Format

The free-field output format is automatically set whenever the calculator is switched on or RESET is pressed. Free-field is also selected whenever run or erase all commands are executed. Each write statement references the free-field format until an appropriate format statement is referenced; then the specifications in the format statement override free-field.

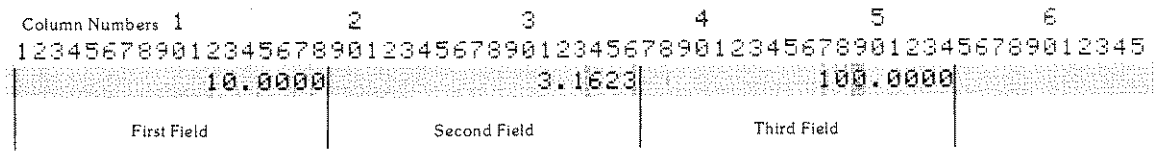
Free-Field Numerics

The free-field format causes each numeric expression to be output, right justified, in an 18 character field. A CR-LF is given after every fourth expression output and after the last expression is output. The form in which expressions appear is determined by the current fixed (fxd) setting.

Executing

```
6: fxd 4;10→I
7: wrt 701,I,√I,I↑2
```

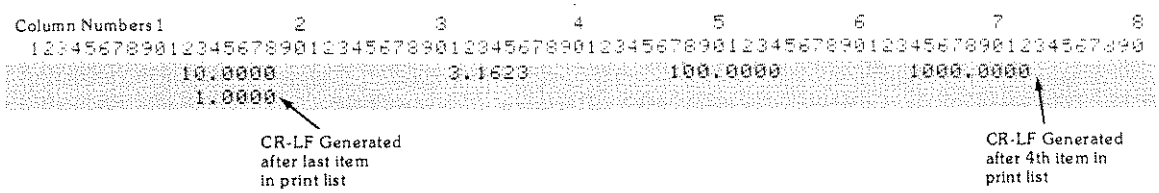
Produces



while executing

```
13: fxd 4;10→I
14: wrt 701,I,√I,I↑2,I↑3,log(I)
```

Produces



Free-Field Strings

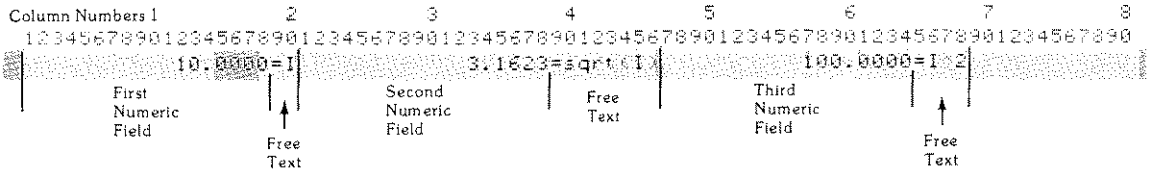
Characters within quotes (literal strings) and string variables are output as "free text", which means that the 18-character fields are not used. The character strings are output with no leading or trailing blanks, which facilitates suffixing labels to data, but requires care in constructing table headings and other text which depends on spacing. The "free text" is output between the 18 character numeric fields if text and numeric fields are intermixed.

Example

Executing

```
21: fxd 4;10→I
22: wrt 701,I,"=I",√I,"=sqrt(I)",I↑2,"=I↑2"
```

Produces



The Write Binary Statement

The Write Binary statement (wtb) sends the binary equivalent of the expressions in the expression list. This is very useful for sending control characters and escape sequences. No CR-LF is sent unless included in the expression list.

Example

Executing

```
29: wtb 701,"message without CR-LF    "
30: wtb 701,"message with CR-LF",13,10
31: wtb 701,"another message with CR-LF",13,10
```

Produces

```
Column Numbers 1          2          3          4          5          6
12345678901234567890123456789012345678901234567890123456789012345
message without CR-LF   message with CR-LF
another message with CR-LF
```

Simple Tables

Where numerous sets of related data are of interest, tabular presentation of the data is useful.

Simple tables with four or fewer columns can be produced if some attention is given to spacing between the column headings. While the numerics are output left justified in 18-character fields, the strings are output with no leading or trailing spaces. Reconciling the difference in outputs is relatively simple, as the following table of I, I^2, $\sqrt{I}$, and Log_{10} of I demonstrates. This program may be found in File 2 on Track 1 of the 9876-9825 Utility Tape.

NOTE

This program requires the Advanced Programming ROM as well as the General I/O ROM.

```
0: "simple table":
1:
2: "generate heading":
3:
4: "label table":wtb 701,"                I  |                  I↑2|"
5: wtb 701,"          SQRT(I)|          LOG(I)",13
6:
7: "break labels from table":for I=1 to 75;wtb 701,"_";next I
8: wtb 701,13,10
9:
10: "print table":
11:
12: for I=1 to 10
13: wrt 701,I,"|",I↑2,"|",rI,"|",log(I)
14: next I
*23966
```

I	I^2	SQRT(I)	LOG(I)
1.0000	1.0000	1.0000	0.0000
2.0000	4.0000	1.4142	0.3010
3.0000	9.0000	1.7321	0.4771
4.0000	16.0000	2.0000	0.6021
5.0000	25.0000	2.2361	0.6990
6.0000	36.0000	2.4495	0.7782
7.0000	49.0000	2.6458	0.8451
8.0000	64.0000	2.8284	0.9031
9.0000	81.0000	3.0000	0.9542
10.0000	100.0000	3.1623	1.0000

CHAPTER 3

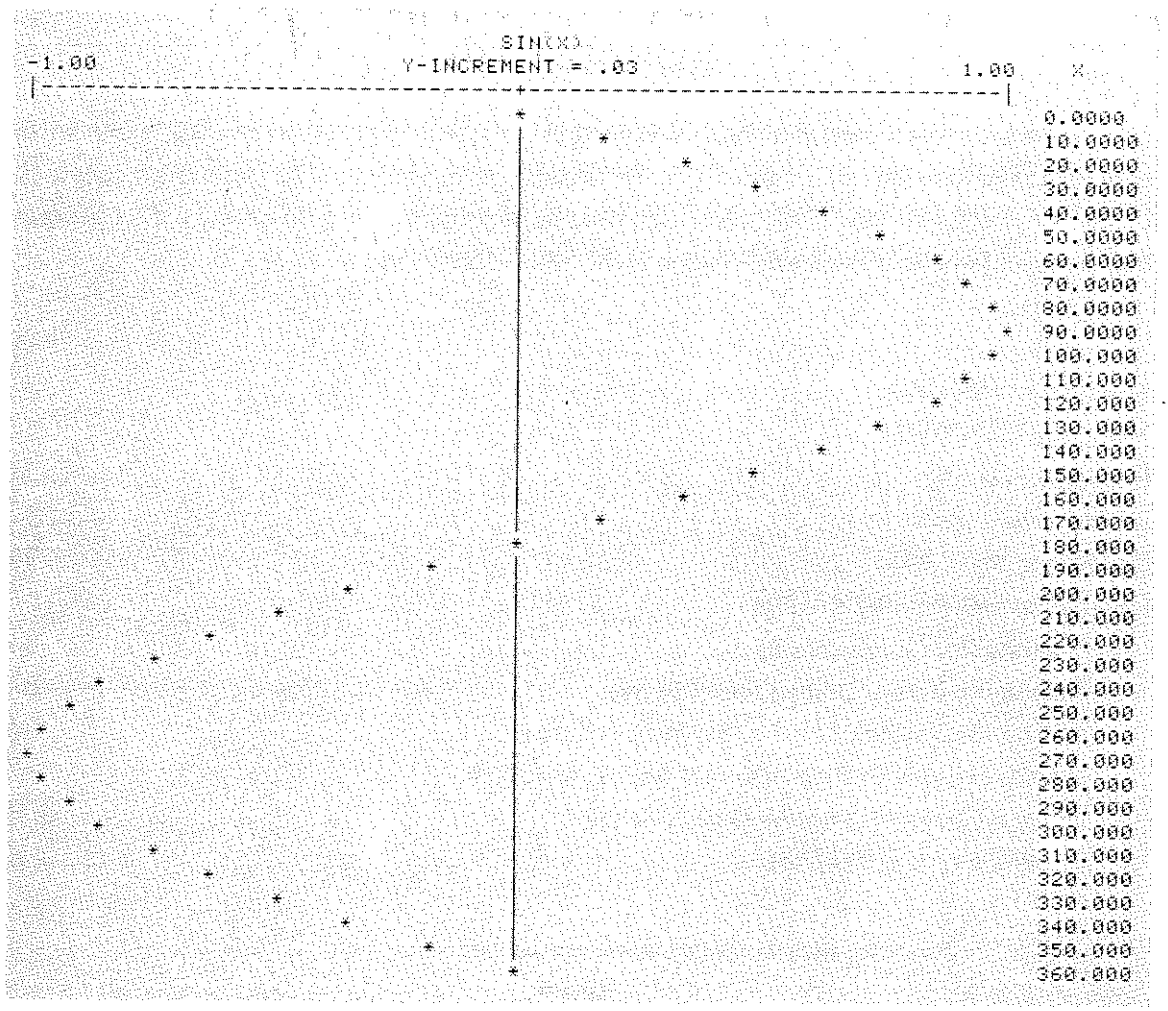
Character Plots of F(X) with a Vertical X-Axis

Introduction

Low resolution (character) plotting is simple if the X-axis is allowed to run the length of the paper (like the time axis on a strip chart). The maximum number of units along the Y-axis is 80. Using fewer units for the Y-axis allows labeling of increments along the X-axis. This program may be found in File 3 on Track 1 of the 9876-9825 Utility Tape.

NOTE

This program requires the String and Advanced Programming ROMs, as well as the General I/O ROM.


SIN(X)
-1.00
Y-INCREMENT = .03
1.00
X
0.0000
10.0000
20.0000
30.0000
40.0000
50.0000
60.0000
70.0000
80.0000
90.0000
100.000
110.000
120.000
130.000
140.000
150.000
160.000
170.000
180.000
190.000
200.000
210.000
220.000
230.000
240.000
250.000
260.000
270.000
280.000
290.000
300.000
310.000
320.000
330.000
340.000
350.000
360.000

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Scaling and Offsetting

Before generating a plot, it is necessary to determine the scale factor and offset value for the Y-axis. The most obvious formula is

$$\text{Scale Factor} = \frac{\text{Number of Y Units}}{\text{Range}}$$

However, this formula produces clipped plots. Therefore

$$\text{Scale Factor} = \frac{\text{Number of Y Units} - 2}{\text{Range}}$$

should be used to plot the function without clipping. Additionally, using an odd number of units allows a representation of the X-axis. By specifying less than the full paper width for the plot, we can allow labels for the X-axis, and avoid the clipping problem. For this plot we will select a Y axis of 71 units, and for our plot of a SIN function with a range of 2, we will use a scale factor of 34.5. An offset of 36 will center the plot on the page (necessary since there are no negative positions in a character string).

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Filling the Clear String

A string of blanks is necessary to clear all characters out of the working output string before each line is constructed. An assignment within a for-next loop is used to fill the 80 character string with blanks.

```
9: for I=1 to 80;" "→B$[I];next I
```

Printing a Heading

The heading is centered above the plot by placing it in a format statement which is then accessed by a `wrt` statement.

```
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
```

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Labeling the Y-Axis

The minimum and maximum Y excursion values are printed above the Y-axis along with the incremental Y-value. Once again, the format statement is found to be the simplest method for positioning the labels.

```
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
```

Generating the Y-Axis

The Y-axis is represented primarily by minus signs (−). Vertical bars (|) are used for maximum and minimum Y-values and a plus sign (+) for the intersection of the X-axis on the Y-axis. This structure is put in a format statement and accessed with a `wrt` statement.

```
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
```

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Calculating and Printing the Plot

Once the Y-axis is generated, we can move on to the actual plot. A `for-next` loop is established to control the value of X throughout its domain. The loop is set up to run from the minimum X-value to the maximum X-value by an appropriate increment (selecting the increment to produce 35 to 40 lines produces a two to one (approximately) aspect ratio for the plot).

Within the `for-next` loop, 6 operations are performed:

1. Clearing the output string.
2. Adding the X-axis to the output string.
3. Adding the X-value label to the output string.
4. Calculating F(X) and scale and offset.
5. Filling a character representing F(X) into the output string.
6. Printing a line of the plot.

The first five operations are required to construct the output string, which is printed in the sixth step.

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Clearing the Output String

The string variable B$ is filled with blanks in line 9 of the program. A simple assignment of B$ into A$ clears A$ and sets it to 80 characters in length (to allow assignments to any position in the string). This provides a clean work space for building the output line.

```
25: "clear the string":B$→A$
```

Adding the X-Axis to the Output String

A vertical bar is added into the string at the center of the plot's Y-values to represent the X-axis. This character is deposited first, so that the character used for plotting can replace it at zero crossings. If you prefer to have the X-axis overwrite the plotted waveform, this assignment should be moved to just before the output string is printed (insert before line 29).

```
26: "add X-axis":"|"→A$[36,36]
```

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Adding the X-value Label to the Output String

The X-value (as determined by the controlling `for-next` loop) is converted into a string and inserted into the last 8 characters of the output string. This provides labels along the X-axis.

```
27: "add X value":str(X)→A$[73,80]
```

Calculating and Scaling F(X) and Adding an Offset

The function (in this case SIN(X)) is evaluated at `X` (as determined by the controlling `for-next` loop) and multiplied by the scale factor. The result is rounded to 0 decimal places using a `prnd` function. This is added to the offset value to produce a Y value normalized for plotting.

```
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
```

F(X) — Scale Factor — Offset

```
0: "program to plot simple functions":
1: "A$ is output string":
2: "B$ is clear string":
3: "Y is scaled value of F(X)":
4:
5: dim A$[80],B$[80]
6: "set degree mode":deg
7:
8: "fill clear string with spaces":
9: for I=1 to 80;" "→B$[I];next I
10:
11: "print heading":
12: fmt 1,34x,"SIN(X)"
13: wrt 701.1
14:
15: "label Y-axis":
16: fmt 2,"-1.00",22x,"Y-INCREMENT = .03",23x,"1.00",4x,"X"
17: wrt 701.2
18:
19: "generate Y-axis":
20: fmt 3,"|",34"-","+",34"-","|"
21: wrt 701.3
22:
23: "calculate & print plot":
24: for X=0 to 360 by 10
25: "clear the string":B$→A$
26: "add X-axis":"|"→A$[36,36]
27: "add X value":str(X)→A$[73,80]
28: "calculate f(x) & scale & offset":prnd(sin(X)*34.5,0)+36→Y
29: "fill point into output string":"*"→A$[Y,Y]
30: "print a line of the plot":wrt 701,A$
31: next X
*4368
```

Filling a Character into the Output String

A character is now filled into the output string at the position representing its Y value as determined above. A string assignment accomplishes this.

```
29: "fill point into output string":"*"→A$[Y,Y]
```

Printing a Line of the Plot

The output string (A$) is now printed, using a wrt statement.

```
30: "print a line of the plot":wrt 701,A$
```

SECTION II

This section will cover two methods for exercising precise control over the printer. The first is formatting and involves the use of `wrt` and `fmt` statements. The formatting functions are resident in the computer. The second technique uses Escape Sequences and Control Characters to control the printer. Format statements are used to control the flow of data, while Escape Sequences and Control Characters are imbedded in the data stream. The HP-IB characteristics of the computer are also discussed.

Formatting — Chapter 4

The `wrt` and `fmt` statements and the format string are discussed. The various character string and numeric specifiers are presented, along with delimiters and various carriage control symbols.

Control Codes — Chapter 5

ASCII control codes and the machine dependent escape character instruction sequences are discussed. Access to the various control features from HPL is covered.

HP-IB Printer Control — Chapter 6

The various features of the printer which are available due to its being an HP-IB device are covered, as well as the methods of dealing with the HP-IB system from HPL.

Advanced Printing

CHAPTER 4

Introduction

Formatted output employs two basic mechanisms. The first is a controlled `wrt` statement, generated by suffixing a decimal point followed by a format reference number to the select code in the `wrt` statement. The second is the format (`fmt`) statement, which defines the pattern to be used in outputting the expression list in the `wrt` statement referencing it.

The Write Statement

The write statement (wrt) outputs the characters, signs, and the decimal point of each item in the expression list to the address specified by the select code. Each item in the list can be a numeric expression, text, or a string name (when the String ROM is in use). The value of each item is output in a free-field format unless a format statement is in effect.

Example

Executing

```
4: fxd 4;10→I
5: wrt 701,I,"=I",√I,"=sqrt(I)",I↑2,"=I↑2"
```

Produces

```
Column Numbers 1         2         3         4         5         6         7         8
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
           10.0000=I          3.1623=sqrt(I)          100.0000=I^2
```

Referencing Formats

Each format statement (fmt) has a number which it can be referenced with by successive write statements (wrt). A write statement references a format statement by suffixing a decimal point followed by the format reference number to the select code in the write statement.

Example

Executing

```
13: fmt 1,f9.2
14: fmt 2,f9.3
15: fmt 3,f9.4
16: fmt 4
17:
18: wrt 701,321.1234567
19: wrt 701.1,321.123456
```

Produces

```
 Column Numbers 1         2         3         4         5         6
 123456789012345678901234567890123456789012345678901234567890123456789012345
fmt 1 ──→321.12
fmt 2 ──→321.123
fmt 3→321.1235
```

If format reference number is not included in a write statement, it references the 0 format, which is set to free-field whenever the calculator is reset (power on, RESET, RUN, or ERASE). The 0 format reference number **can** be reassigned however, and if a format will be used more often than free-field in your program, a small code savings is possible by reassigning the 0 format (see Assigning Format Reference Numbers.

NOTE

If you use the 0 format in your program, be sure to reassign free field to it afterwards.

The Format Statement

The format statement provides flexible and precise control of write statements. A format statement must be executed before the write statement referencing it. The format statement provides a list of specifications for use by the write statement referencing it.

Assigning Format Reference Numbers

A format reference number is used to allow multiple write statements to access the same format statement. Each format number can be an integer from 0 to 9; if not specified, 0 is assumed. Whenever the calculator is reset (power on, RESET, RUN, or ERASE) format number 0 is automatically assigned to specify free-field. A format statement with a format number and no data format specifiers will assign the free-field format to that reference number.

Example

Executing

```
13: fmt 1,f9.2
14: fmt 2,f9.3
15: fmt 3,f9.4
16: fmt 4
17:
18: wrt 701,321.1234567
19: wrt 701.1,321.123456
20: wrt 701.2,321.1234567
21: wrt 701.3,321.1234567
22: wrt 701.4,321.1234567
```

Produces

```
Column Numbers 1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
fmt 0 ------------------>321.1235
fmt 1 ---->321.12
fmt 2 ---->321.123
fmt 3 ->321.1235
fmt 4 ------------------>321.1235
```

If several format statements are executed with the same format number, the most recently executed format is invoked by the reference number.

Example

```
30: fmt 1,f9.2
31: wrt 701.1,321.1234567
32:
33: fmt 1,f9.3
34: wrt 701.1,321.1234567
35:
36: fmt 1,f9.4
37: wrt 701.1,321.1234567
```

Produces

```
 Column Numbers 1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
   321.12
  321.123
 321.1235
```

Data Output Specifications

Data specs determine the form in which data is output. Most data specs determine whether a number is output in fixed point or floating point, the number of digits to the right of the decimal point, and the character field width in which the number appears.

These data specs are available:

[r]fw.d	Specifies fixed-point format.
[r]ew.d	Specifies exponential (floating-point) format.
[r]fzw.d	Specifies fixed-point format with leading zeros in each field. Negative numbers are not allowed with this format.
[r]b	Specifies binary output single characters.
[r]cw	Specifies a character field-width for either text or a string variable (String ROM).

- r is an optional repeat factor (see examples). If r is omitted, 1 is assumed.
- w indicates total field width (in characters). If w is omitted, leading spaces are deleted from the field.
- d indicates the number of digits to the right of the decimal point. If d is omitted, the current fixed or float setting is used.
- w, d and r parameters must be positive integer constants.

For example, the data spec f8.2 specifies a fixed-point number with two digits to the right of the decimal point. The number appears (right-justified) in an eight-character field. If d is 0, the decimal point is not output. A number output under a data spec is always rounded according to the number of decimal places specified. The free-field format is equivalent to 4f18.

Some guidelines should be observed in selecting w and d. The minus sign, decimal point, and exponent are part of each number and must fit in the field width specified by w. For floating point outputs, w should be greater than or equal to d+7. If the calculator cannot output the data in the field width available, the field is filled with $s.

Example

```
45: fmt 1,f9.2
46: wrt 701.1,654321.12345
47: wrt 701.1,-654321.12345
```

Produces

```
 Column Numbers 1         2         3         4         5         6
123456789012345678901234567890123456789012345678901234567890123456789012345
654321.12
-$$$$$$$$
```

Text Character Fields

The c data spec specifies a fixed character field for coresponding text or a string variable in the write statement. The characters are output right-justified in the field. If the field is too small, it's filled with $s.

In the example below, the first line of text fills the field; the second line is too short; the last line is too long.

```
54: fmt 1,c10,f5.1
55: wrt 701.1,"Function =",321.123
56: wrt 701.1,"Phase =",321.123
57: wrt 701.1,"Average Value =",321.123
```

```
Column Numbers 1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
Function =321.1
   Phase =321.1
##########321.1
```

Output Edit Specifications

These edit specs are used to control the placement of output data and to output character strings:

[r] x	Outputs a blank character space.
[r] /	Outputs a CR/LF.
[r] "TEXT"	Outputs the ASCII characters within quotes.
z	Suppresses the automatic CR/LF output after each write statement.

Any combination of specs can appear in the same format statement; each spec must be separated by a comma. Most of the specs can be duplicated r number of times by using the repeat factor.

This example causes the f9.2 format to be referenced twice, followed by 10 spaces, and a f10.1 format.

```
64: fmt 1,2f9.2,10x,f10.1
65: wrt 701.1,321.123,321.123,321.123
```

```
Column Numbers 1         2         3         4         5         6
12345678901234567890123456789012345678901234567890123456789012345
  321.12   321.12                    321.1
```

Tables

Tabular presentation of data can be useful where numerous sets of related data are of interest.

The following table of trigonometric values demonstrates how to build tables using formatting. This program may be found in File 4 on Track 1 of the 9876-9825 Utility Tape.

```
0: "Program to generate Trigonometric tables.":
1:
2: "initialize":0→A
3: "allow for tan(90)":sfg 14
4:
5: "space from top of page":fmt 1,5/;wrt 701.1
6:
7: "label  table":fmt 2,30x,"Trigonometric Table",2/;wrt 701.2
8:
9: "label columns":
10: fmt 3,6x,"Degrees (Radians)",12x,"Sine",10x,"Cosine",12x,"Tangent",/
11: wrt 701.3
12:
13: "body of table":
14: "format for table":fmt 3,9x,f3.0," (",f4.2,")",15x,f6.3,9x,f6.3,10x,e10.3
15:
16: for X=0 to 180 by 10
17: wrt 701.3,X,πX/180,sin(X),cos(X),tan(X)
18: "make spaces every third entry":if (A+1→A)=3;0→A;wrt 701
19: next X
20:
21: "space from bottom of table":wrt 701.1
22:
23: stp
*29745
```

```
                    Trigonometric Table

   Degrees (Radians)          Sine          Cosine          Tangent

         0 (0.00)             0.000          1.000         0.000E 00
        10 (0.17)             0.174          0.985         1.763E-01
        20 (0.35)             0.342          0.940         3.640E-01

        30 (0.52)             0.500          0.866         5.774E-01
        40 (0.70)             0.643          0.766         8.391E-01
        50 (0.87)             0.766          0.643         1.192E 00

        60 (1.05)             0.866          0.500         1.732E 00
        70 (1.22)             0.940          0.342         2.747E 00
        80 (1.40)             0.985          0.174         5.671E 00

        90 (1.57)             1.000          0.000         9.999E 99
       100 (1.75)             0.985         -0.174        -5.671E 00
       110 (1.92)             0.940         -0.342        -2.747E 00

       120 (2.09)             0.866         -0.500        -1.732E 00
       130 (2.27)             0.766         -0.643        -1.192E 00
       140 (2.44)             0.643         -0.766        -8.391E-01

       150 (2.62)             0.500         -0.866        -5.774E-01
       160 (2.79)             0.342         -0.940        -3.640E-01
       170 (2.97)             0.174         -0.985        -1.763E-01

       180 (3.14)             0.000         -1.000         0.000E 00
```

CHAPTER 5

Introduction

The 9876A Thermal Graphics Printer has certain capabilities built into it which are accessed through control codes embedded in the data stream. These codes may be ASCII control characters (the printer recognizes eight of them) or escape code sequences which consist of an Escape control character (`char(27)` or `wtb 701,27`) followed by an identifier and sometimes by a set of parameters.

Viewing the Control Codes

Executing

```
wtb 701,27,"Y"
```

will allow the viewing of the control characters and escape sequences that are sent to the printer. In this mode, the printer ignores all control characters except ^M and ESC Z (which terminates the mode) and treats them as if they were data. Control characters (including ^M and ESC Z) are loaded into the buffer and printed. The printer continues in this mode until

```
wtb 701,27,"Z"
```

is executed. For a complete explanation of this transparent mode of operation, see page 5-8.

This "transparent" mode is useful for debugging programs that send control sequences as it allows the entire data stream to be viewed. This can be very useful for determining what the printer is actually being told to do (as opposed to what you **think** you are telling it to do).

Control Characters

The 9876A Graphics Printer recognizes eight ASCII control characters.

The control characters can be sent to the printer using `wtb` statements, `b` format specifiers in formatted `wrt` statements or `char` functions in `wrt` statements (the `char` function resides in the String ROM). The `char` function and `wtb` statement expressions for the various control characters can be found below, along with a page reference to the detailed description of the operation of each control character.

Title	`char` Expression	`wtb` Access	Page
Carriage Return (^M)	`char(13)`	`wtb 701,13`	5-4
Linefeed (^J)	`char(10)`	`wtb 701,10`	5-4
Formfeed (^L)	`char(12)`	`wtb 701,12`	5-4
Backspace (^H)	`char(8)`	`wtb 701,8`	5-4
Horizontal Tab (^I)	`char(9)`	`wtb 701,9`	5-4
Shift Out (^N)	`char(14)`	`wtb 701,14`	5-4
Shift In (^O)	`char(15)`	`wtb 701,15`	5-5
Escape (ESC)	`char(27)`	`wtb 701,27`	5-5

Carriage Return ␍	wtb 701,13	char(13)

Upon receipt of any character after a carriage return the printer prints the line of characters currently in the printer's buffer and resets the character position to the left margin. The paper is not advanced unless the next character received is a ␊ and the next line of characters printed will be on the same line on the paper if no ␊ is received.

Linefeed ␊	wtb 701,10	char(10)

The linefeed prints the line of characters currently in the printer's buffer and advances the paper one line. The character position remains at its current location.

Formfeed ␌	wtb 701,12	char(12)

The formfeed advances the paper to the top margin of the next form. Formfeed does not affect the character position or print the buffer.

Backspace ␈	wtb 701,8	char(8)

Each backspace received will set the character position one space towards the left margin. The receipt of a ␈ prints the buffer unless the previous character received was a ␈. This causes the printer to overstrike characters rather than replace them when they are backspaced over. The ␈ character is ignored at the left margin. For underlining individual characters, the ␈ requires less code than the auto-underline Escape Sequence.

Horizontal Tab ␉	wtb 701,9	char(9)

The horizontal tab sets the printer to the next preset TAB position that is to the right of the current character position. If a TAB is not set between the current character position and the right margin, a carriage return and linefeed are executed (␍-␊).

Shift Out ␎	wtb 701,14	char(14)

Shift out selects subsequent characters from the character set that is currently designated as the secondary character set until ␏ is received. ␎ and ␏ are only recognized if the printer is set to receive 7 bit ASCII code. If the printer is set for 8 bit ASCII code or if the secondary character set is already selected, ␎ is ignored.

Shift In ^O	wtb 701,15	char(15)

Shift in selects subsequent characters from the primary character set (US ASCII) until ^N is received. ^N and ^O are only recognized if the printer is set to receive 7 bit ASCII code. If the printer is set for 8 bit ASCII code or if the primary character set is already selected, ^O is ignored.

Escape ^[	wtb 701,27	char(27)

The escape character instructs the printer to accept the character sequence following it as an instruction code. Without ^[, the instruction code characters are interpeted as data and printed. The escape sequence must be sent with no blanks between the first 3 characters, as any blanks inserted between the first three characters in the sequence will cause it to be ignored.

Escape Sequence Instructions

When the printer receives an escape character (wtb 701,27), it examines the data following the ESC for an instruction sequence. The sequence may include one or several characters and may be a fixed sequence or a **parameterized** one with user specified components that may be specified according to your application.

NOTE

The escape sequence instruction is read **immediately** following the ESC character. **NO SPACES** are allowed within the first 3 characters of the string. If any are encountered, the printer ignores the Escape and prints the characters after the Escape.

Reset	ESC E	wtb 701,27,"E"

This escape sequence prints the current contents of the buffer and then resets the printer to the following power-on default conditions.

- Vertical Line Spacing is 12 dot rows between character base lines which produces 6.5 lines/inch (2.5 lines/cm).
- The character height is set to normal height characters.
- The top margin is 13 millimetres (3 character lines) from the form perforation to the top of the first line of text.
- The text length is set to 256 millimetres from the top margin (66 lines).
- All horizontal tabs are cleared.
- The Primary character set is selected for printing operations.
- The Secondary Character Set is designated as the HP European Extended Set.
- Both the auto-underline and the auto-overline highlightings are set "off".
- All new characters that have been generated are cleared.
- The Control Code Display mode is not enabled.
- The Graphic Data mode is not enabled.

Print Control Code Mode

The Print Control Code Mode is available for troubleshooting print statements which are designed to excercise control over the printer, as opposed to simply transferring data to it. Two escape sequences control it:

Start Print Control Code Mode	ESC Y	wtb 701,27,"Y"
Stop Print Control Code Mode	ESC Z	wtb 701,27,"Z"

When the printer receives ESC Y, it enters a "transparent" mode in which control codes and escape sequences are loaded into the buffer and printed. One escape sequence (ESC Z) is printed and then executed. ESC Z cancels the transparent mode. A carriage return (CR) received while the printer is in the print control code mode is printed and then a carriage return and linefeed are executed.

TAB Control

Three escape sequences control the creation and elimination of TAB positions in the printer.

Set A TAB	Esc 1	wtb 701,27,"1"

When the printer receives Esc 1, it sets a horizontal TAB position at the current character position. The position may have been reached by the Set to Column sequence discussed below, or by sending a series of spaces to the printer.

Clear A TAB	Esc 2	wtb 701,27,"2"

When the printer receives Esc 2, it clears a horizontal TAB position at the current character position. The position is normally reached by sending a ↹ to the printer.

If a horizontal TAB is not set at the current character position, the printer ignores the instruction.

Clear All TABS	Esc 3	wtb 701,27,"3"

When the printer receives Esc 3, it clears all the horizontal TAB positions currently set. If no TABS are set, the printer ignores the instruction.

Set to Column	Esc & a Column Number 0 to 79 C	wtb 701,27,"&a15C"

When the printer receives Esc &a 15C, it sets the character position to column 15, referenced from the left margin. Column 0 is the leftmost position on the paper and column 79 corresponds to the rightmost position on the paper. This is an absolute reference. This function is useful in setting TAB positions and setting to print in a specific column.

HINT

It is generally easier to use formatted `wrt` statements than to set up internal TABs in the printer. The printer functions are designed for less sophisticated computers that lack adequate formatting and printer control.

Line Control

Three parameters are user definable to control the positioning of lines of print on the page. These are the Top Margin, Line Spacing, and Text Length.

Top Margin	ESC&l 0.4 to 250 millimetres Top Margin T Lower case L	wtb 701,27,"&l4T" Lower case L

When the printer receives ESC&l4T it sets the distance between the top of the first line of print on a page and the top perforations equal to 4 millimetres. This will allow the printer to print the maximum amount on the page without printing over the perforations. The printer advances to the top margin (the first line of print on a page) whenever it receives a formfeed (^L = wtb 701,12), when the next line printed would exceed the text length, or when the bottom perforation passes the print head. (Except for top margin equal to zero.)

Vertical Line Spacing	ESC&l 0 to 60 dot rows between Character Baselines S Lower Case L	wtb 701,27,"&l24S" Lower case L

When the printer receives ESC&l24S from the example above, the spacing is set for 24 dot rows between the baseline of one character and the baseline of the character directly below it. This is twice the default value, and will create a double spaced output.

If the line spacing is set at less than twelve dot rows, the characters begin to overlap. As the line spacing approaches 0, the characters approach 100% overstrike. Full overstrike occurs when the vertical line spacing is zero. For any line spacing less than or equal to 8 dot rows, the printer will backup after each print line for 5x7 characters.

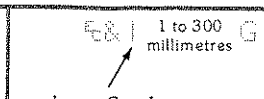
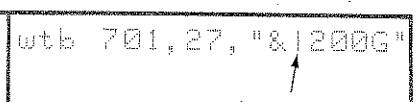
Setting the line spacing to 8 dot rows allows superscripts and subscripts to be printed.

Executing

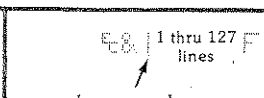
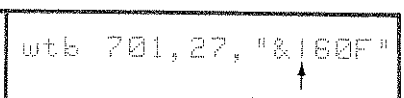
```
1→4: "write superscript":wtb 701," 2"
  5: "set vertical line spacing":wtb 701,27,"&l8S"
  6: "set 150% character height":wrt 701,char(27),"&k1S"
2→7: "print variable":wtb 701,"V"
  8: "clear 150% character height":wrt 701,char(27),"&k0S"
3→9: "print subscript":wrt 701," Max"
  10: "reset printer":wrt 701,char(27),"E"
  *28979
```

Produces

```
          1         2         3         4         5         6
 12345678901234567890123456789012345678901234567890123456789012345
    2←1
2→V
   Max←3
```

Text Length in Millimetres		

When the printer receives %&l200G from the above wtb statement, the text length is set at 200 millimetres. After each line of text is printed, the printer checks the distance from the top margin. If printing the next line would exceed 200 millimetres, the printer executes a formfeed and proceeds to the next top margin before printing the buffer.

Text Length in Lines		

When the printer receives %&l60F from the above wtb statement, the text length is set for 60 lines. Before each line is printed, the printer checks the number of lines printed since the top margin. If the next line printed would exceed 60, the printer executes a formfeed and proceeds to the top margin of the next form before printing the buffer.

Since 70 lines of text is a full page (assuming default line spacing and English paper), the top margin would have to be set to 0 to print 70 lines. Long top margins require short text lengths and vice-versa.

NOTE

Line Spacing will interact with Text Length specified in lines. Greater Line Spacing means fewer lines per page and vice-versa.

Character Control

Several Escape Sequence instructions are available to control the character set and the enhancements available on the printer. These include auto-overline and auto-underline, expanded height characters, user-defined characters, and alternate character sets.

Changing the Secondary Character Set

Designate Secondary Character Set	ESC) Character Set Designator	wtb 701,27,")1K"

When the printer receives ESC)1K from the wtb statement above, it selects Katakana as the secondary character set. Katakana will remain the secondary set until a new set is designated or until the machine is reset (assuming Katakana is not the default secondary set).

The default secondary set is selected at the time of manufacture to be the HP European Extended set. Changing the default secondary set requires internal modifications to the printer and should be left to trained service personnel. Contact your local HP Sales and Service Office for assistance. A list of the offices is in the back of this manual.

To designate a new secondary character set, use the following table:

Character Set	Character Set Designator	Escape Sequence	HPL
Danish-Norwegian	**0D**	ESC)0D	wtb 701,27,")0D"
European Extension	**0E**	ESC)0E	wtb 701,27,")0E"
French	**0F**	ESC)0F	wtb 701,27,")0F"
German	**0G**	ESC)0G	wtb 701,27,")0G"
Katakana	**1K**	ESC)1K	wtb 701,27,")1K"
Spanish	**1S**	ESC)1S	wtb 701,27,")1S"
Swedish-Finnish	**0S**	ESC)0S	wtb 701,27,")0S"
United Kingdom (England)	**1E**	ESC)1E	wtb 701,27,")1E"
US ASCII	**0U**	ESC)0U	wtb 701,27,")0U"

NOTE

The Character Set Designator instruction only designates the character set that will be the Secondary Character Set. The Shift In (^O = wtb 701,15 and Shift Out (^N = wtb 701,14) are used to select primary or secondary character sets in 7 bit ASCII Modes, and the 8th bit is totally ignored unless the printer is in the graphics mode. In 8-bit ASCII mode, ^N and ^O are ignored, and the logical value of the 8th bit selects the character set (a logical value of 1 for the 8th bit selects the secondary character set).

Changing Character Height

Set Expanded Height	^[&k1S	wtb 701,27"&k1S"
Set Normal Height	^[&k0S	wtb 701,27,"&k0S"

When the printer receives ^[&k1S from the first wtb statement above, the character height is increased by 50% until ^[&k0S is received from the second wtb statement. The expanded character height is generated by increasing the distance between vertical dot rows to 1½ times the normal spacing. The 150% spacing is done above and below the baseline, so that characters will line up across the page. The underline position is moved up one row to the line up with a normal height character underline. The width of normal and expanded height characters is the same.

Expanded Height characters use the same 5x7 central matrix that normal characters do, and have the same 2 dot row ascender pattern. However: **the bottom (3rd) descender row is not used with the expanded height characters. Thus the total character matrix is 7x11 for expanded characters, not 7x12 as with normal height characters.**

Selecting Highlights

Select Auto-underline	^[&dD	wtb 701,27,"&dD"
Select Auto-overline	^[&dP	wtb 701,27,"&dP"
Select both Auto-overline & Auto-underline	^[&dT	wtb 701,27,"&dT"
Clear both Auto-overline & Auto-underline	^[&dH	wtb 701,27,"&dH"

NOTE

There are alternate characters for selecting highlights. For a complete list of the characters, see Chapter 3 of the Peripheral Manual.

The printer selects auto-underline and auto-overline off at power on and reset. Once a highlight instruction code is sent to the printer, it remains selected until it is changed by a new instruction code or cleared by resetting or turning off the printer.

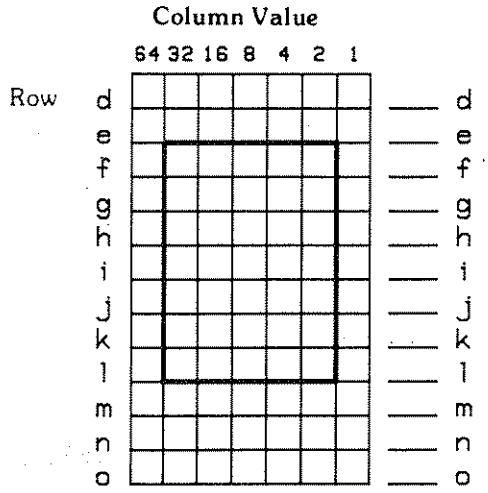
Defining and Clearing New Character Patterns

Define New Character	ESC&n Decimal value of character replaced C Sequence that defines character	wtb 701,27,"&n36c62e34f 16g8h4i8j16k34l62M"
Clear a Redefined character	ESC&n Decimal value of character to be cleared C	wtb 701,27,"&n36C"

When the printer receives ESC&n36c62e34f16g8h48j16k34l62M from the first wtb statement above, the $ character is replaced with the Σ symbol. The new character definition remains in effect until another new character definition is assigned to the same decimal value, or until the decimal value is cleared back to its original assigned value by the second escape sequence given above or by the printer reset (ESCE) which clears all defined characters. The sequence of decimal values to generate the Σ were selected using the following steps:

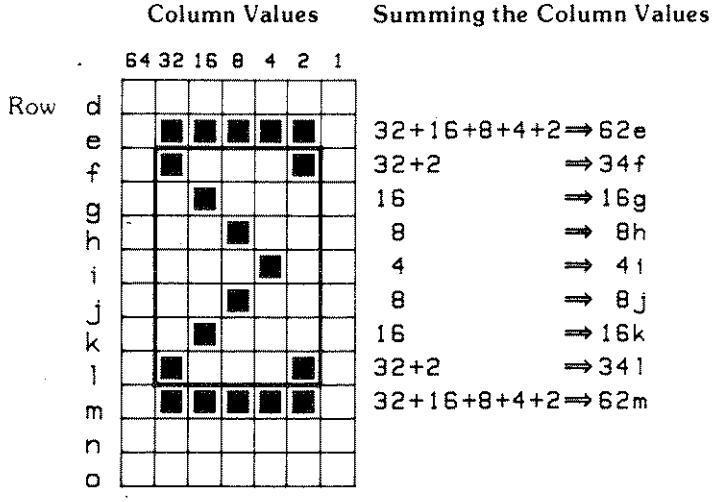
Step 1

Select the dot pattern that will create the new character shape or pattern. This can be done by filling in a seven by twelve matrix (similiar to the one shown below) with the new character's pattern. Additional copies of this matrix form are provided in the Appendix of the Peripheral Manual to be copied and used for designing new character patterns. The outlined 5x7 area represents the area in which a normal character is printed. The new character may utilize the entire 7x12 area shown.



Step 2

Once you have defined a new character pattern on the matrix, add the column values for the columns that contain dots across each row. Each summed value, followed by the character that designates the row position of the value, is used in the instruction code to define the new pattern. Only rows containing dots need to be specified since the printer assumes that any row not specified is blank.



The instruction code that would replace the $ character with the Σ character defined above is as follows:

```
%&n36c62e34f16g8h4i8j16k34l62M
```

Note that rows m,n and o were not specified since they do not contain any dots. Also note that the last row designator (M) in the instruction sequence must be capitilized to terminate the instruction.

The $ character may be replaced by the Σ character by executing

```
wtb 701,27,"&n36c62e34f16g8h4i8j16k34l62M
```

Each value and corresponding row designator pair can be sent to the printer in any order. Thus, the following instruction code results in the same redefinition of $ with Σ as the previous code.

```
%&n62m34l16k8j4i8h16g34f62e36C
```

Each new character pattern that is defined is stored in a memory area within the printer in the form of 12 bytes of data. The memory area can hold a maximum of 84 bytes. Thus a maximum of 7 new character patterns can be stored at one time. Blank lines do not reduce the storage requirements.

The following table lists selected Greek characters and mathematic symbols with the instruction code that defines each one. Note that in each instruction code, the space that has been left before the c (---c) is where the decimal value of the character to be replaced is put.

Table of Character Redefinitions

Symbol		Escape Sequence
Summation Sigma	Σ	E_C&n---c 62e 34f 16g 8h 4i 8j 16k 34l 62M
Integral	∫	E_C&n---c 4e 10f 10g 8 h 8i 8j 8k 40l 40m 16N
Large Left Bracket	[	E_C&n---c 60e 32f 32g 32h 32i 32j 32k 32l 60M
Large Right Bracket	]	E_C&n---c 30e 2f 2g 2h 2i 2j 2k 2l 30M
Large Left Parenthesis	(	E_C&n---c 12e 16f 32g 32h 32i 32j 32k 16l 12M

Large Right Parenthesis	)	E_C &n---c 24e 4f 2g 2h 2i 2j 2k 4l 24M
Root Sign	√	E_C &n---c 7d 4e 4f 4g 4h 4i 100j 20k 12l 4M
Plus or Minus	±	E_C &n---c 8f 8g 62h 8i 8j 62L
Approximate	≈	E_C &n---c 51g 76h 51j 76K
Element	∈	E_C &n---c 30f 32g 64h 126i 64j 32k 30L
Intersects	∩	E_C &n---c 28e 34f 65g 65h 65i 65j 65k 65l 65M
Union	∪	E_C &n---c 65e 65f 65g 65h 65i 65j 65k 34l 28M
Implies	⇒	E_C &n---c 8f 4g 126h 1i 126j 4k 8L
Gamma	γ	E_C &n---c 34i 82j 10k 4l 8m 16n 32O
Delta	δ	E_C &n---c 28e 18f 8g 28h 34i 34j 34k 28L
Theta	θ	E_C &n---c 28f 34g 34h 62i 34j 34k 28L
Mu	μ	E_C &n---c 34h 34i 34j 50k 46l 33m 32N
Pi	π	E_C &n---c 30h 52i 20j 20k 20L
Rho	ρ	E_C &n---c 28h 34i 34j 60k 32l 32m 32n 64O
Tau	τ	E_C &n---c 30h 40i 8j 8k 8L
Upsilon	υ	E_C &n---c 34h 82i 18j 20k 8L
Chi	χ	E_C &n---c 16e 42f 12g 8h 8i 8j 20k 34L
Phi	Φ	E_C &n---c 28e 8f 28g 42h 42i 42j 28k 8l 28M
Psi	ψ	E_C &n---c 8f 8g 8h 107i 42j 42k 28l 2m 8N

Once a character has been redefined, it retains that definition until the machine is reset, turned off, or the Clear New Character Definition Escape Sequence is received. To define all blanks you must specify at least one dot row.

Sending Graphic Data

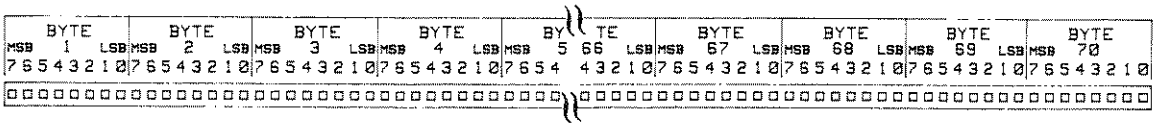
Set Graphics Mode	`E*b` Number of bytes to be interpreted as graphic data 0 thru 70 `W`	`wtb 701,27,"*b70W"`

When the printer receives `E*b70W` from the `wtb` statement above, it interprets the next 70 8 bit bytes as the binary representation of the dots to be printed. The 560 dots are exactly represented by 70 8 bit bytes. (The actual parameter range for this escape sequence is 0→255, but any bytes after the 70th byte are ignored.)

NOTE

You MUST send the number of bytes specified in the Escape Sequence. If you send more bytes than specified, the printer will print the excess bytes as characters. If you send fewer bytes than specified, the printer will interpret text (or the next control codes) as graphic data.

A `char` function may be used to encode the dot pattern with a suitable algorithm. Consideration must be given to the order of bits and bytes representing the dots on the print head. The bits and bytes pattern for the print head is illustrated below.



560 Dots Across the Print Head

To turn on the Nth dot on the print head, two conversions are necessary; one to select the proper byte position and one to generate the character code to fill the position. The byte position may be selected using the following conversion.

Byte position = `sgn (N/8) * int (abs (N/8))+1`

Once the byte position is determined, the proper character can be generated by the formula below.

Character = `char(2↑(7-N mod 8))`

This character can then be filled into a character string which has previously been filled with ASCII nulls (`CHR$(0)`).

`O$[` **Byte position, Byte position** `]` **= Character**

For a complete discussion of the algorithms necessary for actual plotting, see Section III on Advance Plotting.

CHAPTER 6

Introduction

This chapter describes the capabilities of the 9876A HP-IB Printer that are available due to its being an HP-IB device.

Overview of the HP-Interface Bus

The following is a definition of the terms and concepts used to describe HP-IB (bus) system operations.

HP-IB System Terms

1. **Byte** – A unit of information consisting of 8 binary digits (bits).
2. **Device** – Any unit that is compatible with the IEEE Standard 488-1975.
3. **Device Dependent** – A response to information sent on the HP-IB that is characteristic of an individual device's design and may vary from device to device.
4. **Operator** – The person that operates either the system or any device in the system.
5. **Addressing** – The characters sent by a controlling device to specify which device will send information on the bus and which device(s) will receive that information.
6. **Polling** – The process typically used by a controller to locate a device that needs to interact with the controller. There are two types of polling:

- **Serial Poll** – This method obtains one byte of operational information about an individual device in the system. The process must be repeated for each device from which information is desired.
- **Parallel Poll** – This method obtains information about a group of devices simultaneously.

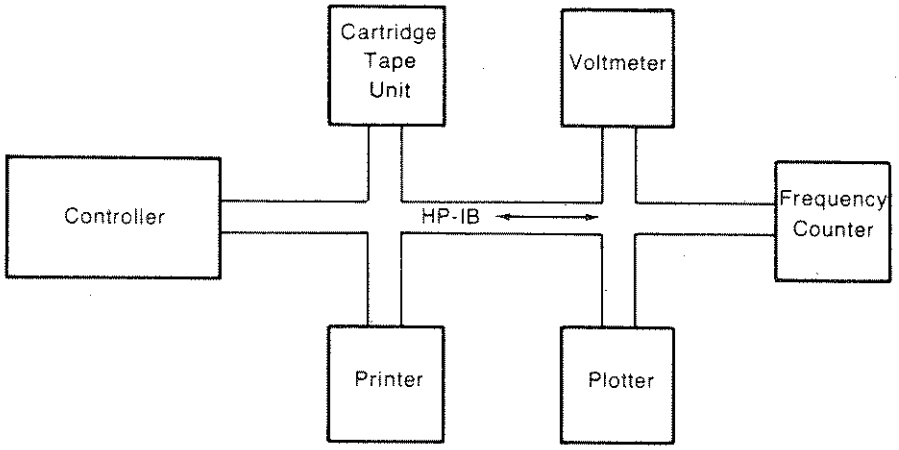
Interface Bus Concepts

Devices which communicate along the interface bus can be classified into three basic categories:

1. **Talkers** – Devices which **send** information on the bus when they have been addressed.
2. **Listeners** – Devices which **receive** information sent on the bus when they have been addressed.
3. **Controllers** – Devices that can specify the talker and listeners for an information transfer. Controllers can be categorized as one of two types:

- **Active Controller** – The current controlling device on the bus.
- **System Controller** – The controller that can take priority control of the bus even if it is not the current active controller. Although each bus system can have only one system controller, the system can have any number of devices capable of being the active controller.

A typical HP-IB System is shown below.



Typical HP-IB System

Message Concepts

Devices which communicate along the interface bus are transferring quantities of information. The transfer of information can be from one device to another device, or from one device to several devices. These quantities of information can easily be thought of as "messages." Typically, each message consists of two basic parts: the address specified by the controller and the information that comprises the message.

In turn, the message can be classified into twelve types. The twelve types of messages are defined as follows with the messages that pertain to the printer highlighted in color.

1. **The Data Message.** This is the actual information (binary bytes) which are sent from one talker to one or more listeners along the interface bus. Data can be either in numeric form or in a string of characters.
2. **The Trigger Message**. This message causes the listening device(s) to perform a device-dependent action.
3. **The Clear Message**. This message causes either the listening device(s) or all of the devices on the bus to return to their predefined device-dependent states.
4. **The Remote Message**. This message causes listening devices to switch from local front-panel control to remote program control.

5. **The Local Message**. This message clears the Remote Message from the listening device(s) and returns the device(s) to local front-panel control.
6. **The Local Lockout Message**. This message prevents a device operator from manually inhibiting remote program control.
7. **The Clear Lockout and Set Local Message**. This message causes all devices on the bus to be removed from **Local Lockout** and revert to **Local**. This message also clears the **Remote Message** for all devices on the bus.
8. **The Require Service Message**. A device can send this message at any time to signify that the device needs some type of interaction with the controller. This message is cleared when the device sends its **Status Byte Message** if the device no longer requires service.
9. **The Status Byte Message**. (This byte represents the status of a single device on the bus.) One bit indicates whether the device sent a **Require Service Message** and the remaining bits indicate operational conditions defined by the device. The byte is sent from a talking device in response to a serial poll operation performed by a controller.
10. **The Status Bit Message**. (This byte represents the operational conditions of a group of devices on the bus.) Each device responds on a particular bit of the byte thus identifying a device-dependent condition. This bit is typically sent by devices in response to a parallel poll operation.

    The Status Bit Message can also be used by a controller to specify the particular bit and logic level that a device will respond with when a parallel poll operation is performed. Thus more than one device can respond on the same bit.
11. **The Pass Control Message**. This transfers the bus management responsibilities from the active controller to another controller.
12. **The Abort Message**. The system controller sends this message to unconditionally assume control of the bus from the active controller. This message terminates all bus communications but does not implement a **Clear Message**.

These messages represent the full implementation of all HP-IB system capabilities. Each device in a system, however, may be designed to use only the messages that are applicable to its purpose in the system. It is important for you to be aware of the HP-IB functions implemented on each device connected to your HP-IB system to ensure the operational compatibility of the system.

Printer Bus Functions

Each device in an HP-IB system may be capable of using all or any portion of the messages just defined. A device's capabilities are grouped under 10 interface functions which are defined by the IEEE Standard 488-1975. Refer to Chapter 4 of the Printer Peripheral Manual (P/N 09876-90000) for a technical description of the functions required for each bus message. The following table shows which messages the 9876A Printer is designed to use:

Bus Message Implementation

Message	9876A Implementation
Data	Receive and Send (status)
Trigger	Not Implemented
Clear	Not Implemented
Local	Not Implemented
Remote	Not Implemented
Local Lockout	Not Implemented
Clear Lockout/Set Local	Not Implemented
Require Service	Send Only
Status Byte	Send Only
Status Bit	Not Implemented
Pass Control	Not Implemented
Abort	Receive Only

Data Messages

The data message refers to the ASCII coded data and the 8-bit graphic data bytes that are sent to the printer on the HP-IB. Most of the data forms and the printer's response to them are described in the first five chapters of this manual as well as in the Peripheral Manual (P/N 09876-90000).

Clearing the Printer

The printer does not respond to the HP-IB message Device Clear or Select Device Clear. The printer can be cleared or reset, however, by using the Reset Escape Sequence (E).

Service Requests and Polling

The printer can send the Require Service Message (SRQ) to the controller when any of the following conditions occur:

- Out of paper.
- Hardware failure.
- Printhead overheated.

The Require Service Message can only be sent if switch 8 on the printer's rear panel is set "on" as described in Chapter 2 of the Peripheral Manual.

Since the printer does not have parallel poll capability implemented it cannot respond to a parallel poll when it has sent the Require Service Message.

It can respond however, to a serial poll, which obtains the Status Byte message from the printer. When the printer is addressed during a serial poll operation, it sends the controller a status byte that defines the condition of the printer. Additionally, the service request line (SRQ) is cleared when the printer responds to a serial poll operation. The following is the definition of the printer's status byte:

MSB							**LSB**
Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
128	**64**	**32**	**16**	**8**	**4**	**2**	**1**
Not Used	Require Service Message Sent	Not Used	Hardware Failure	Printhead Overheated	Out of Paper	Confidence Test in Process	Not Used
Always 0	64=true 0=false	Always 0	16=true 0=false	8=true 0=false	4=true 0=false	2=true 0=false	Always 0

The status byte is sent to the controller as the decimal equivalent sum of the conditions that are true. For example, a status byte with a summed value of 68 would indicate the following printer conditions:

0	Not used.
+64	Require Service Message Sent.
+0	Not used.
+0	The printer does not have a hardware failure.
+0	The printer head is not overheated.
+4	The printer is out of paper.
+0	There is not a confidence test in process.
+0	Not used.

An interrupt routine can be written to allow the Require Service Message to initiate a subroutine which will analyze the status byte from the 9876A to determine the problem generating the Require Service Message. First it is necessary to specify the correct response to an interrupt on the printers select code. This is done with an `oni` statement.

Example

```
8: oni 7,"Service"
```

Immediately before the print routine, a `eir 7` statement is executed, to enable interrupts.

Example

```
9: eir 7
```

The actual routine uses a `rds` function to read the Status Byte and an `if bit` construction to analyze the Status Byte. The `if bit` construction tests the bits directly instead of using a binary reduction of a decimal value. The program then `stops` to allow the user to correct the problem.

Example

```
20: "Service":
21: "*************************************************************":
22: "**** This subroutine first prints a message on the***********":
23: "**** 9825 internal printer that an interrupt has been generated ***":
24: "**** on the HP-IB select code. ********************************":
25:
```

```
26: "**** Next the status byte is obtained using a serial poll operation.":
27: "**** The byte is analyzed with an  if bit  operation. ************":
28: "**** The result of the analysis is printed on the internal printer*":
29: "**** and the program pauses to allow you to correct the problem. **":
30: "**** After the problem is corrected and the user presses CONTINUE,*":
31: "**** the interrupt is re-enabled and program execution returns to *":
32: "**** the main program. ********************************************":
33:
34: prt "*****************"
35: prt "**INTERRUPT ON**"
36: prt "**SELECT CODE 7*"
37: prt "*****************"
38: wtb 16,10,10
39: rds(701)S;fxd 0
40: prt "Status =",S
41: wtb 16,10,10
42: if bit(7,S);gsb "Lunch"
43: if bit(6,S);prt "Require Service Message Sent for"
44: if bit(5,S);gsb "Lunch"
45: if bit(4,S);prt "Hardware Failure"
46: if bit(3,S);prt "-9876 Overheated"
47: if bit(2,S);prt "-Out Of Paper"
48: if bit(1,S);prt "-Confidence Test In Progress"
49: if bit(0,S);gsb "Lunch"
50: wtb 16,10,10
51: prt "Correct The Pro-blem and Press     CONTINUE"
52: wtb 16,10,10
53: stp
54: eir 7
55: iret
56: "":
57: "Lunch":
58: prt "*****************"
59: prt "**Invalid Error******Message****"
60: prt "*****************"
61: ret
*14589
```

NOTE

If a service request is generated during a serial poll operation, the service request will be cleared and will not generate an interrupt in the controller. The status byte value that is transferred to the controller during such a serial poll operation however, will still indicate that both the "Service Request Sent" and the condition that caused the service request are true.

Parallel Polling Considerations

If you use a parallel poll operation in response to Service Request interrupts to determine which device is your system set the SRQ, you must remember that the 9876 does not respond to parallel poll operations. Thus a parallel poll executed as a result of an interrupt routine will show that one of the devices in the system sent an SRQ if the printer was the device requiring service. A serial operation that addresses the printer would then be the proper program response in this situation.

IMPORTANT

The printer sends the Require Service Message (SRQ) as a result of several conditions that stop the printer's operation. Of these conditions, only "OUT OF PAPER" occurs in a manner that typically allows the computer enough time to finish the currently executing program line. The other conditions may or may not allow the computer to finish the currently executing line. Since the computer only reponds to `oni 7` at the end of the currently executing line a program, these other printer conditions may not allow the computer to respond to the printer's request for service (SRQ). This condition can result in hanging up further data transfers on the HP-IB.

The following program allows the computer to respond to a Require Service Message, subject to the limitation stated above. This program may be found in File 5 on Track 1 on the 9876-9825 Utility Tape.

```
0: "*************************************************************************":
1: "* This program demonstrates a set of routines that detect service  *":
2: "* requests that are generated whenever the printer is OUT OF PAPER *":
3: "*************************************************************************":
4:
5: "*************************************************************************":
6: "* The main program sets up the interrupt and enables it.*************":
7: "*************************************************************************":
8: oni 7,"Service"
9: eir 7
10: "*************************************************************************":
11: "* This section is where your main program goes.**********************":
12: "*************************************************************************":
```

```
13: for I=1 to 500
14: wrt 701,"This is line #",I
15: next I
16: end
17: "*************************************************************************":
18: "* This is the end of the main module.*****************************":
19: "*************************************************************************":
20: "Service":
21: "*************************************************************************":
22: "**** This subroutine first prints a message on the******************":
23: "**** 9825 internal printer that an interrupt has been generated ***":
24: "**** on the HP-IB select code. ***************************************":
25:
26: "**** Next the status byte is obtained using a serial poll operation.":
27: "**** The byte is analyzed with an  if bit  operation. *************":
28: "**** The result of the analysis is printed on the internal printer*":
29: "**** and the program pauses to allow you to correct the problem. **":
30: "**** After the problem is corrected and the user presses CONTINUE,*":
31: "**** the interrupt is re-enabled and program execution returns to *":
32: "**** the main program. ********************************************":
33:
34: prt "*****************"
35: prt "**INTERRUPT ON**"
36: prt "**SELECT CODE 7*"
37: prt "*****************"
38: wtb 16,10,10
39: rds(701))S;fxd 0
40: prt "Status =",S
41: wtb 16,10,10
42: if bit(7,S);gsb "Lunch"
43: if bit(6,S);prt "Require Service Message Sent for"
44: if bit(5,S);gsb "Lunch"
45: if bit(4,S);prt "Hardware Failure"
46: if bit(3,S);prt "-9876 Overheated"
47: if bit(2,S);prt "-Out Of Paper"
48: if bit(1,S);prt "-Confidence Test In Progress"
49: if bit(0,S);gsb "Lunch"
50: wtb 16,10,10
51: prt "Correct The Pro-blem and Press     CONTINUE"
52: wtb 16,10,10
53: stp
54: eir 7
55: iret
56: "":
57: "Lunch":
58: prt "*****************"
59: prt "**Invalid Error******Message*****"
60: prt "*****************"
61: ret
*14589
```

The Abort Message

When the printer receives the Abort Message that is sent on the interface clear (IFC) line, it returns to an unaddressed state and waits until it is addressed again before continuing any operations. Any data that is in the printer's buffer when the abort message was received remains there. When the printer is re-addressed after an abort, it continues filling the buffer at the point that it stopped for the abort. The printer is not cleared or reset by an Abort message.

The Abort message can be transmitted by the c l i statement.

Example

```
cli 7
```

HP-IB Worksheet

The following worksheet is provided for use in operating and programming the devices in an HP-IB system. Each device's HP-IB implementation should be filled in an appropriate column of the form. For each of the 12 messages listed, either an R (for receive only), an S (for Send only), an SR (for Send and Receive) or an N (for Not Implemented) should be lettered in the appropriate box to show each device's response to the message.

The printer's response to the various HP-IB Messages has been printed in the appropriate place on the worksheet in the second column. The first column has been filled in for an HP System Desktop Computer. The remaining columns can be used for other devices connected to the bus.

Once the worksheet has been filled in for all of the devices in the HP-IB system, the information can be used as an aid in programming an application. When the program makes use of an HP-IB message, it can be read across the columns for that message and checked to see that the appropriate devices have the necessary capability to respond properly to the message. The worksheet also helps you make sure that each device has a unique address code.

For example, in a program application that utilizes service requests and polling, the printer can send the Require Service Message and respond to a serial poll (Status Byte message) but it cannot respond to a parallel poll operation (Status Bit message). Therefore your program design would have to reflect this situation when preparing service request routines.

In addition, the back of the worksheet provides a place to list the status byte (serial poll) information for the devices in your system. The printer's status byte has been filled in for you. This reference can be useful if you have any service request routines in your program application on the HP-IB.

HP-IB Message Information Worksheet

Device Name	9825	9876A						
Address Code	21	01						
ASCII Listen Address	5	!						
ASCII Talk Address	U	A						
5-Bit Binary Address	10101	00001						
MESSAGES	S - Send Only R - Receive Only	IMPLEMENTATION				SR - Send and Receive N - Not Implemented		
Data	SR	SR						
Trigger	S	N						
Clear	SR	N						
Local	S	N						
Remote	S	N						
Local Lockout	S	N						
Clear Lockout & Set Local	S	N						
Request Service	SR	S						
Status Byte	SR	S						
Status Bit	SR	N						
Pass Control	SR	N						
Abort	SR	R						

HP-IB Polling Information Worksheet

Bit Significance	MSB							LSB	Parallel Poll Bit
Bit Position	7	6	5	4	3	2	1	0	
Decimal Value	128	64	32	16	8	4	2	1	
Status Byte (Serial Poll) Information									
Device **9876A** Address Code **01**	Not Used	Require Service Sent	Not Used	Hardware Failure	Printhead Overheated	Out of Paper	Confidence Test in Process	Not Used	NA
Device Address Code									
Device Address Code									
Device Address Code									
Device Address Code									
Device Address Code									
Device Address Code									
Device Address Code									

SECTION III

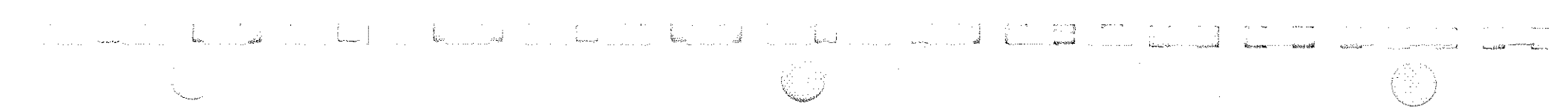
High resolution plots may be generated by accessing the individual dot resistors along the print head. The print head has 560 dots in a 18.5 cm span, or approximately 30.27 dots per centimetre. The dots may be accessed individually by sending `Esb70W` to the printer, followed by a set of 8 bit ASCII characters whose binary values represent the dot pattern to be printed.

High Resolution Plots of F(X) with a Vertical X-axis — Chapter 7

The considerations involved in scaling and offsetting are discussed. The function is evaluated on a line-by-line basis within a `for-next` loop and transformed to a graphic output using the scale and offset factors. Test are employed to prevent errors in the translation from the mathematical model to the graphic output.

Array Plotting — Chapter 8

A method for using Arrays to generate plots with horizontal X-axis is discussed, along with a technique for labeling the plots.



CHAPTER 7

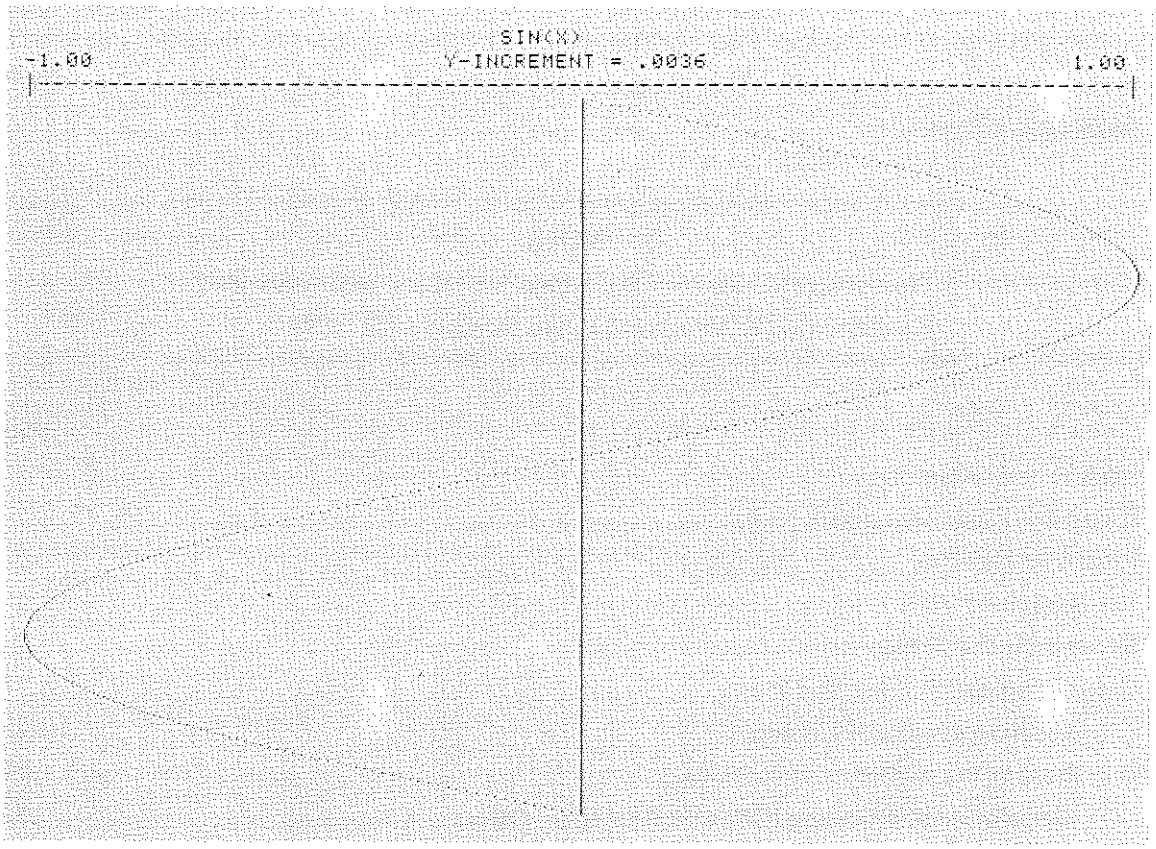
High Resolution Plots of F(X) with a Vertical X-Axis

Introduction

With a few simple modifications it is possible to increase the resolution of the character plot program discussed in Chapter 3 by a factor of ten. The time required for the plot is only twice that of the low resolution plot. This program may be found in File 6 on Track 1 of the 9876-9825 Utility Tape.

NOTE

This program requires the String Variables and Advanced Programming ROM as well as the General I/O ROM.


SIN(X)
-1.00
Y-INCREMENT = .0036
1.00

```
0: "program to plot simple functions":
1:
2: "A$ is output string":
3: "B$ is clear string":
4: "Y is scaled value of F(X)":
5: "J is character position":
6: "0$ contains null character":
7:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701.3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Scaling and Offsetting

Before generating a plot, it is necessary to determine the scale factor and offset value for the Y-axis. As discussed in Chapter 3, the most obvious formula,

$$\text{Scale Factor} = \frac{\text{Number of Y Units}}{\text{Range}}$$

produces a clipped plot. Therefore we will use

$$\text{Scale Factor} = \frac{\text{Number of Y Units} - 2}{\text{Range}}$$

for our plot:

$$\text{Scale Factor} = \frac{558}{2} = 279$$

Using an offset of 280 gives an error margin at either end of the plot.

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701.3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Filling the Clear String with Nulls

First, `char(0)` is used to generate an ASCII Null character, which is filled into `N$`. The clear string (`B$`) is then filled with Nulls in a `for-next` loop. Once the string is filled with nulls it is possible to do random access assignments to any character position in the string.

```
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
```

Printing a Heading

As in the character plot, the heading is placed in a format statement and then accessed with a write statement.

```
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
```

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701.3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y-8)*int(abs(Y-8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Labeling the Y-Axis

The maximum and minimum Y excursion values are printed above the Y-axis along with the incremental Y value. The proper labels, along with the spacing information, are contained in the format statement. The format statement is accessed by a write statement as in the character plot.

```
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
```

Generating the Y-Axis

The Y-axis is represented primarily by minus signs. Vertical bars are used for the maximum and minimum Y values. The entire axis is contained in a format statement which is accessed by a write statement.

```
24: fmt 3,"|",78"-","|"
25: wrt 701.3
```

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701.3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Calculating and Printing the Plot

Once the Y-axis is generated, we can move on to the actual plot. A `for-next` loop is established to control the value of `X` throughout the domain of interest. The loop is set up to run from the minimum X value to the maximum X value by an appropriate incremental step (560 dot rows will produce a square plot – more or less may be used to adjust the aspect ratio of the plot.)

Within the `for-next` loop six operations must be performed, and a branch descision made to avoid overwriting the X-axis.

1. Clearing the Output String.
2. Adding the X-axis.
3. Calculating and Scaling F(X) and Adding an Offset.
4. Branching if on X-Axis.
5. Calculating the character position.
6. Filling the dot into the string.
7. Printing the line.

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701,1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701,2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701,3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Clearing the Output String

The output string is filled with ASCII null characters by a simple assignment. The clear string (B$) has already been filled with nulls in line 13.

```
29: "clear the string":B$→A$
```

Adding the X-Axis

The dot representing the X-axis must be added into each dot row before it is printed. We want the 280th dot to represent the axis. The proper character position has been determined using an integer division function (sgn (Y/8) * int (abs(Y/8)) +1) to be 36. The decimal value to fill into the character is determined using the dot algorithm discussed on page 5-18 (2↑(7-Ymod8)) to be 128. The decimal value is converted to an ASCII character using a char function.

```
30: "add X-axis":char(128)→A$[36,36]
```

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701.1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701.2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701.3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Calculating and Scaling F(X) and Adding an Offset

The function is evaluated for each line at the value of X specified by the `for-next` loop. The result of the evaluation is multiplied by the previously determined scale factor and an offset value is added to this value to make sure only positive numbers are passed to the dot algorithm.

```
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
```

Branching If On X-Axis

If an attempt is made to put two dots in the same location on the same line, the algorithm will shift the dot one position to the left, or make it disappear completely. Since the only source of a double dot is overlaying the function on the X-axis, testing for the Y value equal to the X-axis value (280) is sufficient. If the function lies on the axis, we go directly to printing the line of dots.

```
32: "branch if on axis":if Y=280;gto "print line"
```

```
0: "program to plot simple functions":
1:
8: dim A$[70],B$[70],N$[1]
9: "set degree mode":deg
10:
11: "fill clear string with nulls":
12: char(0)→N$
13: for I=1 to 70;N$→B$[I];next I
14:
15: "print heading":
16: fmt 1,34x,"SIN(X)"
17: wrt 701,1
18:
19: "label Y-axis":
20: fmt 2,"-1.00",25x,"Y-INCREMENT = .0036",26x,"1.00"
21: wrt 701,2
22:
23: "generate axis":
24: fmt 3,"|",78"-","|"
25: wrt 701,3
26:
27: "calculate & print plot":
28: for X=0 to 360
29: "clear the string":B$→A$
30: "add X-axis":char(128)→A$[36,36]
31: "calculate f(x) & scale & offset":prnd(sin(X)*279,0)+280→Y
32: "branch if on axis":if Y=280;gto "print line"
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
35: "print line":wtb 701,27,"*b70W",A$
36: next X
*21380
```

Calculating the Character Position

An integer division function (`sgn(Y/8)*int(abs(Y/8))`) is used to determine the character position the dot lies in. One is added to the result to compensate for the lack of a 0th position in a character string.

```
33: "calculate character position":sgn(Y/8)*int(abs(Y/8))+1→J
```

Filling a Dot into the Output String

The dot algorithm (`2↑(7-Ymod8)`) is used to determine the decimal value of the character that when filled into the position determined above will best represent the `Y` value. A `char` function is used to convert the decimal value into an ASCII character which is then filled into the output string.

```
34: "fill dot into string":char(2↑(7-Ymod8)+num(A$[J,J]))→A$[J,J]
```

Printing a Line

Once the output string contains the correct pattern, it is sent to the printer, preceded by the Set Graphics mode escape sequence (ESC *b70W). Note the use of a wtb statement to encode the escape character (ESC = decimal 27).

```
35: "print line":wtb 701,27,"*b70W",A$
```

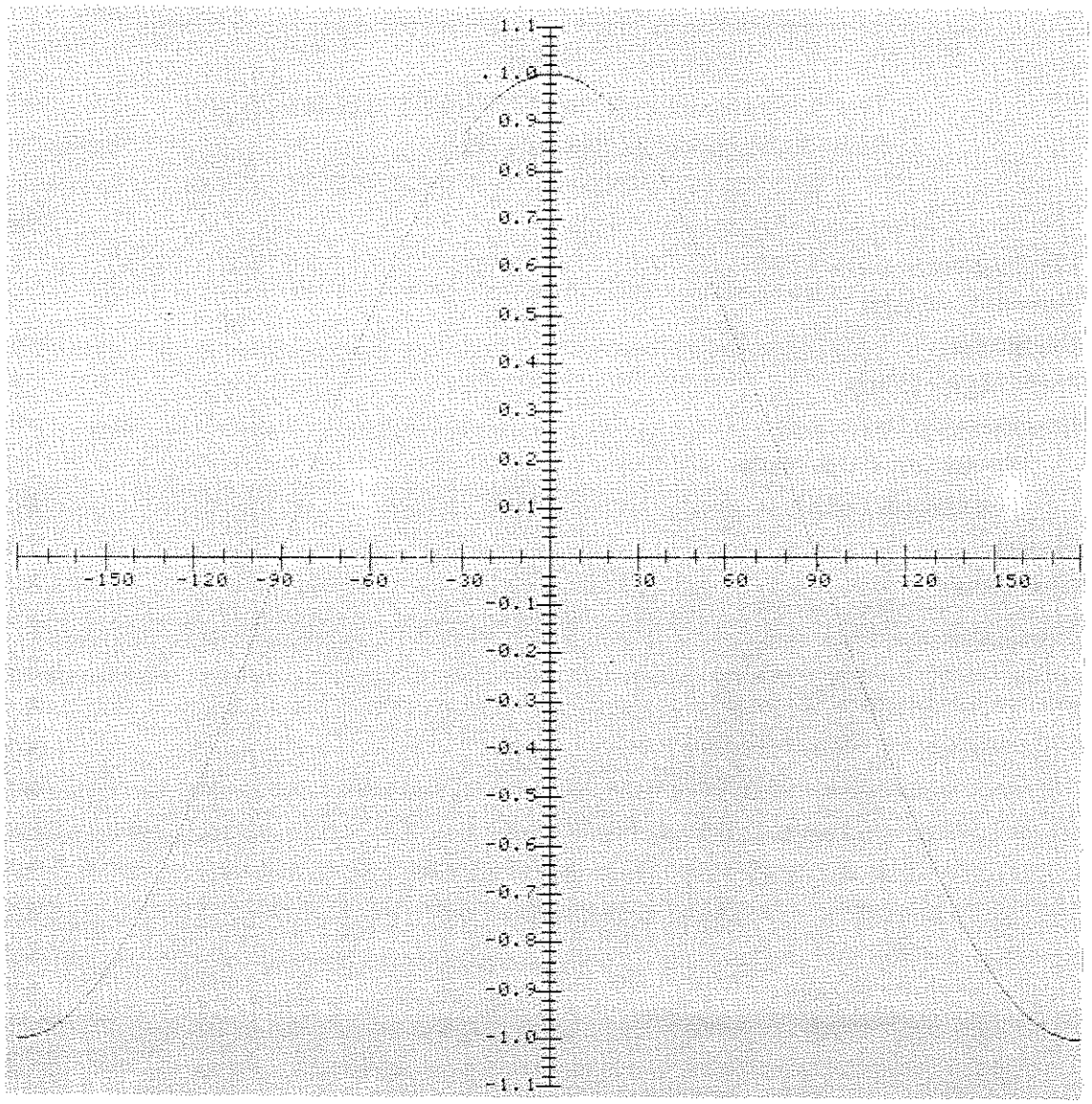
CHAPTER 8

Introduction

Using an array to store data representing points to be plotted, it is possible to generate a plot of a function (or arbitrary data) with the Y-axis running vertically on the paper. This chapter discusses a program to produce such plots. A technique for adding labels in the body of the plot is also discussed. This program may be found in File 7 on Track 1 of the 9876-9825 Utility Tape.

NOTE

This program requires the Advanced Programming and String ROMs as well as General I/O.



```
0: "program for Cos plot ":
1:
2: "setup":
3: dim D[561],R[561],T[6],X[6],Y[6]
4: "D will contain the Domain(X values)":
5: "R will contain the Range (Y values)":
6: "T[1],T[2] are temporary variables":
7:
8: "X[1],X[2],Y[1],Y[2] contain min and max values for display":
9: "set X limits":-180→X[1];180→X[2]
10: "set Y limits":-1.1→Y[1];1.1→Y[2]
11:
12: "X[3],Y[3] are X and Y increments":
13: "calculate X increment":(X[2]-X[1])/558→X[3]
14: "calculate Y increment":(Y[2]-Y[1])/558→Y[3]
15:
16: "X[4],X[5],Y[4],Y[5] contain minor and major tick intervals":
17: "assign X tick intervals":10→X[4];30→X[5]
18: "assign Y tick intervals":.02→Y[4];.1→Y[5]
19:
20: "determine X offset and fill into 0":abs(X[1])→0
21:
22: "dimension and define strings":
23:
24: dim C$[70],X$[7,70],L$[80],T$[1,9]
25:
26: "  C$ is the clear string                X$[5] contains minor Y ticks    ":
27: "  X$[1] contains minor X ticks          X$[6] contains major Y ticks    ":
28: "  X$[2] contains major X ticks          X$[7] is the output string      ":
29: "  X$[3] contains the X Axis             L$ is used to label the X Axis  ":
30: "  X$[4] contains the Y axis dot         T$ is a temporary variable      ":
31:
32: "clear strings":
33: for J=1 to 80
34: " "→L$[J,J];if J<=70;char(0)→C$[J,J]
35: next J
36: C$→X$[1]→X$[2]→X$[3]→X$[4]→X$[5]→X$[6]→X$[7]
37:
38: "set format and precision for labels":
39: fmt 1,z,33x,f6.1,b;fxd 1
40:
41: "HOUSE KEEPING":dsp "HOUSEKEEPING"
42:
43: "generate Y axis dot":0→X;4→N;gsb "make a dot"
44:
45: "generate Y axis tick marks":
46: for I=7 to -7 by -.9
47: "make major tick":I→X[3]→X;6→N;gsb "make a dot"
48: "make minor tick":if I<4 and I>-4;5→N;gsb "make a dot"
49: next I
50:
```

```
51: "generate X axis, X tick marks and evaluate F(X)":
52: for X=X[2] to X[1] by -X[3]
53: "make X Axis":3→N;gsb "make a dot"
54: "make minor ticks":if XmodX[4]<X[3];1→N;gsb "make a dot"
55: "make major ticks":if XmodX[5]<X[3];2→N;gsb "make a dot"
56: "evaluate F(X)":U+1→U;X→T[2];cos(X)→T[1];gsb "SORT"
57: next X
58:
59: "PLOTTING":dsp "PLOTTING"
60:
61: "make a plot":U→I
62: "set X$ index for output string":7→N
63: for Y=Y[2]+6Y[3] to Y[1] by -Y[3]
64: C$→X$[7]
65: gsb "axes"
66: gsb "test data"
67: "print the row of dots":wtb 701,27,"*b70W",X$[7]
68: next Y
69:
70: "go to next perforation":wtb 701,12
71:
72: stp
73:
74: "************************ END OF MAIN PROGRAM ************************":
75:
76:
77: "****************************** SUBROUTINES ******************************":
78:
79: "test data":
80: if I<1;ret
81: if abs(R[I]-Y)>Y[3];ret
82: D[I]→X;gsb "make a dot"
83: I-1→I
84: gto "test data"
85:
86:
87: "axes":
88: "Y axis dot":if Y<=Y[2];X$[4]→X$[7]
89: "minor Y tick":if abs(YmodY[4])<=Y[3] and Y<=Y[2];X$[5]→X$[7]
90: "major Y tick":if abs(YmodY[5])<=Y[3] and Y<=Y[2];X$[6]→X$[7]
91: "next line labels Y axis":
92: if (Y-6Y[3])modY[5]<=Y[3] and abs(Y)>8Y[3];wrt 701,1,prnd(Y-6Y[3],-1),13
93: "generate X Axis":if Y<7Y[3] and Y>-8Y[3];gsb "X-axis"
94: ret
95:
96: "X-axis":
97: "minor X ticks":X$[1]→X$[7]
98: "major X ticks":if abs(Y)>=4Y[3];X$[2]→X$[7]
99: "X axis":if abs(Y)<Y[3]/2;X$[3]→X$[7]
100: "label X axis":if Y<-7Y[3] and Y>-8Y[3];gsb "X labels"
101: ret
102:
103:
```

```
104: "make a dot":
105: "convert X to dot":int((X+0)/X[3])→H
106: "calculate character position":int(H/8)+1→P
107: "find current & new characters":num(X$[N,P,P])→A;2↑(7-Hmod8)→B
108: "add dot if not overlapping":if Amod2B-AmodB=0;char(B+A)→X$[N,P,P]
109: ret
110:
111: "X labels":
112: for X=X[1]+X[5] to X[2]-X[5] by X[5]
113: "set mode":fxd 0
114: "numeric to string conversion":str(prnd(X,0))→T$[1]
115: "decimal to dot conversion":(X+0)/X[3]→H
116: "dot to character position conversion":int(H/8*(8/7))→P
117: "add label to label string":if abs(X)>X[5]/2;T$[1]→L$[P,P+len(T$[1])]
118: next X
119: "print label string":wtb 701,L$,13
120: "restore mode":fxd 1
121: ret
122:
123:
124: "SORT":
125: 1→F;U→C
126: "begin sort":
127: if U=1 or U=F;T[1]→R[U];T[2]→D[U];ret
128: if T[1]<R[1];1→P;gto "ripple up"
129: if T[1]>R[U-1];T[1]→R[U];T[2]→D[U];ret
130: int((C-F)/2)+F→M
131: if T[1]>R[M];gto "above"
132: if T[1]<R[M];gto "below"
133: if T[1]=R[M];M→P;gto "ripple up"
134: dsp "error";stp
135:
136: "above":
137: if T[1]<R[M+1];M+1→P;gto "ripple up"
138: M+1→F;gto "begin sort"
139:
140: "below":
141: if T[1]>R[M-1];M→P;gto "ripple up"
142: M-1→C;gto "begin sort"
143:
144: "ripple up":
145: for Z=U to P+1 by -1
146: R[Z-1]→R[Z];D[Z-1]→D[Z]
147: next Z
148: T[1]→R[P];T[2]→D[P]
149: ret
*31762
```

The Program

Data pairs representing the points to be plotted are stored in two arrays. The X values are contained in the Domain Array (D[I]). The Y value are contained in the Range Array (R[I]). This data is sorted in ascending order by Y values.

The plot is generated one Y dot row at a time. As each Y position is generated in a decrementing `for-next` loop, it is tested against the last Y value in the array (which happens to be the largest value.) If the Y value is most accurately represented by the dot row being generated, the associated X value is used to produce a dot in the output string. The index to the Y array indicating the plotted point is then eliminated from consideration. The next Y value is then subjected to the same test. Testing on presorted data and eliminating points already plotted produces a rapid printout when compared with checking through an entire array of unsorted data for each dot row.

A set of tests based on `mod` functions is used to test for proper positioning of tick marks and labels on the Y-axis and to position the X-axis. The labels are printed by dropping out of the graphic mode and executing a carriage return without a line feed after printing the label, to return the paper to the proper position for continuing the plot.

```
3: dim D[561],R[561],T[6],X[6],Y[6]
4: "D will contain the Domain(X values)":
5: "R will contain the Range (Y values)":
6: "T[1],T[2] are temporary variables":
7:
8: "X[1],X[2],Y[1],Y[2] contain min and max values for display":
9: "set X limits":-180→X[1];180→X[2]
10: "set Y limits":-1.1→Y[1];1.1→Y[2]
11:
12: "X[3],Y[3] are X and Y increments":
13: "calculate X increment":(X[2]-X[1])/558→X[3]
14: "calculate Y increment":(Y[2]-Y[1])/558→Y[3]
15:
16: "X[4],X[5],Y[4],Y[5] contain minor and major tick intervals":
17: "assign X tick intervals":10→X[4];30→X[5]
18: "assign Y tick intervals":.02→Y[4];.1→Y[5]
19:
20: "determine X offset and fill into O":abs(X[1])→O
```

Numeric Constants

The maximum and minimum axes values, dot increments, and tick increments are stored in X[1] through X[5] and Y[1] through Y[5]. The limits of the graph are adjusted by assignments into X[1], X[2], Y[1], and Y[2].

The increments are calculated on the basis of the maximum and minimum axes values and an assumed 558 dots representing each axis dimension. Longer plots (vertical axis) may be generated by increasing the 558 in the calculation of the Y increment. The 558 in the calculation of the X scale factor should not be increased as this will introduce clipping or "crash" the program by attempting to access characters beyond the dimensioned limits of the string variables. The 558 can be reduced for smaller plots along either axis.

The spacing for major and minor ticks is set by assignments into X[4], Y[4] and X[5], Y[5]. Labels are provided for major tick marks. The values contained in X[1] through X[5] and Y[1] through Y[5] are used as constants in remainder of the program.

The X offset is stored in O and is simply the absolute value of the negative X excursion.

```
24: dim C$[70],X$[7,70],L$[80],T$[1,9]
25:
26: "  C$ is the clear string          X$[5] contains minor Y ticks    ":
27: "  X$[1] contains minor X ticks    X$[6] contains major Y ticks    ":
28: "  X$[2] contains major X ticks    X$[7] is the output string      ":
29: "  X$[3] contains the X Axis       L$ is used to label the X Axis  ":
30: "  X$[4] contains the Y axis dot   T$ is a temporary variable      ":
31:
32: "clear strings":
33: for J=1 to 80
34: " "→L$[J,J];if J<=70;char(0)→C$[J,J]
35: next J
36: C$→X$[1]→X$[2]→X$[3]→X$[4]→X$[5]→X$[6]→X$[7]
```

The Strings

The bits representing the patterns to be printed are stored in chracter strings. In addition, strings are provided for labeling the X-axis, and a string of ASCII null characters is provided for clearing the output string after each dot row is printed.

The different lengths are due to the uses the strings are put to. The L$ string is used to store ASCII characters for labeling. There are 80 characters (each 7 dot's wide) across a page. The C$ and X$ strings contain dot patterns which are represented by ASCII characters. All eight bits of each character are used by the printer, so only 70 characters (each representing eight dots) are required.

The "clear strings" module fills the L$ string (used for labeling the X-axis) with 80 blanks. The clear string (C$) is filled will 70 ASCII null characters in the same module.

Preparing for Labels

Labels for the Y-axis are printed near the middle of the plot, adjacent to the Y-axis. This is accomplished through the use of a formatted wrt statement. The format below is referenced for this operation.

```
39: fmt 1,Z,33X,f6.1,b;fxd 1
```

Suppress CR-LF | Space to middle of page | PRINT LABEL | Sends CR | Set proper notational form

```
41: "HOUSE KEEPING":dsp "HOUSEKEEPING"
42:
43: "generate Y axis dot":0→X;4→N;gsb "make a dot"
44:
45: "generate Y axis tick marks":
46: for I=7 to -7 by -.9
47: "make major tick":I*X[3]→X;6→N;gsb "make a dot"
48: "make minor tick":if I<4 and I>-4;5→N;gsb "make a dot"
49: next I
50:
51: "generate X axis, X tick marks and evaluate F(X)":
52: for X=X[2] to X[1] by -X[3]
53: "make X Axis":3→N;gsb "make a dot"
54: "make minor ticks":if XmodX[4]<X[3];1→N;gsb "make a dot"
55: "make major ticks":if XmodX[5]<X[3];2→N;gsb "make a dot"
56: "evaluate F(X)":U+1→U;X→T[2];cos(X)→T[1];gsb "SORT"
57: next X
```

Housekeeping Operations

Certain operations must be performed prior to printing the plot. These "houskeeping" operations include generating various dot patterns for axis and tick marks, evaluating the function over its entire domain, and sorting the resulting data pairs by Y into ascending orders.

Y-Axis and Y Tick Marks

The Y-axis is represented by a single dot. It is generated by calling the `"make a dot"` subroutine to evaluate 0, and giving the routine an index reference to string `X$[4]`, which the pattern will be stored in.

```
43: "generate Y axis dot":0→X;4→N;gsb "make a dot"
```

The Y tick marks are generated in a `for-next` loop running from +7 to −7 dot columns around the Y-axis. Steps of .9 are used rather than 1 to assure no dots are skipped in the process. The `"make a dot"` subroutine is used again, storing the dot pattern for major ticks in `X$[6]` and minor ticks (which extend from −4 to 4 dot rows around the Y axis) in `X$[5]`.

```
45: "generate Y axis tick marks":
46: for I=7 to -7 by -.9
47: "make major tick":I*X[3]→X;6→N;gsb "make a dot"
48: "make minor tick":if I<4 and I>-4;5→N;gsb "make a dot"
49: next I
```

```
51: "generate X axis, X tick marks and evaluate F(X)":
52: for X=X[2] to X[1] by -X[3]
53: "make X Axis":3→N;gsb "make a dot"
54: "make minor ticks":if XmodX[4]<X[3];1→N;gsb "make a dot"
55: "make major ticks":if XmodX[5]<X[3];2→N;gsb "make a dot"
56: "evaluate F(X)":U+1→U;X→T[2];cos(X)→T[1];gsb "SORT"
57: next X
```

The Major X Loop

A `for-next` loop running from the maximum X value (`X[2]`) to the minimum X value (`X[1]`) by steps of minus the X increment (`X[3]`) is used for several purposes. The X axis is generated and stored in `X$[3]`. The patterns representing minor and major ticks along the X-axis are generated and stored in `X$[1]` and `X$[2]` respectively. All three of these operations are performed using the `"make a dot"` subroutine.

```
53: "make X Axis":3→N;gsb "make a dot"
54: "make minor ticks":if XmodX[4]<X[3];1→N;gsb "make a dot"
55: "make major ticks":if XmodX[5]<X[3];2→N;gsb "make a dot"
```

The final operation in the loop is evaluating F(X) over the domain and sorting the resulting X-Y pairs by ascending Y values using the `"SORT"` subroutine. This program spends most of its time sorting.

```
56: "evaluate F(X)":U+1→U;X→T[2];cos(X)→T[1];gsb "SORT"
```

If data from a measuring operation is to be plotted instead of a functional mathematical relation, the F(X) evaluation can be eliminated, and a data reading operation added after this module. If presorted data is used, the `"SORT"` subroutine need not be employed, and the plotting process can be speeded up tremendously. If the data is even partially sorted, the plot will be speeded up significantly, as it will reduce time spent in the `"SORT"` routine.

```
59: "PLOTTING":dsp "PLOTTING"
60:
61: "make a plot":U→I
62: "set X$ index for output string":7→N
63: for Y=Y[2]+6Y[3] to Y[1] by -Y[3]
64: C$→X$[7]
65: gsb "axes"
66: gsb "test data"
67: "print the row of dots":wtb 701,27,"*b70W",X$[7]
68: next Y
```

Plotting

The first step in the plotting operation is assigning 7 to the string index for the "make a dot" routine (X$[7] is the output string).

```
61: "make a plot":U→I
62: "set X$ index for output string":7→N
```

The Y value represented by the current dot row is controlled by a decrementing for-next loop. The loop starts six dot rows above the actual maximum Y value of the plot. This allows the top most Y-tick to be centered relative to its labeling characters (the character matrix is 12 dot rows high).

```
63: for Y=Y[2]+6Y[3] to Y[1] by -Y[3]
```

The output string is cleared by assigning the clear string (C$) to replace the previous contents of the output string (X$[7]).

```
64: C$→X$[7]
```

The "axes" routine is then called. The "axes" routine fills in the appropriate dot patterns representing the axes and tick marks for the plot. Any data plotted will be written over the these patterns.

```
65: gsb "axes"
```

The "test data" routine is called to determine if any data pairs are best represented by points contained in this dot row. If so, they are added using the "make a dot" routine (called from "test data").

```
66: gsb "test data"
```

The dot row is then printed. A wtb statement is used to output the Set Graphics Mode escape sequence (ESC*b70W) and the output string (X$[7]). The 27 in the wtb statement sends an ESC character.

```
67: "print the row of dots":wtb 701,27,"*b70W",X$[7]
```

These steps are repeated for each valuue of Y. After the body of the plot is generated, a wtb statement is used to send a formfeed (decimal value = 12) to the printer. The program execution is then terminated.

Subroutines

"test data":

```
79: "test data":
80: if I<1;ret
81: if abs(R[I]-Y)>Y[3];ret
82: D[I]→X;gsb "make a dot"
83: I-1→I
84: gto "test data"
```

The "test data" routine is called for each value of Y to see if any data pairs are best represented by a dot in the current dot row. The first test is to test the index to the top of the remaining data (I). If it is less than 1, the data list has been exhausted, and we return to the main program.

```
80: if I<1;ret
```

If the data list has not been exhausted, the absolute value of the differance between the current Y value (Y) and the top of the remaining data list (R[I]) is compared to the Y increment value (Y[3]). If the difference is larger, we return to the main program, as no points to be plotted lie on this dot row.

```
81: if abs(R[I]-Y)>Y[3];ret
```

If the difference is less than the Y increment value (Y[3]), the Domain value (D[I]) associated with the Range value (R[I]) is assigned to X, and "make a dot" is called to add the dot to the dot row.

```
82: D[I]→X;gsb "make a dot"
```

One is now subtracted from the data reference Index (since the point has already been plotted) and the tests are done on the next data pair (more than one point may fall in a dot row.)

```
83: I-1→I
84: gto "test data"
```

```
"axes":
```

```
87: "axes":
88: "Y axis dot":if Y<=Y[2];X$[4]→X$[7]
89: "minor Y tick":if abs(YmodY[4])<=Y[3] and Y<=Y[2];X$[5]→X$[7]
90: "major Y tick":if abs(YmodY[5])<=Y[3] and Y<=Y[2];X$[6]→X$[7]
91: "next line labels Y axis":
92: if (Y-6Y[3])modY[5]<=Y[3] and abs(Y)>8Y[3];wrt 701.1,prnd(Y-6Y[3],-1),13
93: "generate X Axis":if Y<7Y[3] and Y>-8Y[3];gsb "X-axis"
94: ret
```

The "axes" subroutine is used to determine which dot pattern is to fill the output string before the plotted points are added. These patterns represent various elements of the axes for the plot. The labels are also controlled by this routine. A series of successive replacements determine which of the axis patterns to use. That is, strings containing various axis patterns replace each other until a test is failed. The pattern assigned by the last sucessful test is used as a base for plotting the points represented by the data pairs found on the particular dot row to be printed.

The first pattern is the dot representing the Y axis. The only condition for its use is a test for being in the plotting area. (Remember: the control loop starts above the active plotting area to allow for labeling the top Y-tick.) If the test is passed, the axis dot (X$[4]) replaces the current contents of the output string (X$[7]).

```
88: "Y axis dot":if Y<=Y[2];X$[4]→X$[7]
```

To determine whether to add a minor tick mark, we add a test based on the mod function using the value for minor tick spacing (Y[4]). If this test is passed, and we are in the active plotting area, the minor tick pattern replaces the Y axis pattern.

```
89: "minor Y tick":if abs(YmodY[4])<=Y[3] and Y<=Y[2];X$[5]→X$[7]
```

The test for a major Y-tick is the same as that for a minor Y-tick, but uses the value for major tick spacing (Y[5]).

```
90: "major Y tick":if abs(YmodY[5])<=Y[3] and Y<=Y[2];X$[6]→X$[7]
```

The three tests described above create the Y axis. Labels for the Y axis are controlled by a mod function based on the major Y-tick value, but six times the Y increment (6Y[3]) is subtracted from the Y value, to provide a six dot row offset – necessary for proper positioning of the labels. The label value is calculated using the same mod structure, and rounded to an appropriate number of decimal places (this will change from plot to plot). The second half of the test for labels is to eliminate printing labels over the X-axis.

```
92: if (Y-6Y[3])modY[5]<=Y[3] and abs(Y)>8Y[3];
                                wrt 701.1,prnd(Y-6Y[3],-1),13
```

The final test in this module calls the "X-axis" subroutine if the Y value is within an appropriate range.

```
93: "generate X Axis":if Y<7Y[3] and Y>-8Y[3];gsb "X-axis"
```

"X-axis":

```
96: "X-axis":
97: "minor X ticks":X$[1]→X$[7]
98: "major X ticks":if abs(Y)>=4Y[3];X$[2]→X$[7]
99: "X axis":if abs(Y)<Y[3]/2;X$[3]→X$[7]
100: "label X axis":if Y<-7Y[3] and Y>-8Y[3];gsb "X labels"
101: ret
```

The "X-axis" routine continues the successive replacement process. The pattern for minor X-ticks replaces the current output string (X$[1]→X$[7]).

```
97: "minor X ticks":X$[1]→X$[7]
```

If the Y value is more than 4 times the Y increment from the X-axis, major X-ticks replace the minor ticks in the output string (X$[2]→X$[7]).

```
98: "major X ticks":if abs(Y)>=4Y[3];X$[2]→X$[7]
```

If the Y value is within one half the Y increment of zero, the dot pattern representing the X-axis replaces the current output string (X$[3]→X$[7]).

```
99: "X axis":if abs(Y)<Y[3]/2;X$[3]→X$[7]
```

A final set of tests is made to position labels for the X-axis. If both tests are passed, the "X-labels" subroutine is called, and the X-axis is labeled.

```
100: "label X axis":if Y<-7Y[3] and Y>-8Y[3];gsb "X labels"
```

"make a dot":

```
104: "make a dot":
105: "convert X to dot":int((X+0)/X[3])→H
106: "calculate character position":int(H/8)+1→P
107: "find current & new characters":num(X$[N,P,P])→A:2↑(7-Hmod8)→B
108: "add dot if not overlapping":if Amod2B-AmodB=0;char(B+A)→X$[N,P,P]
109: ret
```

The "make a dot" routine performs a transformation from a real number (X) to a set of ASCII characters containing a dot pattern representing that number in a graphic sense. This transformation can be broken down into the following steps:

"convert x to dot":

By adding an offset (0) to X, to give all positive values, and dividing by the previously determined increment of $\dot{X}$ (X[3]) it is possible to convert the real number into a "best fit" dot number between 0 and 557. The int function eliminates spurious decimal portions of the number. This value is assigned to H.

```
105: "convert X to dot":int((X+0)/X[3])→H
```

"calculate character position":

An integer division function is used to derive the character position that will contain the dot H. Since character string references begin at 1 instead of 0, we add 1 to the result of the integer division.

```
106: "calculate character position":int(H/8)+1→P
```

```
"find current & new characters":
```

A num function of the X$ string currently indexed (N contains the index) is evaluated at the calculated character position. This is to allow any dots that may already exist in the area of the dot to be added to the string. The num function provides a decimal value equivalent to the ASCII character in the current position. This value is assigned to A.

A mod function is used to determine which dot (0 to 7) will best represent H. This value is subtracted from 7 to make the conversion from least significant digit to most significant digit numbering of H to the most significant bit to least significant bit convention in byte representation of the print head. Two is then raised to this power to convert bit reference to a decimal. This decimal value is assigned to B.

```
107: "find current & new characters":num(X$[N,P,P])→A;2↑(7-Hmod8)→B
```

```
"add dot if not overlapping":
```

As long as the dot patterns represented by decimal numbers A and B have no common elements, they can simply be added. However, if the bit represented by B is already set in A, adding A and B will generate a shift of the bit, thus creating an error in the transformation. Therefore, a test for common bits is performed. If the test yields 0, there are no bits common to A and B, and char (A+B) provides an ASCII character representing the combined bit patterns which is filled into X$[N, P, P]. If the test is non-zero, the bit represented by B is already set in A, and the original pattern in X$[N, P, P] need not be replaced.

```
108: "add dot if not overlapping":if Amod2B-AmodB=0;
                                          char(B+A)→X$[N,P,P]
```

"X labels":

```
111: "X labels":
112: for X=X[1]+X[5] to X[2]-X[5] by X[5]
113: "set mode":fxd 0
114: "numeric to string conversion":str(prnd(X,0))→T$[1]
115: "decimal to dot conversion":(X+0)/X[3]→H
116: "dot to character position conversion":int(H/8*(8/7))→P
117: "add label to label string":if abs(X)>X[5]/2;T$[1]→L$[P,P+len(T$[1])]
118: next X
119: "print label string":wtb 701,L$,13
120: "restore mode":fxd 1
121: ret
```

When the "X-axis" routine determines that it is time to label the X-axis, this routine calculates the labels, converts them into strings, combines all the labels into L$ and prints L$.

A for-next loop from the minimum X value plus the major tick spacing to the maximum X value minus the major tick spacing runs by steps of the major tick spacing. The one tick spacing margin at either end is to prevent attempts to access characters outside the dimensions of the labeling string.

```
112: for X=X[1]+X[5] to X[2]-X[5] by X[5]
```

"set mode":

An appropriate display mode is selected for the labels. This will vary from plot to plot.

```
113: "set mode":fxd 0
```

"numeric to string conversion":

The X value is rounded to an appropriate number of places and converted to a string. This is stored in tempory variable T$[1].

```
114: "numeric to string conversion":str(prnd(X,0))→T$[1]
```

"decimal to dot conversion":

The real value X is converted to a dot reference using the technique discussed in the "make a dot" routine.

```
115: "decimal to dot conversion":(X+0)/X[3]→H
```

"dot to character position conversion":

An integer division function is used find the appropriate character position for the label. The 8/7 factor is to correct for the 80 characters in a line of printed character compared to 70 bytes in dot row.

```
116: "dot to character position conversion":int(H/8*(8/7))→P
```

"add label to string":

If the label passes a test to prevent labeling over the Y axis, the label is added to the label string (L$), from character position P to P+ len(T$[1]).

```
117: "add label to label string":if abs(X)>X[5]/2;
                                        T$[1]→L$[P,P+len(T$[1])]
```

"print label string":

After all the labels have been added to the string, it is printed using a wtb function. A carriage return (decimal value 13) is sent to the printer which prints the label and then rolls the paper back to its original position. This allows the graph to be produced with no discontinuities.

```
119: "print label string":wtb 701,L$,13
```

"restore mode":

The display mode is restored to that selected for Y-labels.

```
120: "restore mode":fxd 1
```

"SORT":

```
124: "SORT":
125: 1→F;U→C
126: "begin sort":
127: if U=1 or U=F;T[1]→R[U];T[2]→D[U];ret
128: if T[1]<R[1];1→P;gto "ripple up"
129: if T[1]>R[U-1];T[1]→R[U];T[2]→D[U];ret
130: int((C-F)/2)+F→M
131: if T[1]>R[M];gto "above"
132: if T[1]<R[M];gto "below"
133: if T[1]=R[M];M→P;gto "ripple up"
134: dsp "error";stp
135:
136: "above":
137: if T[1]<R[M+1];M+1→P;gto "ripple up"
138: M+1→F;gto "begin sort"
139:
140: "below":
141: if T[1]>R[M-1];M→P;gto "ripple up"
142: M-1→C;gto "begin sort"
143:
144: "ripple up":
145: for Z=U to P+1 by -1
146: R[Z-1]→R[Z];D[Z-1]→D[Z]
147: next Z
148: T[1]→R[P];T[2]→D[P]
149: ret
```

A modified binary search is used to determine the correct point for addition of data to the existing data list. Once the proper location is found, any data above the data pair to be added is shifted up to clear a space and the point is added. This method provides better than three to one time advantage over a bubble sort. A linked list approach provides a very slight time advantage but increases the array size. The binary search was selected as an optimized solution for general applications.

Binary Search

A binary search is a fast method for searching through ordered lists. The list is divided in half repeatedly until the element of interest is found. At each iteration, the element of comparison may be above, below, or equal to the list element being examined. If above or below, a new center of the list is created (upper or lower half) and the test is performed again. If equal, the element is found.

In converting the binary search to a sorting technique (possible since we are ordering the list as we create it) it is necessary to add two tests. If the element to be added to the list is above the test point, it must be determined if the element to be added belongs below the element immediately above the test point. This "between" case has a reflection for the below test element case. If either case is true, the list is shifted up to allow this insertion. If the element to be added is equal to the test element, we arbitarialy insert the new element below the test element.

In "SORT", F is the "floor", or bottom of the current list, C is the "ceiling", or top to the current list, and M is the middle. To initialize the sort routine, U, which is the current number of elements in the list, is assigned to C, and 1 is assigned to F.

```
125: 1→F;U→C
```

```
"test for special cases":
```

If the list is just started (U = 1) or if the floor is at the top of the list (U = F), add the new element to the top of the list and return to the main program.

```
127: if U=1 or U=F;T[1]→R[U];T[2]→D[U];ret
```

If the new element is below the bottom of the list, shift all the elements in the list up one position and add the new element (done in "ripple up".)

```
128: if T[1]<R[1];1→P;gto "ripple up"
```

```
"test for already ordered":
```

If the new element is greater than the top of the list, add it as the new top of the list, and return to the main program.

```
129: if T[1]>R[U-1];T[1]→R[U];T[2]→D[U];ret
```

```
"find middle":
```

If neither of the above sets of conditions is true, find the middle of the list and begin sorting.

```
130: int((C-F)/2)+F→M
```

"above":

If the element to be added is above the test element (it has to be to get to this segment) and below the element directly above the test element, set the pointer above the test element, and shift all elements from the pointer up by one and add new element (done in "ripple up").

If the above secondary condition is not true, set the floor to the element above the test element and go back to the beginning of the sort.

```
136: "above":
137: if T[1]<R[M+1];M+1→P;gto "ripple up"
138: M+1→F;gto "begin sort"
```

"below":

If the element to be added is below the test element (it has to be to get this segment) and is above the element immediately below the test element, shift all elements from the pointer up by one and add the new element (done in "ripple up").

If the above secondary condition is not true, set the ceiling to one element below the test element and go back to the beginning of the test.

```
140: "below":
141: if T[1]>R[M-1];M→P;gto "ripple up"
142: M-1→C;gto "begin sort"
```

"ripple up":

This section shifts all the elements above the pointer (P) up one position and adds a new element at the current pointer position.

```
144: "ripple up":
145: for Z=U to P+1 by -1
146: R[Z-1]→R[Z];D[Z-1]→D[Z]
147: next Z
148: T[1]→R[P];T[2]→D[P]
149: ret
```

Non-Factory Address Settings

In a system where the address code 1 on select code 7 is already taken, or where it is necessary to change the HP-IB card select code from 7, it is necessary to modify the select code and/or address in commands such as `list #` and statements such as `wrt` and `wtb`.

Changing the interface card select code in

```
wrt 701
```

from 7 to 6 would give

```
wrt 601
```

Changing the printer address in

```
list # 701
```

from 01 to 00 would give

```
list # 700
```

Changing interface card select code and printer address in

```
wtb 701
```

from interface card select code = 7, printer address = 1 to select code = 6, address = 0 gives

```
wtb 600
```

Non-Factory Address Settings and Option 001 Operation

Option 001 Operation

If a 98032 Option 076 interface is used instead of the HP-IB, all device address code reference must be eliminated.

Thus

```
wrt 701
```

for factory address settings (interface and select code = 07, printer address = 01)

would be changed to

```
wrt 6
```

for the 98032 factory select code setting of 06.

Similar modifications apply to all codes involving output address and select code.

APPENDIX B

The Character Sets

The printer, keyboard, and display for the 9825 use anomolous implementations of U.S. ASCII. This can lead to some complications when using an external printer such as the 9876. The following table can be used to spot differences in the character set and determine the best method for circumventing the problems. Additionally, a set of a Special Function Key definitions are provided on the utility tape to redefine the ASCII equivalents of the four anomolous characters on the 9825 listings to provide listings with the character set you are already familiar with. See the 9876-9825 Utility Pack for information on key definitions provided for listing programs.

Decimal	9825	9876	Notes	Decimal	9825	9876	Notes
0				24			206 [1]
1			185 [1]	25			219 [1]
2				26			207 [1]
3			182 [1]	27			211 [1]
4				28			215
5				29			
6				30			187 [1]
7			183 [1]	31			127 [1]
8				32			
9				33	!	!	
10				34	"	"	
11				35	#	#	
12			[2]	36	$	$	
13				37	%	%	
14			[2]	38	&	&	
15			[2]	39	'	'	
16			[2]	40	(	(	
17			190 [1]	41	)	)	
18			[2]	42	*	*	
19			208 [1]	43	+	+	
20			212 [1]	44	,	,	
21			216 [1]	45	-	-	
22			204 [1]	46	.	.	
23			218 [1]	47	/	/	

1 This number is the 9876A 8 bit European extended character set equivalent value for the 9825 character shown.

2 See the Table of Character Redefinitions on page 5-15.

Decimal	9825	9876	Notes	Decimal	9825	9876	Notes
48	0	0		73	I	I	
49	1	1		74	J	J	
50	2	2		75	K	K	
51	3	3		76	L	L	
52	4	4		77	M	M	
53	5	5		78	N	N	
54	6	6		79	O	O	
55	7	7		80	P	P	
56	8	8		81	Q	Q	
57	9	9		82	R	R	
58	:	:		83	S	S	
59	;	;		84	T	T	
60	<	<		85	U	U	
61	=	=		86	V	V	
62	>	>		87	W	W	
63	?	?		88	X	X	
64	@	@		89	Y	Y	
65	A	A		90	Z	Z	
66	B	B		91	[	[	
67	C	C		92	\	\	2,3
68	D	D		93	]	]	
69	E	E		94	↑	^	3
70	F	F		95	←		
71	G	G		96	'	`	
72	H	H		97	a	a	

2 See the Table of Character Redefinitions on page 5-15.

3 Provided for on redefined keys supplied for program listings – Track 0, File 2.

Decimal	9825	9876	Notes	Decimal	9825	9876	Notes
98	b	b		123	╓	{	2,3
99	c	c		124	╷	|	
100	d	d		125	┼	}	3
101	e	e		126	Σ	~	2
102	f	f		127	├	▒	
103	g	g					
104	h	h					
105	i	i					
106	j	j					
107	k	k					
108	l	l					
109	m	m					
110	n	n					
111	o	o					
112	p	p					
113	q	q					
114	r	r					
115	s	s					
116	t	t					
117	u	u					
118	v	v					
119	w	w					
120	x	x					
121	y	y					
122	z	z					

2 See the Table of Character Redefinitions on page 5-15.

3 Provided for on redefined keys supplied for program listings – Track 0, File 2.

SUBJECT INDEX

Manual Overview

Basic Printing Operations **Section I**

Simple printing operations include listings and catalogues, using WRITE and WRITE BINARY statements, and basic character plotting. This section deals with printing operations that reside entirely in the computer, as opposed to the special capabilities of the 9876A Thermal Graphics Printer.

Listings and Catalogues **Chapter 1**

Listing programs to a 9876A is discussed. Catalogues of Mass Storage Units are also discussed.

Basic Printing **Chapter 2**

The concept of a character field, and the default forms for computer output of data are introduced. How various types of data are dealt with in WRITE (wrt) and WRITE BINARY (wtb) statements are discussed.

Character Plots of F(X) with a Vertical X-axis **Chapter 3**

The necessary methods of scaling the results of the evaluation of a function are presented as well as the procedures for labeling the graph. The algorithm for generating the graph a line at a time is discussed.

Advanced Printing **Section II**

This section will cover two methods for exercising precise control over the printer. The first is formatting and involves the use of WRITE (wrt) and FORMAT (fmt) statements. The formatting functions are resident in the computer. The second technique uses Escape Sequences and Control Characters to control the printer. Format statements are used to control the flow of data, while Escape Sequence and Control Characters are imbedded in the data stream.

Subject Index

a

b

c

d

e

f

g

h

i

l

m

n

o

p

r

s

t

u

V

W

Z