EPSON®

FX SERIES PRINTER

User's Manual

VOLUME 1 TUTORIAL

By
David A. Kater
EduKater

FCC COMPLIANCE STATEMENT FOR AMERICAN USERS

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful:

"How to Identify and Resolve Radio-TV Interference Problems."

This booklet is available from the U.S. Government Printing Office, Washington DC 20402. Stock No. 004-000-00345-4.

P8294017

Preface

The User's Manual for the FX Series printers consists of two volumes: Tutorial and Reference. This volume, the Tutorial, is arranged in the following logical groupings:

Introduction (for everyone)
Programmer's Easy Lesson (for experienced users)
Hardware description: Chapter 1
Software introduction: Chapter 2
Control of the way characters look: Chapters 3 to 6
Control of the way pages look: Chapters 7 to 9
Printer graphics: Chapters 10 to 14
User-defined characters: Chapters 15 and 16
Using everything together: Chapter 17

A complete table of contents for this volume is after this preface. For your convenience, there is an index at the end of each volume covering the complete two-volume set. You can therefore find all the references to any topic in either one.

Conventions Used in This Manual

We provide sample BASIC programs that allow you to see how various commands control the printer's capabilities. Frequently we start with a few program lines and then make several changes and additions to end up with a substantial program. We suggest that you use your SAVE command after each change to prevent losing programs because of power fluctuations or other accidents. When you can RUN a program, we show the results you should expect.

In our sample programs, we use Microsoft'" BASIC, which is widely used in personal computers. Because there are several slightly different versions of Microsoft BASIC and because your computer

may use a version of BASIC other than Microsoft, you may need to modify some of the programs in this manual before they will run. Appendix F offers help, as do the next several paragraphs.

Methods for sending BASIC print and listing commands to the screen and to the printer vary widely. We have used PRINT and LIST as the commands for the screen display, and LPRINT and LLIST as commands for the printer. You may have to change those to the form used by your system.

If, for example, your system uses the PR#1 and PR#0 commands, you will need to change all instances of PRINT in our programs. Since we use PRINT to report progress to your screen and that command does not affect the printing, the easiest modification is to delete such PRINT statements. For example,

```
30 FOR D=1 TO 17: PRINT "ROW";D
```

would become:

```
39 FOR D=1 TO 17
```

because the only purpose of the PRINT statement is to display on the screen information that is not absolutely essential to the program.

Any BASIC system automatically provides a carriage return (and some BASICs add a line feed) after every program line that includes a PRINT or LPRINT command, whether that line prints text or not. To prevent the carriage return, we have you place a semicolon at the end of such program lines. You will see this technique throughout the manual.

A few versions of BASIC use semicolons between any two control codes that fall on one program line, as in:

```
LPRINT CHR$(27);CHR$(52)
```

If you use such a version of BASIC, you will need to add semicolons as appropriate.

After the ESCape code-CHR$(27)-the FX always expects another code. The second code tells the printer which mode to turn on or off, and you may enter it in either of two formats. One format is like the ESCape code-you use a number in parentheses after CHR$, such as CHR$(l). The other format is shorter since it uses only an alphanumeric symbol within quotation marks, such as “E” or “@“. We usually use the latter format.

This format allows you to shorten a program line by combining a command and its print string. In the case of Double-Strike, for instance, the quoted letter "G" turns the mode on and "H" turns it off. To see how combining the code with a print string works, compare:

```
10 LPRINT CHR$(27)"G";"DOUBLE-STRIKE PRINT"
```

with:

```
10 LPRINT CHR$(27)"GDOUBLE-STRIKE PRINT"
```

The second program line may look peculiar, but it gives the same output that the first version does. The G is not printed on the paper; instead, it is interpreted by the printer as part of the ESCape sequence.

In long programs with DATA statements or subroutines, we use END after the line that is executed last, but older BASIC systems require the use of STOP at such points. If yours is one of these, you should change our ENDs to STOPs.

When the presence of one or more blank spaces in a program line is especially important, we use a special character (␣, pronounced 'blank') to represent the spaces. This makes it easier for you to count the number you need. For example, the following:

```
␣␣␣␣␣␣␣␣SAMPLE STRING"
```

means that you should type in one blank space for each ␣:

```
        SAMPLE STRING"
```

The use of the ␣ symbol makes it easy for you to count the eight spaces needed between the quotation mark and the beginning of the first word. The ␣ also calls your attention to single blank spaces that are needed immediately before or after a quotation mark. For example, the following makes clear that you must type one space between the quotation mark and the word "and':

```
LPRINT "␣AND EASY TO TURN OFF"
```

When we include a programming REMark in a program line, it is always preceded by an apostrophe ('), the short form of the BASIC command, REM. For example, we use:

```
10 LPRINT CHR$(27)"@"    'Reset Code
```

and

```
99' Data lines for graphics
```

The computer ignores these remarks; they merely serve to help programmers understand at a glance the way a program is working. You may type them in or not, depending on whether you think you will want them in the future.

We use the caret symbol (^) to indicate exponents. For example:

```
x = Y^2
```

means let X equal Y raised to the second power. Some computer systems use an up-arrow (t), which prints as a left bracket ([) on FX printers.

At the end of each chapter, a Summary section provides a concise review of the chapter's subject matter and a list of the control codes (if any) that have been covered. For listings of the control codes in numerical order and in functional groupings, see Appendixes B and C.

When we refer to an FX mode by name, we capitalize it:

Compressed Mode
Italic Mode
Pica Mode
Script Modes

and, for clarity, we capitalize such names even when the word mode does not appear: Script characters and Italic print.

FX Series Printer User's Manual
Volume 1 Contents

List of Figures

List of Tables

Introduction

FX Features

Epson's MX series of printers attracted enough attention to become the most popular line of printers in the industry. Our FX printers follow in the same grand tradition. The FX printers' power-packed assortment of features includes:

- Upward compatibility with most MX III features
- Several different print modes that can be combined to produce a variety of print styles. These include:

  Roman and Italic print fonts
  Six different print pitches
  Two kinds of bold printing

- Master Select feature for instant use of any one of 16 popular print combinations
- Proportionally spaced characters for professional looking documents
- Easy-to-use Underline and Super/Subscript Modes
- Detailed forms handling capability, including the setting of horizontal and vertical tabs, margins, form length, a skip-over-perforation feature, and variable line feeds
- Up to 233 characters per line with the FX-100™ for spreadsheet users
- User-definable character sets. With this powerful feature you can create your own alphabets and special symbols
- High-resolution graphics capability with six densities to let you create your own charts, diagrams, figures, and illustrations
- International character sets
- Typewriter simulation mode with the FX-80™

- Program debugging mode (hexadecimal dump of codes received from the computer)
- Fast print speed-160 characters per second-for rapid processing of documents
- 2K print buffer for smooth operation
- Adjustable tractor unit for narrow forms
- Both friction- and tractor-feed capability
- Replaceable print head
- Easy-to-reach DIP switches to customize printer features.
- Epson reliability, quality, and support

In short, the FX is loaded with features **that** will challenge your ability to put them to work. This manual can help you use one or all of them.

Inside the Printer

The FX printers contain two kinds of internal memory: ROM (Read Only Memory) and RAM (Random Access Memory). There are 12K bytes (approximately 12,000 characters) of ROM. This unchangeable memory contains all the logic required for the various print features as well as the patterns for all the built-in character sets.

The FX also contains a RAM memory buffer that stores up to 2K bytes of text and printer commands as they are received from the computer. This frees your computer so that you can continue working while the FX is printing. You can also use RAM another way; you can define your own set of characters and then store them in RAM so that you can print them at will.

It is always tempting (though not always wise) to start playing with a new printer the instant it is out of the box. Because the FX is a sophisticated piece of equipment, it is important that you understand what the printer will do and how to operate it before you start printing.

Inside This Manual

This manual will guide you on a carefully planned tour of the various features of the FX printers. In these pages, you **can** learn how to use your computer to control the printer for a large variety of applications.

You can use this manual as a reference, a tutorial study guide, or some combination of the two.

- For those of you who want to use the printer for one simple application (like listing BASIC programs or using a word processing package), a description of the hardware and an overview of the software may be all that's necessary. In this case, you need only Chapter 1, the Quick Reference Card at the back of Volume 2, and a knowledge of the program you are using. You can always learn about the FX's advanced features at a later time. (You might, for instance, someday want to modify a word processing software package so that its printer driver uses special FX features.) The lessons will be waiting for you.

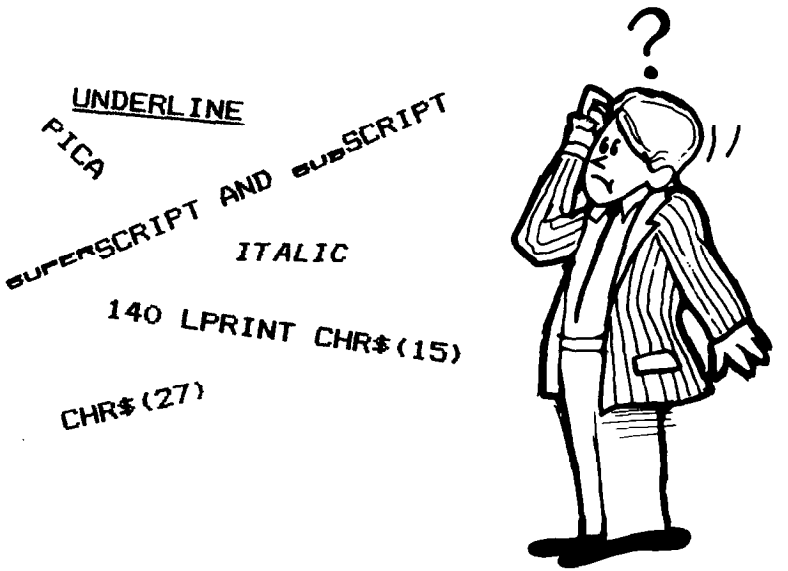
- For those who prefer to roll up their sleeves and see how the printer works, we've included sample programs to demonstrate each of the printer's features.

- For those who want only a quick, and easy reference, the comprehensive Table of Contents, the Appendixes, and the Index provide ready access to information.

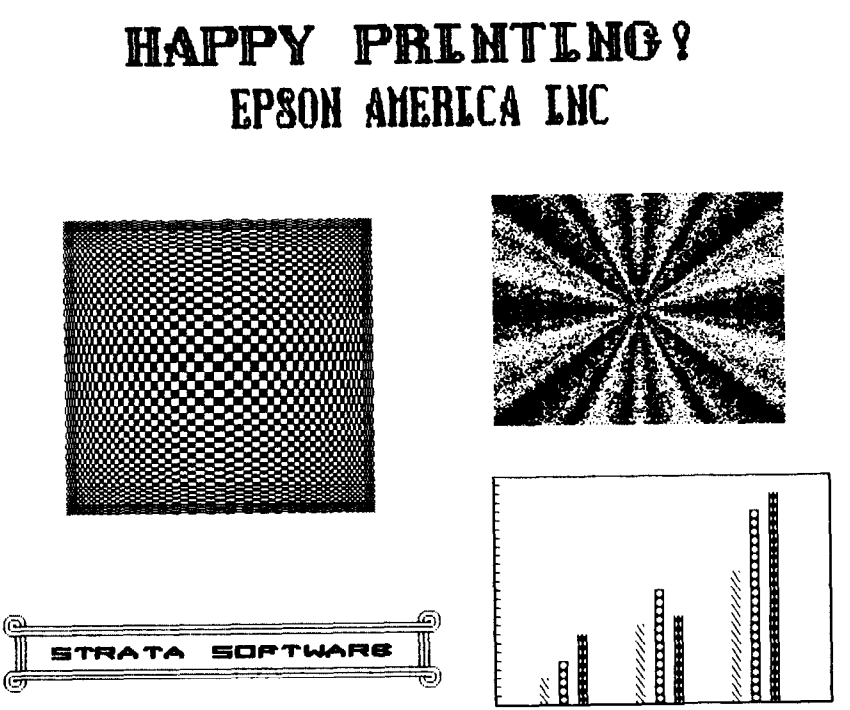
- For computer professionals and other experienced users who simply can't wait to find out what the printer will do, regardless of the consequences, we have a special section entitled "Programmer's Easy Lesson." It gets you up and running fast, then turns you loose on a program that demonstrates several of the printer's features. This program, the Appendixes, and the Quick Reference Card will bring you quickly up to speed.

- For those who are already familiar with the MX or RX series of printers, Appendix E provides a summary of the differences between the FX, the RX, and the MX.

Think of the manual as your personal guide in your exploration of the FX's many features.



For a preview of what your programs can produce, take a look at the following potpourri of print modes and graphics.



PICA

SINGLE-STRIKE 1234567890ABCDEFGHIJKLMN / DOUBLE-STRIKE 1234567890ABCDEFGHIJKLMN
SINGLE-STRIKE EMPHASIZED 1234567890ABC / DOUBLE-STRIKE EMPHASIZED 1234567890ABC
SINGLE-STRIKE EMPHASIZED PROPORTIONAL 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghi
SS EXPANDED 1234567 / DS EXPANDED 1234567
SS EXP EMPHASIZED 1 / DS EXP EMPHASIZED 1
UNDERLINE 1234567 / SUBSCRIPT 1234567 / SUPERSCRIPT 1234567 / *ITALICS 1234567*

ELITE

SINGLE-STRIKE 1234567890ABCDEFGHIJKLMNOPQRSTUV / DOUBLE-STRIKE 1234567890ABCDEFGHIJKLMNOPQRSTUV
SS EXPANDED 1234567890A / DS EXPANDED 1234567890A
UNDERLINE 1234567890A / SUBSCRIPT 1234567890A / SUPERSCRIPT 1234567890A / *ITALICS 1234567890A*

COMPRESSED

SINGLE-STRIKE 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop / DOUBLE-STRIKE 1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnop
SS EXPANDED 1234567890ABCDEFGHIJK / DS EXPANDED 1234567890ABCDEFGHIJK
UNDERLINE 1234567890ABCDEFGHIJK / SUBSCRIPT 1234567890ABCDEFGHIJK / SUPERSCRIPT 1234567890ABCDEFGHIJK / *ITALICS 1234567890ABCDEFGHIJK*

Programmer's Easy Lesson

Before you start, note that we haven't claimed that one easy lesson will make you an FX maestro. It takes more than one lesson to learn the full value of the feature-packed FX printer. In fact, the more time you spend with this manual, the more your printer will cooperate with your every command. But some of you want to see something from a new printer right now-no matter what. The next few pages are especially for you.

If you get stuck, the proper set-up procedures are covered in full in Chapter 1.

First Steps

1. Make all connections with the power OFF! Connect your FX to your computer via the printer cable that you purchased separately. (Some computers require special printer interface boards, also purchased separately).

2. To use continuous-feed printer paper with pin-feed holes, set the friction-control lever and the paper bail toward the front of the printer. If you are using the FX-80, pull the paper under the plastic separator and through the paper path. If you are using the FX-100, you may need to first install the tractor unit, then pull the paper under it. In case the paper starts to jam on either model, refer to Chapter 1 for tips on inserting paper. As you straighten the paper, you will probably need to adjust the pin feeders.

   To use either a single sheet of paper or roll paper without pin-feed holes on the FX-80, first move the pin feeders out of the way. If a tractor unit is installed on your FX-100, you will need to remove it. Then, for either model, push the friction-control lever toward the rear of the printer, pull the paper bail forward, and insert the paper under the plastic separator. Use the manual-feed knob to

feed the paper through. If you use single sheets of paper, the paper-out sensor will cause a beep and stop the printing whenever the bottom edge passes the sensor. You can shut off the sensor by changing DIP switches as shown in Chapter 1.

3. Turn the printer and computer on and load a short BASIC program. Then send a listing to the printer (using LLIST, LIST "P", or whatever your computer's listing command is). You should get a single-spaced listing. If the printout is double-spaced or printed without line spacing, you'll have to change a DIP switch.

Since there are many implementations of the BASIC programming language, it is impossible to write one set of programs that will work on every computer system. This means you may need to modify our programs to suit your system. In Appendix F we discuss such compatibility problems and suggest solutions for several popular computers.

Ticket Program

Here is an example program, written in BASIC, that shows off a lot of the FX printer's features. The program can give you a good survey of print control. If you don't understand one or more features, you can check the index to find what part of this manual covers it.

```
10   N=29: E$=CHR$(27): H$=CHR$(137)
20  LPRINT  E$"1";E$"D"CHR$(26)CHR$(1);
30   LPRINT E$":"CHR$(0)CHR$(0)CHR$(B);
40  LPRINT  E$"%"CHR$(1)CHR$(0);
50  LPRINT  E$"&"CHR$(0)"0:";
60   FOR Y=1 TO 11: LPRINT CHR$(1l);
70   FOR X=1 TO 11: READ D: LPRINT G-K@(D);: NEXT X
80   NEXT Y: LPRINT E$"U1";
90   FOR X=1 TO N: LPRINT CHR$(95);: NEXT X:
     LPRINT  E$"A"CHR$(6)
100 LPRINT "7"H$"   "ø9"; E$"1"
110  LPRINT    "7ø";:  FOR X=1 TO 25: LPRINT ":";:
     NEXT X: LPRINT      H$"ø9"
120  LPRINT "7 :"H$": 9"
```

Figure Easy-1. FX ticket program

```
130  LPRINT "7 :"; E$" !X"; E$"4";    " ØØ' " E$" - 1";
140  LPRINT "TICKET TO SUCCESS!"; E$"!@"; E$"5";
     E$"+0";
150  LPRINT H$": 9": FOR X=1 TO 2: LPRINT "7 :"H$
      ":9": NEXT X
160  LPRINT "7 :";: LPRINT CHR$(14);E$;"E";
     "ØØØ 2";
170  LPRINT CHR$(20);E$"F";E$"S1";    "Ø SERIES";
     E$"T";H$; H$; 9"
180 LPRINT "4 :";: LPRINT CHR$(14);E$;"E";
     "ØØ 1 3";
190 LPRINT CHR$(20); E$"F"; E$"S1";    "Ø PRINTERS";
     E$ "T"; H$;": 5"
200 LPRINT "6 :"; H$;": 8": LPRINT "7 :";H$;": 9"
210 LPRINT "7  :   " ØØ CHR$ (15)     "BY" Ø "" E$"!X";
     "EPSON AMERICA".
220  LPRINT E$"S0"; E$;"!@";      "Ø  INC   "E$"T"; H$;":  9"
230  LPRINT "7 :"; H$;": 9": LPRINT "7      :ØØØØ "E$"!Q";
240   LPRINT "GENERAL ADMISSION"; E$"!@"; H$": 9":
     LPRINT "7:"H$":9",
250  LPRINT   "Ø";:  FOR x=1 TO 25: LPRINT ":";:
     NEXT X: LPRINT H$   " Ø9"
260 LPRINT   "7   "; Ø H$;    "   ØØ    9";  E$"A" CHR$(1)
270 FOR X=1 TO N: LPRINT CHR$(95);: NEXT X: LPRINT
280 LPRINT  E$"@":END
290 '***  FX XXX
300  DATA  64,0,127,0,127,0,64,0,65,0,112   :'0   F
310 DATA   1,0,127,0,127,0,65,0,96,0,0       :'1     F
320 DATA    64,48,72,54,73,6,1,4,72,48,64      :'2      X
330 DATA   1,6,11,48,64,48,73,54,11,6,1      :'3        X
340 '*** TICKET BORDERS ***
350 DATA   64,0,64,0,64,0,32,0,16,8,7        :'4
360 DATA   7,8,16,0,32,0,64,0,64,0,64        :'5
370 DATA  1,0,1,0,1,0,2,0,4,8,112           :'6
380 DATA   127,0,0,0,0,0,0,0,0,0,0,0         :'7
390 DATA    112,8,4,0,2,0,1,0,1,0,1          :'8
400 DATA   0,0,0,0,0,0,0,0,0,0,0,127
410 DATA    73,0,20,34,0,73,0,34,20,0,73     :':
```

Figure Easy-1. FX ticket program (concluded)

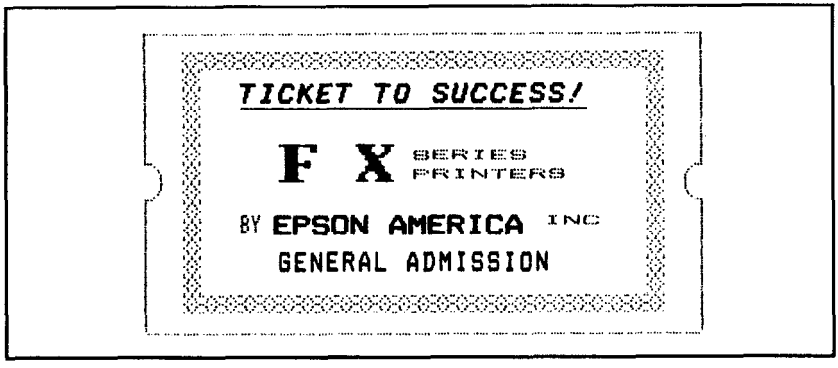


Figure Easy-2. Ticket to success

Ticket Program Description

This is not a complete explanation of the program. That's what the rest of the manual is for. But this brief, line-by-line description should help those of you who wish to analyze the program.

10 Stores values in variables for easy access. E$ holds the ESCape code, CHR$(27).

20 Uses ESCape **"1"** to set the line spacing to 7/216-inch and the ESCape "D" sequence to set a horizontal tab stop at column 26.

30 Uses the ESCape ":" sequence to copy the entire ROM character set into RAM.

40 Designates RAM as the source for the active character set.

50 Prepares the printer to redefine characters "0" through ":".

60 Sets a counter for the **11** letters being defined, and selects the attribute byte of each new character.

70 Reads the data that defines the letters (11 sets of ll).(See Chapter 15 for additional information on lines 30 through 70.)

80 Turns on the Unidirectional Print Mode.

90 Prints the top of the ticket and sets the line spacing to 6/72-inch.

100 Prints the newly defined symbol "7" (left ticket border), tabs to the next stop, prints the other border (9) and sets the line spacing back to 7/72-inch.

110 Prints the outside border, then the top of the inside border (which was defined as the ":" character).

120 Prints another line of borders.

130 Prints more borders, then uses the Master Select to turn on Emphasized Double-Strike Pica. Also turns on Italic and Underline Modes.

140 Prints TICKET TO SUCCESS, then resets the FX to its defaults, including Pica, but does not affect the redefined characters.

150 Produces two more border lines.

160 Prints the upper half of the FX letters in Expanded Emphasized printing. The user-defined character 0 produces the top of the F and 2 produces the top of the X.

170 Turns off Expanded and Emphasized Modes and prints SERIES in Superscript Mode and then prints the right side of the border.

180 Prints the bottom half of the FX letters.

190 Turns OFF the codes, prints PRINTERS in Subscript, then prints a border.

200 Prints borders.

210 Prints borders, then switches to Compressed and prints BY. Sets, with ESCape " !X", Emphasized Double-Strike Pica, and prints EPSON. This new mode automatically turns off Compressed.

220 Sets Superscript Mode (Escape "S0"), returns to normal print (Escape"!@"), prints INC in Superscript, then cancels Script Mode, then prints borders.

230 Prints another line of borders, then sets Master Select with ESCape "!Q", giving Double-Strike Elite.

240 Prints GENERAL ADMISSION, resets the FX to its defaults, and prints right borders.

250 Prints the outside borders and the bottom of the inside border.

260 Prints the outside borders and sets line spacing to 1/72-inch.

270 Prints the bottom of the outside border.

280 Returns the printer to its defaults.

300-330 Provides data for the FX letters as user-defined characters 0-3.

350-410 Provides data for the ticket borders.

Chapter 1
The FX Printers

Once you've unpacked your new printer, the first thing you should do is make sure you have all of the parts. With the FX-80 or FX-100 printer, you should receive the items shown in Figure 1-1:

1. The printer itself
2. A manual-feed knob
3. A paper separator
4. Two protective lids
5. One ribbon cartridge (in a box)
6. This FX Series Printer User's Manual

The FX-80 has a tractor built into its platen for handling continuous-feed paper between 9 ½ and 10 inches in width. To handle narrower continuous-feed paper, you must purchase the optional FX-80 tractor unit.

On an FX-100 you will find these items installed:

7. A tractor unit
8. A dust cover for the tractor unit

As you unpack your printer, you may want to save all protective plastic, paper, and cardboard to use in the future for repacking. Figure 1-2 displays with labels the parts of each model that we discuss in this chapter.

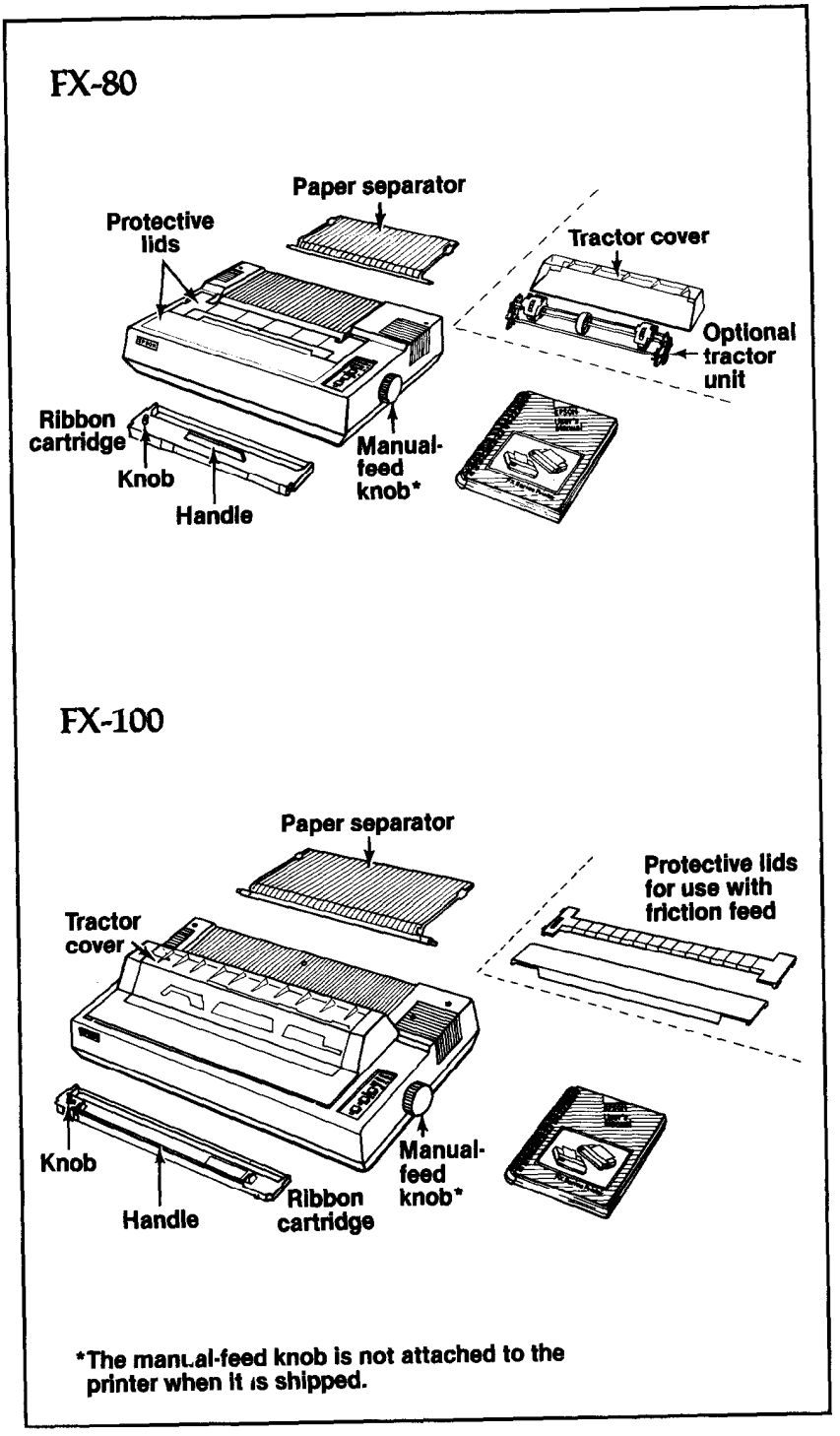
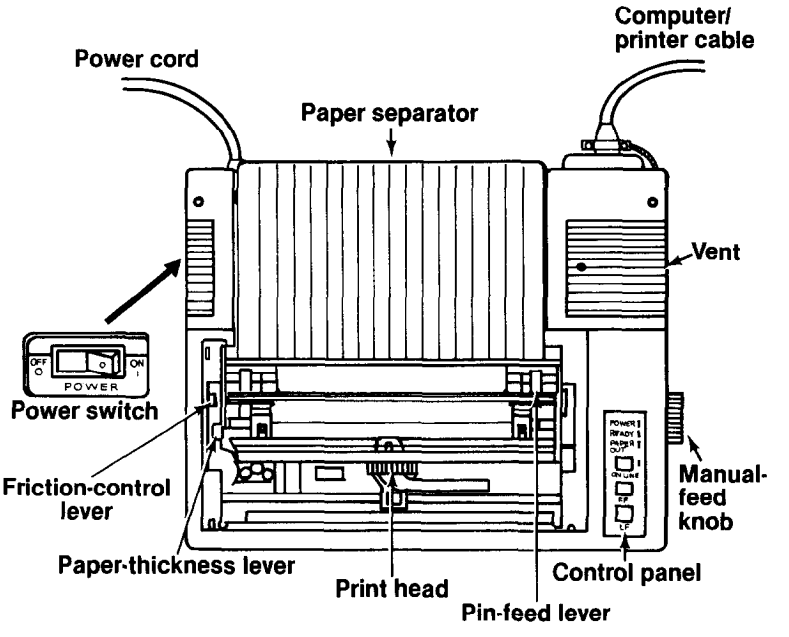


Figure 1-1. The FX-80 and FX-100 printers

FX-80



FX-100

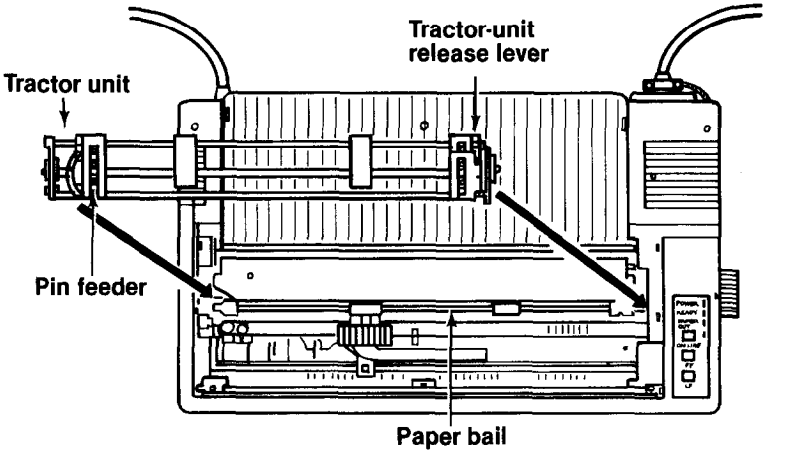


Figure 1-2. Printer parts

Additional Supplies and Accessories

The following items may be purchased separately from your Epson dealer:

Printer cable or interface kit. Each computer system has its own way of connecting to a printer. Some computers need a cable only, others require both a cable and board. The FX printers use the Centronics standard parallel interface scheme described in Appendix K. If your computer expects to communicate through a serial rather than through a parallel interface, you must purchase a serial board for your FX. Your Epson dealer stocks a variety of FX interface boards as well as cables.

Printer paper. FX printers are designed to accommodate several types and sizes of paper. Both printers include tractors so that you can use continuous-feed paper with pin-feed holes and friction mechanisms so that you can use paper without these holes.

Ribbon cartridge replacement. The expected life of a cartridge is three million characters (roughly 1,000 pages of text).

Print head replacement. The expected life of a print head is one hundred million characters (over 30,000 pages of text).

Roll paper holder. For the FX-80, you may purchase an optional roll-paper holder.

Printer Location

Naturally, your printer must sit somewhere near the computer (the length of the cable is the limiting factor), but there are other considerations in finding a choice location for your computer/printer setup. For instance, you may want to find an electrical outlet that is not controlled by a switch-since a switch may be accidentally shut off while you have valuable information stored in memory Be sure the outlet is grounded (do not use an adapter plug). To minimize power fluctuations, avoid using an outlet on the same circuit breaker with large electrical machines or appliances. Finally, for continuous-feed operations you must allow enough room for the paper to flow freely, as in Figure 1-3.

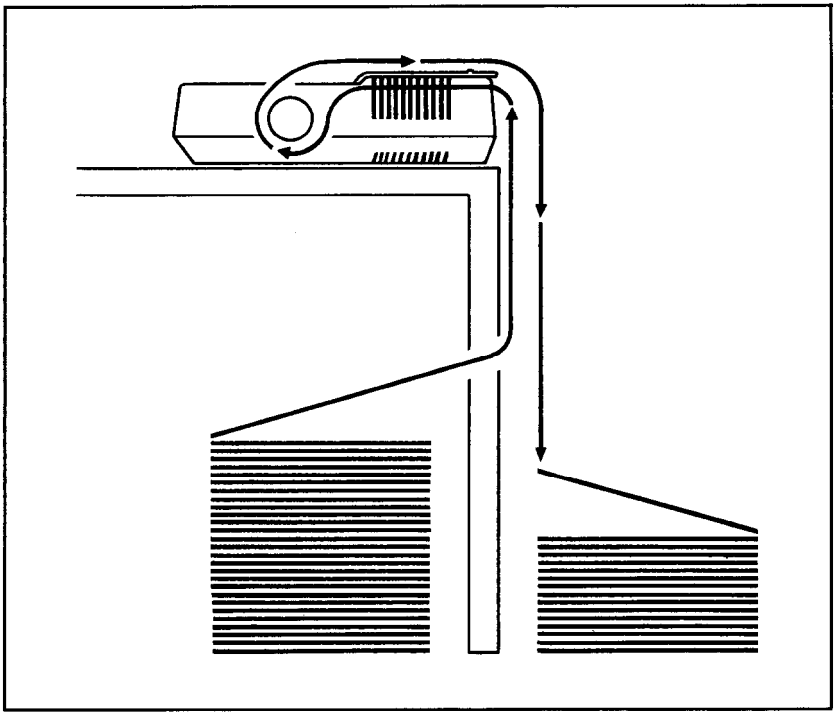


Figure 1-3. Paper path

Printer Preparation

Once you've found a good home for FX, you'll need to do some preparing before you can print. This section describes the first steps, which include installing a few parts, checking the setting of some internal switches, and then inserting the ribbon cartridge.

Note: The printer should be turned OFF during all set-up operations.

Paper separator

To install the paper separator, hold it vertically so that it rests on the two slots at the back of the metal frame as shown in Figure **1-4.** Press down gently but firmly until the separator snaps into place.

To remove the separator, pull up on the left side first, letting the right side slide out of its slot.

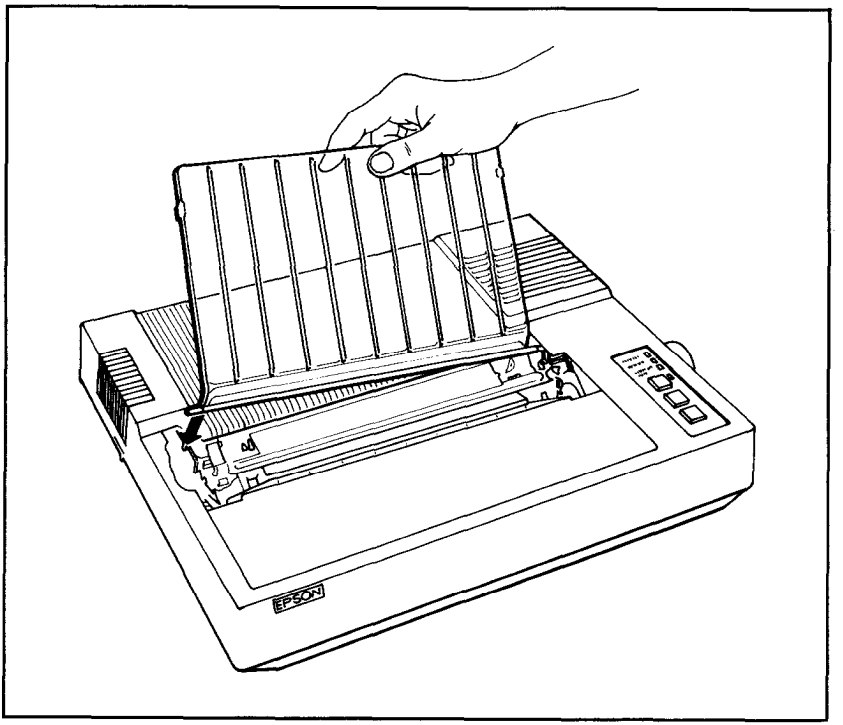


Figure 1-4. Paper separator

Covers

For protection from dust and foreign objects and for quiet operation, FX printers use two types of covers. When you use the friction feed on either the FX-80 or the FX-100 or the built-in tractor on the FX-80, use the pair of flat protective lids (Figure 1-5). When you use the removable tractor unit, use the tractor cover (Figure 1-6).

Install the center protective lid by inserting tabs into slots (one tab per side on the FX-80, two on the FX-100). Fit the left side of the lid over the friction-control lever (you may need to slightly bow the middle of the lid before you can snap the tabs into their slots). When you need to change the setting of the pin feeder on the FX-80 or install a tractor unit on either model, remove this lid by giving it a slight upward tug.

Install and remove either the front protective lid or the tractor cover by using the hinge posts at the front of the printer opening. This arrangement allows you to easily raise and lower the cover to load paper or ribbon. To install the cover, hold it at its full vertical position, slide the right hinge fitting over the right hinge post, and set the left

fitting over its post. Lower the cover. To remove the cover, move it to its full vertical position and then lift it up and a little to the left.

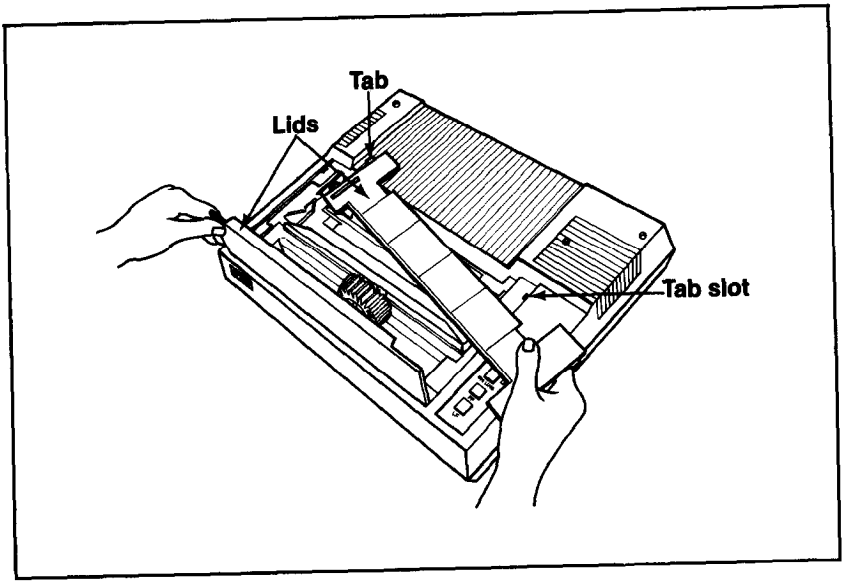


Figure 1-5. Protective lids

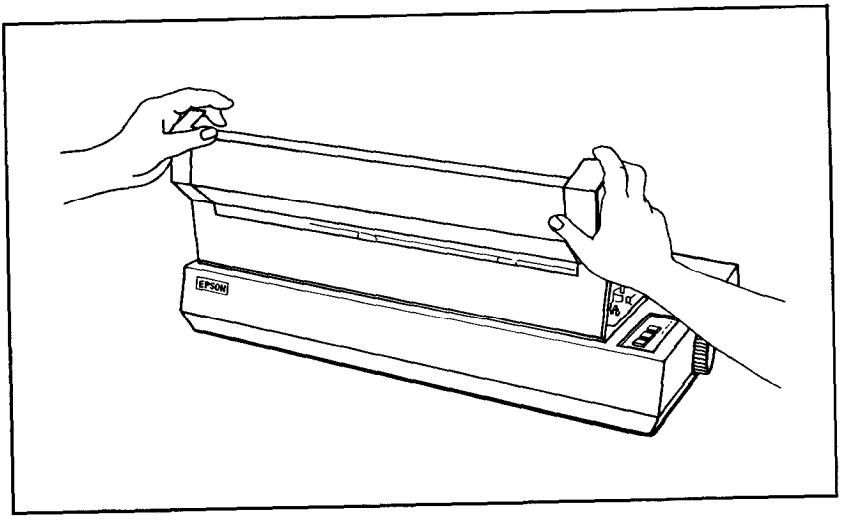


Figure 1-6. Tractor cover

Manual-feed knob

The manual-feed knob (Figure 1-7) can aid you in loading and adjusting paper. To install the manual-feed knob, hold it in position on

the right side and twist until the flat sides of rod and fitting match. Push the knob straight in with a steady pressure. To remove, pull straight out.

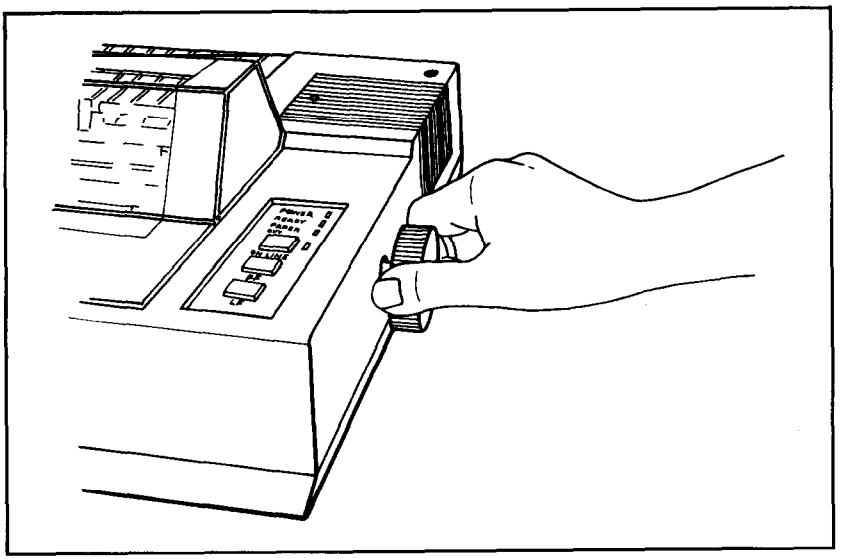


Figure 1-7. Manual-feed knob

DIP switches

Several tiny switches, called DIP (for Dual In-line Package) switches, are located inside the FX. They control a number of important printer functions, such as line-feed adjustment, the paper-out sensor, the beeper, and the default print modes. You can check these switches now, or you can skip to the ribbon section and check the switches later.

The design of the FX printer allows easy access to the internal switches. They are located under the upper-right vent. To remove the vent, you need a Phillip's-head screwdriver. Once the top screw is removed, take the vent off by pressing down and sideways with the palm of your hand (Figure 1-8).

Do not replace the screw because in the course of this manual, we will sometimes suggest that you reset switches. Keep the screw in a safe spot so that you can replace it later.

Locate the two DIP switch assemblies as shown in Figure 1-9 and check that they are set as shown in Figure 1-10.

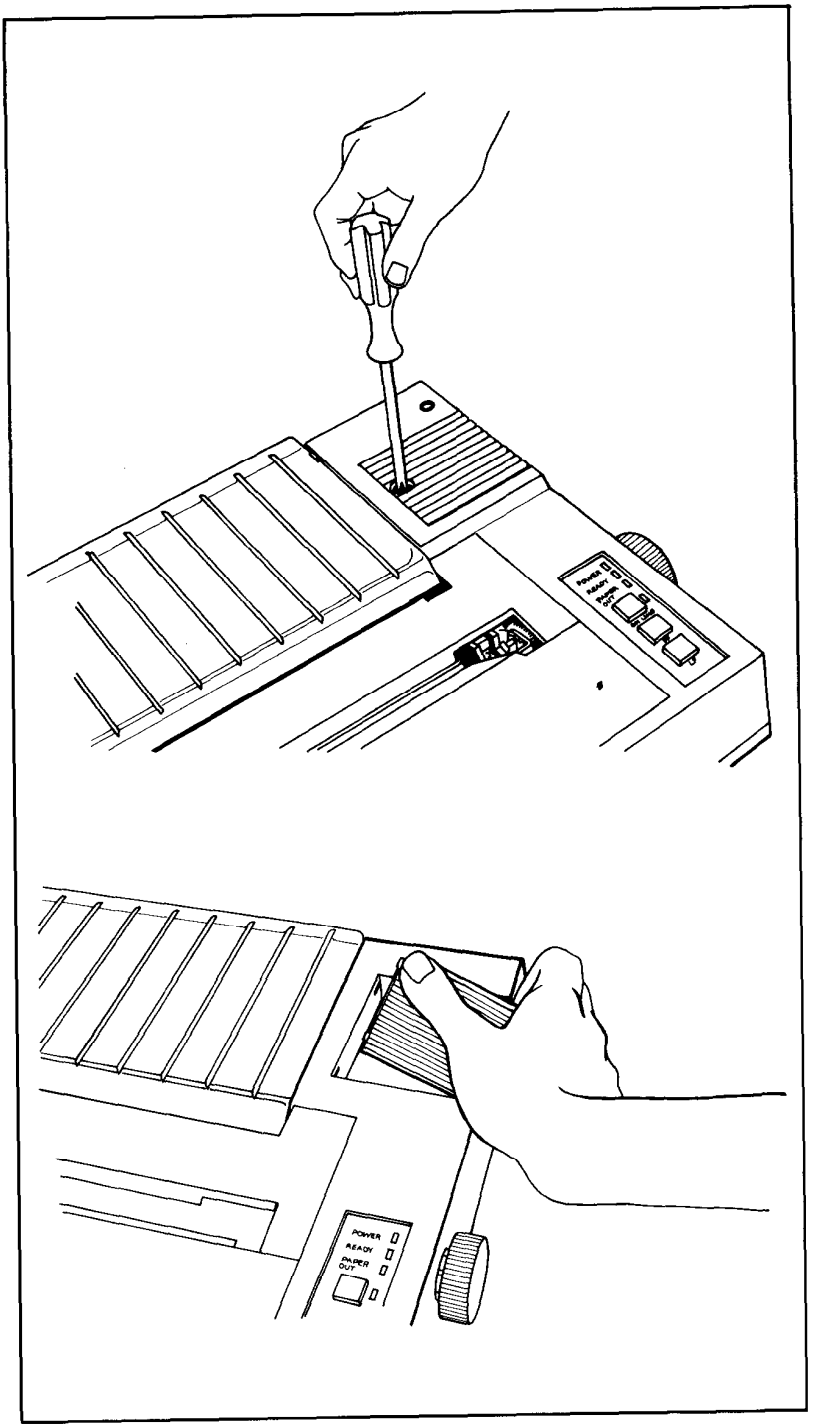


Figure 1-8. DIP switch vent

These switches are set at the factory, and most of them you will never need to touch. You may, however, want to take the time now to match up the switches with their functions, as shown in Table 1-1. For a further discussion of the DIP switches, see Appendix E.

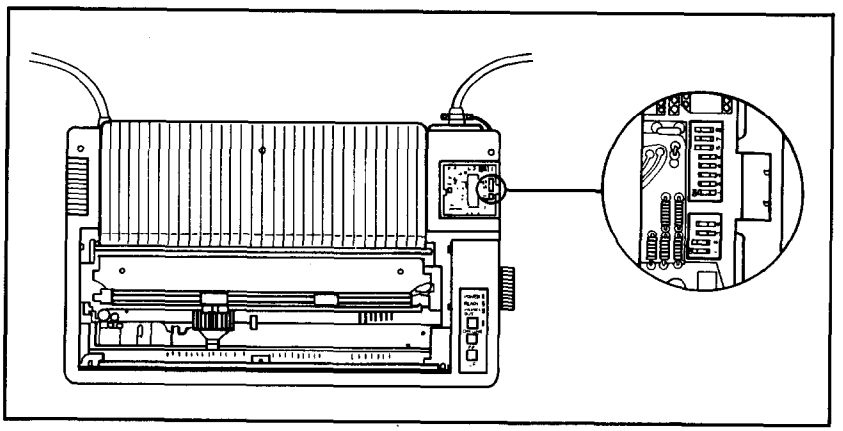


Figure 1-9. DIP switch location

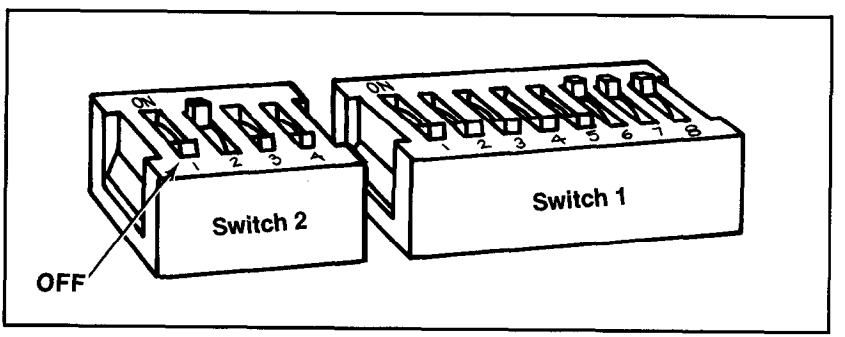


Figure 2-20. DIP switch factory settings

Always turn the power off (with the switch on the left side of the printer) before touching any internal switch. The printer checks most switch settings only at power-up. If you make changes when the power is on, they may be ignored until you turn the printer off, then back on. So set all switches with the power off. Use a non-metallic object, such as the back of a pen, to change the DIP switches.

One switch deserves your immediate attention; it is the switch labelled 4 on switch assembly 2. This switch (let's call it 2-4) adjusts the automatic line feed (the movement of the paper up one line) at the end of each print line to match your computer system's needs.

Table 1-1. DIP switch functions

Switch 1

No.	ON	Function	OFF
1-8	ON	International character	OFF
1-7	ON	International character	OFF
1-6	ON	International character	OFF
1-5	Emphasized	Print weight	Single strike
1-4	2K buffer	RAM memory	User-defined characters
1-3	Inactive	Paper-out sensor	Active
1-2	0 (slashed)	Zero character	0
1-1	Compressed	Print pitch	Pica

Switch 2

No.	ON	Function	OFF
2-4	CR + LF	Automatic line feed	CR only
2-3	ON	Skip-over-perforation feature	OFF
2-2	Sounds	Beeper	Mute
2-1	Active	Printer select	Inactive

Note: The shaded boxes show the factory settings.

Some computer interfaces automatically send a line-feed code to the printer at the end of each print line. Other interfaces send only a carriage return (which returns the print head to its left-most position), and rely on the printer to perform the automatic line feed. Switch 2-4 enables the FX to match either requirement. With switch 2-4 on, the printer automatically adds a line feed to every carriage return it receives from the computer; with the switch off, it expects the computer to provide the line feed. If you are not sure what your system requires, leave the switch the way you find it, but remember that you can adjust this if your first printing occurs either all on one line or with the lines spaces twice as far apart as you requested.

We recommend two changes now. Turning switch 1-2 on adds a slash to the zero character, which makes program listings easier to read. If you are not going to create your own characters, turn switch 1-4 on to take advantage of the internal 2K buffer.

Ribbon installation

First, be sure the printer is turned off and move the print head to the middle of the platen.

Remove the ribbon cartridge from its packing materials. Holding the cartridge by the plastic fin on the top, guide the pair of tabs at each

end of the cartridge into the corresponding slots in the printer frame (Figure 1-11). The cartridge should snap neatly into place.

With the paper bail resting on the platen, you can tuck the ribbon between the metal ribbon guide and the black print head.

As Figure 1-11 suggests, you can ease the ribbon into place with the deft application of a dull pencil. To remove any slack in the ribbon, turn the ribbon knob in the direction of the arrow.

Note: When you replace a ribbon, remember that the print head may be hot from usage; be careful.

Paper Loading

How you load your paper depends on which model of FX you have and which type of paper and feeder you are using. This section covers each type of paper loading and then illustrates the top-of-form position on both models.

Both the FX-80 and the FX-100 include tractors so that you can use continuous-feed paper with pin-feed holes and friction mechanisms so that you can use paper without these holes.

Continuous-feed paper usually comes fanfolded into a box and has pin-feed holes arranged on half-inch tear-off strips at each side. This allows the printer's pin feeders to engage the paper and pull it evenly through the printer. After printing you can remove the tear-off strips and separate the pages.

The FX-80's built-in tractor handles continuous-feed paper that is 9% inches wide, which is the standard 8½-inch width with the tear-off strips removed. The FX printers' removable tractor units (optional on the FX-80, standard on the FX-100) handle continuous paper in widths from 4 inches to the width of the platen - 10 inches on the FX-80,16 inches on the FX-100. The friction feeder on each FX handles all papers narrower than the width of the platen.

To refresh your memory about names of the parts, refer back to Figure 1-2.

FX-80: built-in tractor feed

The FX-80's built-in tractor will accommodate 9%- to 10-inch wide continuous-feed paper with pin-feed holes. You should have few problems loading it if you follow these instructions carefully.

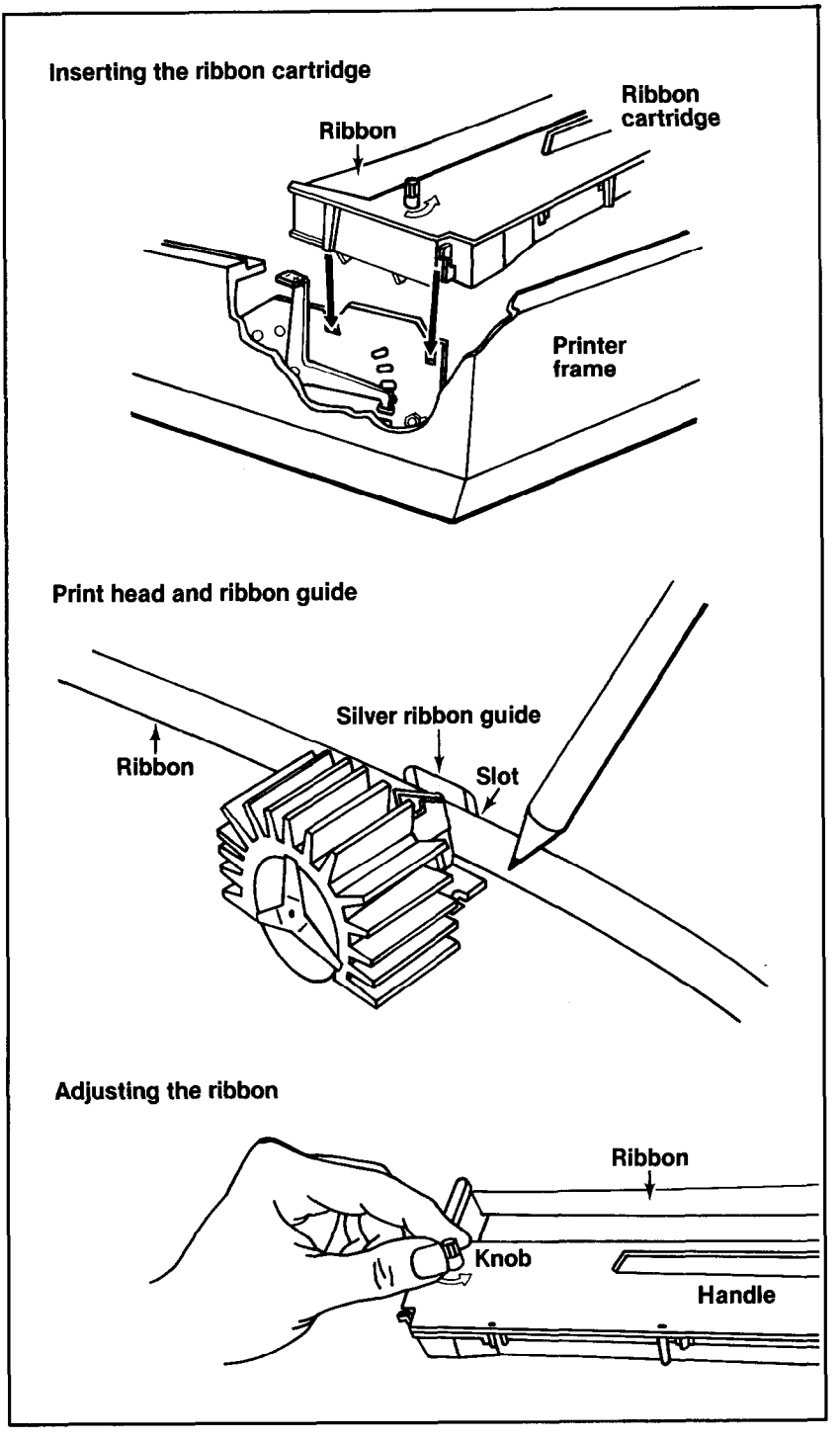


Figure 1-11. Ribbon insertion

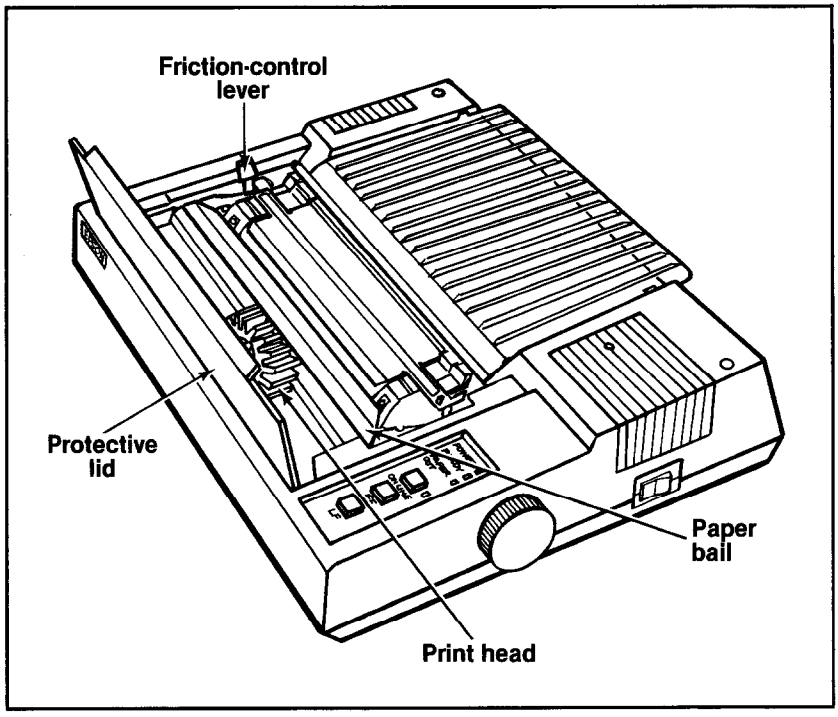


Figure 1-12. Printer readied for ***paper insertion***

- Be sure the printer is turned off. Lift the front protective lid and move the print head to the middle of the platen.
- Remove the center protective lid.
- Pull the paper bail and the friction-control lever toward the front of the printer. Your printer should now look like Figure 1-12.
- Adjust the pin-feed levers to approximate the width of paper you are using. Pull the levers forward to release the pin feeders, move them so their arrows line up with the correct position on the scale (e.g., for 9.5 for 9½-&inch paper), and push the levers backward to lock them into position (Figure 1- 13).

  If you are using fanfold paper, start by positioning your paper directly beneath or behind the printer, as in Figure 1-14, so that the paper won't kink or pull to one side.
- Insert the paper under the plastic separator, guiding it with your left hand while you slowly roll the manual-feed knob clockwise. It is

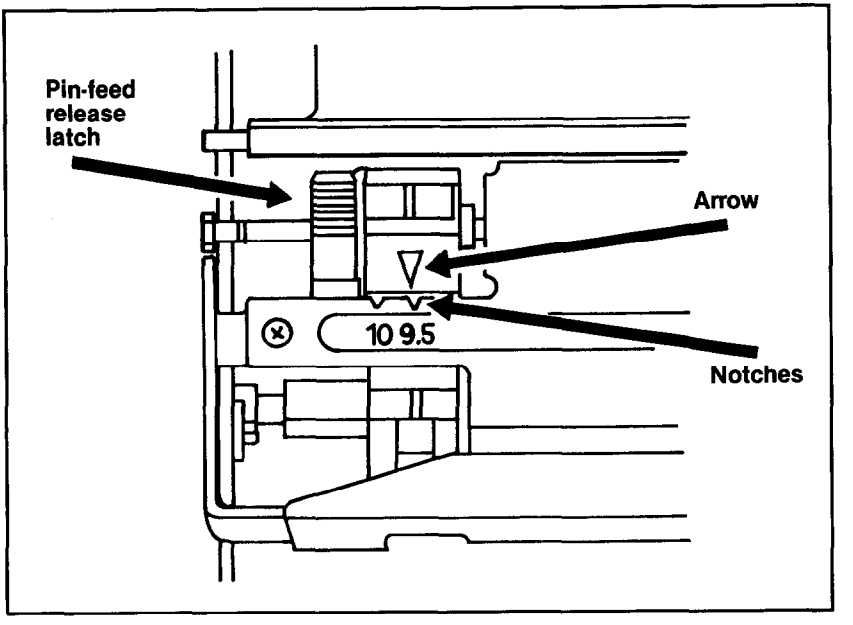


Figure 1-13. Pin feeder ***adjustment***

very important to keep the paper straight so that the pins on both sides engage at the same time. If the paper does not move smoothly, remove it by reversing the manual-feed knob and start again with an unwrinkled sheet.

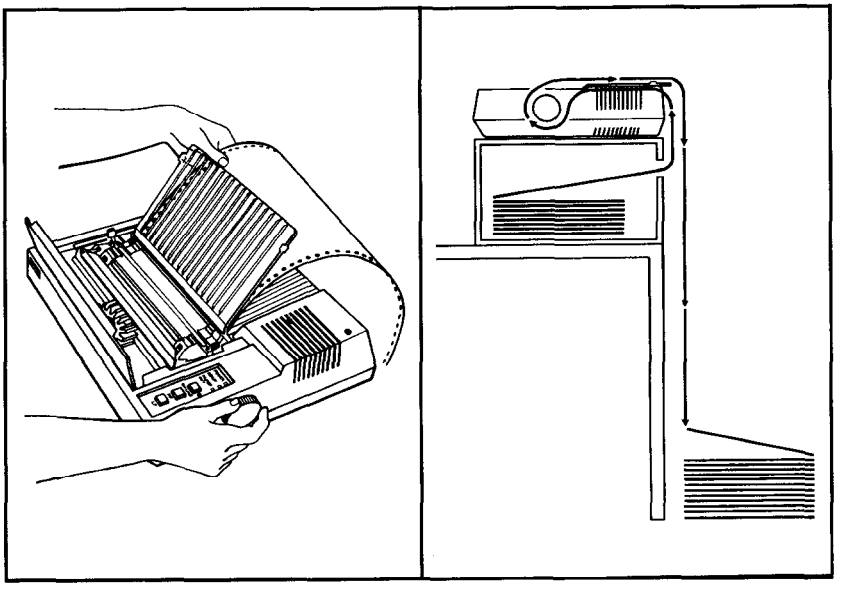


Figure 1-14. Loading the FX-80

- As the paper comes up the front of the platen, watch to be sure that it is feeding under the black edges of the pin feeders. If your paper is wrinkling as it comes through, you may need to readjust the pin feeders.

- Reinstall the center protective lid underneath the paper. Push the paper bail back against the paper and close the front protective lid. You are now ready to set the top of form, as shown at the end of this section.

- The friction-control should remain toward the front of the printer as long as the tractor is used.

FX-80 and FX-100: friction feed

The friction feed is for paper without pin-feed holes.

Before using the friction feed on the FX-80, disengage the pin feeders by pulling the levers forward, then move the feeders as far toward the edges as possible. Remove the center protective lid if it is on.

If a tractor unit is installed on the top, remove it. Push both tractor-unit release levers back (one is shown close up in Figure 1-15). Rock the unit up and back, pulling it gently toward the rear of the printer. The tractor will come off easily-you should not have to tug on the unit to remove it.

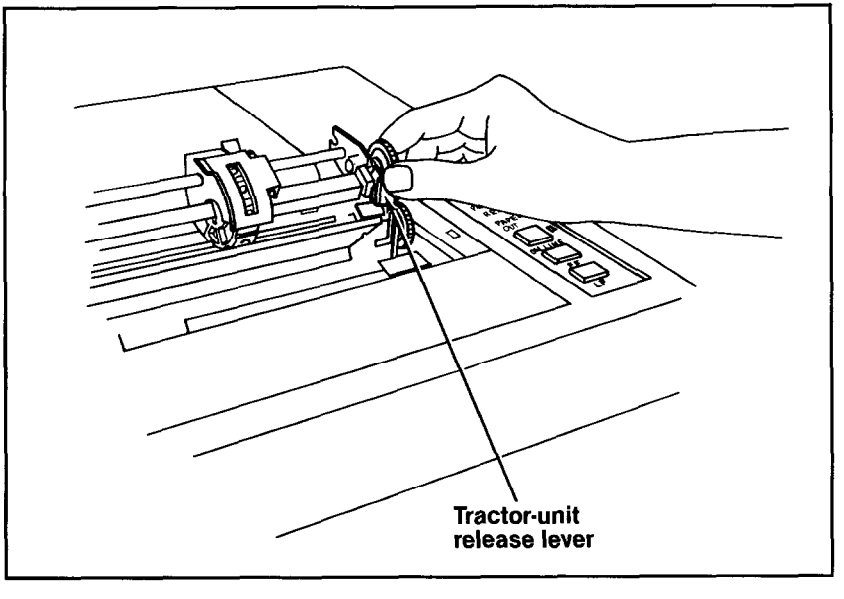


Figure 1-15. Tractor unit release

Now follow these steps to load your paper into the friction feeder:

- Be sure the printer is turned off, Lift the front protective lid and move the print head to the middle of the platen (refer back to Figure 1-12). Pull the paper bail up.
- Engage the friction-control mechanism by pushing the friction-control lever to the back.
- Guide the paper under the paper separator and the platen with your left hand, while turning the manual-feed knob with your right hand.

  If you hear a crinkling noise, stop. This can result from the paper getting slightly wrinkled; it is best to remove the paper and start over with an unwrinkled sheet.
- Do not pull on the paper as it comes up from the platen; leave the friction feeder in charge. Push the paper bail back against the paper, close the front protective lid and reinstall the center protective lid. You are now ready to set the top of form, as shown at the end of this section.

There is one more point to consider if you are printing on single sheets of paper. DIP switch 1-3 is set to active at the factory, and this means that the paper-out feature will sound the FX's beeper and halt printing whenever it senses the bottom of your sheet of paper. In practice, this means that without deactivating the sensor, you won't be able to print on the bottom of a single sheet of paper. You can, usually, deactivate the paper-out sensor easily by changing switch 1-3. Some computer systems, however, ignore the setting of DIP switch 1-3 (See Appendix F.)

FX-80 and FX-100: removable tractor unit (optional on the FX-80)

The removable tractor will accommodate pin-feed paper in any width from four inches to the width of the platen.

To install the optional tractor unit on the FX-80, begin by removing the center protective lid if it is on. Move the built-in pin feeders as far as possible to the right.

To add the tractor unit to either the FX-80 or the FX-100, hold the tractor unit over the printer with the gears to your right as shown in Figure 1-16. Lower the rear hooks over the rear studs as shown in Figure 1-17, pushing the unit back against the studs to ensure that both

sides of the tractor assembly are firmly in place. Rock the front of the unit downward, pressing firmly until it locks into place.

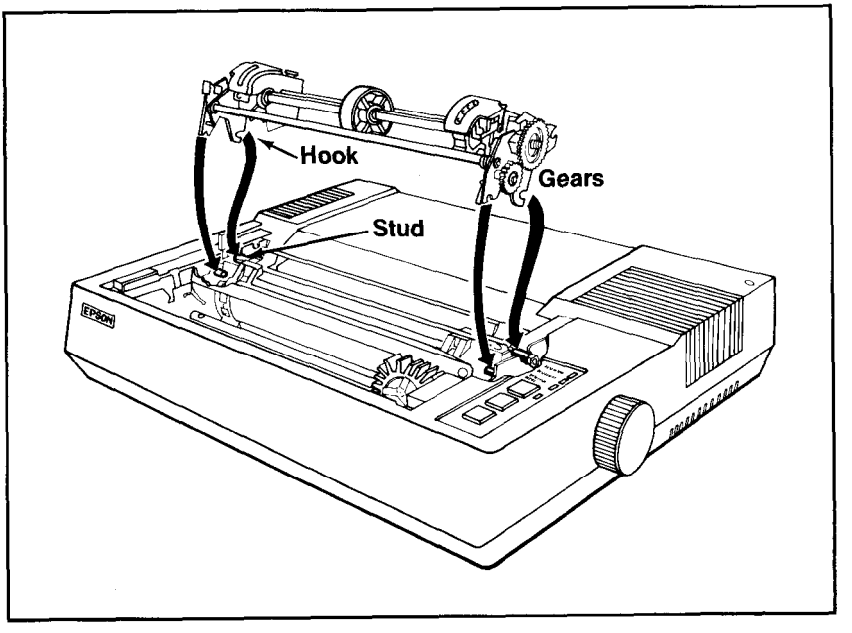


Figure 1-16. Tractor unit installation

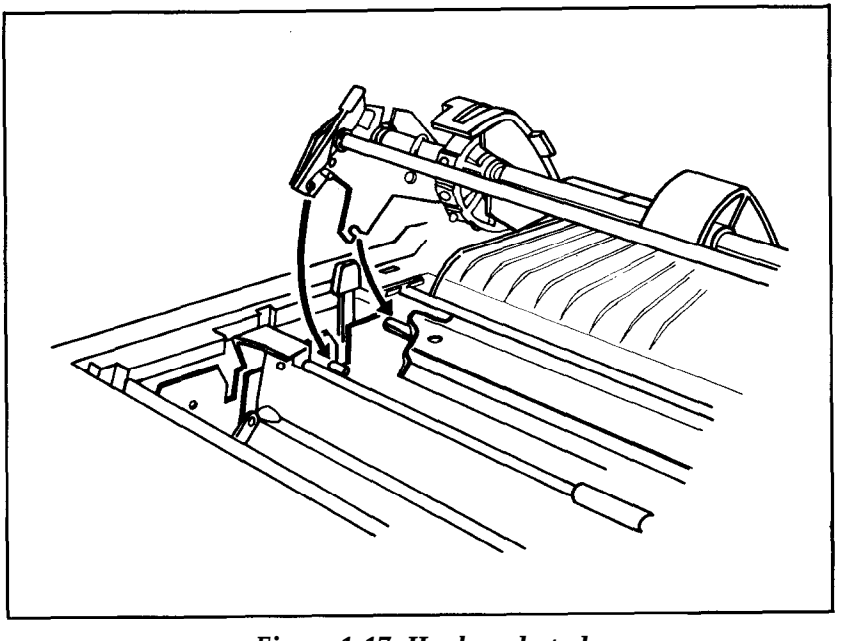


Figure 1-17. Hook and stud

To load the paper into the unit, use this procedure:

- Be sure the printer is turned off; then open the front protective lid to move the print head to the middle of the platen.
- Pull the paper bail and the friction-control lever toward the front of the printer (refer back to Figure 1-12).
- Insert the paper under the paper separator and the platen and push the paper through to the front.
- Position the pin feeders, using the pin-feed locking levers to make the adjustment. One is shown in Figure 1-18.
- Raise the black covers of both pin feeders and ease the paper over the pins. Adjust the paper or pin feeders as necessary so that there are no wrinkles or dips in the paper. Now you are ready to set the top of form.

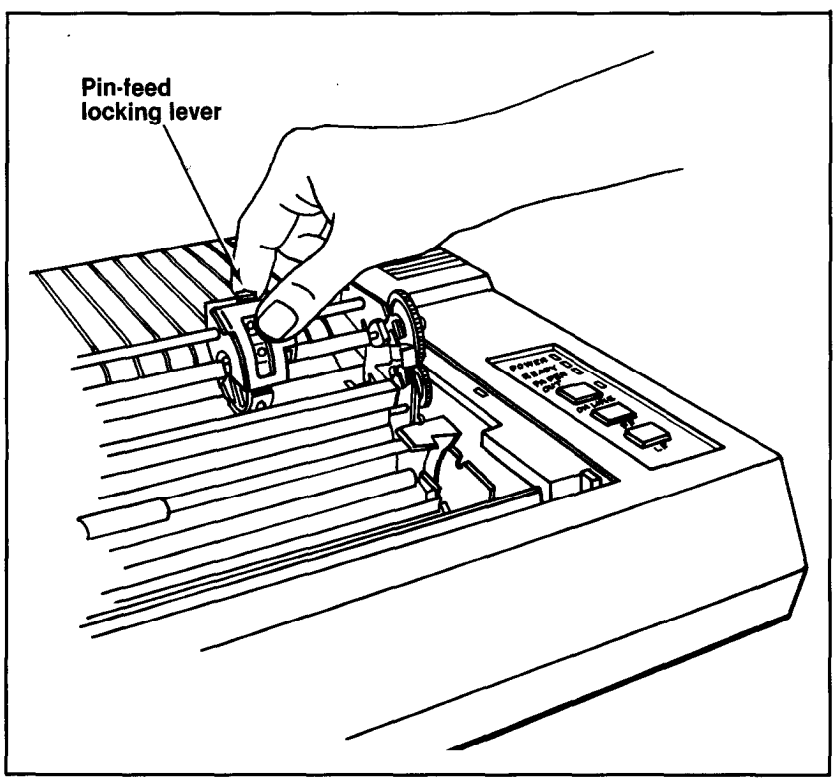


Figure 1-18. Adjusting the pin feeders

Top-of-form position

After you have loaded the paper, you should set it to the top of form, which is the position of the print head when you turn the printer on. (Since the computer term form corresponds to the word page, it may be easier for you to think of this as the top of the page.) To make this setting, advance the paper until a perforation lies slightly below the top of the ribbon.

The relationship between the perforation and the printhead is the same on both models of the printer, as you can see in Figure **1-19. You** need to leave some paper above the ribbon so that the paper moves up smoothly.

When you have the position set, lower the bail and replace or reposition the lids or cover.

Although you **have** arranged **the** paper correctly, you are not done. The printer will not recognize the top of form until the next time you turn it on. The FX considers a form to be 66 lines long unless you change this length as discussed in Chapter 8.

Paper-thickness lever

The paper-thickness lever shown in Figure 1-20 moves the print head to accommodate various paper thicknesses. The factory sets it for ordinary paper (which is 1/500th of an inch thick), but you can adjust it for printing one original and up to two copies. For thick paper or multiple copies, move it toward the front. Do not use the extreme rear setting, however. This position is used for head alignment and will shorten the life of the print head if used in normal operation.

Starting Up

It is finally time to connect the printer to the computer; remember that some computers need interface kits and all need cables. First make sure the power switch is off. Connect the printer end of the printer cable to the connector at the right rear of the FX (as shown in Figure

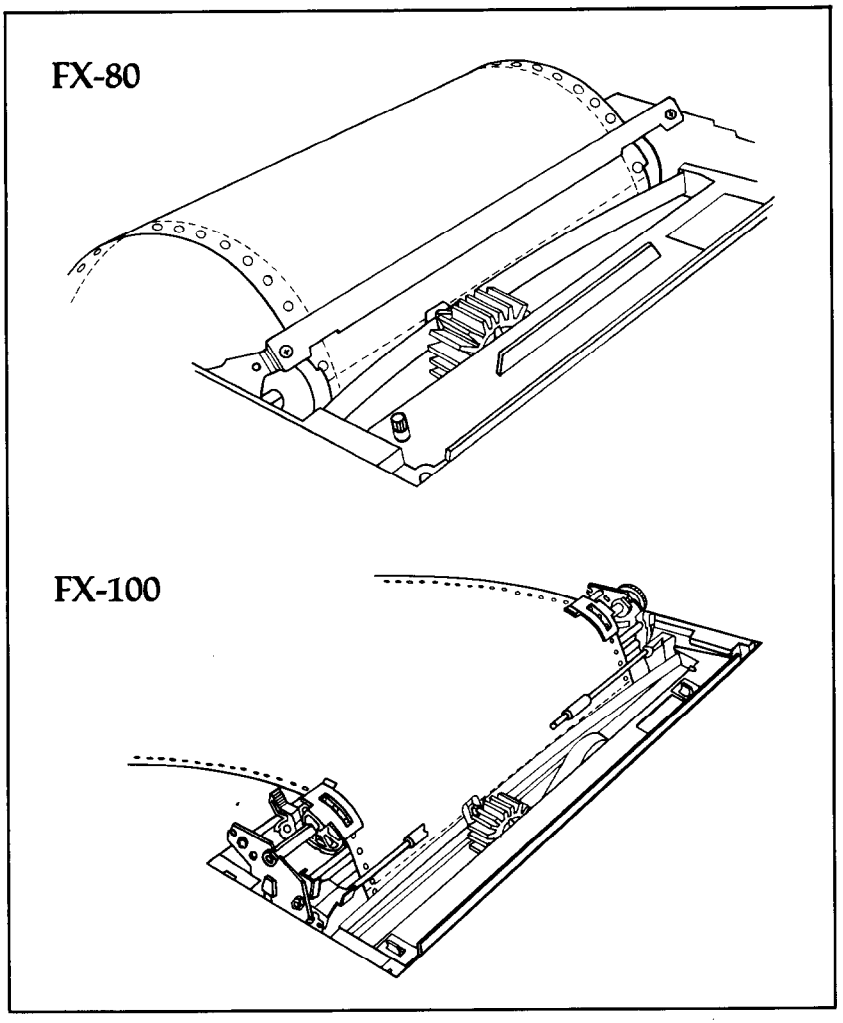


Figure 1-19. Top of form

1-21). The other end of the cable plugs into your computer. If your cable includes grounding wires, be sure to fasten the wires to the grounding screws at each end.

With the paper loaded, turn the printer on with the toggle switch at the left-rear comer of the FX. You get a little dance from the print head and three lights go on: the POWER light, the READY light, and the ON LINE light. If the ON LINE and READY lights are not on, push the button marked ON LINE. If the PAPER OUT light is on, the paper is not loaded correctly.

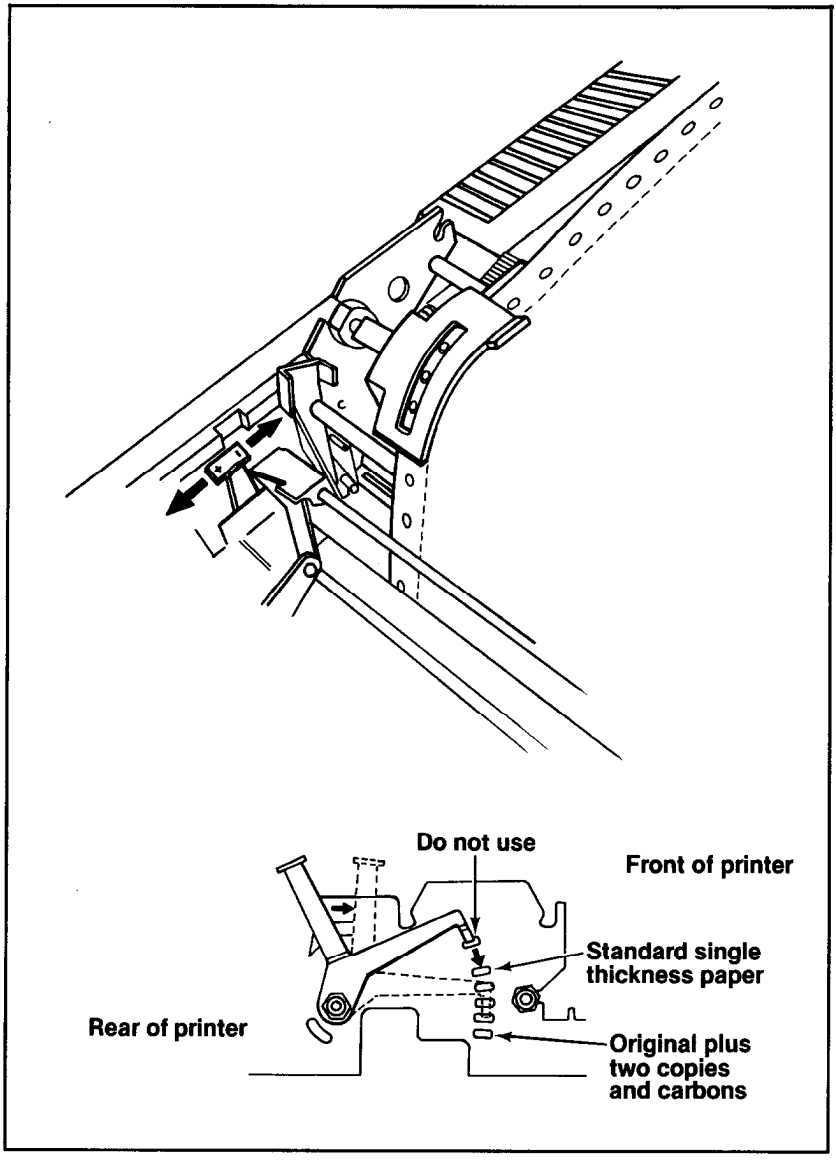


Figure 1-20. Paper thickness adjustment

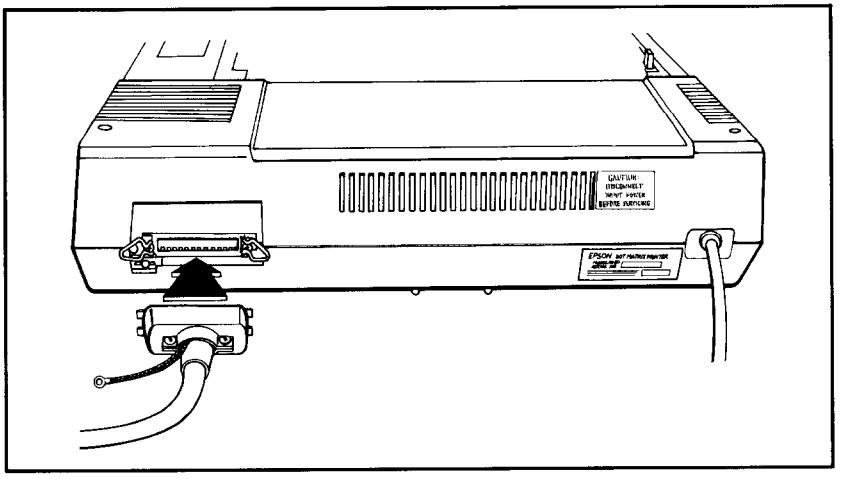


Figure 1-21. Cable connection

Control panel

When the control panel's ON LINE light is on, the printer and computer are in direct communication and the FF (form feed) and LF (line feed) buttons have no effect. Go ahead, try pushing one.

To use the FF and LF button; press the ON LINE button to turn it off. Now you can see what the other buttons do.

Press the LF button briefly, then release it. That produces one line feed. Now hold the LF button down for a moment to produce several line feeds.

The FF button advances the paper one complete page(form). If you hold the button down, it advances several forms. Ideally, you'll set the top-of-form position before you turn on the printer. That way, your printing can start on the first line of the paper. If you turn the printer on while the print head is sitting in the middle of a form, that is precisely where the next form will start when you press the FF button or your program sends the FX a form feed.

Remember that the ON LINE light has to be off for the FF and LF buttons to work.

The FX tests itself

Now it's time to see how your new FX operates. If you're using an FX-100, be sure you have full-width (15-inch printing area) paper

loaded because the printer's test uses all **136** columns. Turn the printer completely off (with the switch on the left side of the printer), press down the LF button, and turn the printer back on again while still holding down the LF button.

Figure 1-22. Sample automatic test

Figure **1-22** shows the FX's automatic test, which prints the standard characters that are stored in the printer. The test pattern continues until you turn the printer off. If you let it run long enough, you'll see two sets (Roman and Italic) of upper- and lower-case letters, plus many other characters.

If you plan to use your FX printer primarily for word processing or with some other commercial software, you are now ready to follow the printer set-up instructions in your software menu or manual. Because most software packages include set-up routines for dot matrix printers, this set up may be merely a matter of specifying which printer you are using.

In case your software asks for specifications that you do not understand, check the appendixes or consult your computer dealer.

Some programs allow you to insert codes to activate such FX features as Emphasized and Compressed printing. Your software manual will tell you how to use this feature, and the Quick Reference Card in Volume **2** will give you the necessary ESCape and control codes.

Chapter 2
BASIC and the Printer

While you read this manual, you'll be testing your FX with programs in the BASIC language. You can, of course, use another language with your printer; see Appendixes A through D for the ASCII and ESCape codes that your software manual will explain how to use. Here we use BASIC because it is the most popular language for personal computers.

One of the simplest things you can do with any FX printer is print listings of your BASIC programs. You merely load a BASIC program into the computer and send the LISTing output to the printer instead of to the screen.

Unfortunately, different computer systems access the printer in different ways. For example, most computers that use Microsoft BASIC send PRINT or LIST commands to the printer by adding a leading L to a screen command: LPRINT, LLIST, etc. Some other computer systems use PRINT # in place of LPRINT Another group uses PR#l to route information to the printer and PR#0 to restore the flow of information to the screen. If you aren't familiar with your system's command conventions, consult its manual.

We will use the LPRINT and LLIST commands for our examples in this manual because the widespread acceptance of Microsoft BASIC makes these commands as close to a standard as exists in this industry. But remember that you may need to modify such commands to match the unique aspects of your system. The Preface and Appendix F can help.

Once you have discovered how your computer communicates with the printer, load a BASIC program into memory. Now list it onto the printer, using your computer's version of the LLIST command. Some examples are shown in Table **2-1.**

Table 2-1. Several computers' print LIST commands

Command	Computer
LLIST	Epson QX-10™, IBM-PC®, and Radio Shack TRS-80®
LIST"COM0:"	Epson HX-20 Notebook Computer™
PR#1 LIST PR#0	Apple® II

If your listing is more than a page long (or if you didn't start the listing at the top of a page), your printer may have printed right over the perforation. Set DIP switch 2-3 to the on position, and the printer will automatically skip over the perforation. We discuss this further in Chapter 8.

Meanwhile, printing a program LISTing is a fundamental function of the printer. Be sure you manage this before continuing (if you have trouble, consult your computer's manual for help).

BASIC Communications

Part of the difficulty in controlling communications between computer and printer is the lack of a completely standard coding scheme. When your computer sends out a numeric code for the letter A, you naturally want your printer to interpret that code as an A. Most manufacturers of computers, printers, and software use the American Standard Code for Information Interchange (ASCII, pronounced ***ask***-ee) to code such frequently used characters as the letters of the alphabet, numerals, and keyboard symbols. Of the 256 ASCII numbers, most are codes for specific characters; some are codes for such computer or printer functions as sounding a beep or performing a carriage return.

The ASCII standard does not yet allow for the advanced features in today's computers and printers. Individual manufacturers therefore adjust the codes to suit their own needs, which means that we are often faced with compatibility problems between printers and computers. (To compare your computer's version of the ASCII table with the FX's version, see your computer manual and this manual's Appendix A.) You can usually overcome the code inconsistencies by sending control codes for advanced printer functions in care of a special ASCII code that is called an ESCape code. The next five subsections discuss these matters in more detail.

Character strings

The character-string (or CHR$) function converts any decimal number from zero through 255 to a character or action. Its format is CHR$ followed by a number in parentheses, for example, CHR$(84). The character-string command follows a PRINT or LPRINT command and causes your computer system to send an ASCII code to the computer's screen or to the printer. What gets printed or performed is determined by the particular modified ASCII table that is used by your system. Where the printing or action happens-on your screen or your printer-depends on the print command that precedes the character-string command.

For a fast check on what your computer does with this function, try printing a few characters on your computer's screen. The usual format for this is PRINT CHR$(n). The n represents one of the numbers from zero to 255, each one of which accesses a unique character or action. Try typing this:

```
10 PRINT CHR$(65)
```

and RUNning it. Since most computers use the numbers from 32 to 127 to mean the ASCII set of characters, you should see a capital A on the screen.

It's the numbers less than 32 and greater than 127 that produce different results on nearly every brand of computer. Try entering:

```
10 PRINT CHR$(193)
```

and RUNning it. If you don't see anything on the screen, don't worry. Remember that this was just a quick check. We are mainly interested in sending that 193 to the printer, and what it prints on the screen is not as important right now.

BASIC print commands

Well then, what happens when you send such a non-standard code as 193 to the printer? To test this out, you need to know what program commands your computer uses to activate the printer. Some typical command sequences are shown in Table 2-2.

Table 2-2. Several computers' printer ***activating commands***

Activating commands	Computer
10 LPRINT CHR$(193)	Epson QX-10, IBM-PC, and Radio Shack TRS-80
5 OPEN "0",#1, "COMØ" 10 PRINT#1, CHR$(193) 99 CLOSE#1	Epson HX-20 Notebook Computer
5 PR#1 10 PRINT CHR$ (193) 99 PR#Ø	Apple II

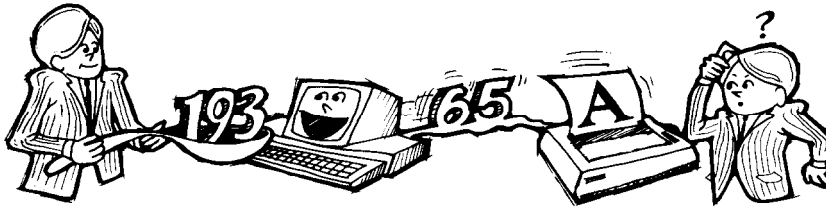
Check your computer's reference manual and type in the commands appropriate to your computer. Then type RUN.

With any luck, you will get an Italic capital A on the printer: A

If nothing prints, it's time to double-check your computer manual and cable connections. Make sure the printer is ON LINE and the READY light is lit.

ASCII and BASIC basics

If you end up with a Roman A: A



instead of an Italic A, pay close attention to the next three paragraphs.

The original ASCII code was designed to use the decimal numbers zero through 127. Computer systems designers soon decided to extend this range (to 0 through 255) in order to make room for more features. Unfortunately, some designers did not anticipate that printers would make use of this extended range. So they designed BASIC printer drivers that intercept any number in the upper half of the range (128 - 255) and automatically convert it to the lower half of the range by subtracting 128.

In these systems such a code as CHR$(193) never makes it to the printer. The printer driver subtracts 128, which means that the code for Italic A gets to the printer as a CHR$(65). The printer then produces a Roman A.

For many applications, you won't need the upper half of the ASCII codes. For others, the inability to generate codes greater than 127 will be an obstacle. Whenever we can, we suggest ways to get around this obstacle. In Chapter 5, for instance, we discuss Italic Mode, which is the FX designers' method of making Italic characters easily accessible to all users.

If you're patting yourself on the back because your printer printed an Italic A, postpone your celebration for a bit. Nearly all computers' BASIC programs intercept codes on their way to the printer and alter some of them. For example, some popular systems intercept a CHR$(l0)-line feed-and send out a CHR§(13)-carriage return-instead. Typical problem codes involve the numbers 0 and 9 to 13. Your computer manual may alert you to these problems, Or experience may have to be your guide.

In order to help computer systems that can't send a zero in a CHR$ command, several printers' instruction sequences allow such options as using "0" (quote-zero-quote) in place of CHR$(0). Besides mentioning some of these solutions within the text of this User's Manual, we have written a troubleshooting appendix, Appendix F.

Control codes

Enough talking about problems. Here's a program line to try:

```
10 LPRINT CHR$(7)
```

Be sure to use the appropriate printer access commands for your system.

Now RUN it. You should hear a short beep. (If you don't hear it, check DIP switch 2-2, using the procedure we gave in Chapter 1.) That's the printer's beeper, which most often sounds to inform you that you've run out of paper (Appendix F lists other causes of beeping). When you produce the beep, you've proved that on your computer certain codes do indeed perform printer functions. Table 2-3 shows the ranges that the FX uses when it interprets ASCII codes for characters and functions.

Table 2-3. ASCII codes on the FX

ASCII code group	FX interpretation
Ø to 31, 127	Printer control codes
32 to 126	Standard (Roman) character set
128 to 159, 255	Additional control codes (Function same as Ø-31, 127)
16Ø to 254	Italic character set

See either Appendix A or the Quick Reference Card for a chart of the FX interpretation of each ASCII code number.

This would be a good time to try printing a few of these codes on your own. And you may want to take a break before you start the next section.

Escape-CHR$(27)-and other CHR$ commands

As more features are added to the printer, even the extended range of codes (0 - 255) is inadequate if only single-code CHR$ instructions can be used. Because of this, Epson has designed the FX printer's logic to understand special sequences of control codes, the ESCape code sequences. You use these code sequences to select one or more printing features, or modes.

Such modes as Italic Mode and Expanded Mode affect the way the characters look. Other modes affect spacing and therefore the formatting of your pages. Appendix C, which is reprinted on the Quick Reference Card, collects the modes into categories for quick reference.

Each ESCape code sequence consists of the ESCape code, which is CHR$(27), plus one or more of the FX's other CHR$ control codes.

Here are two examples of ESCape code sequences:

```
LPRINT CHR$(27)CHR$(71)
LPRINT  CHR$(27)CHR$(38)CHR$(0)CHR$(1)CHR$(3)
```

To see how such sequences work, start a new program now by entering:

```
10 LPRINT CHR$(27)CHR$(52)
20 LPRINT "ITALIC CHARACTER SET"
```

and RUNning it. When you can RUN a program, we show you the results that you should expect:

ITALIC CHARACTER SET

Note: If you haven't yet read the Preface, which includes "Conventions Used in This Manual," this is the time to do it. Especially important now are the passages on semicolons and on saving programs.

The FX interprets the CHR$(27)CHR$(52) sequence in line 10 above as a command to switch from Roman to Italic characters. The LPRINT in line 20 sends a string of characters to the printer to verify that the printer is in Italic Mode.

Now type:

LLIST (or your system's version of the print LIST command)

to check the printer's status as in Figure 2-1:

```
10 LPRINT CHR$(27) CHR$(52)
20 LPRINT "ITALIC CHARACTER SET"
```

Figure 2-1. Italic listing

Since all the text is still printed in Italic characters, you can see that this mode stays on until it is turned off. This is typical of the modes on FX printers: nearly all modes stay on until turned off. We will alert you to the few exceptions.

If your printer is printing one line on top of another or if it is double spacing, you need to change the setting of the FX's DIP switch 2-4.

Change Commands

After you have sent commands to the printer, you will often want to change them, either to turn off one or more modes, or to erase text. To understand what happens when you use one of the several FX methods of making changes, you need to know about two special aspects of the printer, defaults and the printer buffer.

We often talk in these pages about resetting the printer to its defaults. By defaults, we mean the settings that are in effect whenever you turn the printer on. The printer has default settings for such features as the size of a line, the size of a page, and the print mode or mode combination that is in effect (see Appendix G or the Quick Reference Card for a list). Although all of these defaults are originally set at the factory, you can reset some of them by hardware, by changing the appropriate DIP switches. You can also change most features that have defaults by software. But whenever you turn your printer off and back on, you will have reset its default settings.

For example, you can use the software control codes to make your pages be four inches long, but the next time you reset your printer, the page length will return to 11 inches.

Every control code that you send to the printer is stored in the printer's RAM buffer right along with the text. All material goes through this buffer to get to the printed page. This buffer is like a holding tank in which each print line is collected.

The buffer can hold a full line of text characters (on the FX-80, 80 characters for normal-width print, more characters for narrower widths; on the FX-100,136 for normal and more for narrower widths) as well as control codes. All information resides in the buffer until the buffer is filled or a control code that empties it is received. (One such control code issues a carriage return; another type selects graphics, as discussed in Chapter 11.) Then the FX processes the line, one character at a time. As the printer encounters characters and codes, it prints text characters on the page and activates the print modes according to the control codes.

If you understand the concepts of defaults and buffering, you will rarely be surprised by what happens when you send a change code. The next three sections concern change codes: the first two cover changes you make to commands you have sent to the printer; the last concerns making changes to text.

Reset Code

You could turn off the Italic Mode by turning the printer off, then back on. Although turning the printer off resets the printer to its defaults, which include Roman Mode, cycling the printer off and on may disrupt computer/printer communications. FX printers have a Reset Code to avoid that: ESCape CHR$(64).

To see the Reset Code work, add these lines to your budding program:

```
30 LPRINT CHR$(27)CHR$(64)
40 LPRINT "BACK TO ROMAN WITH THE RESET CODE"
```

and RUN it.

```
ITALIC CHARACTER SET

RACK TO ROMAN WITH THE RESET CODE
```

The Reset Code of line *30* turns off all special print modes and resets the printer to its default settings, which include Roman typeface. The top of form is also reset; now it's at the position of the print head when your program issued the Reset Code. You can test this by using the FF button to advance the paper one page.

Notice that there is a blank line between the two lines of text in the printout. This happens because BASIC provides a line feed after every line of print commands unless you put a semicolon at the end of the line. As you will see, we often end lines with semicolons to prevent unwanted line feeds.

The Reset Code is useful when you want to turn off all printer modes. It resets everything to its start-up condition. If you have several different modes active in the printer at one time, they are all shut off by the ESCape CHR$(64).

Mode cancelling codes

The FX printer also provides specific codes to turn off each mode separately. For example, an ESCape CHR$(53) turns off the Italic character set and leaves everything else untouched. To see how the specific cancelling code for Italic Mode works, change the Reset Code in line 30 above to an ESCape CHR$(53):

```
30 LPRINT CHR$(27)CHR$(53);
40 LPRINT "BACK TO ROMAN WITH ITALIC OFF"
```

Notice that CHR$(53) turned Italic off and the semicolon at the end of line 30 eliminated the blank line between the two lines of text.

DELete and CANcel

But suppose you don't want **that** much power. Suppose you only want to erase text in the print buffer without affecting any print modes. Two codes do this: DELete and CANcel. DELete, which is CHR$(127), removes the latest text character from the buffer without affecting control codes. CANcel, which is CHR$(24), is a slightly more powerful code. It removes all the text currently in the buffer, but it also does not affect the control codes.

These codes are seldom used, but they can be helpful if your system sends unwanted characters at **the** beginning of a program listing or a program run. You can then use one or more DELete or CANcel codes at the start of your program to clear out these characters.

Alternate Formats for ESCape Sequences

Activating each of the many FX features with a sequence that consists of the ESCape code plus another character-string command can be cumbersome to use and difficult to remember. Fortunately, there are techniques for shortening the format.

The simplest method is to shorten **the** character-string command that follows the ESCape code. Instead of using CHR$ and a number in parentheses, you can use-inside quotation marks-the ASCII character that corresponds to the number. For example, you have already seen that you can send the Reset Code with CHR$(27)CHR$(64). Because the @ symbol is the ASCII equivalent of 64, this command can also be typed in as CHR$(27) " @ " .

We use this shorter format whenever possible. To see it in your current program, change lines 10 and 30. The ASCII symbol for the 52

of line 10 is the number 4, and the symbol for the 53 of line 30 is the number 5, so enter the following:

```
10 LPRINT CHR$(27)"4"
30 LPRINT CHR$(27)"5";
```

Now use RUN to make sure that both ESCape sequences work as before.

You can also shorten your programs by storing the ESCape code in a character string. If you enter A$=CHR$(27) in an early line of a program, you can simply enter A$ each time you want the ESCape code. For example, the following lines produce the same results as the previous ones:

```
5 A$=CHR$(27)
10 LPRINT A$"4"
30 LPRINT A$"5";
```

This technique can help if you use the ESCape code frequently.

If you use a certain ESCape sequence often in a program, you can store it in a character string as shown in this example:

```
5 B$=CHR$(27)+"4"
10 LPRINT B$
```

Note that in this case you must use a plus sign between the elements of the ESCape sequence.

Summary

You use CHR$ to send numeric codes from BASIC, and you use CHR$(27), the ESCape code, to earmark the printing or action to occur on the FX printer. You turn on an FX mode by using the ESCape code and adding to it either the character-string command and an appropriate FX control code or else the ASCII equivalent of the control code enclosed by quotation marks.

Nearly all printer modes stay active until they are turned off (we cover the exceptions later). You can turn off one printer mode or set of modes by shutting the printer off or by sending the Reset Code, either of which resets the FX to its default modes. In addition, each mode has its own cancelling code that turns off only that particular mode.

You can delete all the text characters in the FX's buffer by using the CANcel code or the latest text character by using the DELete code. Neither affects the control codes.

See the Preface for a list of the conventions used in this manual, Appendix A for a table of the ASCII codes, and Appendixes B and C for tables of the control codes. Appendix F offers programming solutions to interfacing problems, while Appendix E lists the defaults and shows the DIP switch settings. See also the Quick Reference Card.

Here are the DIP switches that we mentioned in this chapter:

Switch 2-2	Controls the beeper
Switch 2-4	Controls the number of vertical lines that the printer spaces at the end of a print line

Here are the codes that we have covered in this chapter, listed in the order of their appearance:

CHR$(7)	Causes a beep
CHR$(27)	Prepares the printer to accept other control codes (Escape code)
CHR$(27)“4”	Turns Italic Mode ON
CHR$(27)”@”	Resets the printer to its defaults (Reset Code)
CHR$(27)“5”	Turns Italic Mode OFF
CHR$(127)	DELetes the latest text character in the print buffer
CHR$(24)	CANcels all text in the print buffer

Chapter 3
Print Pitches '

One of the big advantages an FX printer has over a daisy-wheel printer or a typewriter is the ability it gives you to choose from a variety of widths, or pitches, for your characters. To use this feature well, it's important to understand just how an FX prints. The technique used by an FX printer is called dot-matrix printing.

Dot-Matrix Printing

A dot matrix is a grid or graph that someone who designs a character set for a dot matrix printer uses. The dot-matrix designs for the characters, which may be letters of the alphabet, numbers, or symbols, are stored in the printer's read-only memory (ROM).

The FX's dot matrix is nine rows of dots high and six columns of dots wide. Look at any letter **on** your printout-it's made up of a series of dots. And, as you can see in Appendix A, every letter fits inside this six by nine grid.

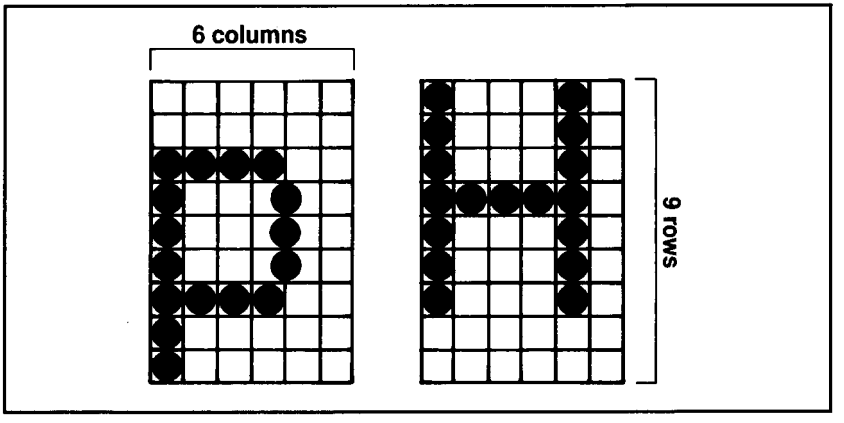


Figure 3-1. Dot-matrix characters

Figure 3-1 shows one each of lower- and uppercase letters. The p gives an example of the way a few lowercase letters use the bottom two rows of the matrix. All numbers, uppercase letters, and most symbols are formed within the top seven rows of the matrix.

Main columns

The construction of the print head restricts the maximum height of any character to nine dots. As shown in Figure 3-2, the print head uses a vertical column of nine pins (actually, wires). Because there is only one column of pins, the head must move sideways sequentially to **each** of the different column positions of the matrix, then fire the appropriate pins. Electrical impulses cause the FX to fire pins at the paper. As a pin is fired, it presses against the ribbon to produce one dot of the matrix. At each position, the printer fires only the pins that are necessary to print the current column of the character.

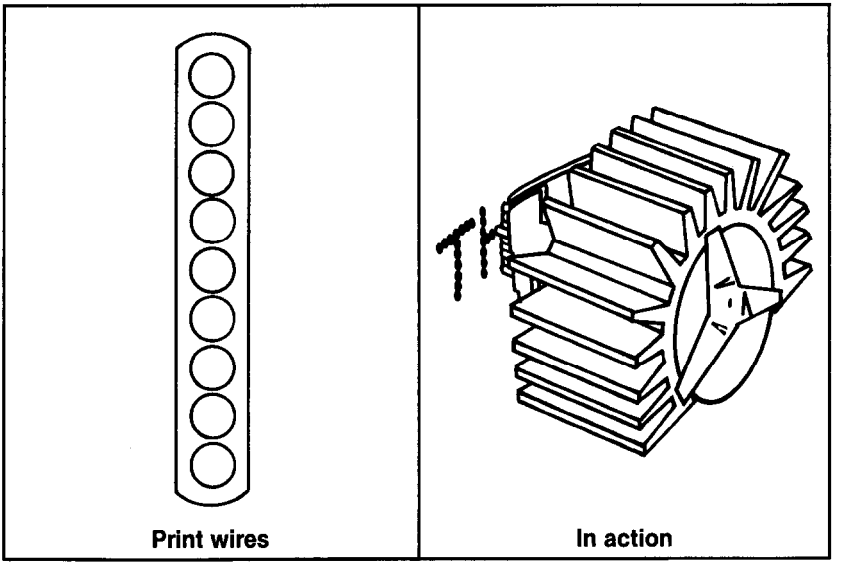


Figure 3-2. The print head

To print a capital H as in Figure 3-2, the print head fires pins 1 through 7 in column **1;** pin 4 in columns 2, 3, and 4; and pins 1 through 7 again in column 5.

Intermediate positions

FX characters are designed to be five or fewer columns wide. Leaving the sixth column blank allows for space between letters. Figure 3-3 shows the 6 main columns, numbered 1, 3, 5, etc.

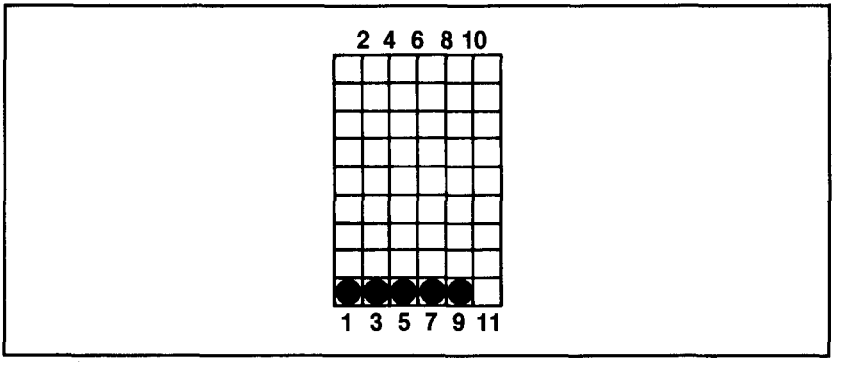


Figure 3-3. Main columns

Because the use of 5 dots does not give quite enough detail for the highest quality characters, an FX prints some dots half way between the main columns in the 6-dot-wide matrix. This enhancement results in **a** matrix grid that is actually 11 dots wide-6 main columns with 5 intermediate columns. You can count the 11 positions on the grid shown as Figure 3-4.

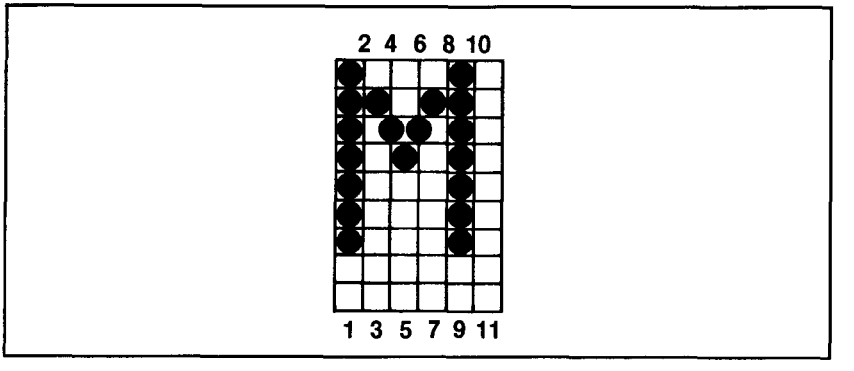


Figure 3-4. Intermediate positions

The dots printed in intermediate column positions would overlap with those in the main columns if they were printed in the same row.

If you look through Appendix A, you'll notice that none of the FX's characters use dots in consecutive main and intermediate columns in the same row. There is a reason for this: the printer's speed. The FX recalls a character's dot-matrix pattern from ROM and prints it in 1/160th of a second. At that speed, the print head is simply moving too fast to pull the pins back and forth in time to print an overlapping dot. This fact is critical when you design characters, as you will see in Chapter 15.

Modes for Pitches

Several of the FX printer's modes produce characters in different widths, or pitches. You may recognize two of these pitches as standard character widths used on typewriters; a third produces a narrower character. Each of these three is covered in one of the subsections below, followed by a discussion of how the FX assigns priorities to its print modes. We discuss other pitch modes later.

Pica and Elite Modes

At start-up the FX prints 10 characters per inch (cpi). This is the same pitch as that of a typewriter's Pica character set. The FX can also print characters in an Elite pitch (12 cpi). When you print in Pica on the FX-80, you can get up to 80 characters on a line; changing to Elite Mode gives you 96. When you print in Pica on the FX-100, you get up to 136 characters per line; Elite Mode gives you 163.

Changing the print pitch between Pica and Elite does not change the number of ***columns*** in each character, nor does it change the ***pattern*** used to create each character. It simply moves the columns closer together or farther apart to change the width of each character. To see how this works, compare the Pica and Elite letters shown in Figure 3-5. Since the columns used to print Elite characters are packed more closely than those for Pica, more of them fit on one line of print. Notice that Elite pitch compresses spaces as well as characters.

Elite Mode (which produces Elite pitch) is set with an ESCape "M". Try it in this program:

```
NEW
10 LPRINT CHR$(27)"M"
20 LPRINT  "ØØØØØØØØØØ    COMPARE ELITE PITCH WITH THE
   PICA BELOW"
```

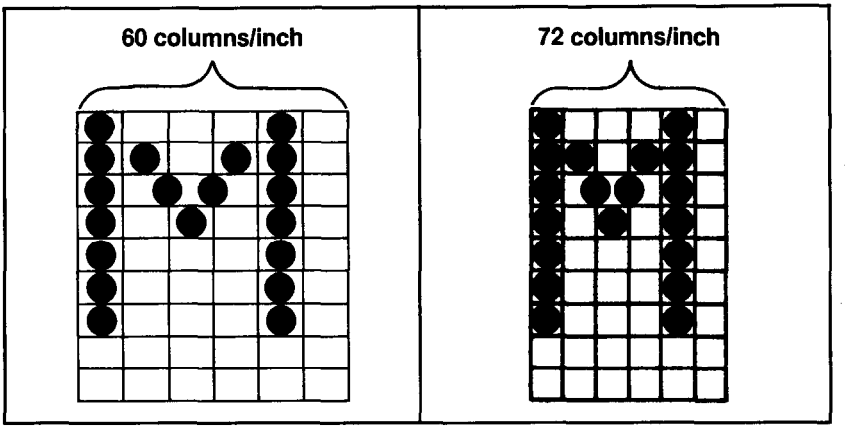


Figure 3-5. Pica and Elite letters

```
3Ø LPRINT  CHR$(27)"P";
4Ø LPRINT      "ØØØØØØØØØØ                PICA PITCH IS THE NORMAL PRINT
    WIDTH"
```

When you RUN it, you should get:

```
COMPARE ELITE PITCH WITH THE PICA BELOW
  PICA PITCH IS THE NORMAL PRINT WIDTH
```

Figure 3-6. Pitch comparison

The 10 blank spaces in line 20 above print as 10 Elite spaces; the 10 corresponding spaces in line 40 print as Pica spaces. This means, as you can see in Figure 3-6, that the different width modes affect spaces as well as characters. The modes also affect tabs, which we will discuss in Chapter 9.

The ESCape "P"' in line 30 turns Elite Mode off and returns the printer to Pica pitch. Notice that, because Pica is the factory-set default, it comes on whenever you turn the printer on (unless you've changed the pitch default by changing a DIP switch).

Compressed Mode

To see a third handy print pitch that is available on FX printers, replace your current program with this one:

```
NEW
2Ø LPRINT CHR$(15)"COMPRESSED MODE IS SET WITH
   CHR$(15)"
3Ø LPRINT "IT WILL STAY ON UNTIL YOU CANCEL IT"
4Ø LPRINT CHR$(l8)"PICA AGAIN"
```

```
COMPRESSED MODE IS SET WITH CHR$(15)
IT WILL STAY ON UNTIL YOU CANCEL IT
PICA  AGAIN
```

Notice that we had you use only CHR$(l5) to turn Compressed Mode on-that is, we didn't have you type in an ESCape code first. If you prefer consistency to brevity, you may add one and use ESCape CHR$(15) to get the same effect.

As do most other print modes on FX printers, Compressed stays on until you turn it off. And there is a specific code that turns Compressed Mode off-CHR$(18). The Reset Code would work just as well, but remember that it also resets all other current printer modes to the defaults. The FX gives you a choice of resetting codes one at a time or all at once.

At 17.16 characters per inch, Compressed Mode is the narrowest character pitch available on FX printers. FX-80 users can fit 132 Compressed characters into each line (up to 137 by changing the right margin setting; see Chapter 9). FX-100 users can get 233 Compressed characters per print line. Compressed Mode is especially useful for printing spreadsheets or reports that require several columns of information.

If you find yourself using this pitch more often than not, you can change the default pitch from Pica to Compressed Mode by setting

DIP switch 1-1 on. This adjustment will make the printer reset to Compressed Mode, after which you can switch to other modes as needed. You could get Pica Mode with control codes, for instance, by using the Compressed shut-off code: CHR$(18). Then you could return to Compressed with either of the usual commands-CHR$(15) or ESCape " @ "-or by turning the printer off and back on.

Mode priorities

These first three pitches-Pica (10 cpi), Elite (12 cpi), and Compressed (17.16 cpi)-are mutually exclusive. That is, only one can be in use at a given time. When a program activates conflicting modes, one of them will take precedence. In the case of Elite, Compressed, and Pica Modes, for example, Elite has the highest priority. To check this, try RUNning this line in your current program. In this example and many others throughout the manual we use the format that combines the command with the print string. The printer will interpret the letter M as part of the command and will not print it. This format is further explained in the preface.

```
10 LPRINT CHR$(27)"MELITE PITCH"
```

```
ELITE PITCH
COMPRESSED MODE IS SET WITH CHR$(15)
IT WILL STAY ON UNTIL YOU CANCEL IT
PICA AGAIN
```

This time the printer doesn't make it to Compressed Mode. Your entire printout is in Elite pitch even though Compressed Mode is turned on in line 20. Does that mean the printer ignores the CHR$(15) when it is in Elite Mode? Let's find out. Change line 30 to read:

```
30 LPRINT CHR$(27)"PCANCEL ELITE TO SEE COMPRESSED"
```

```
ELITE PITCH
COMPRESSED MODE IS SET WITH CHR$(15)
CANCEL ELITE TO SEE COMPRESSED
PICA AGAIN
```

ESCape "P" did cancel Elite Mode, which caused line 30's text to print in Compressed. So the printer did recognize the command for Compressed Mode-CHR$(15)-in line 20, but its program masked that command as long as Elite was in effect.

Don't take this lesson lightly-it is a good example of how print modes interact on FX printers.

Pitch Mode Combinations

The previous three modes can't be mixed, but the next mode can be used in combination with any one of them. And you can add it to a printout for either of two durations, for one print line or for a longer passage.

Expanded Mode

Expanded Mode doubles the width of the current pitch mode. Since it can be combined with all previous pitches, Expanded Mode doubles the number of available print pitches to six.

You can turn on the Expanded print feature for either of two durations. If you activate it with CHR$(14), it turns off after each print line. If you want Expanded Mode to stay on line after line (continuously), you activate it with ESCape "W" followed by CHR$(1).

Also notice that ESCape "W" CHR$(1) is turned off with ESCape "W" CHR(0). CHR$(14), on the other hand, can be turned off either with CHR$(20) or with ESCape "W "CHR$(0).

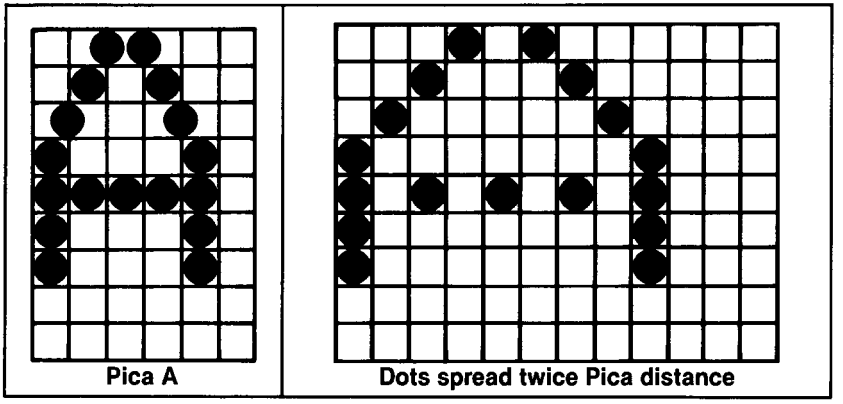
To see Expanded characters, type in:

```
NEW
10 LPRINT CHR$(14)"EXPANDED PRINT"
20 LPRINT "TURNS OFF AFTER EACH LINE WITH CHR$(14)"
30 LPRINT CHR$(27)"W"CHR$(1)"EXPANDED PRINT STAYS
   ON"
```

```
40 LPRINT "CONTINUOUSLY WITH ESCAPE W"
50 LPRINT CHR$(27)"W"CHR$(0)
```

```
EXPANDED PRINT
TURNS OFF AFTER EACH LINE WITH CHR$(14)
EXPANDED PRINT STAYS ON
CONTINUOUSLY WITH ESCAPE W
```

The printer extends the dot matrix by spreading the dots horizontally to twice their normal distances apart, and then it adds a duplicate of each dot to the next main dot column (see Figure **3-7).**



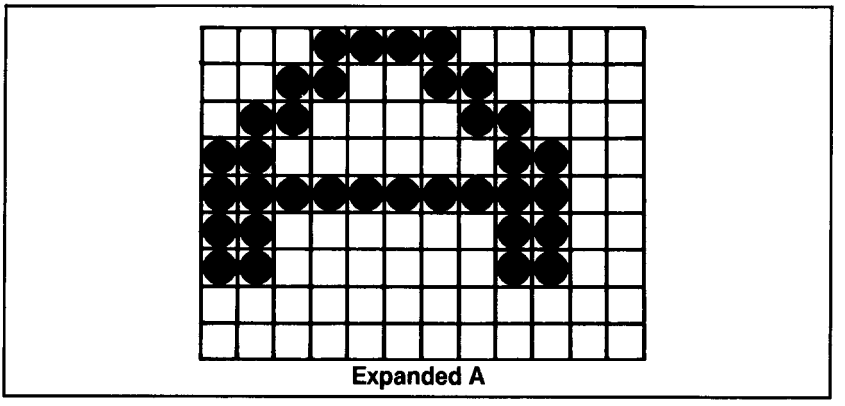


Figure 3-7. Pica and Expanded letters

Those of you who like compact program lines and those of you whose computer systems have difficulty sending a CHR$(0) or

CHR$(1) can use an alternative form for this pair. For continuous Expanded, and for the other modes which use CHR$(1) and CHR$(0) as a toggle switch, you can use an abbreviation. Here, for example, you can use:

```
LPRINT CHR$(27)"W"CHR$(1)
```

or you can use:

```
LPRINT CHR$(27)"W1"
```

for the same result.

Expanded Mode works equally well with any of the three basic pitches. Type and RUN this program:

```
NEW
10 LPRINT "YOU CAN PRINT EXPANDED:"
20 LPRINT CHR$(27)"W1PICA"
30 LPRINT CHR$(15)"COMPRESSED"
40 LPRINT  CHR$(27)"MAND ELITE"
50 LPRINT CHR$(27)"@CHARACTERS"
```

```
YOU CAN PRINT EXPANDED:
PICA
COMPRESSED
AND ELITE
CHARACTERS
```

Most of the FX modes stay on continuously, and so your program must turn off each mode when you are finished with it. We tell you about the few exceptions to this rule-such as the CHR$(14) version of Expanded-as they come up.

Multiple print pitches on one line

Suppose that you want to emphasize just one word within a line by printing it in a different pitch. The following program shows how it can be done:

```
10 LPRINT "YOU CAN MIX: ø";
20 LPRINT "PICAP ø";
30 LPRINT CHR$(14)"EXPANDED,"
40 LPRINT CHR$(20)CHR$(15)  "øCOMPRESSED";
50 LPRINT CHR$(14)  "ø EXPANDED,";
60 LPRINT CHR$(20)CHR$(27)"M AND ELITEP ø";
70 LPRINT CHR$(14) "EXPANDED"
80 LPRINT CHR$(27)"@CHARACTERS ON THE SAME LINE'
```

```
YOU CAN MIX: PICA EXPANDED,
COMPRESSED EXPANDED, AND ELITE EXPANDED
CHARACTERS ON THE SAME LINE
```

By deleting the semicolon at the end of line 10 and adding a semicolon to the end of line 30, you can mix all six print pitches on a single print line.

In program lines 30 to 70, CHR$(14) and CHR$(20) move the printer in and out of Expanded Mode.

This program turns Compressed Mode on in line 40, and Compressed stays on until the Reset Code turns it off in line 80. It gets replaced (masked) in line 60 when the program turns on Elite Mode, which has a higher priority.

Since doubling the width of the three standard pitches adds more dots to the matrix of each character, doubling gives you a darker print. There are other ways you can get darker print, plus many other ways you can change the looks of the type, and we discuss these techniques in the next three chapters.

Summary

The FX uses a dot matrix to plot the characters it prints. Vertically, the matrix consists of 6 main and 5 intermediate columns. Horizontally, the matrix consists of 9 rows. Appendix A shows how each of the FX's characters fits into this matrix.

Using these potential positions, the print head is capable of printing in a large variety of modes. In this chapter we introduced three sets of commands that add five pitches (widths) to the factory-set default, Pica. You can change the default to Compressed by changing DIP switch 1-1.

Table 3-1 shows six pitch modes, their densities in characters per inch, and the commands you use to turn them on. In this table, we assume that you are using Pica as the default. If you have changed to Compressed, you activate Pica with CHR$(18).

Table 3-1. ***Summary of print pitches***

Print sample	CPI	Entry code
← 1 inch →		
PICA PRINT	10.00	Default mode (Roman Pica)
ELITE PRINT	12.00	CHR$(27)"M"
COMPRESSED PRINT	17.16	CHR$(15)
EXPANDED PICA PRINT	5.00	CHR$(27)"W1" or CHR$(14)
EXPANDED ELITE PRINT	6.00	CHR$(27)"W1" CHR$(27)"M"
EXPANDED COMPRESSED PRINT	8.58	CHR$(27)"W1"CHR$(15)
← 1 inch →		

Here is the DIP switch that we mentioned in this chapter:

Switch 1-1	Allows you to change the pitch default from Pica to Compressed

Here is a list of the commands that we introduced in this chapter, listed in the order of their appearance:

CHR$(27)"M"	Turns Elite Mode ON
CHR$(27)"P"	Turns Elite OFF
CHR$(15)	Turns Compressed Mode ON
CHR$(18)	Turns Compressed (set either by software or by hardware) OFF
CHR$(14)	Turns Expanded Mode (1-line) ON
CHR$(20)	Turns Expanded (1-line) OFF
CHR$(27)"W1"	Turns Expanded Mode (continuous version) ON
CHR$(27)"W0"	Turns Expanded (either the 1-line or the continuous version) OFF

Chapter 4
Print Quality

In the last chapter you learned how to change the width of the printed characters to achieve six different print ***pitches.*** The FX printer also offers several modes that improve print ***quality*** without affecting pitch. The three new modes that we discuss in this chapter are Double-Strike, Emphasized, and Proportional. After we cover these modes separately, we discuss combining them with pitch modes.

Bold Modes

Each of the modes we discuss here produces darker characters than do the Single-Strike modes we discussed in Chapter 3, and each gets its bold effect by printing overlapping dots. Because Proportional Mode includes Emphasized, it also produces bold characters. The difference between Double-Strike and Emphasized Modes lies in the direction the print head moves before it prints the overlapping dots.

Double-Strike Mode

The Double-Strike Mode prints each line twice. You turn Double-Strike on with ESCape "G". It stays on until you turn it off with ESCape "H". To see its effect, try this:

```
NEW
10 LPRINT CHR$(27) "GDOUBLE-STRIKE PRINT IS DARKER";
20 LPRINT CHR$(27) "H THAN SINGLE-STRIKE"
```

and RUN it:

```
DOUBLE-STRIKE PRINT IS DARKER THAN SINGLE-STRIKE
```

In this mode the individual dots are less noticeable, and the characters are more fully formed than in Single-Strike.

The way Double-Strike gets this result is rather clever: the FX prints each character in the regular fashion until it reaches either the end of the line or the point at which you have Double-Strike turn off. Then the FX shifts the paper up slightly and prints the Double-Strike passage again. This means that every dot in each row of the character gets a shadow (see Figure 4-1). Double-Strike Mode fills in some of the more visible gaps between the dots of a character. The end result is better looking print.

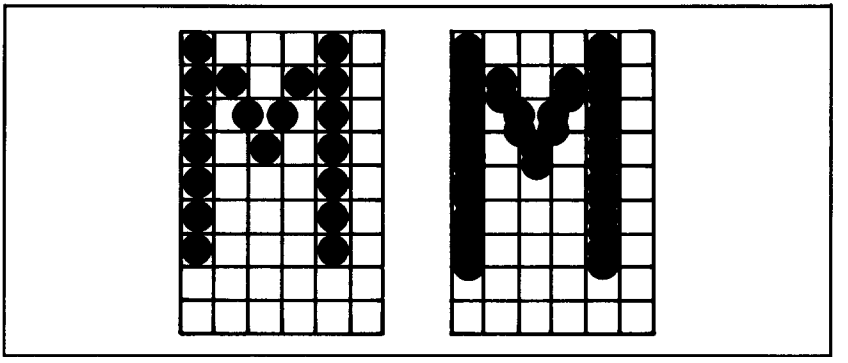


Figure** 4-1. **Single-Strike and Double-Strike letters

Differences between Double-Strike and Single-Strike printing don't stop with the quality of print. Since each passage prints twice, the throughput of the Double-Strike Mode is less than that of Single-Strike. It's the old trade-off between speed and print quality. With **a** normal print speed of 160 characters per second (cps), the FX still moves along pretty quickly in the Double-Strike Mode.

Emphasized Mode

There is yet another way you can increase the boldness of your printed characters. ESCape "E" produces what we call Emphasized print. As in Double-Strike, each character gets two sets of dots. In Emphasized, however, the print head does not make two passes and does not move down the page. Instead, it slows down so that it can print overlapping dots, and prints each dot twice, the second time slightly to the right of the first, as illustrated in Figure 4-2.

To see Emphasized, add these lines to your program:

```
30 LPRINT CHR$(27)"EEMPHASIZED ADDS A TOUCH OF
   CLASS"
60 LPRINT  CHR$(27)"@"
```

DOUBLE-STRIKE PRINT IS DARKER THAN SINGLE-STRIKE
EMPHASIZED ADDS A TOUCH OF CLASS

That's right, Emphasized is very similar to Expanded print, except that Expanded Mode prints a duplicate set of dots a full (rather than a half) column to the right of the initial set. Figure ***4-2*** shows Expanded and Emphasized characters.

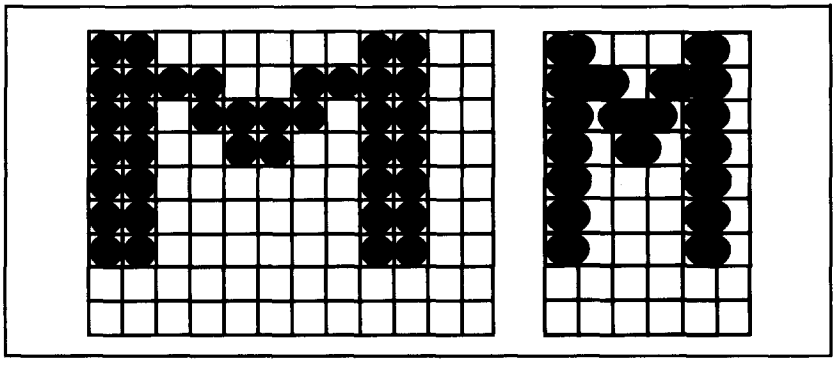


Figure 4-2. Expanded and Emphasized letters

Although the print head slows to half normal speed (i.e., 80 cps) in Emphasized Mode, the increase in print quality is well worth it. If you like Emphasized so well that you want to use it for most of your printouts, you can set it as a default with DIP switch 1-5. This adjustment will make the printer automatically reset to Emphasized Mode, after which you can switch to other modes as needed.

Whether you turn Emphasized on with ESCape "E" or by DIP switch, you can turn it off with ESCape "F".

Emphasized Mode is always a variation of Pica; it can never be combined with either the Elite or the Compressed Mode. It can, however, be added to Double-Strike. See this by adding:

```
40 LPRINT CHR$(27)"GCOMBINED THEY CAN'T BE BEAT"
```

to your program.

DOUBLE-STRIKE PRINT IS DARKER THAN SINGLE-STRIKE
EMPHASIZED ADDS A TOUCH OF CLASS
COMBINED **THEY** CAN'T **BE BEAT**

Emphasized Mode (line 30) stays on until you shut it off. Double-Strike comes on (line 40) before Emphasized is turned off. You see the result above.

Proportional Mode

Have you ever wondered why most computer printouts don't look as good as typeset books, even when you use bold characters? It's because most dot-matrix printers use a uniform width for each character (monospacing) whereas typesetting machines set the width for each character proportional to its size. That is, narrow characters like i and ! are printed without the excess space that would be used if they were printed in the same width as m and w.

Now the FX offers you a Proportional Mode on a dot-matrix printer. In this mode characters are printed with a uniform amount of blank space between them. ESCape "p1" turns on the Proportional Mode. If your computer system doesn't allow you to use lowercase letters, you may use a longer form instead: ESCape CHR$(112) "1".

Here's an example of the difference between Monospaced and Proportional Modes. Enter:

```
NEW
10 LPRINT If"!!!!!!!!!!!!!";
20 LPRINT CHR$(27)"p1"
40 LPRINT "!!!!!!!!!!!!!PROPORTIONAL ON"
60 LPRINT CHR$(27) "p0";
70 LPRINT "!!!!!PROPORTIONAL OFF"
80 LPRINT CHR$(27)"@"
```

```
! ! ! ! ! ! ! ! ! ! !
!!!!!!!!!!!!PROFORTIONAL ON
!!!!!PROPORTIONAL OFF
```

Lines 10 and 40 print the same number of exclamation marks, but the characters from line 40 are packed more closely. Proportional Mode prints the characters in Emphasized and strips off all unused space between characters.

The shut-off command for Proportional print is either ESCape "p0" or ESCape CHR$(112)"0". Either puts the printer back into the mode that it was in before it entered Proportional Mode. As an example, if the printer enters Proportional Mode from Compressed Mode, ESCape "p0" returns the printer to that mode.

Since all Proportional characters are Emphasized, it makes sense that Proportional characters, like Emphasized, can only be printed in Pica pitch, not Elite nor Compressed. In addition, Proportional Mode cannot be mixed with Double-Strike.

The cost of all this high-powered printing is the slower speed of printing and the wear and tear on the ribbon. Understandably, these dense modes shorten the life of a ribbon compared to Single-Strike printing. Used sparingly, however, they can give you increased capability for little increased cost.

Mixing Modes

Much of the fun of owning an FX is trying out all the different mode combinations. In the process you'll find that some modes make magic together, while others cannot be mixed at all. Most important, you'll gain an insight into the way the print modes work, and you'll attain a higher level of control over your printer.

With the many modes available on the FX, there are enough mode combinations to suit just about any palate. It's impossible, however, for each print mode to get along with all others as well as Emphasized does with Double-Strike. For example, you cannot mix either Emphasized or its variation, Proportional, with Elite or Compressed pitches for a very good reason.

Emphasized characters already violate the rule that two overlapping dots cannot be printed in the same row. Since the FX prints Emphasized in Pica Mode at half normal speed, it can make an exception. But the dots in Elite and Compressed characters are already so closely packed that printing them in Emphasized print as well is not possible, even at the slower speed. The print head simply cannot fire and retract the pins fast enough.

So what does the printer do when it receives a request for two conflicting modes? Ignore one of them, take a vacation, or beep loudly? The answer is: none of the above. It turns both modes on internally, but-based on a factory-set priority list-selects only one of them for printing text.

As you found out in Chapter 3 when you cancelled Elite to see Compressed print, the FX remembers that a mode is on even though that doesn't affect the current print line. Just to produce a little more printout evidence, change your program to turn on Double-Strike at the same time as Proportional, Then, when you turn Proportional off,

the printer will prove that Double-Strike has been turned on all the time.

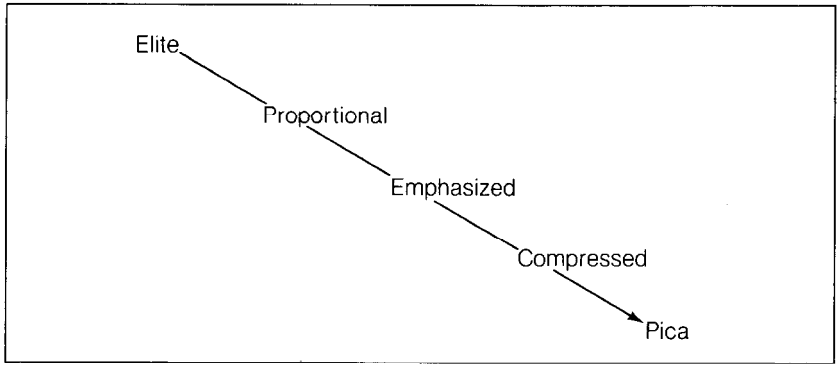
Add lines 30 and 50, and make some changes to line 70:

```
30 LPRINT CHR$(27)"G";
50 LPRINT "WHEN PROPORTIONAL GOES"
70 LPRINT "!!!!!!!!!!!!!OFF, DOUBLE-STRIKE CAN COME
   ON"
```

```
 !!!!!!!!!!!!
!!!!!!!!!!!!PROPORTIONAL  ON
WHEN PROPORTIONAL GOES
!!!!!!!!!!!!!OFF,  DOUELE-STRIKE CAN COME ON
```

Even though Proportional Mode will not permit Double-Strike to affect lines 40 and ***50*** (since Proprotional Mode has priority), Double-Strike does take hold as soon as Proportional is shut off.

Figure 4-3 shows the priorities for the modes we've covered so far.



Note: Each mode takes precedence over the modes beneath it.

Figure 4-3. Mode priorities

Not all mode combinations create conflicts. You've already seen how well Emphasized and Double-Strike combine. In the next chapter we show you more modes to mix with these.

Summary

Double-Strike, Emphasized, and Proportional Modes can be used to produce bold characters. Proportional, which prints in Empha-

sized, strips excess space from between characters. Double-Strike can be combined with all other modes except Proportional, whereas Emphasized, and thus Proportional, cannot be combined with either Elite or Compressed.

Mode combinations are governed by the FX's priority list. This list determines which mode gets printed when two or more conflicting modes are active at the same time.

Table 4-1 shows the modes we have covered so far.

Table 4-1. Summary of modes

Type of mode	Mode name
Typeface	Roman (default) Italic
Pitch	Pica (default) Elite Compressed Expanded
Weight	Single-Strike (default) Double-Strike Emphasized
Spacing	Monospaced (default) Proportional

Note: Pitch and weight together make up print density. The defaults are those set at at the factory. By changing DIP switches, you can change the pitch default from Pica to Compressed and the weight default from Single-Strike to Emphasized.

Here is the DIP switch that we mentioned in this chapter:

Switch 1-5 Allows you to change the weight default from Single-Strike to Emphasized

Here are the commands that we introduced in this chapter:

CHR$(27) "G" Turns Double-Strike Mode ON. Double-Strike prints in Half-Speed

CHR$(27) "H" Turns Double-Strike OFF

CHR$(27) "E" Turns Emphasized Mode ON. Emphasized prints in Half-Speed

CHR!$(27) "F" Turns Emphasized OFF

CHR$(27) "p1" or CHR$(27) CHR$(112)"1"
Turns Proportional Mode ON. Proportional prints in Emphasized and thus Half-Speed.

CHR$(27) "p0" or CHR$(27) CHR$(112)"0"
Turns Proportional OFF

Chapter 5
Dress-Up Modes and Master Select

In the first three subsections of this chapter, we cover four more print modes: Underline; two Script Modes-Superscript and Subscript; and Italic. Each of these modes allows you to add a particular finishing touch to your printouts. After we show how you can quickly select 16 combinations of pitch and weight by using the Master Select feature, we demonstrate combining the combinations.

Four Modes

These four modes give you capabilities that are unusual for dot-matrix printing: an underline of any character or blank space, true Super- and Subscript characters, and an Italic Mode that any computer system can select without software adjustments.

Underline Mode

In the old days, dot-matrix printers could not underline words. Even in the not-so-old days, printer users like you had to use all kinds of tricks to underline words. The technique usually involved using either the hyphen (-) or underscore (_), along with either a change of line spacing or the use of backspace. When it worked, the right words did get underlined, but those methods were tedious and time consuming.

Those days are now gone, thanks to the FX. It has a built-in Underline Mode that makes underlining very easy. You can toggle the control code for Underline, ESCape "-", on and off just as you toggle the code for Expanded Mode. To turn Underline on, you use:

```
CHR$(27)"-"CHR$(1) or    CHR$(27)"-1"
```

You can turn Underline Mode off with:

```
CHR$(27)"-"CHR$(0) or    CHR$(27)"-0"
```

Enter and RUN this program to see what FX underlining looks like:

```
NEW
20 LPRINT CHR$(27)"-1UNDERLINING IS SIMPLE";
40 LPRINT CHR$(27)"-0 TO TURN ON/OFF"
```

```
UNDERLINING IS SIMPLE TO TURN ON/OFF
```

You can underline virtually anything you want-even a series of blank spaces. Try adding this line, typing in the spaces where we have used Ø characters:

```
30 LPRINT    "ØØØØØØØ";
```

```
UNDERLINING IS  SIMPLE          TO TURN ON/OFF
```

Being able to underline a blank space really helps when you want to make a form that has lines to be filled in later with signatures or data. The FX Underline Mode can even underline leading spaces with BASIC TABS-although it doesn't work with the FX's internal tabs that are set with ESCape "D" (see Chapter 9).

Underline Mode does not use the FX's underline character. Since the underline character is itself only five dots wide, it would not print in the spaces that separate the text characters. See this by printing a row of underline characters above the text in the current program:

```
10 FOR X=1 TO 42: LPRINT CHR$(95);: NEXT X: LPRINT
```

```
UNDERLINING IS SIMPLE TO TURN ON/OFF
```

When the FX is in Underline Mode, it is simply using the ninth pin to underscore the entire passage.

How far under a character its underline appears depends on the model of FX that is printing. The FX-100 prints a character and its underline at the same time, and the underline occurs on the ninth row of the nine-row character matrix. This means that the underline prints over the bottom row of the descender for a few lowercase letters, such as y and g.

The FX-80, on the other hand, can perform a reverse line feed, and it uses this capability to place the underline one row lower than any text dot. To do this, the FX-80 prints the text to be underlined, moves the print head down the paper one row's worth to print the underline, then moves the print head back up to the original text line.

To see which method your printer uses, delete line 10 and replace line 30:

```
30 FOR X=1 TO 5: LPRINT CHR$(l03);: NEXT X
```

```
UNDERLINING IS SIMPLEggggg TO TURN ON/OFF
```

On the FX-80, the underlining does occur below the descenders of the gs.

Script Modes: Super and Sub

Print pitches, such as Pica, Elite, and Compressed, affect the width of characters: both Elite and Compressed Modes may be seen as squeezing Pica characters ***horizontally.*** FX printers can also squeeze a character vertically-to about half its normal height. These short characters are called Script characters (not to be confused with calligraphic manuscript lettering). Because the Script characters are so short, they can be placed high or low on the line; thus you can use them for either SUPERscripts or SUBscripts.

This command turns Superscripting on:

```
LPRINT CHR$(27)"50"
```

This one turns Subscripting on:

```
LPRINT CHR$(27)"S1"
```

Try this program to see both kinds of Script characters:

```
20 LPRINT CHR$(27)"S0SUPER";
30 LPRINT CHR$(27)"TSCRIPT AND";
40 LPRINT CHR$(27)"S1 SUB";
50 LPRINT CHR$(27)"TSCRIPT"
60 LPRINT "CAN EVEN GO ON THE SAME LINE"
```

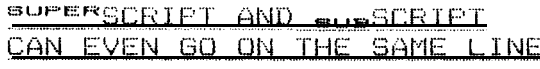
```
SUPERSCRIPT AND SUBSCRIPT
CAN EVEN GO ON THE SAME LINE
```

Notice that ESCape "T" turns either kind of Script Mode off and also that both versions of Script Mode are automatically printed in Double-Strike. Since Double-Strike prints at half speed, so do the Script Modes. And since Double-Strike can't mix with Proportional, neither can either type of Script.

If you are using the FX-100 and you switch in and out of Double-Strike Mode several times in the same line, you will see that the print line slopes slightly to the right. This happens because Script prints in Double-Strike, for which FX printers use two passes. After one pass to print the characters on the current print line, the print head moves down the page one-third of a dot to print the second set of dots.

The FX-80 can use reverse feed to return to the original print line, but the FX-100 cannot. This does not make much difference when you use Double-Strike infrequently. If you are using an FX-100, you can add Underline Mode to your program to see exactly where the jog comes:

```
10 LPRINT CHR$(27)"-1"
```



We mention this now because it is with Script characters that you are most likely to move in and out of Double-Strike several times on one line. If you do that on an FX-100, you'll get a line that declines.

If Double-Strike was the current mode before the printer entered Script Mode, the printer exits from Script to Double-Strike. If not, the printer returns to Single-Strike.

Italic Mode

This is the last print mode that we introduce in this chapter. Italic characters are printed in a completely different typeface than are the more usual Roman characters. Appendix A shows the dot-matrix patterns used to define both Roman and Italic characters, along with their corresponding ASCII numbers. As you saw in Chapter 2, the Italic characters' numbering begins in the top half of the ASCII range (at 160, to be exact). And some computer systems won't let the numbers of the top half through to the printer.

Whether your computer system is one of these or not, with ESCape "4" you can print Italic ***characters.*** Prove it by adding these lines to your program:

```
10 LPRINT CHR$(27)"4"
70 LPRINT CHR$(27)"@"
```

```
SUPERSCRIPT AND SUBSCRIPT
CAN EVEN GO ON THE SAME LINE
```

When you want to turn off only the Italic Mode, you use ESCape **"5"** (instead of line 70's Reset Code) in your program.

More Mode Combinations

With all the handsome print types we've covered up to this point, you're probably wondering how many different print combinations are waiting in your FX. Well, hang onto your hat. By combining the various print modes, the FX can print text in **128** type styles.

Instead of counting these one by one, we'll begin by describing a special FX feature, the Master Select, which lets you choose any one of **16** popular mode combinations more simply than by calling up each mode separately. Then we'll demonstrate mixing Master Select choices with the modes of this chapter.

Master Select

For Master Select you use the code sequence of ESCape " ! " followed by CHR$ and a number in parentheses. With Master Select you can quickly produce any possible combination of the following:

Single-Strike
Pica
Elite
Compressed
Emphasized
Double-Strike
Expanded

These are the modes for pitch and weight. Master Select will accept any number between 0 and **255.** Since there are only **16** unique combinations, however, we've devised a program (Figure **5-1)** with which you can print out a handy reference chart (Figure **5-2)** listing one set of Master Select numbers.

```
NEW
20 Y$(1)="SINGLE-STRIKE ø": Y$(2)="SNGL-STRIKE
   EMPHASIZED ø"
30 Y$(3)="DOUBLE-STRIKE ø": Y$(4)="DBL-STRIKE
   EMPHASIZED ø"
40 Z$(1)="PICA ø": Z$(2)="ELITE ø":
   Z$(3)="COMPRESSED ø"
50, FOR X=1 TO 2
60 FOR Y=1 TO 4
70 FOR Z=1 TO 3
80 READ N: IF N<0 THEN 130
90 LPRINT CHR$(27)"!"CHR$(0);: IF N<10 THEN LPRINT
   " ";
95 ' OK to substitute CHR$(2) for CHR$(0)
l00 LPRINT N; CHR$(27)"!"CHR$(N);
110 LPRINT Y$(Y);: IF X=2 THEN LPRINT "EXPANDED ø";
120 LPRINT Z$(Z)
130 NEXT Z: NEXT Y: NEXT X
140 LPRINT CHR$(27)"!"CHR$(0)
150 DATA 0,1,4,8,-1,-1,16,17,20,24,-1,-1
160 DATA 32,33,36,40,-1,-1,48,49,52,56,-1,-1
```

Figure 5-1. Master Select Program

If you want to see this in underlined Italic add the following line:

```
10 LPRINT CHR$(27)"-1"CHR$(27)"S0"CHR$(27)"4";
```

Now when you want to use Double-Strike Emphasized Expanded Pica, you do not have to type out the long sequence:

```
LPRINT  CHR$(27)"G"CHR$(27)"E"CHR$(27)"W1"
```

since you can get the same result by using:

```
LPRINT CHR$(27)"!"CHR$(56)
```

or even (after consulting the ASCII table for the equivalent of 56):

```
LPRINT CHR$(27)"!8"
```

Because this latter format is the ultimate in simplicity, we use it in Table 5-1 below. You can find the Master Select code for any valid combination of pitch and weight by reading across in the row for the pitch you have selected and down in the column for the weight you want. Where the two intersect you will find the ASCII symbol to use in the simplified format. For example, to combine Compressed with

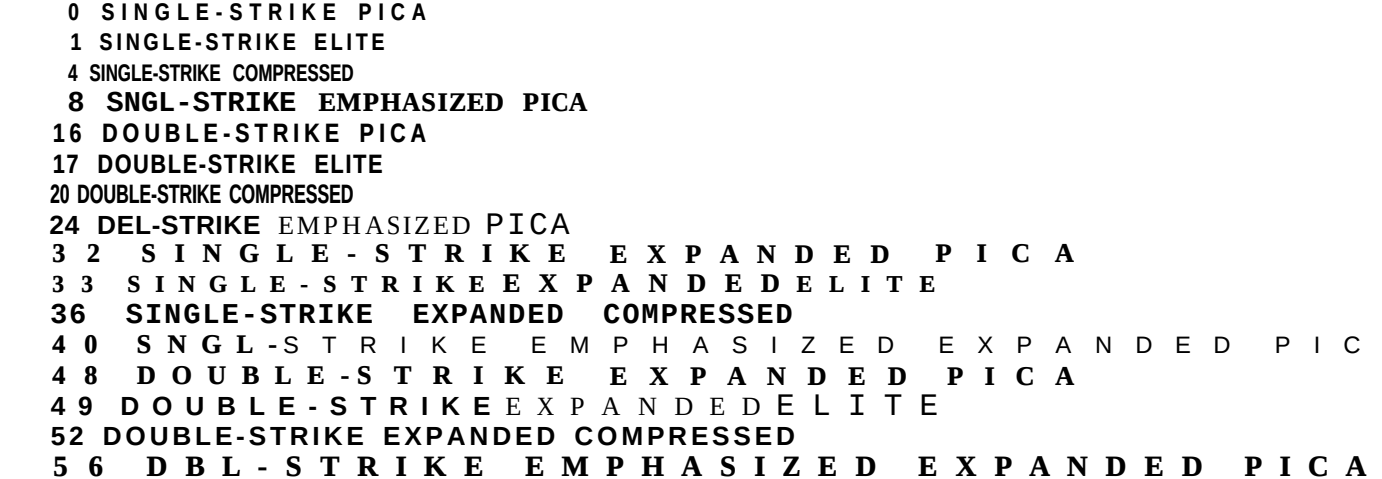


Figure 5-2. Master Select choices

Double-Strike, use LPRINT CHR$(27)"!T". N/A indicates that the two modes cannot be combined.

Table 5-1. Master Select Quick Reference Chart

PITCH	WEIGHT			
	Single Strike	Emphasized	Double Strike	Double Strike Emphasized
Pica	@	H	P	X
Elite	A	N/A	Q	N/A
Compressed	D	N/A	T	N/A
Expanded Pica	℔	*	0	8
Expanded Elite	!	NIA	1	N/A
Expanded Compressed	$	N/A	4	N/A

If you rely on Master Select to change print modes, you don't have to turn each mode off separately when you want to change to a different combination. When you select a new mix by Master Select, it takes care of all resetting. Because the Master Select also overrides the DIP switch settings, you use the same codes even if you have reset some of the DIP switches.

Master Select combinations

At the beginning of this chapter we introduced you to four print modes. Figure 5-3 illustrates the 11 different ways these can be combined.

If you add the default Pica Mode, there are 12 different combinations. Four of these 12 combinations (the ones that don't involve Super- or Subscript) can be combined with any of the 16 Master Select modes. Four times 16 gives you 64 combinations. The other eight combinations that involve Super- and Subscript Modes can be combined with the eight Single-Strike Master Select modes. Eight times eight gives another 64 combinations. Add these to the first 64, and you get the total of 128 unique print modes that we mentioned previously.

To create a new combination that uses both Master Select and one or more of the dress-up modes, the simplest method is to choose a

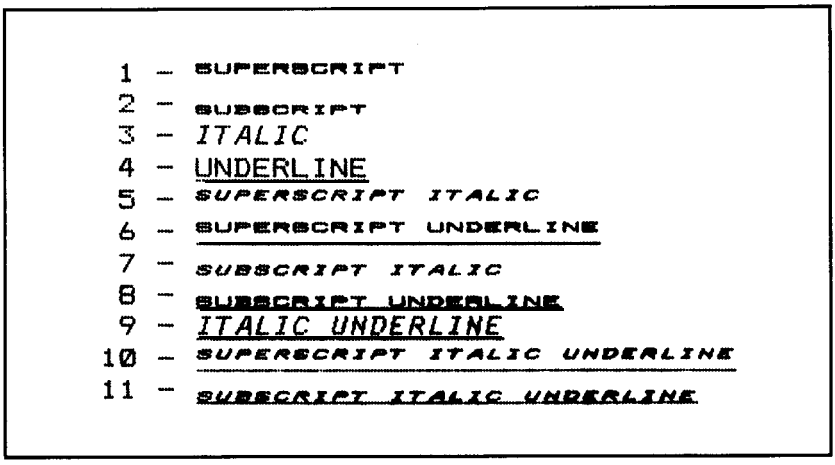


Figure 5-3. Dress-up combinations

Master Select base and then add the sequence(s) that you want to embellish it. Here is a program that does just that, several times.

```
NEW
10 N=4: GOSUB 70: LPRINT CHR$(27)"S1THE
   FX"CHR$(27)"T"
20 N=17: GOSUB 70: LPRINT CHR$(27)"-1PRINTERS"
30 N=8: GOSUB 70: LPRINT CHR$(27)"-0HAVE EVER"
40 N=49: GOSUB 70: LPRINT "EXPANDING"
50 N=56: GOSUB 70: LPRINT  CHR$(27)"4POSSIBLITIES"
60 LPRINT CHR$(27)"@": END
70 LPRINT CHR$(27)"!"CHR$(N);: RETURN
```

```
THE FX
PRINTERS
HAVE EVER
EXPANDING
POSSIBILITIES
```

Notice that we've stored the Master Select sequence as a subroutine in line 70 to save typing. As we've called up the subroutine and given it a new number, we've three times added the ESCape codes for Subscript (line 10), Underline (line ***20),*** and Italic (line ***50)*** Modes. The Reset in line ***60*** returns the printer to its defaults.

To aid you in quickly finding your way around in this forest of possibilities, we offer Table ***5-2.***

	PICA PRINT	SUPER SCRIPT	SUB SCRIPT	ITALICS	UNDER LINE	SUPER SCRIPT ITALICS	SUPER SCRIPT UNDER LINE	SUB SCRIPT ITALICS	SUB SCRIPT UNDER LINE	ITALICS UNDER LINE	SUPER SCRIPT ITALICS UNDER LINE	SUB SCRIPT ITALICS UNDER LINE
SINGLE-STRIKE PICA	ABCD	.....	.....	*ABCD*	ABCD	.....	.....	.....	.....	*ABCD*	.....	.....
SINGLE-STRIKE ELITE	ABCDE	.....	.....	*ABCDE*	ABCDE	.....	.....	.....	.....	*ABCDE*	.....	.....
SINGLE-STRIKE COMPRESSED	ABCDEFG	.....	.....	*ABCDEFG*	ABCDEFG	.....	.....	.....	.....	*ABCDEFG*	.....	.....
SINGLE-STRIKE EMP PICA	ABCD	.....	.....	*ABCD*	ABCD	.....	.....	.....	.....	*ABCD*	.....	.....
DOUBLE-STRIKE PICA	ABCD	ABCD	ABCD	*ABCD*	ABCD	*ABCD*	ABCD	*ABCD*	ABCD	*ABCD*	*ABCD*	*ABCD*
DOUBLE-STRIKE ELITE	ABCDE	ABCDE	ABCDE	*ABCDE*	ABCDE	*ABCDE*	ABCDE	*ABCDE*	ABCDE	*ABCDE*	*ABCDE*	*ABCDE*
DOUBLE-STRIKE COMPRESSED	ABCDEFG	ABCDEFG	ABCDEFG	*ABCDEFG*	ABCDEFG	*ABCDEFG*	ABCDEFG	*ABCDEFG*	ABCDEFG	*ABCDEFG*	*ABCDEFG*	*ABCDEFG*
DOUBLE-STRIKE EMPHASIZED PICA	ABCD	ABCD	ABCD	*ABCD*	ABCD	*ABCD*	ABCD	*ABCD*	ABCD	*ABCD*	*ABCD*	*ABCD*
SINGLE-STRIKE EXPANDED PICA	AB	.....	.....	*AB*	AB	.....	.....	.....	.....	*AB*	.....	.....
SINGLE-STRIKE EXPANDED ELITE	AB	.....	.....	*AB*	AB	.....	.....	.....	.....	*AB*	.....	.....
SINGLE-STRIKE EXPANDED COMPRESSED	ABC	.....	.....	*ABC*	ABC	.....	.....	.....	.....	*ABC*	.....	.....
SINGLE-STRIKE EMPHASIZED EXPANDED PICA	AB	.....	.....	*AB*	AB	.....	.....	.....	.....	*AB*	.....	.....
DOUBLE-STRIKE EXPANDED PICA	AB	AB	AB	*AB*	AB	*AB*	AB	*AB*	AB	*AB*	*AB*	*AB*
DOUBLE-STRIKE EXPANDED ELITE	AB	AB	AB	*AB*	AB	*AB*	AB	*AB*	AB	*AB*	*AB*	*AB*
DOUBLE-STRIKE EXPANDED COMPRESSED	ABC	ABC	ABC	*ABC*	ABC	*ABC*	ABC	*ABC*	ABC	*ABC*	*ABC*	*ABC*
DOUBLE-STRIKE EMPHASIZED EXPANDED PICA	AB	AB	AB	*AB*	AB	*AB*	AB	*AB*	AB	*AB*	*AB*	*AB*

Table 5-2. Print types

Summary

You select Script, Underline, and Italic Modes with ESCape codes. You can combine these modes with a Master Select combination of modes for pitch and weight to form 128 print styles.

Appendix A shows the dot matrixes for all characters in the Italic typeface.

Here are the commands that we introduced in this chapter.

CHR$(27) "-1"	Turns Underline Mode ON
CHR$(27)"-0"	Turns Underline OFF
CHR$(27)"S1"	Turns Subscript Mode ON. Script characters print in Double-Strike and at half speed
CHR$(27)"S0"	Turns Superscript ON
CHR$(27)"T"	Turns either version of Script Mode OFF
CHR$(27)"4"	Turns Italic Mode ON
CHR$(27)"5"	Turns Italic OFF
CHR$(27)"!"CHR$(n) or CHR$(27)"!n"	Selects one of the 16 Master Select combinations, where n stands for a number between 0 and 255

Chapter 6
Special Printing Features

In this chapter you'll discover several new features that will enhance your control over the printer. Backspacing, for example, allows you to combine characters. You can use a set of software commands to switch in and out of international character sets, and you can control the speed of printing.

Backspace

The backspace function is handy for making overstrikes. Because it moves the print head backward one character, you can print two characters in one print position. You can also shift the print head slightly to print offsets. You activate backspace by typing a CHR$(8) between the two characters.

Overstrikes

Typically, backspace is used for single-character overstrikes. Here are a few examples that create some useful mathematical symbols:

```
NEW
10 LPRINT "=" CHR$(8) "/" ' Not-equal
```

≠

For the not-equal symbol,‘ the backspace moves the print head back over the equal sign so that the slash can be printed on top of it. You can

use the same technique to produce the plus-or-minus symbol:

```
10 LPRINT CHR$(27)"S0+"CHR$(8); 'Plus/minus
20 LPRINT CHR$(27)"S1-"
30 LPRINT CHR$(27)"@"
```

±

How about that, and it only took three lines. Next try this approximately equally short program:

```
10 LPRINT CHR$(126)CHR$(8); ' Approximately equal
20 LPRINT CHR$(27)"J"CHR$(11)CHR$(126)
30 LPRINT CHR$(27)"@"
```

≈

This program prints CHR$(126), a diacritical mark used in Spanish that is called a tilde. You backspace the print head by using CHR$(8), then force a partial line feed by using Escape "J" CHR$(11). The FX prints the second tilde just below the first one to form the desired figure.

Offsets

The backspace function works in all pitches, which opens up some interesting possibilities. To see what happens when you mix backspacing with different pitch modes, we've prepared a special program that shows off the backspace function in two passes: first in Pica and then in Expanded Pica. We cause all backspacing to be done in Compressed Mode to create a slight offset. Enter:

```
NEW
10 FOR J=0 TO 1
20 LPRINT CHR$(27)"W"CHR$(J); ' Expanded when J=1
30 LPRINT "BACKSPACES"CHR$(15); ' Compressed
40 FOR X=1 TO 17: LPRINT CHR$(8);: NEXT X
50 LPRINT CHR$(18)"BACKSPACES" ' Compressed off
60 NEXT J: LPRINT CHR$(27)"@" ' Reset
```

and RUN the program.

BACKSPACES
BACKSPACES

The 17 backspaces (line 40) are printed in Compressed Mode. The difference in character widths makes the second printing of the word BACKSPACES be offset from the first.

In the next program, the offset is a little more dramatic. Change the following lines:

```
30 LPRINT  "BACKSPACE"CHR$(15);
40 FOR X=1 TO 15: LPRINT CHR$(8);: NEXT X
50  LPRINT  CHR$(18)"BACKSPACE"
```

```
BACKSPACES
BACKSPACES
```

After the FX prints each BACKSPACE, it moves the print head 15 Compressed positions backward. Instead of bold characters, you get a shadow effect. You could spend all day mixing pitch modes to get different eye-stretching effects.

Unidirectional Mode

The FX printers provide high quality printouts in their normal bidirectional print mode. However, there may be situations in which vertical columns printed in Elite or Compressed Mode get slightly misaligned. The printer has a Unidirectional Mode to prevent such problems .

To see how effective Unidirectional Mode can be, let's create a long vertical line. First we'll print it in the usual, bidirectional manner. Begin by typing:

```
20  LPRINT  CHR$(27)"1"CHR$(27)"1"CHR$(40)
40 FOR X=1 TO 10: LPRINT CHR$(124): NEXT X
50 LPRINT  CHR$(27)"@"
```

Line 20 sets the line spacing to seven dots and the left margin to 40. To create the vertical line, line 40 prints one character (represented by ASCII 124) 10 times (on 10 print lines). Line 50 resets the printer to its defaults.

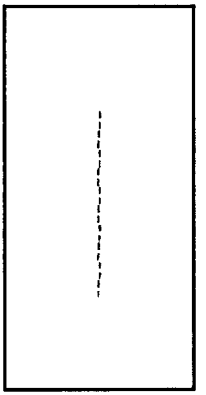


Figure 6-1. Bidirectional line

Look carefully at your printout or at the version we show as Figure ***6-1.*** See how the line seems to quiver? Now turn on Unidirectional printing to see how much difference it makes. Add line 10 and RUN the program again:

```
10 LPRINT CHR$(27)"U1"
```

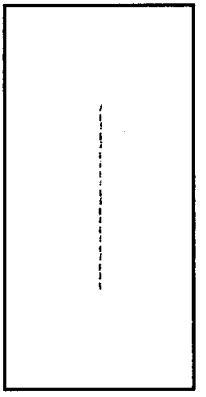


Figure 6-2. Unidirectional line

CHR$(27) “U1" turns on the Unidirectional printing whose results we show as Figure 6-2, and CHR$(27)“U0” turns it off.

Did you watch the print head as it printed? In every row it moved from left to right instead of alternating directions as usual.

You can get the same effect for one line at a time by using the command CHR$(27)"<”. This command causes a carriage return with no

line feed, which means that the subsequent movement of the print head will be from the left margin to the right. To see this in action, delete line 10 and change line 40 to read:

```
40 FOR X=1 TO 10: LPRINT CHR$(27)"<"CHR$(124):
   NEXT X
```

When you RUN it, you can watch the print head move to its leftmost position after it prints each line.

Unidirectional print motion straightens out the slight misalignment of characters that results from printing bidirectionally in the Elite or Compressed Mode. The difference is subtle, but often it's such subtleties that make a topnotch graphics display possible.

International Characters

Remember the tilde that you used at the beginning of this chapter to create the approximately equal sign? Well, that tilde is only one of the many uncommon characters that are stored in the FX as components of the nine international character sets.

We mentioned the international character sets in Chapter 1, when we explained that the factory-set defaults for the DIP switches include the selection of the USA character set. Besides the pound sign (#), dollar sign ($), and at sign (@), the USA set includes the nine other symbols shown in Table 6-1; one or more of these nine may not appear on your computer's keyboard.

Table 6-1. ***Some special characters***

91	[	Left bracket
92	\	Back slash
93	]	Right bracket
94	^	Caret
95	_	Underline
96	`	Accent grave
123	{	Left brace
124	¦	Flat colon
125	}	Right brace
126	~	Tilde

But that's not all. Your FX has the makings of a world correspondent. Packed in the ROM are nine sets of letters and special characters

that are used in different countries. These international characters can be accessed with:

```
LPRINT CHR$(27)"R"CHR$(n);
```

where n is a number from zero to eight. The ESCape "R" sequence selects one of these nine countries:

0	USA	3	United Kingdom	6	Italy
1	France	4	Denmark	7	Spain
2	Germany	5	Sweden	8	Japan

Once you have selected a country, you can use its special characters.

Choosing a new international character set, however, does not give you a completely new set of 256 characters. There are 64 international characters stored in the ROM, 32 in Roman and 32 in Italic typeface, and of these you can only use 12 characters at a time. Each set of 12 is stored as the following ASCII numbers:

```
35, 36, 64, 91, 92, 93, 94, 96, 123, 124, 125, 126
```

The next program selects each of the international character sets in turn. When you RUN it, the printout (shown here as Table 6-2) displays the special symbols of each set (if you have problems, consult Appendix F).

```
NEW
10 DIM ARRAY (12): LPRINT CHR$(27)"M"
20 LPRINT CHR$(27)"D"CHR$(l0)CHR$(0)
30 LPRINT CHR$(137);
40 LPRINT "35 36 64 91 92 93 94 96
   123 124 125 126"
50 FOR X=1 TO 12: READ ARRAY(X): NEXT X
60 DATA 35,36,64,91,92,93,94,96,123,124,125,126
70 FOR Y=0 TO 8: LPRINT CHR$(27)"R"CHR$(Y);
80 READ C$: LPRINT C$CHR$(137)CHR$(14);
90 FOR X=1 TO 12: LPRINT CHR$(ARRAY(X)) "ø";
100 NEXT X: LPRINT: NEXT Y
110 DATA "USA","FRANCE","GERMANY","U.K.","DENMARK"
120 DATA "SWEDEN","ITALY","SPAIN","JAPAN"
130 LPRINT CHR$(27)"@"
```

There are a few items in the program that haven't been covered yet; we'll cover them later. The important thing now is to understand how you can use the CHR$(27)"R" to gain access to the international characters.

Table 6-2. International characters in Roman *typeface*

	35	36	64	91	92	93	94	96	123	124	125	126
USA	#	$	@	[	\	]	^	`	{	¦	}	~
FRANCE	#	$	à	°	ç	§	^	`	é	ù	è	¨
GERMANY	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
U.K.	£	$	@	[	\	]	^	`	{	¦	}	~
DENMARK	#	$	@	Æ	Ø	Å	^	`	æ	ø	à	~
SWEDEN	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	à	ü
ITALY	#	$	@	°	\	é	^	ù	à	ò	è	ì
SPAIN	₧	$	@	¡	Ñ	¿	^	`	¨	ñ	}	~
JAPAN	#	$	@	[	¥	]	^	`	{	¦	}	~

This program provides an easy reference to the international characters; you'll probably want to keep the printout handy.

You can also print international characters in Italic Mode. Change these two lines:

```
80 READ C$: LPRINT C$CHR$(137)CHR$(14)CHR$(27)"4";
100 NEXT X: LPRINT CHR$(27)"5": NEXT Y
```

to get the result shown in Table 6-3:

Table 6-3. International characters in Italic *typeface*

	35	36	64	91	92	93	94	96	123	124	125	126
USA	#	$	@	[	\	]	^	`	{	¦	}	~
FRANCE	#	$	à	°	ç	§	^	`	é	ù	è	¨
GERMANY	#	$	§	Ä	Ö	Ü	^	`	ä	ö	ü	ß
U.K.	£	$	@	[	\	]	^	`	{	¦	}	~
DENMARK	#	$	@	Æ	Ø	Å	^	`	æ	ø	à	~
SWEDEN	#	¤	É	Ä	Ö	Å	Ü	é	ä	ö	à	ü
ITALY	#	$	@	°	\	é	^	ù	à	ò	è	ì
SPAIN	₧	$	@	¡	Ñ	¿	^	`	¨	ñ	}	~
JAPAN	#	$	@	[	¥	]	^	`	{	¦	}	~

When could you use this program? Well, you can print . . .

```
. . . in italiano,
. . . en français,
. . . in English,
. . . auf deutsch,
. . . på svenska,
. . . paa dansk,
. . . en español.
```

and if you want to use one of the foreign sets all the time, you can change your printer's default.

The factory setting of a default international character set-for the USA-is shown in line 1 of Table 6-4. You can change this by resetting some of the FX's DIP switches. Three switches:

Switch 1-6
Switch 1-7
Switch 1-8

generate eight combinations. Table 6-4 shows the switch settings.

Table 6-4. International DIP switch settings

Country	Switch 1-6	Switch 1-7	Switch 1-8
USA	On	On	On
France	On	On	Off
Germany	On	Off	On
United Kingdom	On	Off	Off
Denmark	Off	On	On
Sweden	Off	On	Off
Italy	Off	Off	On
Spain	Off	Off	Off

That leaves the Japanese set unaccounted for. If you want the characters for it, you must choose the set by using the ESCape "R" statement.

Special Speeds

You can control the speed of the FX's printing in a couple of ways. You can set it to print at half its usual speed, and FX-80 users can cause it to print one character at a time, immediately upon input. You turn

either of these capabilites on and off, as a mode, with an ESCape sequence.

Half-Speed Mode

The FX can print at the fine rate of 160 characters per second (cps). But it will also print more slowly if you want it to: the Half-Speed Mode prints at 80 cps. The command sequence uses lowercase s plus zero and one as a toggle:

```
LPRINT CHR$(27)"s1"
```

turns Half Speed Mode on, and, as usual, the zero version of the command turns the mode off. If your system can't send lowercase letters, use the longer format:

```
LPRINT  CHR$(27)CHR$(115)CHR$(1)
```

to get the same results.

The FX uses Half-Speed, as we explained in Chapter 4, to enable extra-dense printing. But why would you want to make the printer work at half its normal speed? The main advantage to Half-Speed printing is a quieter run (e.g., for those late night printing sessions).

Immediate-Print Mode (FX-80 only)

The FX-80 can move even more slowly-at the speed of your typing. In the Immediate-Print Mode, the print head prints one character at a time, as you send it. The FX-80 also moves the paper up so that you can see the current line and then down to continue printing. This kind of instant feedback can be especially helpful in telecommunications.

You turn Immediate Print on with CHR$(27)"i1" or, to avoid the lowercase i, CHR$(27)CHR$(105)"1". But before looking at it, let's review the normal operation of the printer buffer. Enter this program:

```
NEW
20 A$="": INPUT "TYPE A LETTER ø",A$
30 IF A$="" THEN 50
40 LPRINT A$;: GOT0 20
50 LPRINT
```

Now type several characters, and after each press the RETURN key. True to form, the printer just stuffs the characters into its buffer while it waits for a carriage-return code. To end this version of the program

and print the contents of the buffer, press RETURN alone. Now add this line:

```
10 LPRINT CHR$(27)"i1"
```

And RUN the program. Your FX-80 responds to your typing-immediately.

When you are finished, press RETURN alone, then use the zero version of the command to return to full speed.

Summary

You can use the Backspace Mode to overstrike one or more characters. In this manner, you can combine two or more characters to form completely new ones. Another way you can use backspacing is to sandwich it between two printings, each in a different print mode, to create characters with offsets.

You can select any one of the nine international character sets by using the control code sequence given below; or you can designate any one of eight of the sets as the default set by changing a DIP switch.

You can cause any FX to print in Half Speed, and you can cause the FX-80 to react as though it were a typewriter by using the Immediate Print Mode. Here are the DIP switches we mentioned in this chapter:

Switches 1-6
1-7
1-8

Any change to these switches causes a change of the default international character set.

Here are this chapter's commands:

CHR$(8) Causes a backspace

CHR$(27)"R"CHR$(n) Selects an international character set, where n = 0-8:

0 = USA	5 = Sweden
1 = France	6 = Italy
2 = Germany	7 = Spain
3 = United Kindgom	8 = Japan
4 = Denmark	

CHR$(27)“s1”	Turns Half-Speed ON; If your system can’t send lowercase letters, use CHR$(115) **CHR$(1)**
CHR$(27)“s0”	Turns Half-Speed OFF
CHR$(27)“i1”	For the FX-80 only, turns Immediate-Print Mode ON. If your system can’t send lowercase letters, use CHR$(105)CHR$(1).
CHR$(27)“i0”	Turns Immediate-Print Mode OFF

Chapter 7
Line Spacing and Line Feeds

Up to this point in the manual, we have not discussed the way the printer moves a page so that it doesn't print lines of text right on top of each other. Now we do. In this chapter you will learn how to change the distance that the paper moves; the movement is called a line feed, and the distance is called a line space. The ability to change line spacing is vital to printing graphics, as you will see in later chapters.

Line Spacing

Each line feed must move the paper a specific distance, but that distance need not remain the same for every application, or even every line.

The FX gives you three types of commands to change the size of a line space. You can use one type to select the most common sizes. You can use a second type of command to vary line spacing in 72nds of an inch. And you can use the third to adjust lines microscopically, in 216ths of an inch.

Preset line spacing

The default setting for one line space is 1/6-inch, which produces 6 print lines per inch. Since this size of line feed is equivalent to 12 rows of dots in the standard character matrix (Figure 7-1), we call this 12-dot line spacing.

When you have been using another size of line spacing and want to return to 12-dot, you use this code: ESCape "2".

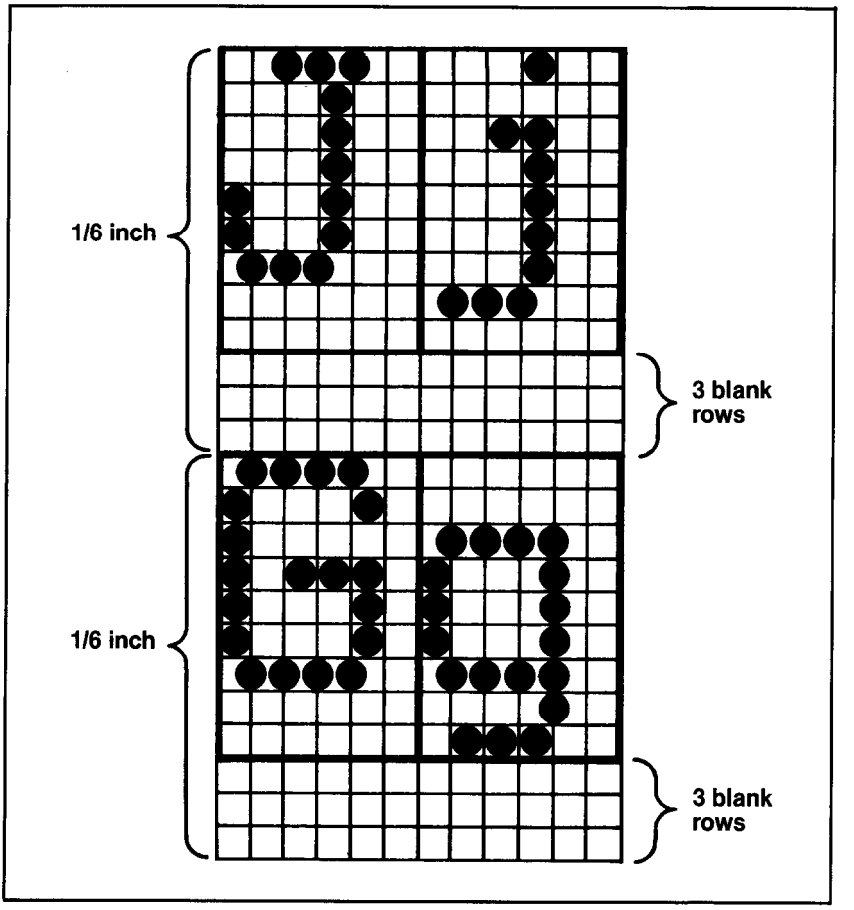


Figure 7-1. *Default line spacing*

To see 12-dot spacing, reset the printer (to clear any previous modes), and enter:

```
NEW
20 FOR X=0 to 4
30 LPRINT TAB(6*X)"STAIR STEPS"
40, NEXT X
```

```
STAIR STEPS
      STAIR STEPS
            STAIR STEPS
                  STAIR STEPS
                        STAIR STEPS
```

Your first STEPS print in 12-dot spacing. Now tighten up the line spacing by adding lines 10 and 50:

```
10 LPRINT CHR$(27)"0"
50 LPRINT CHR$(27)"2"
```

The CHR$(27)"0" of line 10 changes the usual 12-dot (1/6-inch) line spacing to a handy variation: 9-dot (l/B-inch) spacing. Nine-dot spacing is especially useful in the 9-pin Graphics Mode that we introduce in Chapter 11.

Another convenient line spacing is 7-dot (7/72-inch). To see this one, change line 10 to:

```
10 LPRINT CHR$(27)"1"
```

Although 7/-dot spacing is not suitable for text, it does allow you to print graphics designs with no gaps. You'll use it extensively in the graphics programs presented later.

Variable line spacing

The FX includes the three line spacing commands shown above mainly for convenience. Besides printing in the preset 7-, 9-, and 12-dot spacing, the FX allows you to vary line feeds from 0 to 85 dots worth (0/7-inch to 85/72-inch). ESCape"A"CHR$(n), where n represents n/72-inch, changes the distance that a line feed covers to n dots.

To show what varying n can mean, the following program increases the line spacing by one dot's worth on each line feed:

```
20 FOR X=0 TO 24
30 LPRINT  TAB(X)"STAIR"CHR$(27)"A"CHR$(X+128)
   "STEPS"
40 NEXT X
50 LPRINT  CHR$(27)"2"
```

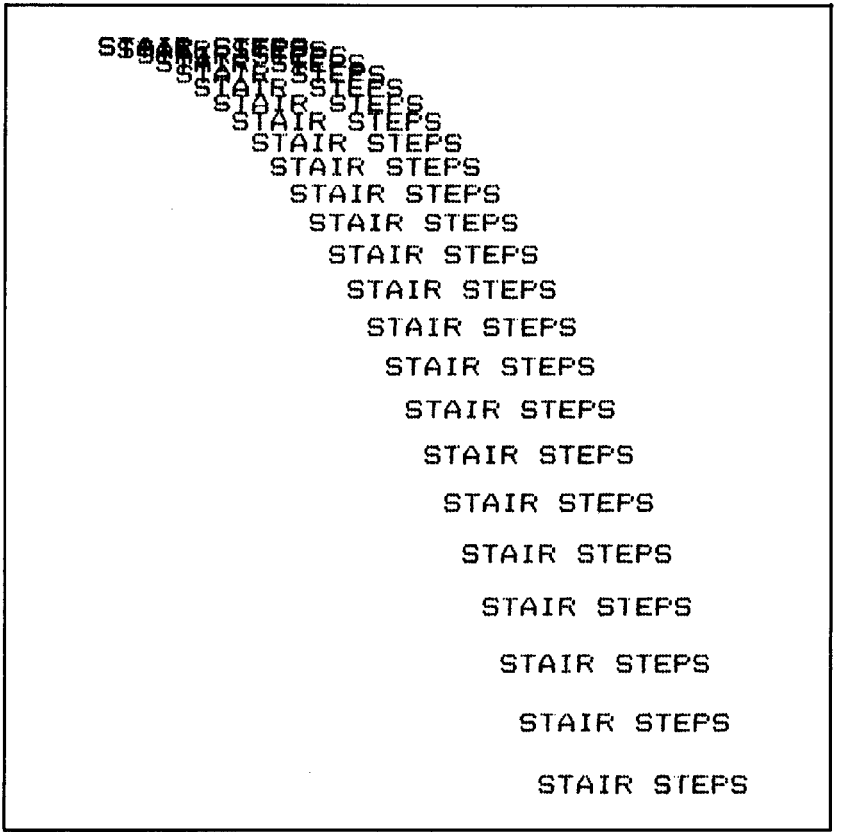


Figure** 7-2. **Cascading STAIR STEPS

Figure 7-2 shows that the loop in line 20 and the ESCape "A" command in line 30 gradually increase the line spacing. Because many computer systems have difficulty with one or more numbers below **13** in character-string commands, we have used X+128 in the line-spacing command to avoid those problems.

The ESCape"A"CHR$(n) command sets the line spacing to n/72-inch if the n is any number from 0 through 85. If n is between 85 and 128, the line spacing is 85/72-inch. At 128 the sequence starts again, with 128 giving the same result as 0, 129 the same as 1, and so on. Therefore, the X+128 in line 30 produces a change in line spacing from 0 to 24/72-inch.

In summary, ESCape "A"CHR$(n) selects line spacing in 72nds of a inch, where:

n =		line spacing =	
	0 - 85		0 - 85/72-inch
	85 - 127		85/72-inch
	128 - 213		0 - 85/72-inch
	214 - 255		85 /72-inch

Since the ESCape "A" sequence lets you specify any 72nd of an inch (from 0 - 85) and since each dot of a dot matrix fills 1/72-inch, you can use this command instead of the preset commands that we covered above. You can set 12-dot line spacing either by specifying 12 to the "A" sequence or by using the preset command:

CHR$(27)"A"CHR$(12) or CHR$(27)"2"

Notice the position of the ESCape "A" sequence in line 30-right between the strings STAIR and STEPS. We placed it there to demonstrate that the FX doesn't execute the line-feed command until the end of the print line. Otherwise it would print the two strings on different lines.

Also notice that the lowest valid line spacing is zero dots. You can use this when you want to make multiple overstrikes on one line. To see an example, make these changes to your program:

```
20 FOR X=0 TO 4
30 LPRINT  TAB(6*X)"STAIR"CHR$(27)"A"CHR$(0)
   "Ø STEPS"
```

STAIRS STAIRS STAIRS STAIRS STAIRS STEPS

The reason for emphasizing 72nds of an inch is that the distance between the centers of any two pins on the print head is 1/72-inch. So these line spacing commands are simply telling the printer just how many dots to space between the lines.

Microscopic line spacing

There is also a way to space at smaller intervals than 72nds. Using a CHR$(27)"3" will set the spacing to increments of 216th of an inch; l/216-inch is one-third the distance between the pins of the print head (center to center). That means the printer can position a specific line one-third of a dot lower than the previous line. In fact, that's exactly how the Double-Strike Mode operates. One word of caution. As you can imagine, total accuracy is not guaranteed for such fine settings as 1/216- and 2/216-inch.

You specify this finer line spacing in much the same way as you did the variable line spacing that we showed above with CHR$(27)"A". The format is CHR$(27)"3"CHR$(n), where n can range from zero to 255. Here's an example using the l/216-inch line spacing five times. Change lines 10 and 30 to read:

```
10 LPRINT CHR$(27)"3"CHR$(1)
30 LPRINT "ABCDEF"
```

ABCDEF

These letters are very heavy looking. To carry this idea to an extreme, increase the upper limit of the loop to 10 or 15.

The ability to adjust line spacing in increments of 1/216-inch gives you tremendous control in your vertical formatting. You can use this control to fine-tune your graphics printouts. If the 7-dot line spacing leaves gaps in the figures, just tighten it up by changing it to 6-2/3-dot (20/216-inch):

```
CHR$(27)"3"CHR$(20)
```

We'll use this technique in a later chapter.

Line Feeds

Besides being able to change the size of line spaces, you can change other aspects of a line feed. You can send it immediately and for one line only, or you can send it as above, as a continuous feature. And if you are using an FX-80, you can send a reverse line feed to make the print head move back up the page.

One-time, immediate line feed

The FX has a special line feed that executes a new size of line feed once, then reverts back to the size of the previous line feed. And that's not all-it is executed immediately rather than at the end of the print line as all the other line spacing commands are. The format is:

```
LPRINT CHR$(27)"J"CHR$(n)
```

where n represents a distance of from zero to 255/216-inch.

Put this one to the same test you gave the ESCape "A" command with the following program:

```
20 FOR X=10 TO 30
30 LPRINT TAB(X)"STAIR"CHR$(27)"J"CHR$(X)'STEPS"
40 NEXT X
50 LPRINT CHR$(27)"2"
```

Compare Figure 7-3 to Figure 7-2. In the ESCape "A" program of Figure 7-2, the FX prints STAIR and STEPS on the same line, and performs the variable line feed at the end of the line. In the ESCape "J" program of Figure 7-3, the FX first performs a line feed without a carriage return ***between*** the strings (see how STEPS begins to sag below STAIR?), and then a standard 12-dot line feed plus carriage return at the end of each line.

To summarize, the ESCape "J" does not require a shut-off code as the other line-feed control codes do: the FX executes it once, then forgets it. Another difference between this line-spacing command and the others is that this one does its work without performing a carriage return: the print head does not move to the left margin.

Reverse feed (FX-80 only)

The FX-80 has another one-time, immediate line feed, the reverse feed. It operates the same way that ESCape "J" does, as its code suggests: ESCape "j". The only difference is that reverse feed moves the print head back up the paper (it actually moves the paper, but the effect is of the print head moving). The following program lets you watch ESCape "j" in action. When you RUN it, the FX-80 will first

Figure 7-3. Staggering STAIR STEPS

print the two lines of text and then move the print head up the page to print the line of hyphens above the first line.

```
10 LPRINT "REVERSE FEED"
20 LPRINT
30 LPRINT "ARE YOU WATCHING?"
40 LPRINT CHR$(27)"j"CHR$(140);
50 LPRINT "__________"
```

```
----------
REVERSE FEED

ARE YOU WATCHING?
```

If your system cannot send lowercase letters to the FX, use the numeric equivalent of " j " -CHR$(106).

Don't use reverse feed with mailing labels in the printer-they can either move on their gummed paper or peel off and get stuck inside the FX.

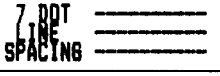
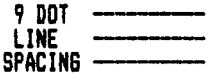
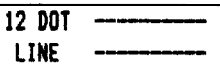
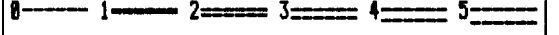
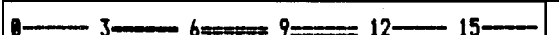
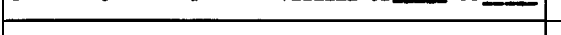
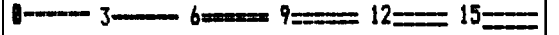
Summary

The FX provides line spacing in increments of 0/72- to 85/72-inch and 00/216- to 255/216-inch. You can cause the FX to change the size of a line space to one of three preset sizes, or you can specify a size in fractions of an inch. You'll use variable line spacing primarily for graphics.

The FX also gives you commands to produce an immediate line feed, either forward or backward.

Table 7-1 summarizes the line-spacing commands and gives a sample of each.

Table 7-1. Line-spacing commands

Example	Line spacing	Command
	7/72"	CHR$(27)"1"
	9/72' (1/8")	CHR$(27)"0"
	12/72" (l/6")	CHR$(27)"2" (default)
	n/72"	CHR$(27)"A"CHR$(n) n = 0 - 255
	n/216"	CHR$(27)'3"CHR$(n) n = 0 - 255
	n/216"	CHR$(27)"J"CHR$h) n = 0 - 255 (immediate)
	n/216"	CHR$(27)"j'CHR$(n) n = 0 - 255 FX-80 only (immediate)

Chapter 8
Forms Control

The FX has several features that make it easy for you to print on any size of page and to determine where on the page the printing will appear. Because they are needed most often for creating forms or for printing on pre-printed forms, these features are called forms control. With the FX you can easily change the length of a page, the margin settings, and the horizontal and vertical tabs. In this chapter and the next two, you will learn about these forms-control features.

This chapter covers the way you control form feeding, which is the movement of the paper from one paging unit to the next. You can control the distance the paper moves, the positioning of the print head at the top of each page, and the printer's response to your use of single sheets of paper.

Form Length Control

The FX's default form feed is 66 lines, which (in the default 12-dot line spacing) equals the length of a standard form: `11` inches. When you want to use a different size of paper, you can change the length of a form feed. The next three subsections cover these factors of form feeding.

Form feed distance

When the printer is not on line, using the FF button on the top of the printer feeds the paper to the top of the next form. If you want to advance the paper during a program run without stopping everything, pushing the FF button, then starting again, you can use an ASCII code that performs a form feed whenever you want one.

For the form feed command to be of any use, however, you must

first tell the printer where the top of form is. In most cases you'll want the printer to use the first line below the paper perforation as the top-of-form line. To get this result, turn the printer off and feed the paper through (using the manual-feed knob) until a perforation lines up with the top of the ribbon (see Figure 8-1 or consult Chapter 1).

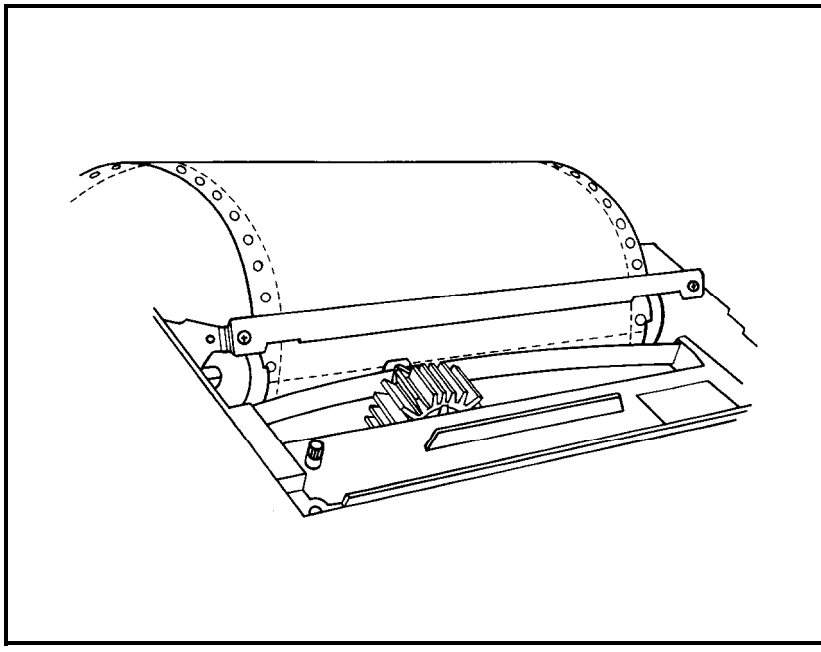


Figure 8-1. Setting the top of form

Turn on the printer. The FX will now remember this position on the paper as the top of form. Each form feed will move the paper to the corresponding position on the next sheet. Try pressing the FF button (with the printer off line). Sure enough, the FX moves the paper right to the top of form.

Now press the LF button a few times, then turn the printer back on line and type:

```
LPRINT CHR$(12);
```

ASCII 12 is the low-order form-feed code; some users will be better off using the high-order 140.

CHR$(12) sends the paper to the top of the next form. It gives the same result as the FF button so long as you end the line with a semi-colon to prevent BASIC from adding a line feed to the LPRINT line.

Not-so-standard forms

The printer's default length for a form feed is 11 inches. But what if you decide to use a different form length, say 2 or 14 inches? The printer has no way of measuring the length of your paper. You must tell the FX about your shorter (or longer) form.

The CHR$(27) "C" command gives you two ways to change the form length: by inches or by lines.

CHR$(27)"C"CHRS(0)CHR$(n)	Sets the form length to n ***inches*** (1-22)
CHR$(27)"C"CHRS(n)	Sets the form length to n ***lines*** (1-127)

CHR$(0) makes the difference between the two commands.

To see the first format in action, run the following program. Enter:

```
NEW
10 LPRINT CHR$(27)"C"CHR$(2)CHR$(2);
20 FOR X=1 TO 4
30 LPRINT "TWO-INCH FORM"CHR$(12);
40 NEXT X
50 LPRINT CHR$(27)"@"
```

CHR$(27)"C" is the key. It changes the form length in the printer's memory so that the string TWO-INCH FORM is printed at the top of each new form. It also resets the top of form to the current position of the print head on the paper. It works just as if you'd turned the printer off and on, but without the resetting of other defaults that happens when you set the top of form by using CHR$(27)"@" or by cycling power.

In the program above, line 10 uses the inches format to set the form length to two inches. The CHR$(12) in line 30 activates the form feed to move the paper to the top of the next two-inch form.

The second format-the one without CHR$(0)--sets the form length by counting, in the current line spacing, the number of lines.

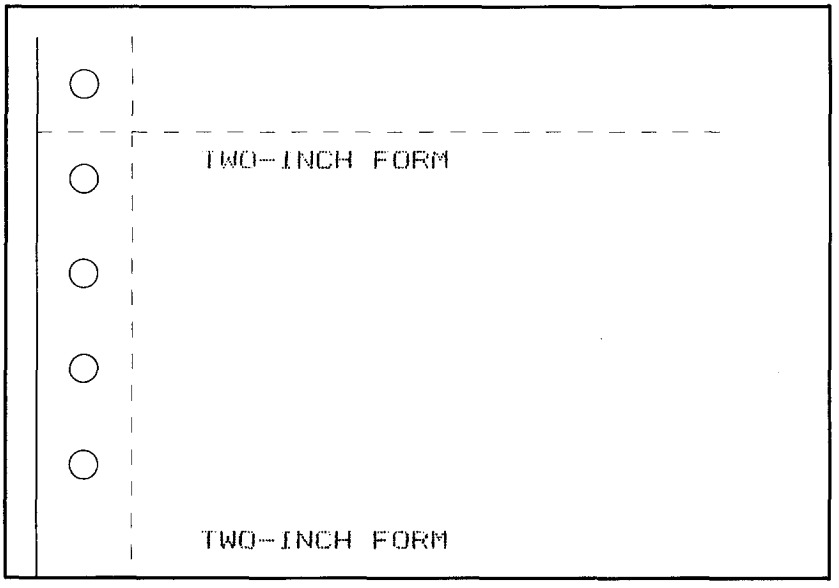


Figure 8-2. Two-inch form feed

Check it by changing your program lines as shown below and RUNning the program again; see if your printout matches Figure 8-3.

```
10 LPRINT CHR$(27)"C"CHR$(2);
30 LPRINT "TWO-LINE FORM"CHR$(12);
```

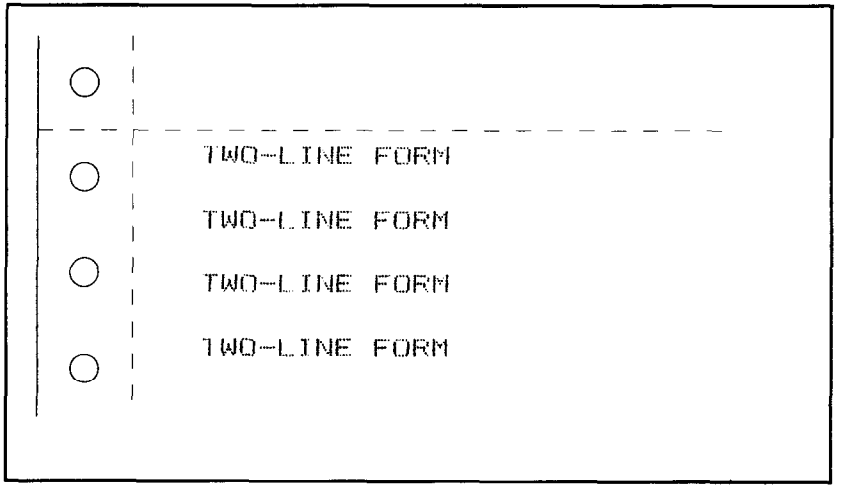


Figure 8-3. Two-line form feed

Why does the printer give you two options? In some cases, setting the form length by inches is more convenient. If you know how many inches long the form should be, the printer will calculate the correct setting for you, regardless of the current line spacing. On the other hand, setting the form length by number of lines is the only way you can set extremely long form lengths.

If you set the line spacing to its maximum of 85/72-inch and you also set the form length to its maximum of 127 lines, you get a form that is nearly 150 inches long. Compare this to the maximum of 22 inches that you get when you set form length in inches. Except for this difference, your choice of form setting command is a matter of personal preference.

Either format of CHR$(27)"C" sets the form length in the line spacing that is in effect when the command is given. Subsequent changes in line spacing will not affect it. This can be important in the graphics programs later in this manual.

Paper Perforation Skip

You can avoid printing on the paper perforation by setting the top of form properly and making sure the printer is informed of the correct form length. Then your appropriately placed form feeds will keep the printing off the perforation.

But there are some situations in which you will find that sending form feeds on every page is inconvenient, and others in which you can't control the process with a BASIC program. A good example of the latter is printing a listing of a long BASIC program. If your computer system's LLIST command doesn't automatically skip perforations, most printers can't compensate. Your FX, however, can; it can automatically output a form feed at the bottom of ***every*** page to clean up those listings. You can think of it as an automatic skip-over-perforation command.

Skip command

ESCape "N", the skip-over-perforation command, can automatically skip lines at the bottom of each page to avoid printing right on the perforation, but you must first inform the printer of two things: 1) where the correct top of form is and 2) how long the current forms are.

For standard 11-inch forms, just position the paper correctly before turning on the printer; for other form lengths, use the CHR$(27)“C” command.

It’s time to try this out. Make sure the perforation is even with the top of the ribbon (as in Figure **8-1),** reset the printer, then type:

```
NEW
10 LPRINT CHR$(27)"N"CHR$(6);
20 FOR X=1 TO 70
30 LPRINT "SIX-LINE SKIP . . . LINE";X
40 NEXT X
```

Figure 8-4 shows the skip.

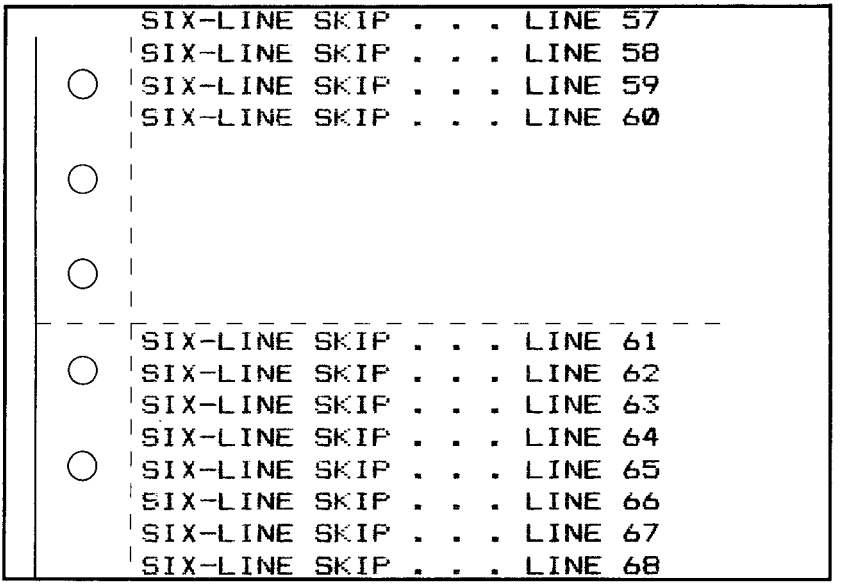


Figure 8-4. Standard skip

The CHR$(27)“N”CHR!$(6) in line 10 tells the printer to skip 6 lines (1 inch), which moves the paper across the perforation to the next top of form. You can set the number of lines to any value from 1 to 127, as long as the value is less than that for the current form length (set by default or CHR$(27)“C”). If the value is greater than or equal to the form length, CHR$(27)“N” is ignored.

When you use the skip-over-perforation command, you may want to change your top of form. No matter what number you use as a skip-over-perforation setting, the printer skips that many blank lines from the last text line to the new top of form. In other words, when you set your top of form the usual way, you will have all of your blank space at the bottom of each page.

To get equal amounts of blank space on the top and bottom of each page, you can set the top-of-form position below the perforation. For the standard six-line (1-inch) gap, setting the top of form to half an inch below the perforation will provide equal amounts of blank space at the bottom of the old page and the top of the new page. You can turn the skip-over-perforation feature off with ESCape "O" (that's the letter oh). You also cancel it by resetting the printer or by setting a new length with ESCape "C".

DIP switch skip

You may find that you want the skip-over-perforation feature to become a default feature on your printer. Nothing could be easier.

Turn the printer off (so that your new setting will take effect when you turn the FX on again). Now remove the vent covering the DIP switches as described in Chapter 1 and change DIP switch 2-3 to the on position.

Set a new top of form as described above. Now replace the vent and turn the printer on. Then delete line 10 and put in this new line 30:

```
30 LPRINT "AUTO SKIP COMING UP . . . LINE";X
```

Except for the positioning of the blank lines and the wording of the text, this printout should match Figure 8-4.

An automatic six-line skip-over-perforation will now be active every time you turn the printer on. And you can still turn it off with the CHR$(27)"O" or change the setting with CHR$(27)"N"CHR$(n).

If you don't want to keep the skip-over-perforation feature active, reset switch 2-3 and your top of form before proceeding.

Single-Sheet Adjustment

For instructions on loading single sheets of paper into the friction feeder, refer back to Chapter 1.

If you use single-sheet paper on your FX printer and run to the end of the form, the paper-out sensor prevents the printer from accidentally printing on the platen. The sensor automatically sounds the beeper and shuts down the printing until you load another sheet and continue. While the sensor saves wear of print head, ribbon, and platen, it also prevents you from printing on the last quarter of a page. If you need to print on this part of the page, you can deactivate the paper-out sensor by setting DIP switch 1-3 on.

Alternatively, you can disable the sensor temporarily by using a software code: ESCape "8". You can see how this works by running the next example. Because you need to see it in action, we do not provide a figure for either run of this program.

With the paper-out sensor active (switch 1-3 off) and the beeper alive (switch 2-2 on), load a single sheet of paper and RUN this program to see the sensor work:

```
10 LPRINT
20 INPUT "HIT RETURN WHEN READY",A
30 FOR X=1 TO 60
40 LPRINT "PRINTING WILL STOP BEFORE PAPER RUNS OUT"
50 NEXT X
```

The printer quits about three-fourths of the way down the page. Good, the sensor is doing its job. Now let's override it. Load in a new sheet of paper and change line 10 to read:

```
10 LPRINT CHR$(27)"8"
```

When you RUN this new program with most computer systems, the printer prints all 60 lines, ignoring the paper-out condition. Some systems, however, ignore both the ESCape "8" code and the setting of DIP switch 1-3. (See Appendix F.)

You'll want to use this code with caution, but it's nice to have when needed. To restore the sensor to full power, either send an ESCape "9" command or reset the printer.

Summary

The printer automatically considers each page to be 11 inches or 66 lines of 12-dot line spacing long. You can change the length of each form (in inches or lines) with an ESCape code. You can turn the skip-over-perforation feature, the paper-out sensor, and the alarm on or

off, either by changing DIP switches or by sending the printer ASCII codes as summarized below.

Switch 2-2	When on; activates the beeper; when off, deactivates it
Switch 1-3	When off, makes the paper-out sensor active; when on, deactivates it
Switch 2-3	When off, turns the skip-over-perforation feature OFF; when on, produces an automatic 1-inch skip over every perforation

Check to see if you want to reset any switches before going on to Chapter 9.

Here are the commands we covered in this chapter:

CHR$(12)	Produces a form feed; CHR$(140) is the alternate, high-order version
CHR$(27)"C"CHR(0)CHR$(n)	Sets the form length to n inches, where n is 1 - 22
CHR$(27)"C"CHR$(n)	Sets the form length to n lines, where n is 1 - 127
CHR$(27)"N"CHR$(n)	Turns the skip-over-perforation feature ON, set to n lines, where n is 1 to less than the current form length in lines. Is cancelled by CHR$(27)"@" and CHR$(27)"C" as well as CHR$(27)"O"
CHR$(27)"O"	Turns the automatic skip-over-perforation feature OFF
CHR$(27)"8"	Turns the paper-out sensor OFF
CHR$(27)"9"	Used in conjunction with CHR$(27)"8", turns the paper-out sensor back ON
CHR$(27)"@"	The Reset Code resets the form length to 11 inches, the top of form to the current line, and the skip-over-perforation to the DIP switch setting

Chapter 9
Margins and Tabs

At power-up, your FX contains specific default settings for margins and for horizontal and vertical tabs. You can make changes to any of these. Since it is best to change margins before tabs, we discuss margins first, then three aspects each of horizontal and vertical tabs.

Margins

Most word processing programs have commands that let you set the left and right margins. If yours doesn't, or if you ever need to change margins from a BASIC program, the FX has just the thing-margin control. For Pica pitch, the default margins are 80 characters for the FX-80 and 136 for the FX-100. You can change each margin separately to suit your needs.

Left margin

If your word processor cannot control the left margin, this command is the one you've been waiting for. The command for setting the FX's left margin uses the lowercase letter el:

```
CHR$(27)"1"CHR$(n)
```

where n is the column number for your new left margin. Use CHR$(l08) in place of the "1" if your system has trouble with lowercase characters.

Here are the ground rules. The allowed values for n on the FX-80 range from 0 to 78 for Pica, 0 to 93 for Elite, and 0 to 133 for Compressed Mode. On the FX-100 the ranges are 0 to 134 for Pica, 0 to 160 for Elite, and 0 to 229 for Compressed. The printer ignores all invalid settings, such as those greater than the current page width. New margin settings go into effect at the next carriage return, and they stay in effect until you change them.

Try out the left margin command with:

```
NEW
10 LPRINT "LEFT MARGIN"
253 LPRINT CHR$(27)"1"CHR$(10)
30 LPRINT "LEFT MARGIN SET AT 10
40 LPRINT
```

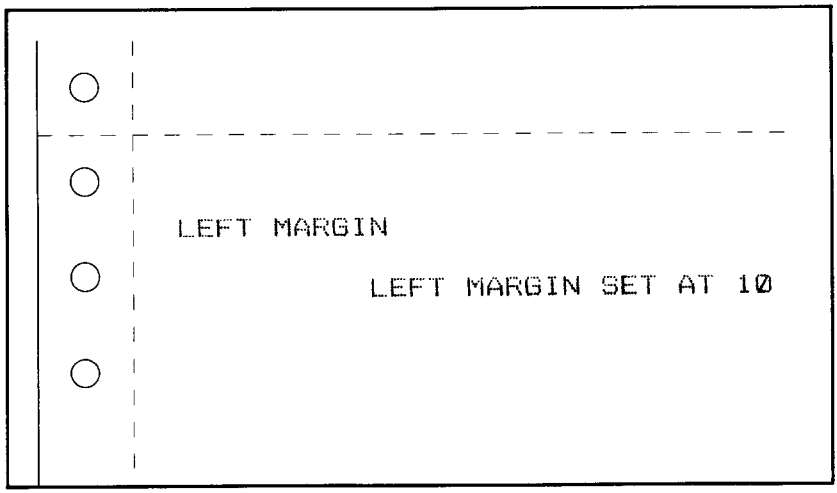


Figure 9-1. Left margin setting

As Figure 9-1 shows, line 10 prints at the default (zero) left margin, and line 30 makes the new left margin start 10 spaces to the right of the default.

You may have noticed that we did not have you reset the left margin to zero at the end of the program. To see if the new margin is still in effect, type your computer's LLIST command. Does your printout page now look like Figure 9-2?

The left margin is still set at 10, and it will stay at this setting either until it is reset by some software code or until the printer is turned off.

Margins and pitches

Margin settings are not affected by changing the width of the print ***after*** they are set. This means that if you set the margin in Pica Mode

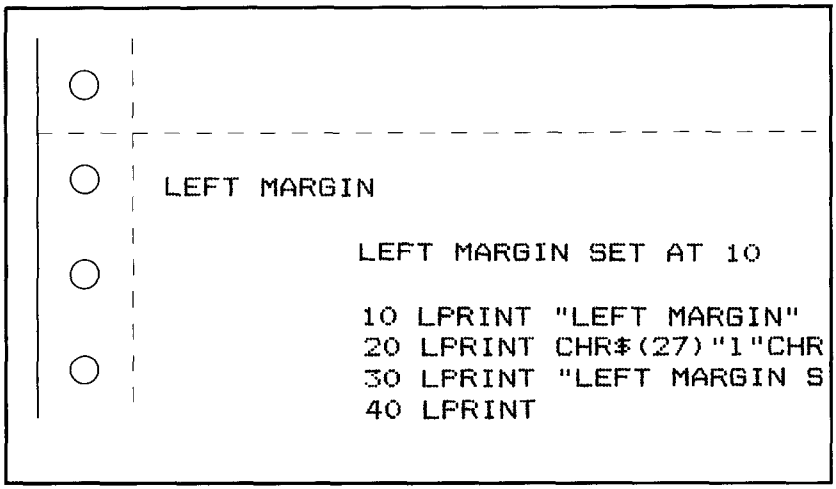


Figure 9-2. Listing at new margin

and then switch to Compressed, the left margin stays the same distance from the edge of the paper. To see an example, type:

```
LPRINT CHR$(15)
```

and then your computer's print listing command. Figure 9-3 shows the page with this addition.

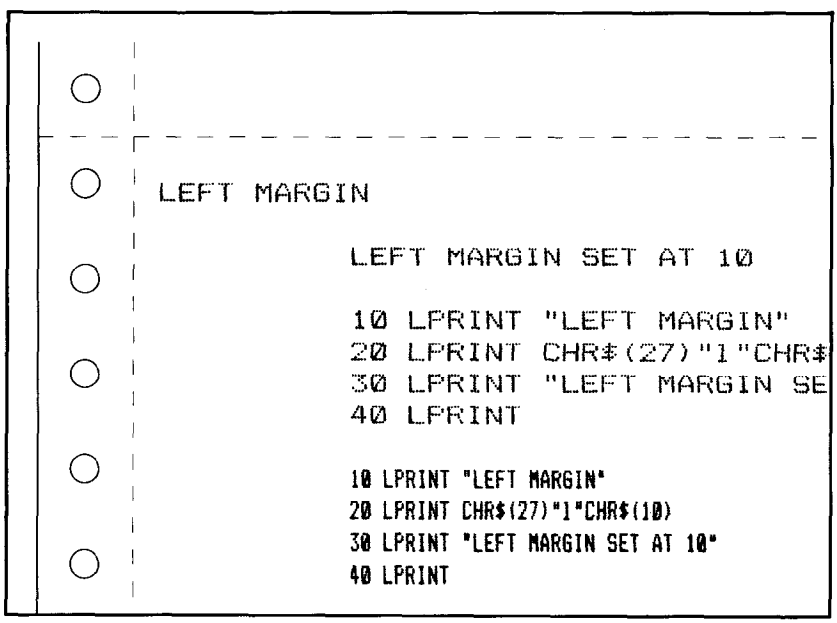


Figure 9-3. Absolute left margin

The text prints in ***Compressed*** Mode, but the left margin is still set at ***10 Pica*** spaces.

Right margin

The general format for the right margin is:

```
CHR$(27)"Q"CHR$(n)
```

For the FX-80, n can range from 2 to 80 in Pica, 3 to 96 in Elite, and 4 to 137 in Compressed Mode. For the FX-100, n can range from 2 to 136 in Pica, 3 to 163 in Elite, and 4 to 233 in Compressed. The lower limits may seem strange when compared with the Os allowed for left margin limits. We'll investigate the limits on the right margin shortly, but first let's try out the command.

The margins can be set either from a BASIC program or from the BASIC command level (without line numbers). The following line sets the right margin to 5 in Pica. Type:

```
LPRINT  CHR$(18)CHR$(27)'Q'CHR$(5)
```

and

```
58 LPRINT "RIGHT MARGIN WHERE?"
```

Now RUN your program.

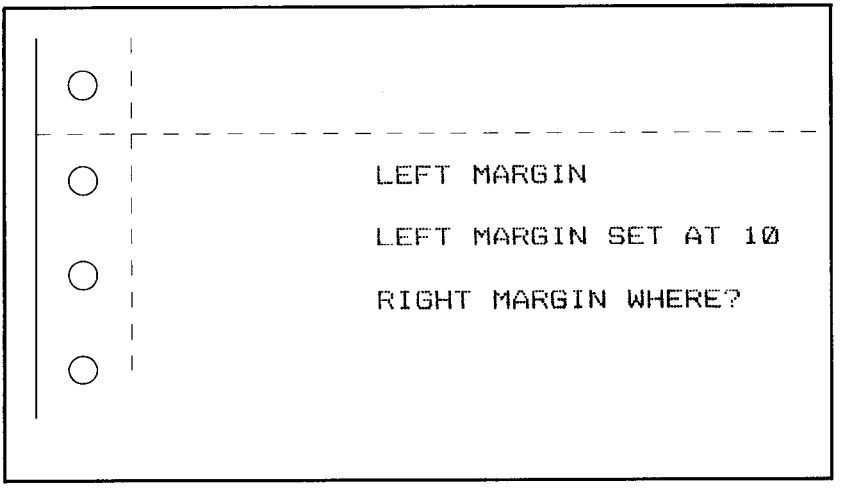


Figure 9-4. Right margin set incorrectly

Figure 9-4 shows the new listing, which did not print out at the position you specified. What happened? Well, the CHR$(18) turned off Compressed Mode, but there was no change in the margin because the new right margin setting would have occurred on the wrong side of the current left margin (which is still set at 10). Remember, the FX simply ignores impossible settings.

Use a workable number to reset the right margin:

```
LPRINT CHR$(27)"Q"CHR$(22)
```

Then RUN the program.

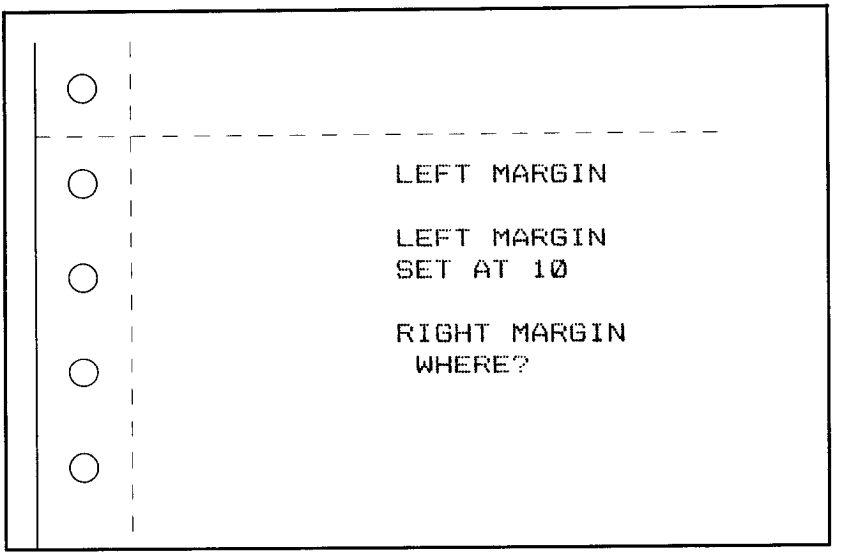


Figure 9-5. Right margin set correctly

As shown in Figure 9-5, all the characters get printed between columns 10 and 21 (counting from 0).

You can use the right margin setting to increase your printing width in Compressed Mode on the FX-80. The default right margin in that mode is 132, which is slightly less than 8 inches. If you set the right margin to 137 after giving the command for Compressed, you get a full S-inch print line. You may also need a WIDTH statement; see Appendix F.

Both margins

Notice that the left and right margin commands use different numbering systems. In Pica Mode the left margin command counts from 0 to 79 while the right margin command counts from 1 to 80. Keep this difference in mind when you use the two commands together.

Another difference between the two margin commands is that the minimum left margin setting is 0, regardless of pitch, but the minimum right margin is the value of the left margin setting plus 2 in Pica, 3 in Elite, or 4 in Compressed.

The left and right margin commands can make a mess of your program if you aren't aware of three more factors. First, since these commands clear text out of the buffer (as CANcel-CHR$(24)-does), you should not issue margin commands at the end of a program line that produces a print line. A good rule is: always send your new margin command before you send the print line.

Second, the left margin command has a profound effect on horizontal tabs. It moves the tab columns horizontally, based on the new left margin as the zero column. In other words, use this order:

- set margins first
- set tabs as needed
- then send the print line.

The third factor that can affect your new margins and tabs is resetting. Since the FX returns to its default settings whenever you reset the printer, you must be careful about using the Reset Code-CHR$(27)"@"--in a program and about turning the printer off during a series of runs.

Tabs

Your printer contains default horizontal tabs set at every eight spaces and default vertical tabs set at every other line. You can issue one command to use these default tabs, or you can issue other commands to change them. You may change tabs in either a regulated (evenly spaced) pattern or so that the tabs vary. You may also set vertical tabs in sets, called channels.

In this manual, we often use the terms column and row to refer to the positioning of ***dots*** within a matrix or, in graphics, on a page. In this chapter we will use the terms column and row instead to refer to the positioning of ***characters*** on a page.

Horizontal tab usage

The FX has the ability to tab horizontally, and it has default tabs set in the current pitch at columns 8, 16, 24, 32, . . . every eight Pica spaces on out to the current width of the page. We will show you how to change the tabs to suit your needs more closely, but first let's see how the printer's tabs work.

You can move the print head from any position on the print line to the next tab stop with the ASCII horizontal tab code, CHR$(9). You use CHR$(9)--or CHR$(137) if 9 is a number your system does not send-to move from stop to stop, whether the stop is a default tab or a tab that you have set.

Using the exact line numbers shown, enter this sample program:

```
10 H$=CHR$(137): A$="0123456789"
30 FOR X=1 TO 3: LPRINT A$;: NEXT X: LPRINT
40 FOR J=l TO 3
50 LPRINT H$;"TOP";
60 NEXT J: LPRINT
120 LPRINT CHR$(27)"@"
```

and compare your RUN with Figure 9-6.

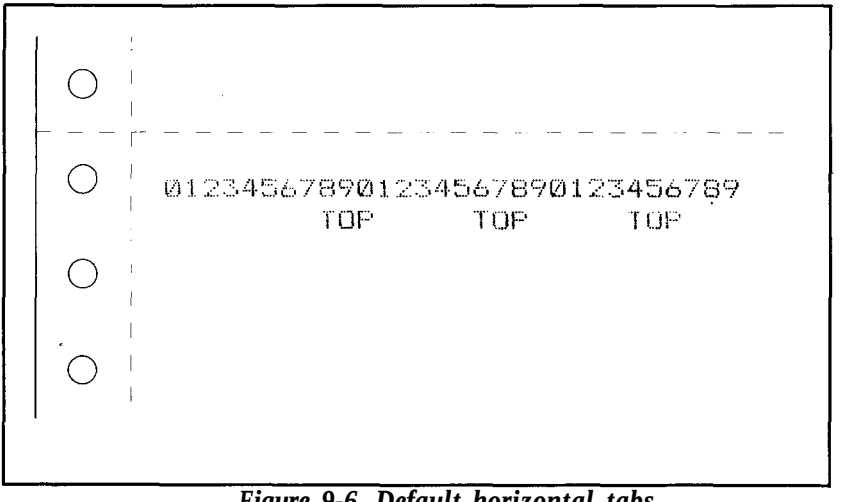


Figure 9-6. Default horizontal tabs

This shows that the default tabs (each represented by the T of the word TOP) are set in columns 8, 16, 24, 32, etc. Remember that the column count starts at 0.

Note that many BASICs handle numbers differently from strings. This difference is most evident when you are printing columns that contain mixtures of numbers and strings: many BASICs automatically add spaces both before and after each number. You may have to make adjustments if you want to have a column of numbers line up. Test this out on your machine with the following changes:

```
70 FOR J=1 TO 9
80 LPRINT H$;J;
90 NEXT J: LPRINT
```

Figure **9-7** shows the text heading (TOP) centered above each column of numbers. Since each column is three spaces wide and each number consists of a single numeral, the automatic space placed before each number does the centering. Remember the way your system handles numbers so that you can make adjustments as necessary.

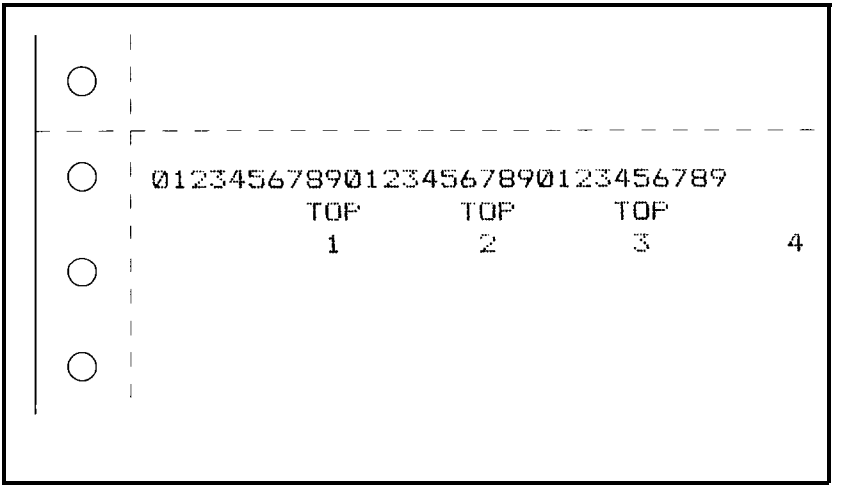


Figure 9-7. Tabs with text and numbers

When you want to use a tab stop that is not your first stop, you simply enter an extra character string tab command for each stop that you want to skip over. In your current program, for instance, you would put your first column at the second stop by making line 50 read:

```
50 LPRINT H$;H$;"TOP";
```

Since we use this technique within a program later, we do not include a printout here.

Variable horizontal tabs

You can change the default horizontal tab settings by specifying new tab stops. To do this, use the format:

```
CHR$(27)"D"CHR$(n1) . . . CHR$(nk)CHR$(0)
```

where n_1 and n_k stand for the first and last of a series of new tab stops, and the CHR$(0) informs the printer that you are through setting tabs. The FX can store up to 32 tab stops; you may specify one or all of these. You may also add stops to one or more of the default tabs, as in the next version of your current program.

Delete lines 40 through 90 and add:

```
40 LPRINT CHR$(27)"D"CHR$(6)CHR$(12)CHR$(20)CHR$(0);
50 LPRINT H$;"ONE";H$;"TWO";H$;"END"
```

RUN it to see if you get the results of Figure 9-8.

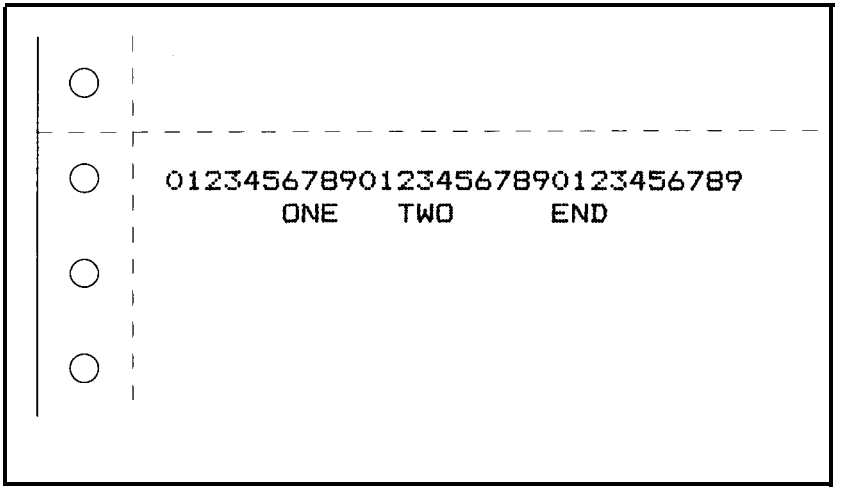


Figure 9-8. Variable horizontal tabs

In line 40 you set new tab stops at **6, 12,** and **20,** terminating the setting process with CHR$(0). The H$ sends the tabbing command.

For your horizontal tab settings, you cannot use a number larger than the one that represents the last column of the page. If you have not reset the margins, for the FX-80 this is:

79 in Pica, 95 in Elite, and **131** in Compressed.

For the FX-100 this is:

135 in Pica, 162 in Elite, and 232 in Compressed.

Don't forget that resetting the margins automatically returns the tabs to their default settings. Set margins before tabs.

Tabs are set in the currently active pitch, and subsequent changes in pitch do not affect the tab positions. Here's proof. Add these lines to your current program:

```
60 LPRINT CHR$(15);
70 LPRINT H$;H$;"TWO MORE";H$;"THE END"
```

and RUN it.

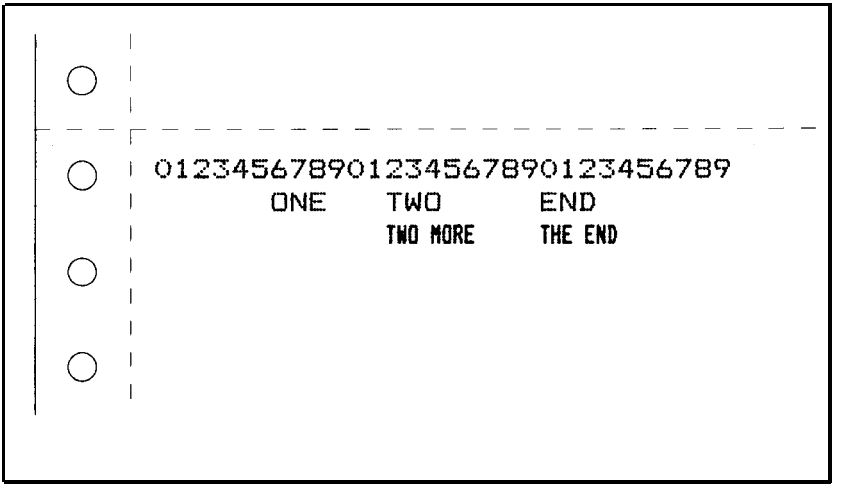


Figure** 9-9. **Absolute horizontal tabs

Figure 9-9 shows two more factors of tabbing. First, it demonstrates the way to skip a tab. Since in line 70 you specified a tab stop (H$) with no text string, the printer moved to the second stop. Second, this figure shows that the tabs are set absolutely. Although you changed the pitch to Compressed in line 60, which changed the width of characters and spaces, the width of the tab did not change. You will need to keep both these factors in mind as you use tabs.

Vertical tab usage

You use vertical tabs much as you do horizontal ones. With the vertical tab code, CHR$(11), you move from tab to tab. You can use

the default vertical tabs, which are set for every other line, or you can set tabs in one of two ways, in a single set or, for forms, in up to 8 sets, called channels.

Ordinary vertical tabs

Most often you probably will only need one series of vertical tabs. You set them with ESCape "B" in this format:

CHR$(27)"B"CHR$(n_1) . . . CHR$($n_k$)CHR$(0)

where n_1 to n_k represent up to 16 numbers that specify the lines that get tab stops. The process is terminated by CHR$(0). If your system won't send a 0, use any number lower than n_k.

The allowable range of tab settings in the ESCape "B" sequence is 1 to 255, but a number greater than the current form length is ignored. Therefore, you would use a setting as large as 255 only when the line spacing is less than three dots (3/72-inch).

Because the top line of a page is line 0, you can set tabs at line 1 through 65 with the default form feed of 66 lines.

You can set up all the tabs you'll need for a form without having to use all of them at once. Here, for example, is the beginning of a program that will result in a form used by a rental agent:

```
NEW
10 V$=CHR$(11)
20  LPRINT  CHR$(27)"B"CHR$(8)CHR$(18)CHR$(18)CHR$(27)
    CHR$(37)CHR$(48)CHR$(0)
30 LPRINT V$; "RENTAL MAINTENANCE REQUEST";
50 LPRINT V$
90 LPRINT V$; "TENANT"
```

So that your program will be concise, line 10 defines the vertical tabbing code as V$. Then line 20 specifies the six lines to which the printer will tab. When you RUN this program, the two sets of text of lines 30 and 90 print out at the first and third stops, while line 50 causes the printer to skip over the second stop. Figure 9-10 shows the printout.

You set the first tab for line 3, but the first printing is on the fourth line. This is because the printer counts the first line as the zero line.

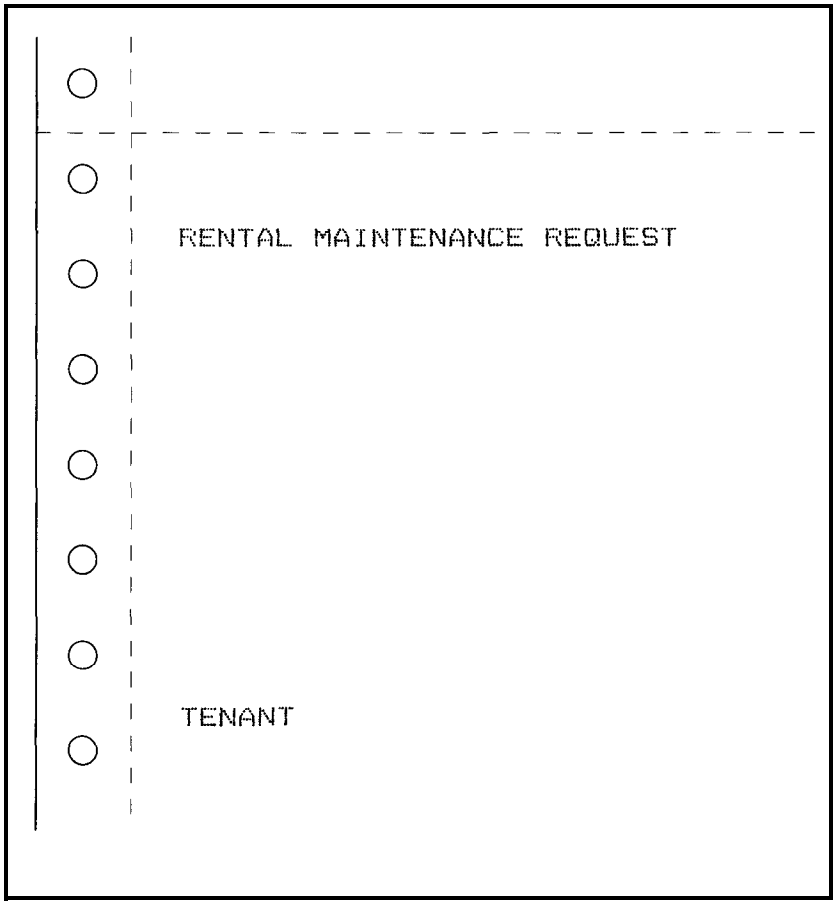


Figure 9-10. ***Ordinary vertical tabs***

Once you have tabbed to a stop, you can print more than one line of text at that position. See this by changing line 50 and adding the three lines shown below to your current program. If you enter the number of spaces that we have indicated with Ms, the entries will line up neatly.

```
50 LPRINT V$;"LOCATION"
60 LPRINT    ØØØ. ADDRESS:"
70 LPRINT    ØØØØØØC CITY:"
80 LPRINT  ØØØØØ STATE:"
```

Your printout should look like Figure 9-11.

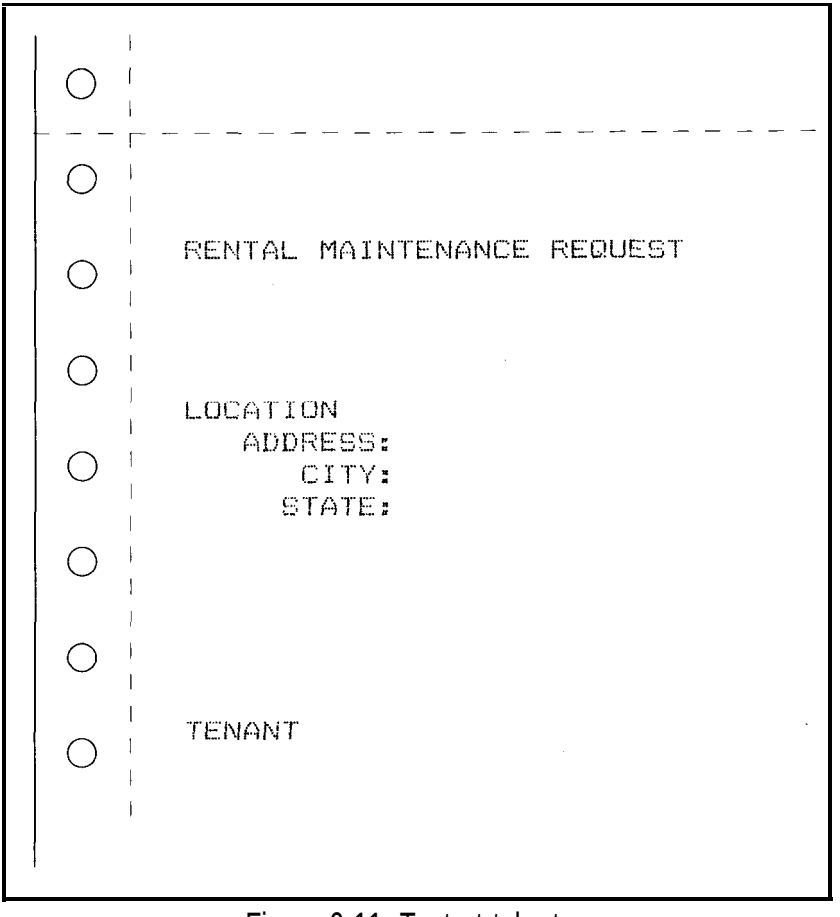


Figure 9-11. Text at tab stop

Just as for horizontal tabs, vertical tab settings are absolute: they do not change when you change the size of a space. For example, suppose you want to add to this form a graphics logo that uses special line spacing. If you forget to return to 12-dot spacing before the FX prints the next text after the logo, the line spacing will go awry-but each tab stop will remain the same distance from the top of form.

See that the tab stops are absolute by adding these two lines to your program:

```
25 LPRINT CHR$(27)"A"CHR$(4)
250 LPRINT CHR$(27)"@"
```

Line 25 changes the line spacing from 12/72-inch to 4/72-inch; line 250

uses the Reset Code to return the FX to 12-dot spacing. RUNning this program produces a printout to match Figure 9-12.

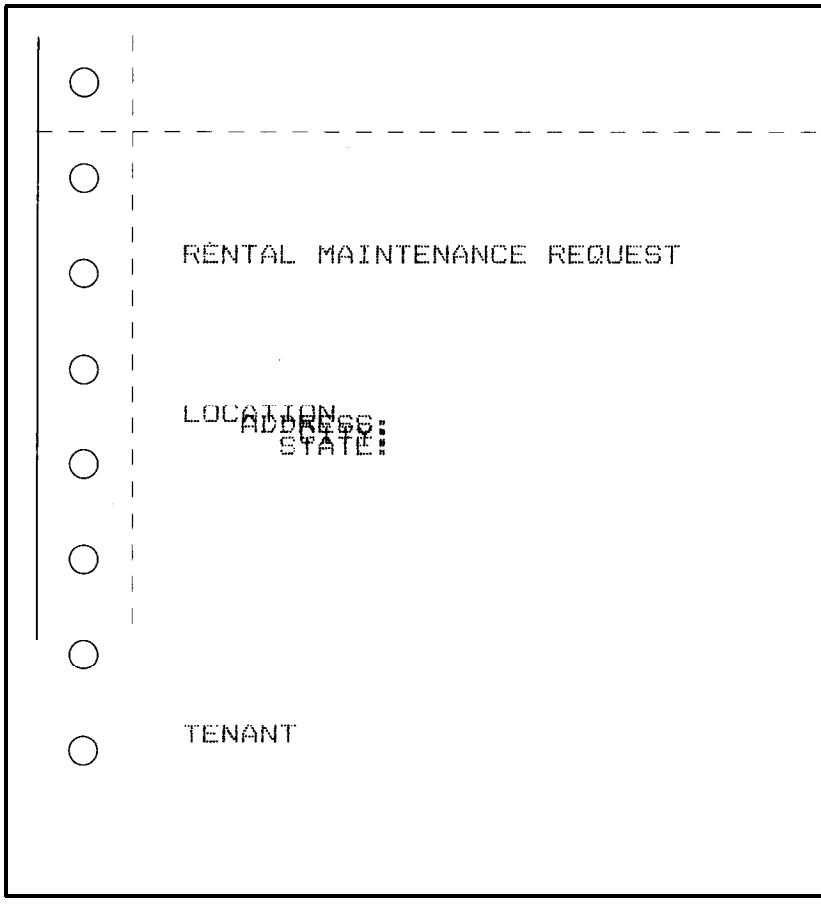


Figure 9-12. Absolute vertical tabs

Be sure to delete line 25 after you've seen its effect.

Vertical tab channels

Vertical tab channels are especially helpful in two situations. The first occurs when you are writing a program to accompany a pre-printed form that can accommodate various types of responses. The second occurs when you create a multipage form or report with different vertical tabs on each page.

You can store up to eight channels of tab stops, numbered from 0 to 7. You use a format that is similar to the one for a single set:

CHR$(27)"b"CHR$(N)CHRR$($n_1$) . . . CHR$(n_k)CHR$(0)

where N stands for a reference number between zero and seven under which this channel will be stored. If you have already stored a set using ESCape "B", the FX has labelled it as channel 0.

If your system won't send lowercase letters, substitute CHR$(98) for the "b". As for ESCape "B", you can store up to 16 stops, and you can use numbers between 1 and 254. You use either 0 or a number smaller than n_k to terminate the setting process.

Because the channels are stored, you must make the printer recall one before you can use it. You use this format:

CHR$(27)"/"CHR$(n)

where n stands for the number of the channel (0 - 7). After you have used this format, you perform tabbing by using CHR$(11) as usual.

Of the two reasons we mentioned for using channels, the programming of a multipage report form is the simpler. The program shown demonstrates the way you could set up an outline to be filled in by someone else or at a later date. Figure 9-13 shows the printout of that program.

```
NEW
10  LPRINT  CHR$(27)"b"CHR$(1)CHR$(15)CHR$(40)CHR$(1);
20 ' SET TABS AT 15,40 FOR CHANNEL #1
20  LPRINT  CHR$(27)"b"CHR$(2)CHR$(25)CHR$(30)CHR$(1);
40     ' SET TABS FOR CHANNEL #2 AT 25,30
50 LPRINT CHR$(27)"b"CHR$(3)CHR$(30)CHR$(45)CHR$(1);
60     ' SET TABS FOR CHANNEL #3 AT 30,45
100 ' ****** START OF MAIN PROGRAM ******
110 FOR Z=1 TO 3: READ X
120 LPRINT "TOP OF PAGE"
130  LPRINT  CHR$(27)"/"CHR$(X);
140    ' SET CURRENT CHANNEL "PAGE"
150 LPRINT CHR$(11)"TAB #1 FOR CHANNEL"X
160 LPRINT CHR$(11)"TAB #2 FOR CHANNEL"X
170 LPRINT CHR$(140);
180 NEXT Z
190 DATA 1,2,3
2000  LPRINT  CHR$(27)"@"
10000 END
```

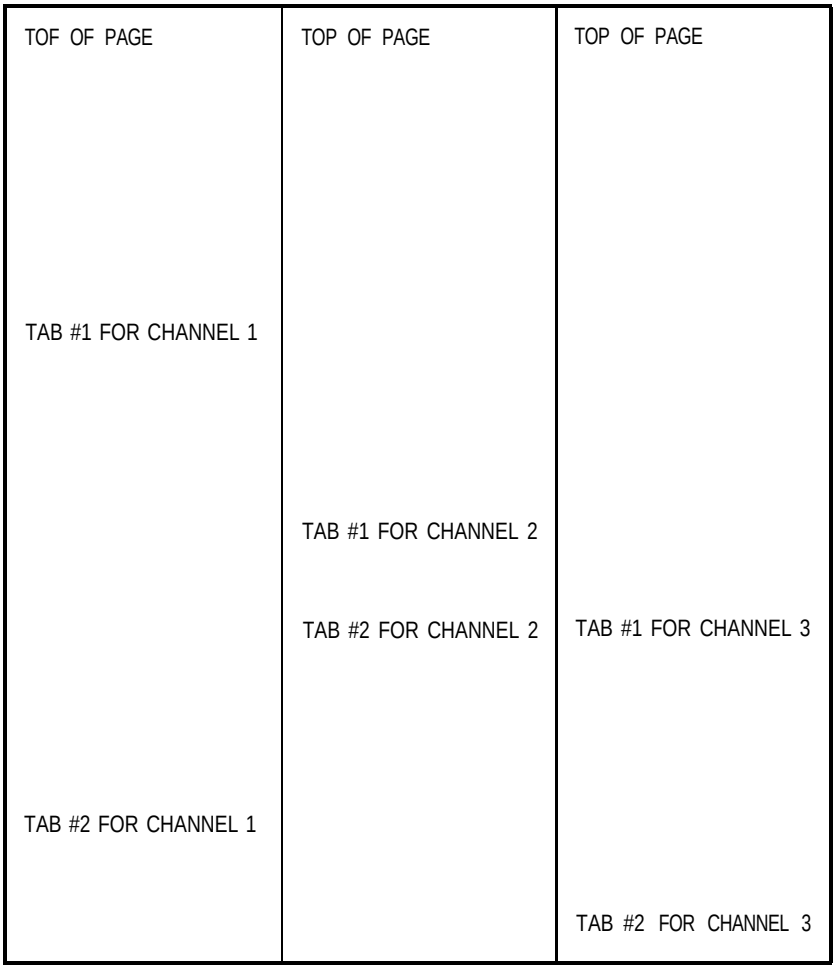


Figure 9-13. Printout of multipage channels

Summary

The FX gives you the ability to set margins and to use default, regulated and variable tabs; you can set tabs in either the horizontal or vertical direction. The default horizontal tabs occur in Pica, regardless of the current pitch. You set horizontal tabs in the current pitch. The default vertical tabs occur at every other line in 12-dot line spacing. All tabs are absolute once set; subsequent changes in pitch or line spacing do not change the tab positions.

CHR$(27)“1”CHR$(n) Sets the left margin to n. (If you can’t use lowercase letters, use CHR$(108) in place of “1”.) Limits are 0 - 78 in Pica, 0 - 93 in Elite, and 0 - 133 in Compressed

CHR$(27)“Q”CHR$(n) Sets the right margin to n. Limits are 2 - 80 in Pica, 3 - 96 in Elite, and 4 - 137 in Compressed

CHR$(9) or CHR$(U7)Moves the print head to the next horizontal tab.

CHR$(11) Moves the print head to the next vertical tab

CHR$(27)“B”CHR$(n_1)CHRS(n_k) . . . CHR$($n_k$)CHR$(0)
Sets vertical tabs at lines, n_1,n_2. . . n_k. Terminates with CHR$(0) or any number less than n_1.

CHR$(27)“D”CHR!$(n_1)CHR$($n_2$) . . . CHR$(n_k)CHR$(0)
Sets horizontal tabs at columns n_1, . . . n_k. Terminates with CHR$(0) or any number less than n_k

CHR$(27)“b”CHR$(N)CHR$($n_1$)CHR$(n_2) . . . CHR$($n_k$)CHR$(l)
N sets the vertical channel number. N = O-7; 0 is the same as CHR$(27) “B”

CHR$(27)"/“CHR$(n) Selects channel n

Chapter 10
Introduction to Dot Graphics

Welcome to the world of Epson graphics. To get you off to a solid start, we use this chapter to discuss all the fundamentals of dot graphics, from the number of dots per page to the way to position one dot, before we show you several patterns that you can print by using one dot at a time.

Dots and Matrixes

Imagine the blank printout page as a huge canvas that is made up of a series of dot matrixes. You can use this page like an artist's canvas to create your own graphic images. Where the painter uses brush and paint, you will use the computer and printer to express your artistic ideas.

Think of the page as a series of matrixes. For an FX-80 page and Pica characters, for example, you can calculate the number of main columns across a page by multiplying:

```
    80 matrixes wide
x    6 columns per matrix
=  480 dots per row
```

Then you can calculate the numbers of rows down a page by multiplying:

```
    66 lines per page
x   12 dots high per line
=  792 dots per column
```

A final multiplication:

```
    480 main columns
x   792 rows
```

gives you a grand total of 380,160 dot positions per FX-80 page. And that doesn't even take into account intermediate columns, the FX-100's ability to print 136 Pica matrixes, or both models' ability to use graphics density settings to increase the number of dots across the page and microscopic line spacing to increase the number of dots down the page.

Since there is a huge number of such dot positions on each page, this sounds like a giant task. It won't be, however, because we'll give you methods to shorten the time and steps to good graphics design.

Print Head

Printing high-resolution graphics on the FX requires a mode that is very different from the text modes. In any of the several versions of Graphics Mode, none of the predefined characters or symbols in the printer's memory are used. Instead, you create the patterns of dots that are printed. Thus, you control where and when each and every dot is printed.

To do this, you look at the page as a series of dot columns, arranged in rows. For each column position on a print line, the print head impresses the pattern of dots that you have specified. Before you can start designing these patterns, however, you need to know a little more about the way the print head works.

Even though there are nine pins on the print head, each column can be only eight dots high. That's because printer/computer communications are based on eight data lines, with each of the top eight pins of the print head corresponding to one of the eight lines. So each sweep of the head prints eight or fewer dots, depending on your computer system and your specifications (see Figure 10-1).

Seven-bit systems can control only the middle seven pins. Because we want users of such systems to be able to use these programs, we rarely put the top pin to work in the programs that follow. When we do, we tell you so.

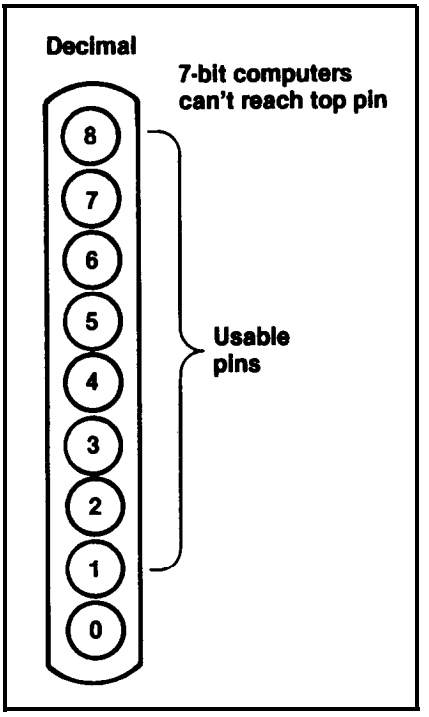


Figure 10-1. Pins numbered sequentially

Each time the print head makes a horizontal pass, it prints a pattern of dots. To print figures taller than 7 or 8 dots, the print head must make more than one sweep. If you use the 12-dot (default) line spacing, the print head will leave gaps between the graphics lines, just as it does between text lines. To avoid such gaps in your pattern, adjust the line spacing to 7- or 8- dot and print consecutive lines until the figure is complete. Figure 10-2 shows a before and after example (we work with this pattern later).

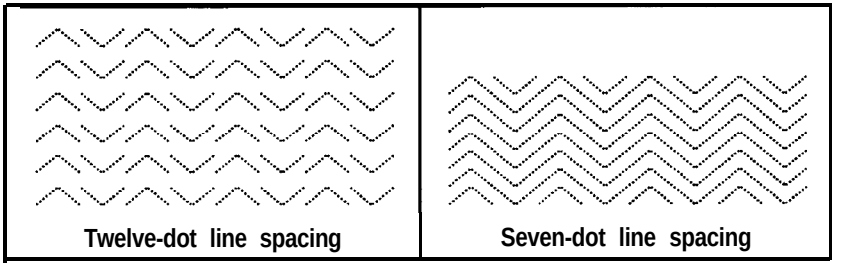


Figure 10-2. Dot pattern in two line spacings

Each pass of the print head contains one piece of the total pattern, which can be as tall or short as you desire. You don't have to use the whole page or even an entire line for your graphics figures. In fact, you can reserve as little or as much space as you like for a figure-and position it anywhere on the page.

Graphics Mode

Multi-line figures are printed in lines that are either seven or eight rows tall. For each graphics figure, you must first enter one of the seven versions of Graphics Mode and then tell the printer the number of columns you wish to print on each line. In other words, you specify the density of the dots.

Here is the format for entering Single-Density Graphics Mode (60 dots per inch).

```
LPRINT CHR$(27)"K"CHR$(n1) CHR$(n2);
```

The two number slots (n_1 and n_2) determine the number of columns reserved for graphics.

Why two numbers instead of one? To get around a limitation of the BASIC CHR$ function, which with only one reservation slot would not let you print a figure across the entire width of the page. An 8-inch page can hold up to 480 Single-Density graphics dots per row. But since the BASIC CHR$ function is limited to numbers from 0 to 255, you can't send a number as large as 480 directly to the printer.

That's where the second number slot fits in. You use the two sets of numbers together to send large numbers to the printer. The first number that you specify (n_1) indicates a number of columns (0 -255), as you'd expect. A 255 in that position says "reserve 255 columns for graphics," which means that any Single-Density figure less than half a page wide can be handled easily by the first number alone. Although you may sometimes need only this first slot for specifying width, you must not stop with that specification. You must still satisfy the printer, which always expects a value for the second slot (n,). Send it a 0.

Often a figure needs more than half a line. To reserve more than 255 columns for graphics, the second number (n,) must be greater than 0. But n_2 does not represent a number of single dots; it represents a number of groups, each of which contains 256 dots. Using a 1 in the second slot means "reserve one group of 256 dots plus whatever is in the first slot." A 2 in that spot means "reserve two groups of 256 dots (512) plus . . ." and so on-up to 7 times 256 (or 1792) dots on the FX-80 and up to 12 times 256 (or 3072) on the FX-100.

Actually, the FX-80 will accept numbers larger than 7 for n_2, but it is pointless to send them because the printer treats them as modulo 8. That is, 8 works the same as 0, 9 as 1, 10 as 2, etc.

Similarly, the FX-100 treats numbers above 12 as modulo 13. The maximum number of dots you can reserve on the FX-80, then, is:

```
CHR$(27)"K"CHR$(255)CHR$(7);
```

which is 255 dots plus 7 times 256 dots . . . for a total of 2047 dots per row. On the FX-100, you can reserve 255 plus 12 times 256 . . . for a total of 3327 dots per row.

But on a Single-Density print line you can only fit 480 dots.

For now, we'll stick with Single-Density, which means that we won't use numbers over 480. Later we'll see that the FX does have Graphics Modes of greater density.

Some systems, such as those for the IBM-PC and the Epson QX-10, also require WIDTH statements for longer lines. See your system documentation.

Pin Labels

Once you put the printer into Graphics Mode, your next step is to tell the print head which pins to fire at each new position. You do this by sending numbers via the CHR$ function. Each number that you send represents a unique combination of pins.

You might expect that the eight pins would be numbered 1 through 8 (as they are in Figure 10-1). But that won't work because you are going to send one number to represent all the pins to fire in one column. Using the 1-through-8 system, you would send such a total as 10 and the FX wouldn't know if this meant pins 4 and 6 or pins 2,3, and 5.

Since computers use the binary numbering system (Os and 1s only), it is most efficient for each pin to correspond to the decimal equivalent of one bit in an &bit binary number: 1, 2, 4, 8, 16, etc. (see Figure 10-3).

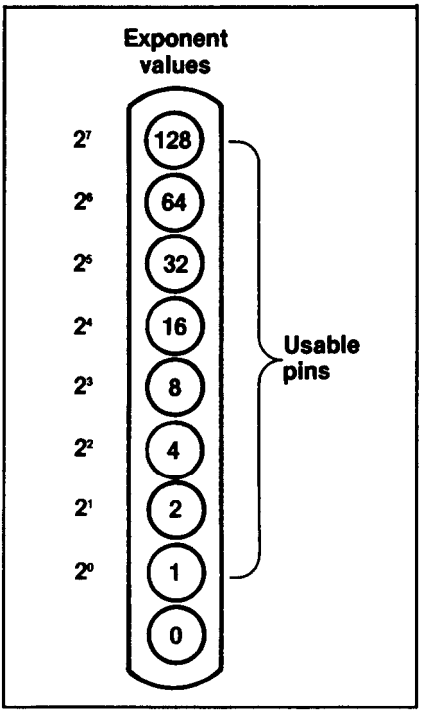


Figure 10-3. ***Pins labelled uniquely***

The hardware makes this the most practical labelling system. Each pin corresponds to one of the eight data lines from the computer, and each data line corresponds to one bit in a binary number. The place values in a binary number are nothing more than the series made up of powers of two. Figure 10-4 shows how you use a decimal sum--74--to fire a particular pattern of pins.

If you try adding several sets of pins together, you'll see that with this system you get no duplicates. The number 6 represents pins 2 and **4** (since 2 + 4 = 6) and there is no other way to get 6 by adding powers of two. This means that any combination of the eight pins adds up to a unique decimal number which falls within the range 0 to 255.

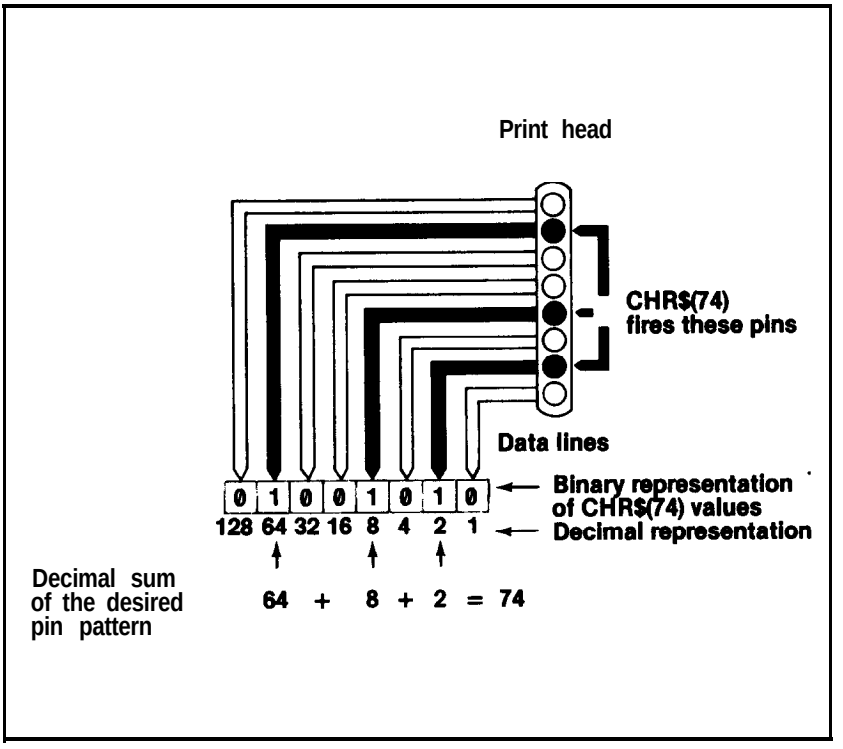


Figure 10-4. Pin combinations

Now that you, know the labels for the pins, how would you fire the top pin? Why, by sending LPRINT CHR$(128), of course. And how about the bottom graphics pin? That's right, LPRINT CHR$(l). If you wanted to fire only the top and bottom pins, you'd simply add 128 and one, then send LPRINT CHR$(129). By adding the appropriate label numbers together, you can fire any combination of pins you want.

Now you can see why not being able to send an ASCII code above 127 is the same as not being able to fire the top pin.

First Graphics Programs

The next few exercises show you what you can do by printing with one pin per column. That may sound simple, but with one pin at a time you can create interesting figures; you move from a straight line to a slash, a large caret, a wave pattern, and finally to a diamond pattern.

In the programs that follow (except the first), we shorten the process of specifying pins by using the fact that their labels represent powers of two. (Refer back to Figure 10-3 to refresh your memory about the relationship of ordinal numbers to powers of two and the exponential labels for the pins.) We use the caret (^) to represent exponentiation; for example, 2^6 means raise two to the sixth power. Some computer systems use, instead of a caret, an up-arrow (t), which prints as a left bracket ([) on the FX.

Straight line

Your first testing of the FX's graphics potential will consist of firing the bottom graphics pin. Enter and RUN this program (be careful to include the semicolons):

```
NEW
10 LPRINT CHR$(27)"K"CHR$(100)CHR$(0);
20 FOR X=1 TO 100
30 LPRINT CHR$(l);
40, NEXT X
50 LPRINT
```

If your system won't send CHR$(0), use CHR$(8).

This program deserves a full discussion:

Line 10 prepares the printer to accept 100 columns of graphics data.

Line 20 starts a loop for the LPRINT statement. Note that the loop must match the number of columns specified in line 10. The printer is expecting 100 bytes of data; it interprets everything it receives as graphics data until this quota is filled.

Line 30 sends a one to fire the bottom graphics pin. The semicolon at the end of the line is necessary to suppress the carriage return and line feed (ASCII 13 and 10), because otherwise they are sent automatically at the end of each LPRINT line. Without that semicolon, the printer would receive the sequence 1,13,10,1,13,10 . . . instead of l,l,l,l.

Line 40 completes the loop.

Line 50 doesn't print anything-it just forces a carriage return at the end of the print line, overriding the semicolon of line 30. Forcing the carriage return is not really necessary since the line is the last one of this program. It's just a good habit to develop.

Notice that the printer doesn't print each time it receives a CHR$(l). The FX stores data in its print buffer until it receives as many numbers as it expects-in this case, 100.

Slash

Using the form 2^X, you can fire individual pins by letting X vary between 0 and 7 (0 and 6 for 7-bit systems). Here's how it works. To exercise the pins in a pattern (a slash) that shows off their placement, enter:

```
NEW
40 LPRINT CHR$(27)"K"CHR$(7)CHR$(0);
80 FOR X=0 TO 6
ll0 LPRINT CHR$(2^X);
120 NEXT X
```

When X equals 0,2^X is 1-so the bottom graphics pin is fired. When X equals 1,2^X is 2--so the second pin is fired. This pattern continues right up through X equals 6, which fires the seventh pin. We purposely omit X equals 7 to accommodate systems that are limited to 7 bits.

Large caret

The next step is to change the direction of the slash. Can you guess how it's done? Sure, just reverse the order of the exponents, and the same routine can be used. In fact, let's turn it into a subroutine:

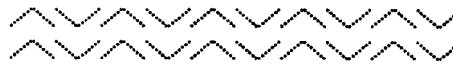
```
10 F=O
40 LPRINT CHR$(27)"K"CHR$(14)CHR$(0);
50 GOSUB 80: F=1: GOSUB 80
70 LPRINT CHR$(27)"@": END
80 FOR X=0 TO 6
90 N=X: IF F=1 THEN N=6-X
```

```
110 LPRINT CHR$(2^N);
120 NEXT X: RETURN
```

On the first pass of the loop (line SO), N equals X and the exponents increase in order from 0 to 6. The second time the routine is called, N equals 6 minus X, which reverses the order (from 6 down to 0). The flag F of line 50 activates the change of direction, and line 90 reflects the value for the exponent.

This two-directional slash routine can be repeated indefinitely. For an interesting variation, alternate the direction of the slashes each time a pair is printed by changing the flag F in line 50:

```
20 FOR L=l TO 2
30 FOR J=0 TO 9
50 GOSUB 80: F=1-F: GOSUB 80
60 NEXT J: LPRINT: NEXT L
```



In this version of the program, Line 50 makes F alternate between zero and one. The J loop repeats pairs of diagonals on one line, while the L loop adds a second line.

Wave pattern

For most graphics programs, you'll want to change from the normal l2-dot line spacing to 7-dot--or B-dot spacing if you can use the top pin. Add line 10 to make your listing look like this:

```
10 F=0: LPRINT CHR$(27)"1"
20 FOR L=1 TO 2
30 FOR J=0 TO 9
40 LPRINT CHR$(27)"K"CHR$(14)CHR$(0);
50 GOSUB 80: F=1-F: GOSUB 80
60 NEXT J: LPRINT: NEXT L
70 LPRINT CHR$(27)"@": END
80 FOR X=0 TO 6
90 N=X: IF F=1 THEN N=6-X
110 LPRINT CHR$(2^N);
120 NEXT X: RETURN
```

See what a big difference the line spacing makes? All of the multiple-line graphics programs in this manual use this line spacing.

Diamond pattern

In this next and final version of the program, you exercise even more control over the slashes. This program varies not only their direction, but also their sizes (length and height) on the print line. Although the program still uses only one subroutine, it prints 24 different patterns, 12 on each of the 2 print lines. The differences in the patterns are achieved by IF-THEN tests in a subroutine.

To get the final listing, add line `100` and change lines `10, 30, 40, 50,` 80, 90, and 120:

```
10 LPRINT CHR$(27)"1"
20 FOR L=1 TO 2
30 FOR J=0 TO 1
40 LPRINT  CHR$(27)"K"CHR$(27)CHR$(0);
50 GOSUB 80
60 NEXT J: LPRINT: NEXT L
70 LPRINT CHR$(27)"@": END
80 FOR X=1 TO 6: Y=X: IF J=1 THEN Y=7-X
90 FOR Z=0 TO Y: N=Z: IF J=1 THEN N=Y-Z
100 IF L=2 THEN N=7-N
110 LPRINT CHR$(2^N);
120 NEXT Z: NEXT X: RETURN
```

If you didn't watch the printing when you ran this program, the printout may not make it obvious that there are two lines of print-but there are. The first line prints slashes by moving from bottom to top and then from top to bottom, whereas the second line prints the slashes first from top to bottom and then from bottom to top.

And how do the IF-THEN statements fit into the picture? The one in line 80 changes the length of the slashes. The one in line 90 controls the direction and height. The IF-THEN in line `100` makes the bottom row a mirror image of the top.

Summary

You enter Graphics Mode with the CHR$(27)"K" **CHR$(n_1)CHR$($n_2$)** command. You determine the number of graphics columns by filling the two reservation slots, n_1, and n_2. You fire your pin patterns by adding up the pin labels, which consist of powers of two.

Here is the command we introduced in this chapter:

CHR$(27)"K"CHRS($n_1$,)CHR$(n_2);
Enters Single-Density Graphics Mode and specifies width setting. Width = $n_1+256*n_2$ where n_1 is 0 - 255 and n_2 is 0 - 7 on the FX-80 and 0 - 12 on the FX-100

Note: Single-Density graphics dots are printed 60 per inch horizontally and 72 per inch vertically.

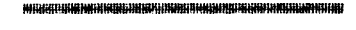
Chapter 11
Varieties of Graphics Density

We introduced you to FX graphics by having you use Single-Density Graphics Mode and a single pin per column. In this chapter we cover six more graphics densities and provide examples of designs that use pin combinations. We also offer you several tips for programming graphics.

Graphics Programming Tips

Let's start with a program that fires the four low graphics pins in each column. Since these pins are labelled 1,2,4, and 8, and since the sum of these four labels is 15, send a CHRS(15) to the printer:

```
NEW
20  A$=CHR$(27)+"K"+CHR$(100)+CHR$(0)
30 B$=CHR$(l5)
40 LPRINT A$;:  FOR X=1 TO 100: LPRINT B$;: NEXT X
80 LPRINT CHR$(27)"@"
```



Sure enough, CHR$(15) fires the four low pins.

This is basically the same program as the.one you used to print the first slash in the last chapter, but we've added a new wrinkle. We have you store the concatenated string for entering Single-Density Graphics Mode in the variable A!$ and the string for data in B$. The first variable makes it easy to enter a Graphics Mode several times in one program, and the second, to send data repeatedly.

Graphics and the Reset Code

You may have wondered about the semicolon we had you place after the B$ in line 40. If this line were text rather than graphics data, this semicolon would cause the text and the Reset Code (CHR$(27)"@") in line 80 to be placed in the same text buffer and consequently the Reset Code would wipe the text out. This doesn't happen with the graphics command and its data because the CHR$(27)"K" command itself dumps everything from the buffer to the printer. By the time the Reset Code makes its appearance, both the command and its data are gone.

To see this in action, add a line to your program:

```
50 INPUT A
```

The INPUT causes the program run to pause for you to see that the graphics design is printed before the program reaches the Reset Code in line 80. To let the program finish, press your carriage-return key.

There's an important correlation to note. Anything that was in the buffer prior to the graphics string remains unscathed, and anything left behind is erased. To verify this, make these changes in your program:

```
35 LPRINT "PROTECTED TEXT";
50 LPRINT "THIS WILL NOT PRINT";
```

and RUN it once.

PROTECTED TEXT

In general, you need to remember that the Reset Code doesn't affect graphics data, but it may affect companion text.

Graphics and low ASCII codes

Sending the codes between zero and 31 from BASIC can be a problem when you are printing graphics just as it is when you are printing text. The problem again is that most computer systems handle some of these codes in a special way instead of delivering them to the printer.

For example, one computer system handles form feeds by itself. It counts lines to keep track of the top of form. If a program sends the CHR$ command with the number that ASCII designates as the code

for form feed-CHR$(12)--to the printer, the computer system intercepts it and sends instead a series of line feeds-CHRS(10). It does this whether the CHR$ commands represent true commands, parameters for commands, or data. It screens out all instances of its reserved numbers. You can see how this could upset the printing of graphics. In this example, you would get pins 2 and 4 (whose sum is 10) when you wanted pins 3 and 4 (whose sum is 12).

Does this mean that if your computer system transmutes some control codes, you can't use the pin patterns of those numbers in your graphics programs? Well . . . yes, it does. At least, not via the usual CHR$ function. But you can often design around these problems by using other numbers with similar patterns, or you can POKE the trouble codes directly to the FX. Learning how to cope with these problems is part of learning how to use the printer with your system. See Appendix F for help.

Density Varieties

The FX printer offers you the 16 text densities that we printed out as Figure 5-2, and seven graphics density settings that we list later in this chapter (Table 11-1). Remember that you can choose a text density by specifying it in the Master Select ESCape code sequence, and that you can switch densities later by changing one parameter in that code.

The FX has a similar command sequence for specifying and changing modes for its graphics densities. Here is the commands format:

```
LPRINT  CHR$(27)"*"CHR$(m)CHR$(n1)CHR$(n2);
```

where m indicates the number (0 - 6) of the desired Graphics Mode, and the settings n_1 and n_2 are the usual graphics width settings. The seven modes include six densities and the two speeds for Double-Density.

Before you try out any of the new graphics density settings, record a sample line of Single-Density. To do so, enter this new program:

```
20  A$=CHR$(27)+"*"+CHR$(0)+CHR$(50)+CHR$(0)
30  B$=CHR$(85)+CHR$(42)
40 LPRINT A$;: FOR X=1 TO 25: LPRINT B$;: NEXT X
50 LPRINT "Ø SINGLE-DENSITY GRAPHICS Ø";
60 LPRINT A$;: FOR X=1 TO 25: LPRINT B$;: NEXT X
70 LPRINT
80 LPRINT CHR$(27)"@"
```

The printer fires pins 1, 3, 5, and 7 (with the respective exponential values of 1, 4, 16, and 64) in the first column and pins 2, 4, and 6 (exponential values 2, 8, and 32) in the second. And it alternates that sequence for 50 columns-50 columns in Single-Density.

This program also mixes graphics and text on one line. It does that by using semicolons to keep both kinds of output on the same print line.

High-Speed Double-Density Graphics Mode

Now let's print the same pattern in twice the normal (Single) density. Change the first 0 in line 20 to a 2 and retype the text in line 50 as shown below:

```
20  A$=CHR$(27)+"*"+CHR$(2)+CHR$(50)+CHR$(0)
50 LPRINT "  ø HIGH-SPEED DOUBLE-DENSITY GRAPHICS ø";
```

If your computer system requires a WIDTH statement to prevent the printer from issuing a carriage return before the graphics line is complete, add it now:

```
7 WIDTH LPRINT 255
```

The format for this statement may be different for your BASIC; see your software documentation.

HIGH-SPEED DOUBLE-DENSITY GRAPHICS

The program still prints 50 columns of dots, but now it presses them together and prints them in half the space. They print at the same speed as in Single-Density Graphics.

This High-Speed Double-Density Graphics Mode has one drawback. Because the dots are so closely packed, two dots in the same row cannot appear in two consecutive columns. In the above program we avoided the problem by never calling up one dot twice in succession. If you try to print two consecutive dots, the printer simply ignores the second one (see Figure 11-1).

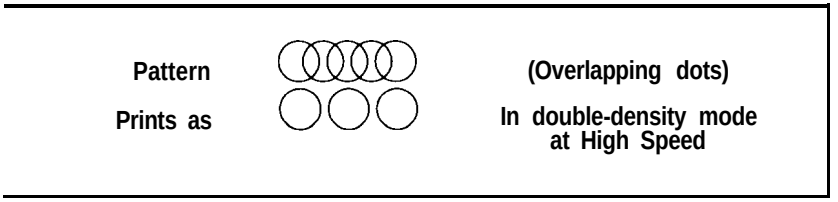


Figure 11-1. High-Speed Double-Density dots

To check this out, change the pin patterns in line 30 from 85 to 127, the sum of the labels for pins 1 through 7:

```
30 B$=CHR$(127)+CHR$(42)
```

As Figure 11-2 shows, the repeated dots, the ones called for by the CHR$(42)--pins 2, 4, and 6--are not printed at all. The print head is moving too fast to retract the pins and then instantly fire them again, so the FX's program suppresses them.

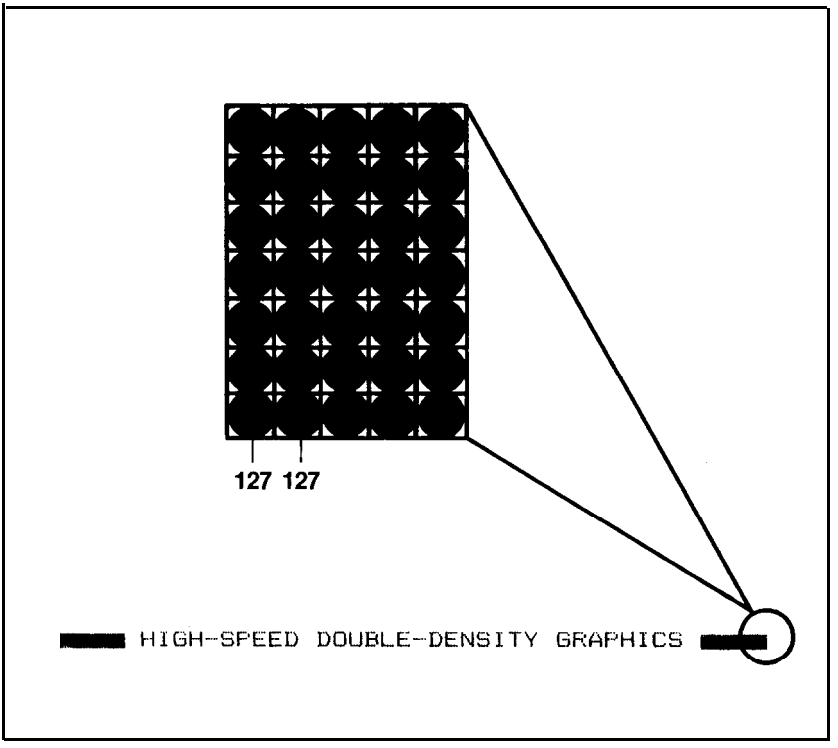


Figure 11-2. No overlapping dots

Low-Speed Double-Density Graphics Mode

Ah, but the FX has a special print mode to take care of this very problem. It's called the Low-Speed Double-Density Graphics Mode. Change the 2 of line 20 to a `1` and change the text in line 50 once more:

```
20 A$=CHR$(27)+"*"+CHR$(l)+CHR$(50+CHR$(0)
50 LPRINT "PLOW-SPEED DOUBLE-DENSITY GRAPHICS ø";
```

Take note of the print speed when you RUN your program this time. It's the same density as the previous mode, but printed at half the speed. As you can see in Figure `11-3` or from looking very carefully at your printout, this time the CHR$(42) columns are printed as requested.

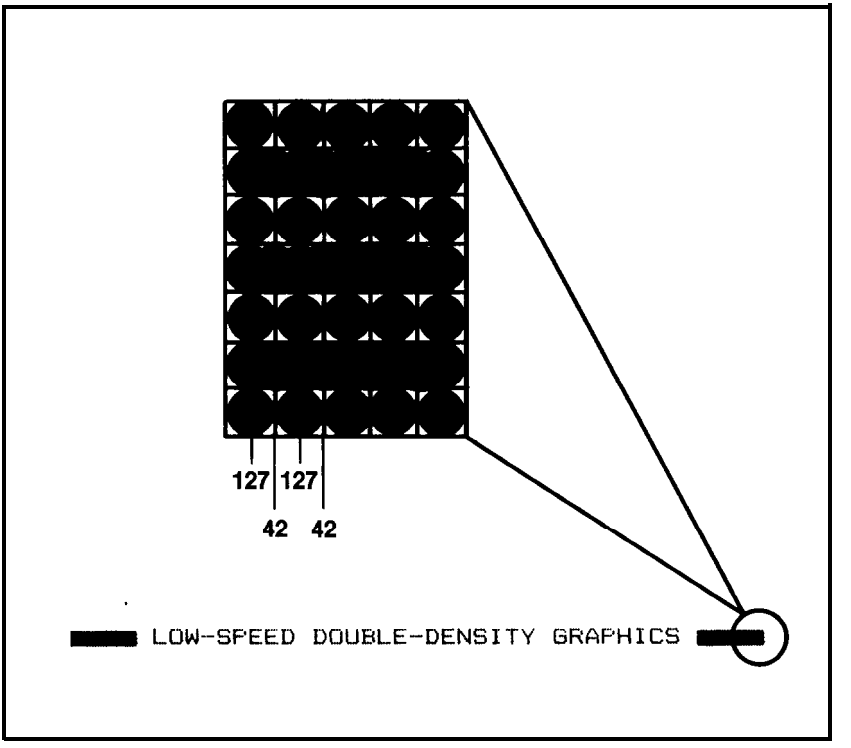


Figure 11-3. Overlapping dots

Quadruple-Density Graphics Mode

The FX also gives you the ability to print dots four times as densely as in Single-Density. Change the 1 line 20 to a 3 and lines 30 and 50 to read:

```
20  A$=CHR$(27)+"*"+CHR$(3)+CHR$(50)+CHR$(0)
30 B$=CHR$(85)+CHR$(42)
 50 LPRINT    "QUADRUPLE-DENSITY GRAPHICS ";
```

QUADRUPLE-DENSITY GRAPHICS

In Quadruple-Density Graphics Mode, any FX can print 480 times 4 (or 1920) columns of dots on a single 8-inch line and the FX-100 can print 816 times 4 (or 3264) columns on a 13.6-inch line. The graphics setting required to fill an entire &inch line in this mode is n_1 equals 128 and n_2 equals seven (since 7 times 256 plus 128 equals 1920).

At this density the same limitation as for High-Speed Double-Density Graphics Mode applies-you can't print two adjacent dots in the same row.

More densities

Yes, there are more densities. Besides the three graphics densities we've covered so far (Single, Double, and Quadruple), the FX can print in two graphics densities that match the screen densities of two types of CRTs. One of these may allow you to develop designs and then print them in the same weight. Additionally, the FX can simulate the density that plotters use-called one-to-one aspect ratio. This mode prints the same density horizontally as vertically, 72 dots per inch each way, which allows you to print true circles.

You can print a sample of all the modes with the following program:

```
10 FOR M=0 TO 6
20  A$=CHR$(27)+"*"+CHR$(M)+CHR$(120)+CHR$(0)
30  B$=CHR$(85)+CHR$(42)
 50 LPRINT "MODE #";M;"";
60 LPRINT A$;: FOR X=1 TO 60: LPRINT B$;: NEXT X
70 LPRINT: NEXT M
80 LPRINT CHR$(27)"@"
```

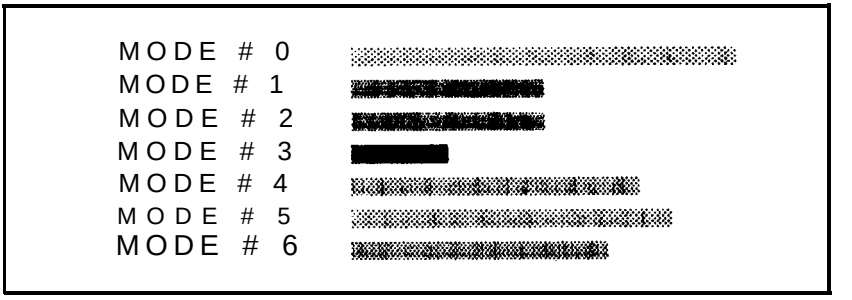


Figure 11-4. Seven density modes

Figure 11-4 displays all seven of the FX modes that affect graphics density. Table 11-1 describes them.

More Graphics Programming Tips

The next two sections discuss two modes that the FX offers to help you solve potential graphics problems. A reassigning code allows you to change the density for graphics programs that use one of the four alternate codes. The 9-pin Graphics Mode allows you to use all nine pins on each line and thus speed up screen dumps.

Reassigning alternate graphics codes

The FX provides a command to reassign one of the alternate graphics codes-K, L, Y, or Z-so that it represents any other of the seven Graphics Modes. The command and its format are:

```
LPRINT CHR$(27)"?s"CHR(n);
```

where s is one of the four symbols, K, L, Y, or Z, and n is one of the numbers used with the ESCape " *" command, 0 to 6. There are several instances in which you may use this sequence.

The first occurs if you have written a program to be printed in one Graphics Mode and now want to print it in another. If you have used concatenation to store your Graphics command strings in one short character string, that will not be difficult. You can simply change the mode number or alternate code in the definition of the character string.

In the program for Figure 11-4, for example, you did this in line 20 by storing the Graphics Mode command sequence in A$ and making

Table 11-1. Graphics Modes

Mode	Density	Alternate code	Description	Head speed (in./sec.)
0	Single	CHR$(27)"K"	60 dots per inch; 480 dots per 8" line 816 dots per 13.6" line	16
1	Low-Speed Double	CHR$(27)"L"	120 dots per inch; 960 dots per 8" line 1632 dots per 13.6"line	8
2	High-Speed Double	CHR$(27)"Y"	Same density as Mode 1, but faster. The printer does not print consecutive dots in any one row.	16
3	Quadruple	CHR$(27)"Z"	240 dots per inch; 1920 dots per 8" line; 3264 dots per 13.6"line The printer does not print consecutive dots in any one row.	8
4	Epson QX-10	none	Matches the screen density of the QX-10: 80 dots per inch; 640 dots per 8" iine; 1088 dots per 13.6" line. (This makes it easv to do screen dumps.)	8
5	One-to-one (plotter)	none	72 dots per inch 576 dots per 8" line; 979 dots per 13.6" line. Produces the same density horizontally as vertically, which makes circles look round	12
6	Other CRT screens	none	90 dots per inch; 720 dots per 8" line; 1224 dots per 13.6" line Matches the Corvus Concept and DEC® screens.	8

the mode number a variable, M. As M varied, so did the graphics density. It was your use of the "*" command that allowed you to use the variable.

Suppose you had used the "K" command instead. You could use the reassigning code to make it work as though it were "*". To see this work, enter the following lines for your current program:

```
15 LPRINT CHR$(27)"?K"CHR$(M);
20   A$=CHR$(27)+"K"+CHR$(l2@l)+CHR$(@)
```

You should get another printout of Figure 11-4.

A second time you can make good use of the reassigning code occurs when you want to change a program in which you have not concatenated the graphics codes. Using the "?" sequence allows you to change every instance of your graphics command by entering only one line.

A third type of use occurs when you want to use a program developed for a different model of Epson printer. Suppose you have a program for circles written for an MX that uses Double-Density-the "L" variety-and you want to refine it by switching to the one-to-one aspect ratio. In this case, you would use the following line at the beginning of your program:

```
LPRINT CHR$(27)"?L"CHR$(5);
```

and the FX will do the rest.

Nine-Pin Graphics Mode

Recall that in the last chapter we said that the bottom (zero) pin of the print head is not normally used in the Graphics Modes. That's because most microcomputers communicate with parallel-type peripheral devices using eight data lines (even if they have 16-bit processors). When the peripheral is a printer, each data line corresponds to one pin on the print head. Thus each byte sent will fire up to eight pins.

But the printer has 9 pins available. So how do you fire the ninth pin with only 8 data lines? In fact, do you really want to bother with just one extra pin? Well, for such graphics-intensive applications as screen dumps, printing 9 pins at a time can speed up the process considerably, For this purpose, the FX has a special 9-Pin Graphics Mode (it won't, however, work with 7-bit computer systems). In this mode the printer takes 2 bytes to fire all 9 pins-as shown in Figure 11-5.

Since computers are faster than printers, there is no significant time loss in printing a single line of graphics with 9 pins. You get 9 dots per line in about the same time as you get 8 dots in the other Graphics Modes. Not a bad deal.

The format for entering 9-Pin Graphics Mode is:

```
LPRINT  CHR$(27)"^"CHR$(d)CHR$(n1)CHR$(n2);
```

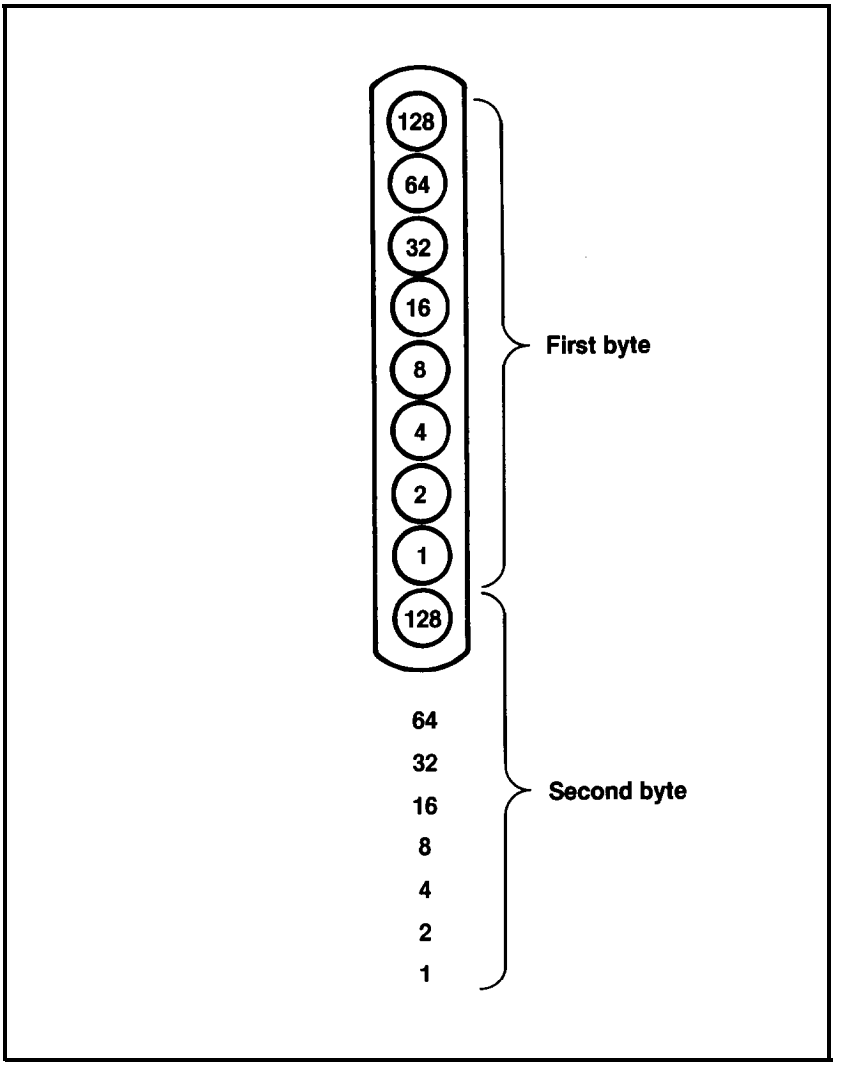


Figure **11-5.** *Nine-pin usage*

(Use CHR$(94) if you can't generate the caret symbol (^) from your system.) The d determines the density of the graphics: d set to 0 produces Single-Density; d set to **1** produces Double-Density.

In this format, n_1 and n_2 represent the usual width settings, but each print pattern requires two bytes (instead of one). This means that when you want to print 60 columns of graphics, you must send 120 data bytes.

Firing 9 pins with 8 data lines is just a shade more difficult than firing 7 or 8 pins. It takes 2 bytes to define each 9-dot pin pattern: the first byte determines the pattern of the top 8 pins in the usual way and only the top bit of the second byte is used. Thus any second byte of 128 or greater fires the bottom pin of the print head; anything less does not. Try this sample program:

```
20  A$=CHR$(27)+CHR$(94)+CHR$(0)+CHR$(60)+CHR$(0)
30 B$=CHR$(85)+CHR$(0)+CHR$(l70)+CHR$(128)
60 LPRINT A$;: FOR X=1 TO 30: LPRINT B$;: NEXT X
80 LPRINT CHR$(27)"@"
```

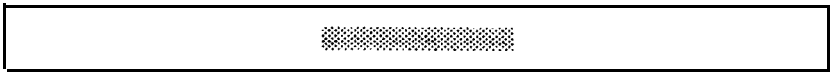


Figure** 11-6. **Printout using bottom pin

Compare this with the densities in Figure 11-4 (this one is Single-Density). Look closely at Figure 11-6; you'll see that the bottom pin prints in every other column. If you want to see Double-Density, change the first 0 of line 20 to a 1. For fans of 9-Pin Graphics, the CHR$(27)"0" line spacing is ideal: it sets the line spacing to 9/72-inch 9-dot).

Pin Combination Patterns

The next phase in printing graphics is to arrange pin firing sequences into meaningful designs. Figure 11-7 shows how you might design a dot pattern on graph paper.

In Figure 11-7, we show on the side of the figure the pin labels for each row of dots. At the bottom of each column we show the sum of those labels. These sums are the numbers you send to print this pattern.

Once you've calculated the numbers for a pin pattern, you can store them in DATA statements. You separate items in a DATA statement with commas. A program reads these items from the DATA statement into variables with a READ statement.

Now begin a new program with a READ statement and the values for the pattern of Figure 11-7. Enter the following lines, but don't RUN the program yet.

```
50 READ N
90  DATA  3,7,31,63,126,124,112,96,92,66,33,25,5,3
```

Line 50 reads the first data number into the variable N. To read the rest of the numbers, line **50** must be executed in a loop. Add these lines to the program:

```
20  A$=CHR$(27)+"K"+CHR$(14)+CHR$(0)
30 LPRINT A$;
```

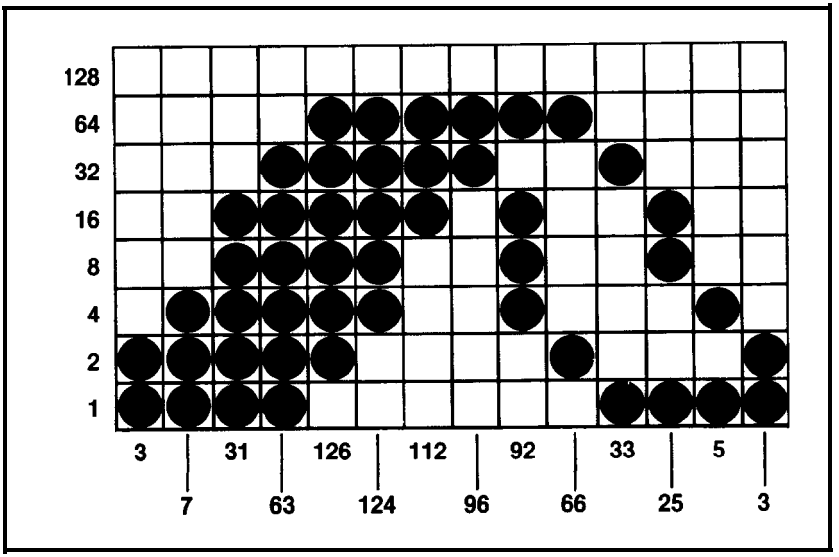


Figure 11-7. Curling design

```
40 FOR X=1 TO 14
60 LPRINT CHR$(N);
70 NEXT X
80 LPRINT CHR$(27)"@": END
```

Perfect! Just like the design.

Repeated patterns

Now how would you go about repeating this pattern-add more data? Or store the data in program variables to be recalled as needed? Here you don't need to go to that much trouble. For simple programs

like this one, you can just get the program to reread one set of data by using a RESTORE statement. To see this, change two lines and then RUN the program:

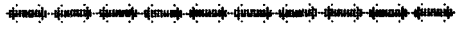
```
30 FOR Y=l TO 19: RESTORE: LPRINT A$;
79 NEXT X: NEXT Y
```

Although the new loop in line 30 repeats the pattern 10 times, you don't need 10 repetitions of the DATA statements. The RESTORE statement in line 30 tells the program to read the same data again. If you use it carefully, the RESTORE statement can reduce the amount of data that you need to type in for a graphics pattern.

Repeated DATA numbers

Sometimes you will want to repeat the same DATA number several times in succession. Here's a change that gives an instance:

```
90 DATA 8,28,62,93,28,28,28,28,28,28,93,62,28,8
```



At the center of this program line you fire the middle three pins-with CHR$(28)-six times in succession. Six 28s was boring enough to type. Can you imagine typing the data for 20 or 50 repetitions of the same number? There has to be a-better way-and there is.

You can enter the number of repetitions right into the DATA lines, coded as a negative number. Then change the READ routine to test for a negative number. When the program reads a negative number, it transfers control to a special subroutine that does the repeating. (Or you could use a STRING$ statement.)

Change lines 50 and 90, and add lines 100, 110, and 120 so that your listing looks like this:

```
20  A$=CHR$(27)+"K"+CHR$(14)+CHR$(0)
30 FOR Y=1 TO 10: RESTORE: LPRINT A$;
40 FOR X=1 TO 14
50 READ N: IF N<0 THEN 100
60 LPRINT CHR$(N);
70 NEXT X: NEXT Y
```

```
80 LPRINT CHR$(27)"@": END
90 DATA 8,28,62,93,-6,28,93,62,28,8
100 READ R: FOR J=1 TO -N
110 LPRINT CHR$(R);: NEXT J
120 X=X-N-1: GOT0 70
```

RUN it again. Same arrow pattern, right? And with less data. The number of repetitions (6) is entered into the DATA line as a negative number that is followed by the pattern (28) to be repeated. Yet even with this short cut, graphics designs do require that you plan and enter lots of data. In the next few chapters, we'll show you more ways to take advantage of design patterns to reduce the amount of data needed.

Summary

We began this chapter by adding up pin labels and using the total to fire four graphics pins at one time. We changed that program to show that the Reset Code does not affect either graphics commands or data. And we commented on problems that may arise from software interfacing when you try to send codes that represent added-up pin labels.

After this general discussion, we introduced two new graphics densities that have specific commands. You can print Double-Density at one of two speeds: 160 cps (High) and 80 cps (Low). The second of these modes, Quadruple Density, prints only at Low.

We then introduced a new single command with which you can access any one of the seven FX graphics modes; Table 11-1 summarizes them.

Here are the commands that we introduced in this chapter:

CHR$(27)"*"CHR$(m)CHR$($n_1$)CHR$(n_2)
: Enters Variable-Density Graphics Mode and specifies a mode and the width setting, where m is mode 0 - 6. Width = $n_1+(256*n_2)$, where n_1 is 0 - 255 and n_2 is 0 - 7

CHRS(27)"Y"CHR$($n_1$)CHR$(n_2)
: Enters High-Speed Double-Density Graphics Mode and specifies the width setting. Width = $n_1 + (256*n_1)$, where n_1 is 0 - 255 and n_2 is 0 - 7

CHR$(27)“L”CHR!$(n $_1$)CHRS(n_2)

Enters Low-Speed Double-Density Graphics Mode and specifies the width setting. Width = n_1+(256*n_2), where n_1 is 0 - 255 and n_2 is 0 - 7

CHR$(27)“Z”CHR!$(n $_1$)CHR$($n_2$)

Enters Quadruple-Speed Graphics Mode and specifies the width setting. Width = n_1+(256*n_2), where n_1 is 0 - 255 and n_2 is 0 - 7

CHR$(27)“?s”CHR$(n)

Reassigns an alternate code to a new code number so that it produces a different Graphics Mode, where s is the sequence letter K, L, Y, or Z and n is 0 - 6

CHRS(27)"^"CHR$(d)CHR$(n_1)CHR$($n_2$)

Enters 9-Pin Graphics Mode and specifies the width setting, where d is 0 for Single-Density and 1 for Double-Density. Width = n_1+(256*n_2) where n_1 is 0 - 255 and n_2 is 0 - 7

Chapter 12
Design Your Own Graphics

In this chapter we take you through the development of two graphics programs, from design to implementation. The two programs use entirely different techniques. The first program uses a method of storing and recalling data similar to that of the curling program in the last chapter. You store pin patterns and their repetition factors in DATA statements to produce a pattern that is suitable as a logo.

The second program does not read pin patterns from DATA statements at all. Instead, you code the few necessary pin patterns into the program as constants. You do store values in DATA lines-to control the number of times each pattern is to be repeated.

These examples show how easy it is to create high-resolution dot graphics on your FX. We hope they inspire you to include graphics in your own programs.

Planning Process

It should be apparent by now that printing high-resolution images requires careful planning and lots of data. Programs available for some computers enable users to draw figures on the screen, then echo them to the printer. Without such a program, there really is no quick and easy way to calculate the data required for pin patterns.

Once you have a design, you can usually plan your graphics program most easily by following these steps:

1. Plot the design, dot by dot, on graph paper.

2. Translate the dots into their appropriate pin numbers, seven or eight rows (depending on your computer system's capability) at a time.

3. Figure out the easiest way to send those numbers to the printer.

Once you get the hang of it, the whole process is easy. It does require some patience, but sometimes, when regular patterns form your designs, you can use the computer to do most of the tedious work. It can calculate the pin patterns.

Follow the development of the programs in this chapter, then get yourself some graph paper and a pencil and have a good time.

STRATA Program

This program prints (in Double-Density Graphics Mode) a sample logo that will be used again in Chapter 17. As shown in Figure 12-1, we drew the design on graph paper, using six horizontal rows consisting of 7 vertical dots to correspond to the 7-dot line spacing we will use. Each of these rows corresponds to one pass of the print head, making it easier to calculate the pin patterns.

Here is the DATA line for the first row. Be sure to enter the line numbers exactly as shown. Type:

```
NEW
800 DATA 0,1,2,4,11,18,36,72,-16,16,
    64,8,64,8,32,16,0,-7,0,0,128
```

The program's handling of the data is fairly straightforward. In most cases, the program merely reads a number and sends it to the printer. To make this happen, add three lines:

```
610 READ N: IF N=128 THEN 650
620 IF N>=$ THEN PRINT CHR$(N;: GOTO 610
650 LPRINT
```

Since 127 is the largest number that can fire seven pins, you will place a 128 at the end of each DATA line as a "stopper" number to signal the end of the print line. Line 610 tests for a stopper. When it reads an N of 128, it passes control to line 650, which produces the necessary line feed. This makes line 650 the only exit from the continuous loop.

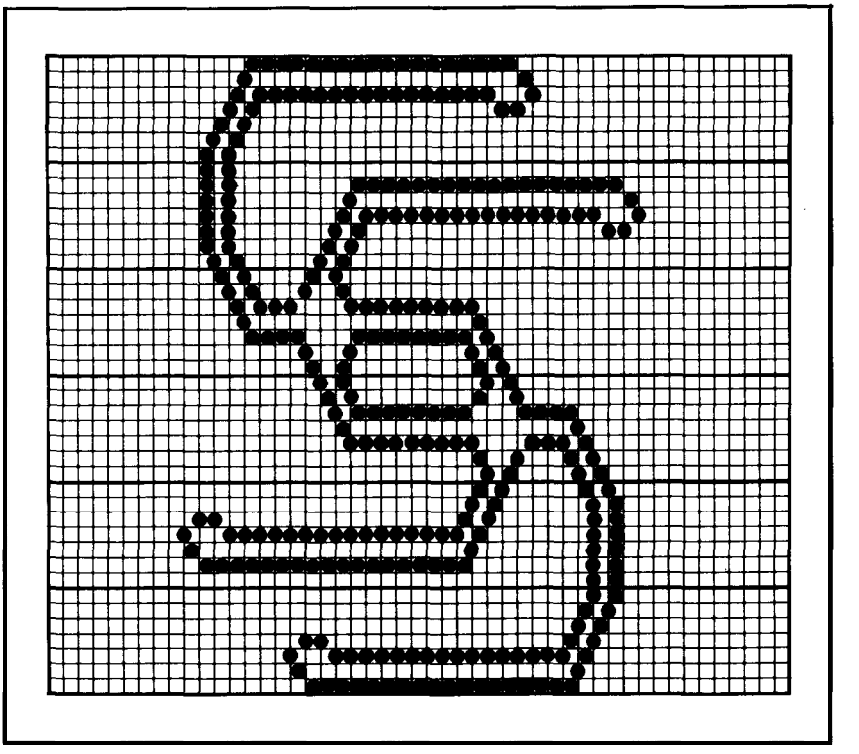


Figure 12-1. STRATA layout

In most cases the program reads a number greater than or equal to zero and sends it to the printer (line 620). Control then returns to line 610, which reads the next number.

If N is negative, the program bypasses the LPRINT ***in*** line 620 and goes on to line 630. Negative numbers in the DATA lines represent repeat factors as they did in the last chapter, but the repeat routine is slightly different. This program repeats numbers in pairs:

```
630 READ P,R: FOR J=l TO -N:
    LPRINT CHR$(P)CHR$(R);: NEXT J
640 GOT0 610
```

For example, when the negative 16 in line 800 is read into N, control passes to line 630. Line 630 reads the next two numbers (16 and 64) into P and R and prints them 16 times. Control then returns to line 610, which reads the next number.

There's only one thing left to do before you can print the first line-enter a Graphics Mode:

```
600 LPRINT CHR$(27)"L"CHR$(60)CHR$(0);
```

Now RUN the program.

That's a good start. To complete the program, add 7-dot line spacing, a loop to process the last five lines of data, and the data lines:

```
100 LPRINT CHR$(27)"1"
590 FOR K=1 TO 6
650 LPRINT: NEXT K: LPRINT CHR$(27)"@": END
799 ' <<< LOGO DATA >>>
810 DATA  0,126,1,0,126,1,-5,0,0,1,2,4,11,
    18,36,-16,8,32,4,32,4,16,8,0,128
820 DATA  0,0,0,64,32,16,72,36,-3,16,4,34,
    65,0,0,65,34,-8,16,4,18,11,4,2,1,0,-9,0,0,128
839 DATA  -8,0,0,64,32,16,72,36,16,-7,
    4,16,36,65,0,0,1,66,36,16,-3,4,16,4,18,11,4,2,1,
    -2,0,0,128
840 DATA  0,32,16,64,8,64,-15,8,32,72,16,32,64,-6,
    0,0,0,127,0,0,127,0,0,0,128
850 DATA  -7,0,0,0,8,4,16,2,16,-15,
    2,8,18,36,72,16,32,64,-2,0,0,128
```

Now RUN the STRATA program.

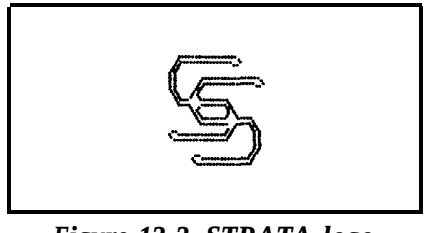


Figure 12-2. STRATA logo

And there you have it . . . Figure 12-2, a genuine logo that you could use on all types of business printouts. If your printout doesn't match this, print a listing and check it against Figure 12-3.

```
l00 LPRINT CHR$(27)"1"
590 FOR K=1 TO 6
600 LPRINT  CHR$(27)"L"CHR$(60)CHR$(0);
610 READ N: IF N=128 THEN 650
620 IF N>=0 THEN LPRINT CHR$(N);: GOT0 610
630 READ P,R: FOR J=1 TO -N:
    LPRINT CHR$(P)CHR$(R);: NEXT J
640 GOT0 610
650 LPRINT: NEXT K: LPRINT CHR$(27)"@": END
799 ' <<< LOGO DATA >>>
800 DATA  0,1,2,4,11,18,36,72,-16,16,
    64,8,64,8,32,16,0,-7,0,0,128
810 DATA  0,126,1,0,126,1,-5,0,0,1,2,4,11,
    18,36,-16,8,32,4,32,4,16,8,0,128
820 DATA  0,0,0,64,32,16,72,36,-3,16,4,34,
    65,0,0,65,34,-8,16,4,18,11,4,2,1,0,-9,0,0,123
830 DATA  -8,0,0,64,32,16,72,36,16,-7,
    4,16,36,65,0,0,1,66,36,16,-3,
    4,16,4,18,11,4,2,1,-2,0,0,128
840 DATA  0,32,16,64,8,64,-15,8,32,72,16,32,64,-6,
    0,0,0,127,0,0,127,0,0,0,128
850 DATA  -7,0,0,0,8,4,16,2,16,-15,
    2,8,18,36,72,16,32,64,-2,0,0,128
```

Figure 12-3. STRATA program

We have you use this logo as part of a larger program in Chapter 17.

Three-Dimensional Program

Using a little imagination and creativity, you can do a lot with dot graphics. Take a look at the three-dimensional design of Figure 12-4, which spells but the name of your favorite printer.

In this section we work through an FX-80 design; if you own an FX-100, do not think that we have forgotten you. As we go along we note which lines will differ for you, and at the close of the section we show you a complete FX-100 program.

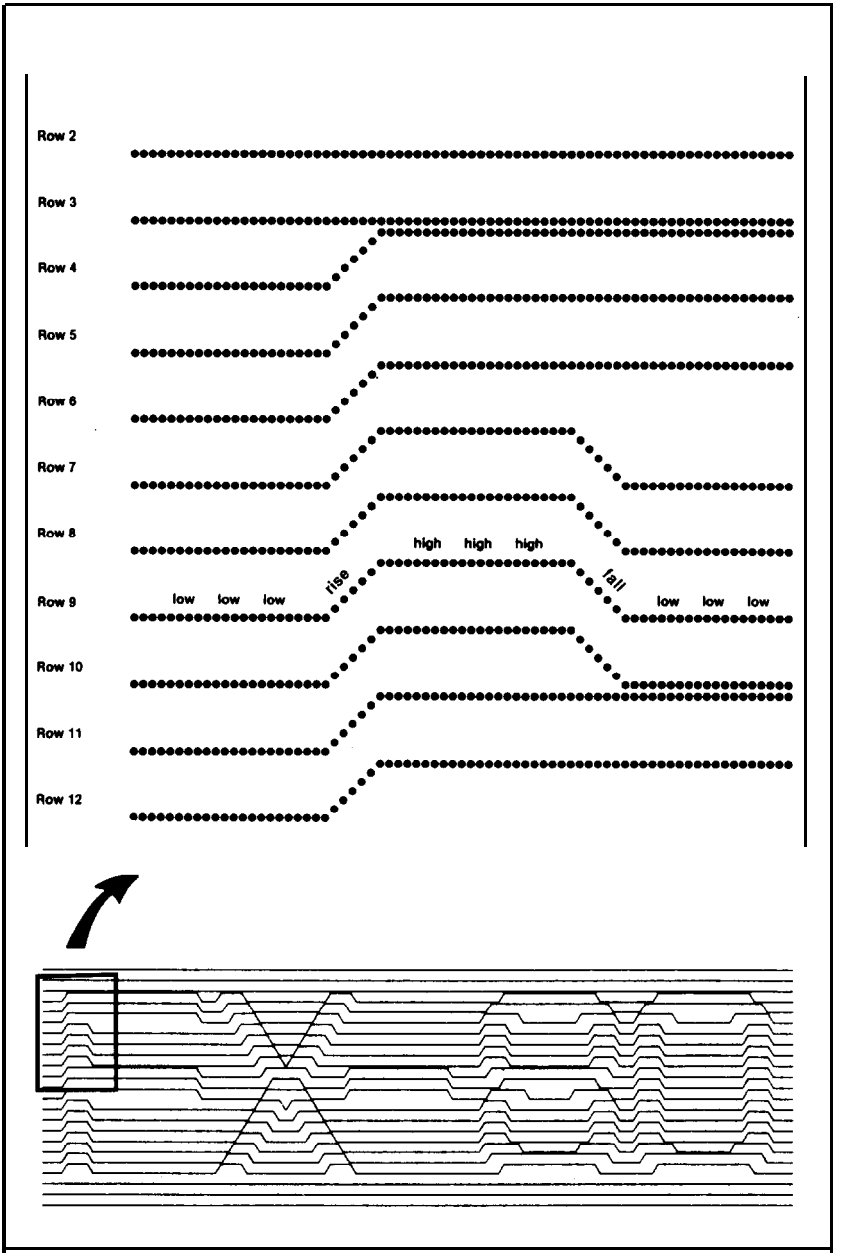


Figure 72-4. Corner of the FX-80 design

A very few pin patterns are needed for this program. In fact, each "pattern" consists of only one pin, making the numbers easy to calculate:

1	for the low pin
64	for the high pin
1, 2, 4, 8, 16, 32, 64	for the diagonal rise
64, 32, 16, 8, 4, 2, 1	for the diagonal fall

As you will see in the next few pages, these pin patterns are coded right into the program. You'll only need to store as data the number of repetitions for the low and high sections.

A close look at Figure 12-4 reveals that most of the lines can be produced by repeating a d-step pattern:

1. Fire the bottom pin (pin 1), repeat L times.
2. Draw a diagonal rise (pins 1 - 64).
3. Fire the top pin (pin 64), repeat H times.
4. Draw a diagonal fall (pins 64 - 1).

This pattern is repeated several times. Printing the figure is mainly a matter of reading the length of the low and high sections, then printing the four-part cycle.

First version of 3D program

We have you enter this program in portions that are easy to discuss as units, so please don't try to RUN it until we give the word.

Might as well start off with the easy stuff. Set the line spacing for 7-pin graphics:

```
NEW
l0 LPRINT CHR$(27)"1"
```

Note: If your system leaves gaps in 7-dot graphics printing, you will prefer to use the 6-2/3-dot line spacing, CHR$(27) "3'CHR$(20).

Next up are the three straight lines at the start of the design. There's no need for anything fancy-just a single dot printed across the page. For that, add these lines (lines 20 and 170 are different for the FX-100):

```
29 G$=CHR$(27)+"L"+CHR$(51)+CHR$(3): GOSUB 160
158 LPRINT CHR$(27)"@": END
160 FOR X=1 TO 3: LPRINT G$;
```

```
170 FOR Y=1 TO 819: LPRINT CHR$(l);: NEXT Y
180 LPRINT: NEXT X: RETURN
```

Now RUN the first trial:

Line 20 stores the graphics entry string in G$. This produces Low-Speed Double-Density dots for 819 columns [51+(3x256) = 819]. Line 170 fires the bottom graphics pin 819 times. The X loop (lines 160 and 180) repeats the routine to print the line three times.

Because you will use this line-printing routine (lines 160 - 180) again to print the bottom three lines of the figure, we have you set it aside as a subroutine. It is called by the GOSUB in line 20 and separated from the rest of the program by the END in line 150.

So far, so good! Now for the rest of the figure. The following lines control the four-step process:

```
80 FOR X=1 TO L: LPRINT CHR$(l);: NEXT X
100 LPRINT CHR$(l)CHR$(2)CHR$(4)CHR$(8)
    CHR$(16)CHR$(32)CHR$(64);
110 FOR X=1 TO H: LPRINT CHR$(64);: NEXT X
120 LPRINT CHR$(64)CHR$(32)CHR$(16)
    CHR$(8)CHR$(4)CHR$(2)CHR$(l);
```

Lines 100 and 120 print the 7-dot rise and fall. The lengths of the low and high sections are stored in DATA statements, then read into the variables L and H. Line 80 prints the bottom pin L times; line 110 prints the top pin H times.

The next step is to add the READ portion of the program:

```
50 READ L,H
60 L=L*7: H=H*7
70 IF L=0 THEN 90
90 IF H<0 THEN LPRINT: GOT0 150
130 GOTO 50
```

Line 50 reads numbers from the data statements in pairs: the first is stored into L, the second into H. L and H are then each multiplied by seven; this extends the width of the figure without increasing the size of the data numbers. This enlargement factor must be the same as the number of dots in the rise and fall or the design will not line up properly.

If L is read as zero, line 70 causes the program to ignore line 80. This enables the printer to print the center portion of the X, where the diagonal fall meets the diagonal rise at a point and no low section is required.

Line 90 serves two purposes. It forces a line feed each time a negative number is read, and it skips the last three steps of the four-step cycle so that each print line can end on a low.

If your computer system requires a WIDTH statement to prevent the printer from issuing a carriage return before the graphics line is complete, add it now:

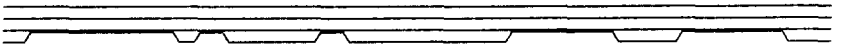
```
7 WIDTH LPRINT 255
```

The format for this statement may be different for your BASIC; see your software documentation.

To print the program with one line of data, add lines 40 and 190 (the latter differs for the FX-100):

```
40 LPRINT G$;
190  DATA  3,20,2,3,12,3,22,14,8,14,6,-1
```

and RUN it.



The negative number at the end of line 190 signals the end of the print line.

The last program changes add a loop to print 17 lines and enter the data statements. (All the data lines differ slightly for the FX-100):

```
30 FOR D=1 TO 17: PRINT "ROW";D
90 IF H<0 THEN LPRINT: GOT0 140
140 NEXT D: GOSUB 160
200 DATA 3,20,3,3,10,3,21,18,4,18,4,-1
210 DATA 3,20,4,3,8,3,21,5,8,5,2,5,8,5,3,-1
220 DATA 3,3,22,3,6,3,22,3,12,3,2,3,12,3,3,-1
230 DATA 3,3,23,3,4,3,23,3,12,3,2,3,12,3,3,-1
240 DATA 3,3,24,3,2,3,24,3,12,3,2,3,12,3,3,-1
250 DATA 3,3,25,3,0,3,25,3,12,3,2,3,12,3,3,-1
260 DATA 3,20,9,6,5,15,5,18,3,3,12,3,3,-1
270 DATA 3,20,10,4,6,15,7,14,5,3,12,3,3,-1
280 DATA 3,20,9,6,5,15,5,5,6,5,3,3,12,3,3,-1
```

```
290  DATA  3,3,25,3,0,3,25,3,12,3,2,3,12,3,3,-1
300  DATA  3,3,24,3,2,3,24,3,12,3,2,3,12,3,3,-1
310  DATA  3,3,23,3,4,3,23,3,12,3,2,3,12,3,3,-1
320  DATA  3,3,22,3,6,3,22,3,12,3,2,3,12,3,3,-1
330  DATA  3,3,21,3,8,3,21,5,8,5,2,5,8,5,3,-1
340  DATA  3,3,20,3,10,3,21,18,4,18,4,-1
350  DATA  3,3,19,3,12,3,22,14,8,14,6,-1
```

Yes, indeed, high-resolution graphics does require a large amount of data. Okay, now RUN the program:

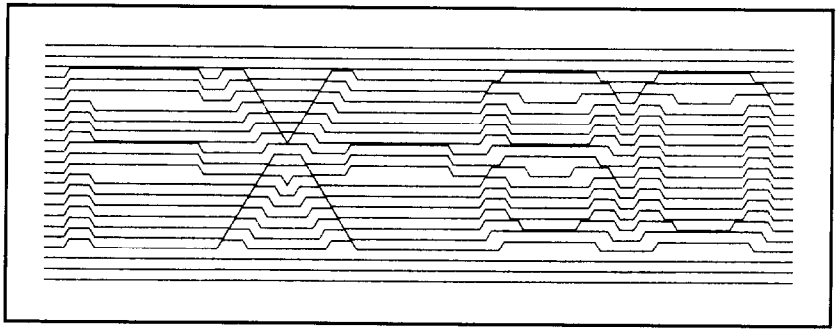


Figure 12-S. FX-80 figure

Success! In your printout (which should look like Figure 12-5), can you see the 3D effect of the letters? The modifications to follow will make the letters easier to read.

Type LLIST to see what you've got up to this point. It should match Figure 12-6. Figure 12-7 shows the printout and Figure 12-8 shows the listing for the FX-100.

```
7 WIDTH LPRINT 255
10 LPRINT CHR$(27)"1"
20 G$=CHR$(27)+"L"+CHR$(51)+CHR$(3): GOSUB 160
30 FOR D=1 TO 17: PRINT "ROW";D
40 LPRINT G$;
50 READ L,H
60 L=L*7: H=H*7
70 IF L=0 THEN 90
80 FOR X=1 TO L: LPRINT CHR$(1);: NEXT X
90 IF H<0 LPRINT: GOT0 140
```

Figure 12-6. Program *for* FX-80 ***figure***

```
100 LPRINT  CHR$(1)CHR$(2)CHR$(4)CHR$(8)CHR$(16)
    CHR$(32)CHR$(64);
110 FOR X=1 TO H: LPRINT CHR$(64);: NEXT X
120 LPRINT  CHR$(64)CHR$(32)CHR$(16)CRR$(8)CHR$(4)
    CHR$(2)CHR$(1);
130 GOTO 50
140 NEXT D: GOSUB 160
150 LPRINT CHR$(27)"@": END
160 FOR X=1 TO 3: LPRINT G$;
170 FOR Y=l TO 819: LPRINT CHR$(1);: NEXT Y
180 LPRINT: NEXT X: RETURN
190 DATA  3,20,2,3,12,3,22,14,8,14,6,-1
200 DATA  3,20,3,3,10,3,21,18,4,18,4,-1
210 DATA 3,20,4,3,8,3,21,5 ,8,5,2,5,8,5,3,-1
220 DATA  3,3,22,3,6,3,22,3,12,3,2,3,12,3,3,-1
230 DATA  3,3,23,3,4,3,23,3,12,3,2,3,12,3,3,-1
240 DATA  3,3,24,3,2,3,24,3,12,3,2,3,12,3,3,-1
250 DATA  3,3,25,3,0,3,25,3,12,3,2,3,12,3,3,-1
260 DATA  3,20,9,6,5,15,5,18,3,3,12,3,3,-1
270 DATA  3,20,10,4,6,15,7,14,5,3,12,3,3,-1
280 DATA  3,20,9,6,5,15,5,5,6,5,3,3,12,3,3,-1
290 DATA  3,3,25,3,0,3,25,3,12,3,2,3,12,3,3,-1
300 DATA  3,3,24,3,2,3,24,3,12,3,2,3,12,3,3,-1
310 DATA  3,3,23,3,4,3,23,3,12,3,2,3,12,3,3,-1
320 DATA  3,3,22,3,6,3,22,3,12,3,2,3,12,3,3,-1
330 DATA  3,3,21,3,8,3,21,5,8,5,2,5,8,5,3,-1
340 DATA  3,3,20,3,10,3,21,18,4,18,4,-1
350 DATA  3,3,19,3,12,3,22,14,8,14,6,-1
```

Figure 72-6. Program for FX-80 figure (concluded)

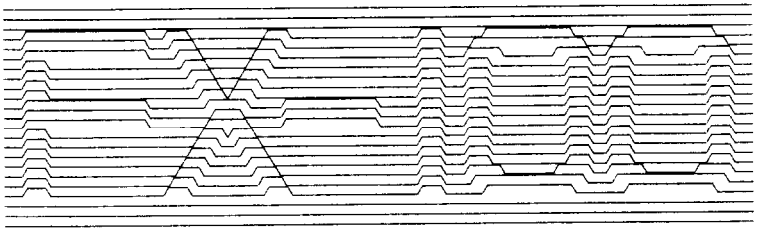


Figure 12-7. FX-100 figure

```
7 WIDTH LPRINT 255
10 LPRINT CHR$(27)"1"
20 G$=CHR$(27)+"L"+CHR$(l2l)+CHR$(3): GOSUB 160
30 FOR D=1 TO 17.: PRINT "ROW";D
40 LPRINT G$;
50 READ L,H
60 L=L*7: H=H*7
70 IF L=0 THEN 90
80 FOR X=1 TO L: LPRINT CHR$(1);: NEXT X
90 IF H<0 THEN LPRINT: GOT0 140
100 LPRINT CHR$(1)CHR$(2)CHR$(b)CHR$(8)CHR$(16)
    CHR$(32)CHR$(64);
110 FOR X=1 TO H: LPRINT CHR$(64);: NEXT X
120 LPRINT CHR$(64)CHR$(32)CHR$(16)CHR$(8)
    CHR$(4)CHR$(2)CHR$(1);
130 GOTO 50
140 NEXT D: GOSUB 160
150 LPRINT CHR$(27)"@": END
160 FOR X=1 TO 3:LPRINT G$;
170 FOR Y=1 TO 889: LPRINT CHR$(1);: NEXT Y
180 LPRINT: NEXT X: RETURN
190   DATA 3,20,2,3,12,3,21,3,6,14,8,14,6,-1
200   DATA 3,20,3,3,10,3,22,3,4,18,4,18,4,-1
210   DATA 3,20,4,3,8,3,23,3,3,5,8,5,2,5,8,5,3,-1
220   DATA 3,3,22,3,6,3,24,3,3,3,12,3,2,3,12,3,3,-1
230   DATA 3,3,23,3,4,3,25,3,3,3,12,3,2,3,12,3,3,-1
240   DATA 3,3,24,3,2,3,26,3,3,3,12,3,2,3,12,3,3,-1
250   DATA 3,3,25,3,0,3,27,3,3,3,12,3,2,3,12,3,3,-1
2.60  DATA 3,20,9,6,5,15,6,3,3,3,12,3,2,3,12,3,3,-1
270   DATA 3,20,10,4,6,15,6,3,3,3,12,3,2,3,12,3,3,-1
280   DATA 3,20,9,6,5,15,6,3,3,3,12,3,2,3,12,3,3,-1
290   DATA 3,3,25,3,0,3,27,3,3,3,12,3,2,3,12,3,3,-1
300   DATA 3,3,24,3,2,3,26,3,3,3,12,3,2,3,12,3,3,-1
310   DATA 3,3,23,3,4,3,25,3,3,3,12,3,2,3,12,3,3,-1
320   DATA 3,3,22,3,6,3,24,3,3,3,12,3,2,3,12,3,3,-1
330   DATA 3,3,21,3,8,3,23,3,5,8,5,2,5,8,5,3,-1
340 DATA 3,3,20,3,10,3,22,3,4,18,4,18,4,-1
350 DATA 3,3,19,3,12,3,21,3,6,14,8,14,6,-1
```

Figure 12-8. Program for FX-100 figure

Other versions

Now that you've typed in all that data, it's time to have some fun. You can change a few of the pin patterns to make a dramatic effect on results. These changes work the same on either model's figure.

These ***changes*** fill in the diagonals as illustrated in Figure 12-9:

```
100 LPRINT CHR$(1)CHR$(3)CHR$(7)CHR$(15)
    CHR$(31)CHR$(63)CHR$(127);
120 LPRINT CHR$(127)CHR$(63)CHR$(31)CHR$(15)
    CHR$(7)CHR$(3)CHR$(1);
```

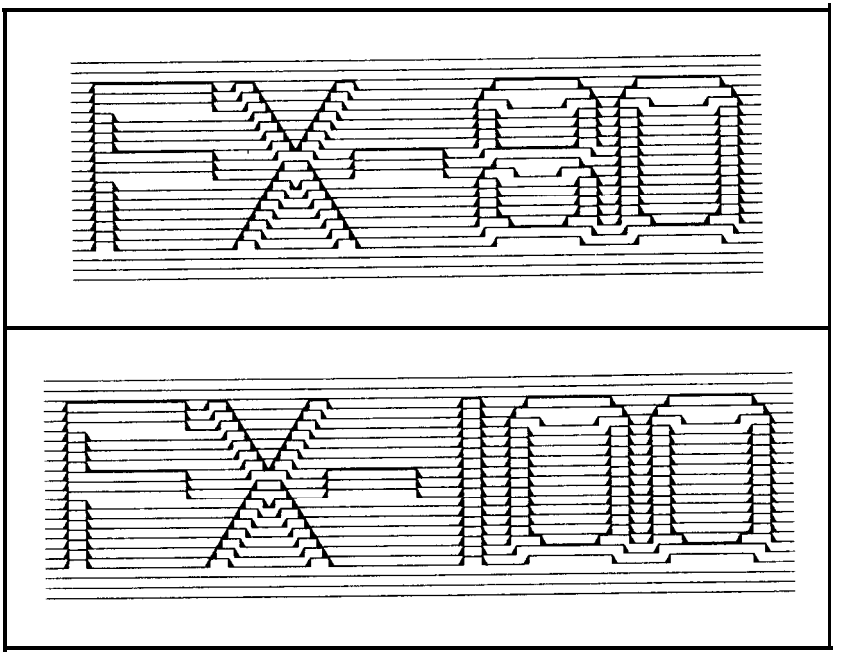


Figure 12-9. ***More distinct version***

And one additional change fills in the entire text (Figure 12-10):

```
110 FOR X=1 TO H: LPRINT CHR$(127);: NEXT X
```

Can you vary the program to produce a complete black/white reverse like the one in Figure 12-1l?

Summary

We did not introduce any commands in this chapter. Instead, we demonstrated two ways of programming dot graphics. For the first example we used DATA statements to store pin patterns and repetition factors. For the second example we stored the pin patterns as constants and used DATA statements only to store repetition factors.

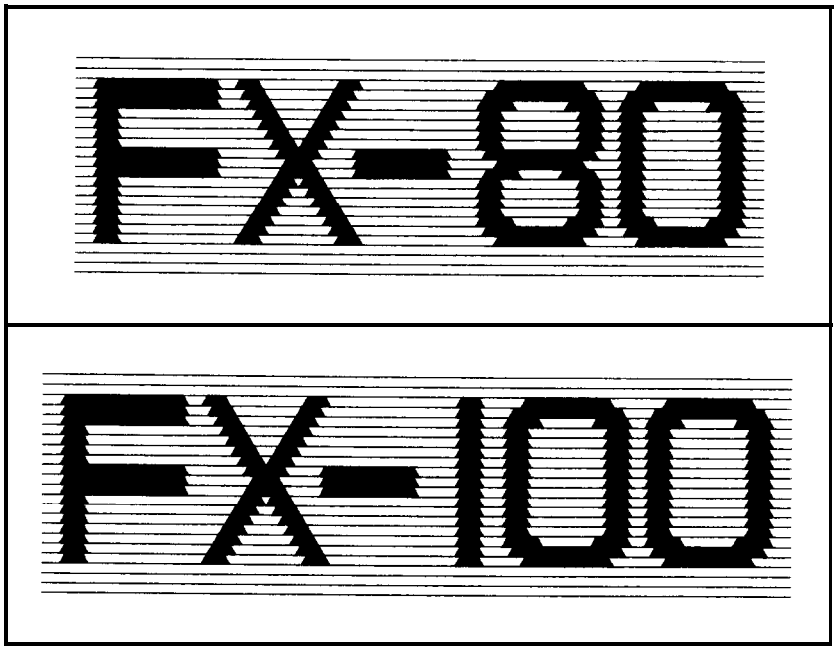


Figure 12-10. Most distinct version

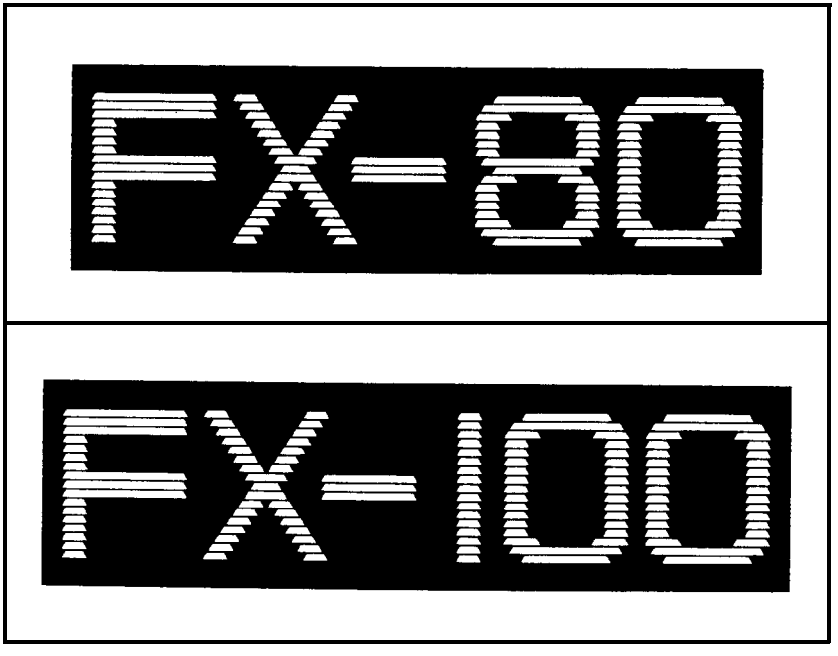


Figure 12-11. Reversed version

Chapter 13
Plotter Graphics

As you work with dot graphics, you may run into printer limitations because dot-matrix printers are designed primarily for fast printing of text. The FX, however, can also print high-resolution graphics, as you saw in the STRATA program. But the side-to-side motion of a dot-matrix printer makes it virtually impossible to place the print head in the middle of a page and trace out a lazy spiral or even a circle.

Does that mean the FX can't create the same type of figures that a plotter does? Certainly not. You just have to approach the problem a little differently than you would in working with a plotter. In fact, with a little advance planning, you can use the dot graphics modes to simulate the activity of a full-fledged plotter.

The secret to bringing out the plotter in your FX is to apply the capabilities of your computer system. You can use its memory as a sketch pad. With mathematical formulas, you can design any sort of pattern. Once you've got your picture complete in memory, you can send it line by line to the printer. When you use this approach, the printer doesn't have to act like a plotter to draw like one.

The figures printed in this chapter show how easy it is to simulate plotter graphics with the FX. You start by using the computer's memory as a sketch pad for the plotting. To do this, you set up a correspondence between a pattern of dots on paper and a set of values that you arrange in the computer's memory (Figure 13-1).

Arrays

The structure in computer memory in which you store your pattern of dots is called an array. Think of an array as an ordered set of cubby

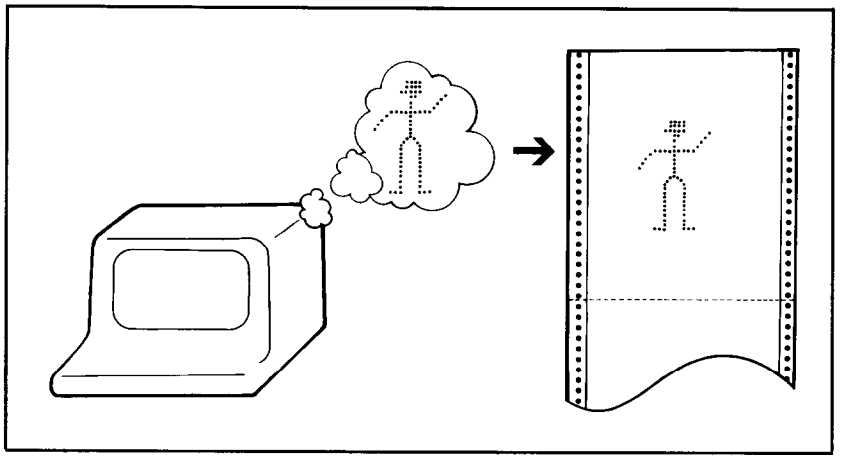


Figure 13-1. Computer memory as sketch pad

holes or cells arranged in rows and columns, as Post Office boxes are. Each cell of the array corresponds to a dot position on the paper (Figure 13-2).

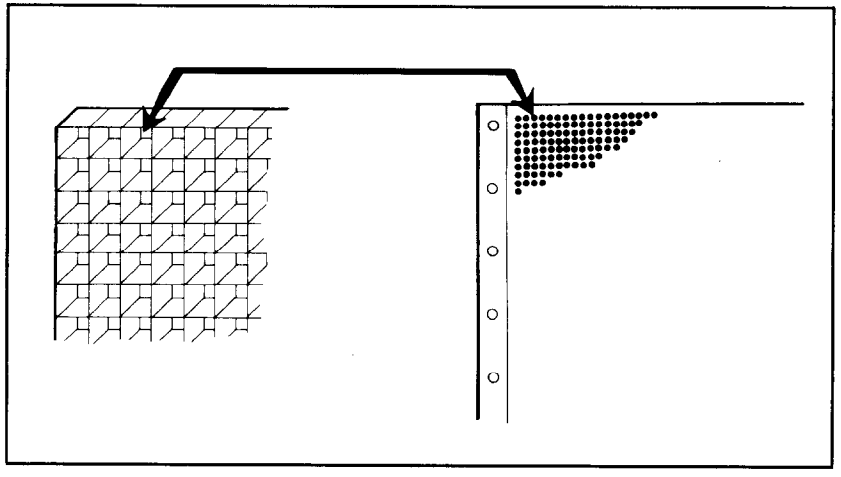


Figure 13-2. Array in memory and on paper

Although the cells in a numeric array can hold nearly any numeric value, you use only the binary numbers (0 and 1) for this graphics program. Figure 13-3 demonstrates using a 1 to represent a dot and a 0 to represent no dot.

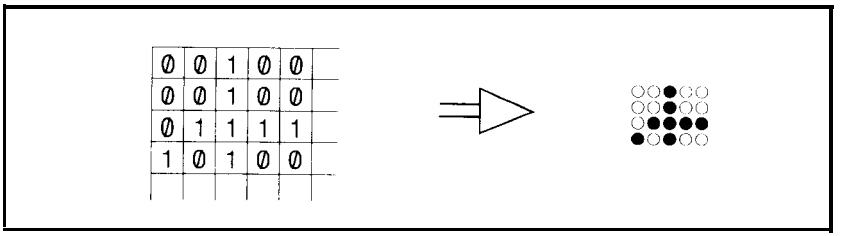


Figure* 13-3. *Ones and zeros become dots and blanks

Why all this fuss and stew about arrays? We want to show you that the FX can simulate a plotter. And once the correspondence between array cells and dot positions is firmly established, you can easily plot in any direction.

Let's look at the way each cell is named. The cells are arranged in rows and columns, so each cell can be easily pinpointed by its row and column position.

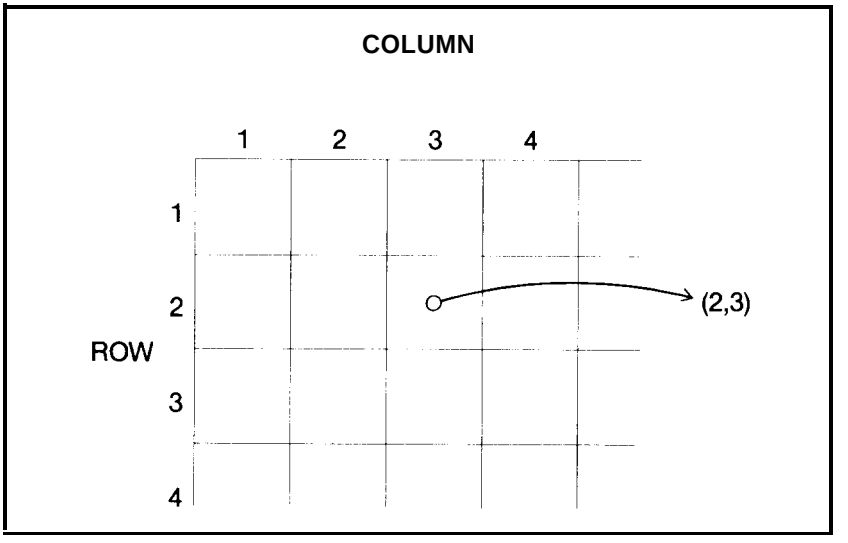


Figure* 13-4. *Labelled cell

The labelled cell of Figure **13-4** sits at the intersection of row **2** and column **3,** so you can label it by its address: cell **(2,3).** In BASIC, you give the entire array a name, then append the address to the name. Thus, if you name the array of Figure **13-4** array A, the cell name is A(2,3)

DIMension and arrays

Most BASICs allow you to use up to 10 rows and 10 columns in an array without any special preparation of the computer's memory. Since arrays use up lots of memory, you must inform the system if you intend to use a larger array. In BASIC, this is done with the DIMension statement, which is contained in the first line of the next program. Enter:

```
NEW
10 DEFINT A: N=21: DIM A(N,N)
```

If your system rebels at line 10, use:

```
10 DEFINT A: N=21: DIM A(21,21)
```

The DIM statement in line 10 reserves enough memory for 21 rows and 21 columns of numbers. That gives you a total of 441 cells. Each cell takes up 2, 4, or 8 bytes, depending on the precision of the variables you use. The DEFINT restricts all variables that start with the letter A to be of the integer type (2 bytes); this definition saves memory.

Filling arrays

Most computer systems set to 0 all numeric variables, including cells of arrays, at the beginning of a program. If your system does not automatically do this, use the following lines to perform the same function:

```
15 FOR X=0 TO N: FOR Y=0 TO N
17 A(X,Y)=g: NEXT Y: NEXT X
```

To plot a figure in memory after all the cells are set to zero, you simply deposit ones in the correct positions using LET statements. For example, the statement A(2,3)=l will place a one in location (2,3).

Suppose you want to plot a circle of radius 10 in the 21 row by 21-column array-definitely a job for a plotter! You can, however, use the standard distance formula (as in Figure 13-5) to calculate the distance from the center cell (11,ll) to each of the surrounding cells. If this distance is equal to 10, the cell content is changed to one; otherwise, the cell value remains zero.

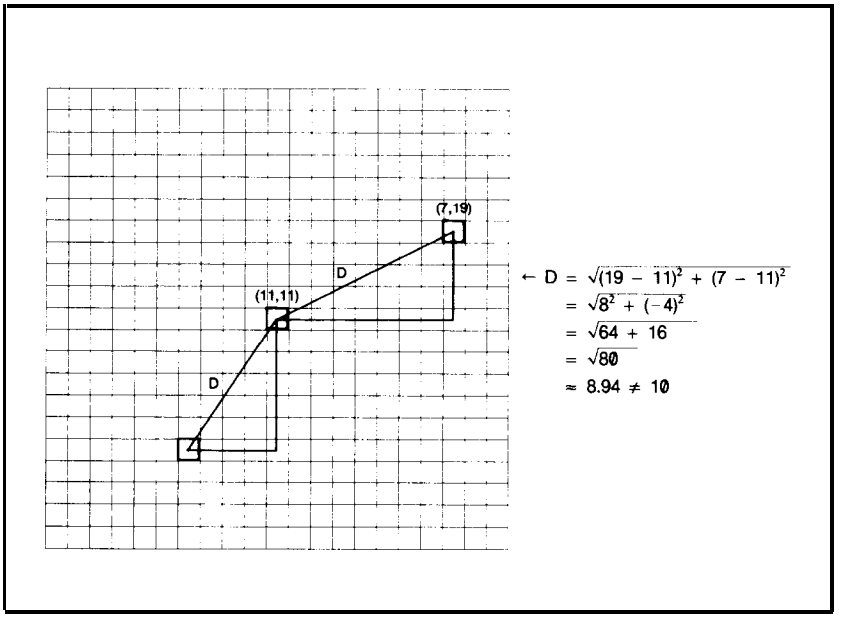


Figure 13-5. Plotting a circle

Circle Plotting

You can have your program examine the cells of an array in any order; the following program scans them row by row, using two loops:

```
20 FOR R=1 TO N: FOR C=1 TO N
```

At each cell, line 30 calculates the cell's distance from a center point by using the distance formula:

```
30 D=SQR((R-11)^2+(C-11^2)
```

Next the program compares this distance with a number (10) that specifies the radius of a circle. Notice that this formula commonly results in a number that includes a decimal fraction, e.g., *8.94.*

If the distance for the current cell A(R,C) is close to 10, you set its contents equal to one: otherwise, you leave it alone.

```
40 IF INT(D+.S)=10 THEN A(R,C)=1
```

Line *40* sets any cell whose distance is between *9.5* and 10.5 equal to one.

The final step to plotting a circle in an array is to close the loops and display the contents of the array. Add these three lines to your program:

```
50 LPRENT A(R,C);: NEXT C: LPRINT
60 PRINT "ROW";R: NEXT R
170 LPRINT CHR$(27)"@": END
```

and RUN it.

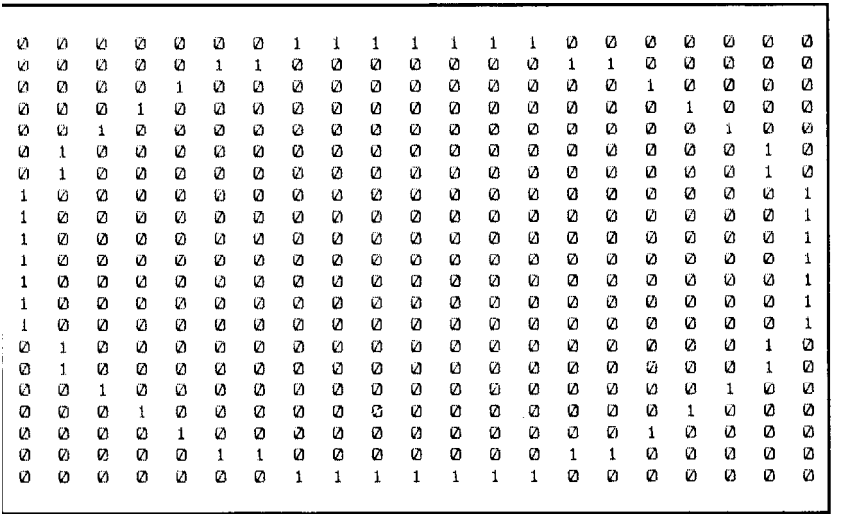


Figure 13-6. Displaying an array

The printout shows, in terms of ones and zeros, the results of your planning and your program's plotting.

Ones become dots

Next you need to translate the contents of the array to dots on the paper. First, modify line 50 so that it no longer prints out the contents of the array:

```
50 NEXT C
```

Then fill the usual graphics prescription for 7-dot line spacing:

```
70 LPRINT CHR$(27)"1"CHR$(7);
```

The CHR$(7) sounds the beeper to tell you when the array is full.

The next line loads the beginning (B), ending (E), and step (S) values for the loop that will read and print the array.

```
100 B=1: E=N-6: S=1
```

We have you use variables here so that you can change them later. That way you can make your program read the array in a number of directions.

Using 7 pins of the print head on each pass, the program will take 3 passes to print a 21-row array. If you change the array size, remember to use a multiple of 7.

Continue by adding these five lines:

```
110 FOR P=B TO E STEP 7*S
120 PRINT "LOADING ROWS";P;"TO";P+6*S
130 LPRINT  CHR$(27)"*"CHR$(0)CHR$(N)CHR$(0);
150 FOR C=1 TO N: GOSUB 180: NEXT C
160 LPRINT: NEXT P
```

Line 110 loads the array rows from beginning (B) to end (E) in sets of seven.

Line 120 prints to the screen an update of the computer's progress.

Line 130 enters Graphics Mode and reserves N columns for graphics (N is the width of the array).

Line 150 accesses the subroutine (to be added next) that calculates the pin patterns for each column.

And line 160 closes the loop for each pass (P) of the print head.

Pin firing sequences

The last step before printing the figure is to convert those ones and zeros to pin firing sequences. Add this subroutine to your program:

```
170 LPRINT CHR$(27)"@": END
180 F=0: FOR R=P TO P+6*S STEP S
190 IF A(R,C)=1 THEN F=F+2^ABS(P+6*S-R)
200 NEXT R
220 LPRINT CHR$(F);: RETURN
```

Understanding the subroutine is easy if you take it one step at a time. This subroutine calculates the pin firing pattern (F) for each column of seven dots. It examines the array vertically, one cell at a time. When it

encounters a one, it adds the appropriate power of two to F (line 190). The exponent is the difference between the current row (R) and the last row in this pass of the print head (P+6*S). Line 220 sends F to the printer as a graphics pin pattern.

```
10 DEFINT A: N=21: DIM A(N,N)
20 FOR R=1 TO N: FOR C=1 TO N
30 D=SQR((R-11)^2 + (C-11)^2)
40 IF INT(D+.5)=10 THEN A(R,C)=1
50 NEXT C
60 PRINT "ROW";R: NEXT R
70 LPRINT CHR$(27)"1"CHR$(7);
100 B=1: E=N-6: S=1
110 FOR P=B TO E STEP 7*S
120 PRINT "LOADING ROWS";P;"TO";P+6*S
130 LPRINT CHR$(27)"*"CHR$(0)CH$(N)CHR$(0);
150 FOR C=1 TO N: GOSUB 180: NEXT C
160 LPRINT: NEXT P
170 LPRINT CHR$(27)"@": END
180 F=0: FOR R=P TO P+6*S STEP S
190 IF A(R,C)=1 THEN F=F+2^ABS(P+6*S-R)
200 NEXT R
220 LPRINT CHR$(F);: RETURN
```

Check your listing against the program above to make sure you have it all. If you do, type RUN.

The array looks like this printout when it's translated into dots.

If all went well, skip to the "Higher resolution" section, below. If not, take the time to find out a code solution.

Code solutions

If your printout doesn't look much like ours, it's likely the problem involves the codes from nine to 13. Remember that the program determined the dot patterns to send to the printer by the figure stored in the array. The cleanest way to get the figure to print correctly is to either POKE the codes directly or use a printer driver that allows the codes to pass through as sent. See Appendix F.

If neither is possible, there is a third way. You can avoid these codes during printing without doing too much damage to the figure. The test

below picks off any potential problem codes and changes them to less dangerous numbers.

```
210 IF F>8 AND F<14 THEN F=F-5
```

This line takes any number between 8 and 14 and subtracts 5 from it, putting it out of the trouble range. Adjust this test to fit your system.

You may see another problem with the figure. The standard 7-dot line spacing may be off just enough to add a slight gap every seven rows. An easy fix for this is to adjust the line spacing as needed with the CHR$(27)"3" command. This gives you the ability to make adjustments as fine as one-third of a dot. For example, in this program you can set a 6-2/3-dot line spacing by changing line 70 to:

```
70 LPRINT CHR$(27)"3"CHR$(20)CHR$(7);
```

Higher resolution

If everything did go according to plan, your printed figure resembles a circle of radius 10 dots, but the resolution isn't all that you might hope for. For one thing, the individual dots are clearly visible. To make a more continuous figure, you need a graphics mode of higher density. To get it, change line 130 to:

```
130 LPRINT CHR$(27)"*"CHR$(1)CHR$(N)CHR$(0);
```

This does give a more satisfactory density, but now the program distorts the circle. Mathematically inclined folks can adjust for this distortion by creating an ellipse in the array (the horizontal compression creates a circle). If you pursue this course, keep in mind that the FX has several graphics densities available.

With figures this small, it is difficult to obtain a smooth curve. The solution is to draw larger circles.

Unfortunately, larger arrays gobble up memory. For example, an array wide enough to stretch clear across an g-inch page in Single-Density would contain over 200,000 cells (480 x 480). Considering that each cell takes up at least two bytes, there is not much hope of tucking the entire array into memory all at once.

So how can you squeeze more print out of your arrays? For non-symmetric designs, you are pretty well stuck with the memory limitation of two bytes per cell unless you are willing to resort to more

drastic measures. One such measure would be to let each bit of the numbers stored in the array cells represent one graphics dot. This would increase the storage ability but tremendously complicate the programming.

For symmetric designs such as the circle, you can use a different measure. Take advantage of the symmetry to increase your output four-fold without increasing the size of the array one iota. How? By using the array to plot one-fourth of the circle in memory, then modifying the array three times to generate the remaining three parts (Figure 13-6).

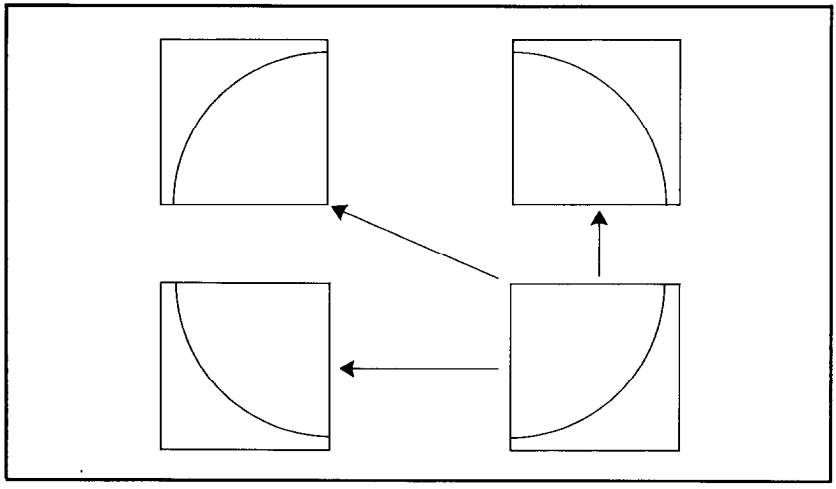


Figure 13-7. Divide and conquer

Try this out with the current program. Here are the changes you need to make to your current program in order to plot the lower-right corner :

```
30 D=SQR(R^2+C^2)
40 IF INT(D+.5)=20 THEN A(R,C)=1
130 LPRINT CHR$(27)"*"CHR$(5)CHR$(N)CHR$(0);
```

By moving the center of the circle from `(11,11)` to (0,0) and increasing the radius from `10` to ***20,*** you enlarge the figure.

Reflections

Once the desired image is stored in the array you can rotate and reflect it in several different directions. It's all done with mirrors; at least, it looks like mirror reflections when you are done. You create the mirror effect by reading the array in different directions.

Currently your program reads the array from left to right, seven rows at a time, but it is just as easy to read it in the reverse order. Add this line:

```
140 FOR C=N TO 1 STEP -1: GOSUB 180: NEXT C
```

and double the graphics width setting in line 130:

```
130 LPRINT- CHR$(27)"*"CHR$(5)CHR$(2*N)CHR$(0);
```

Now RUN it.

The left half of the figure mirrors the right half. With a few more changes, the program can read the array upside down and double the output again. Add these two lines:

```
80 B=N: E=7: S=-1
90 FOR Z=1 TO 2
```

and change these three:

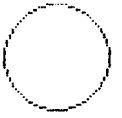
```
60 PRINT "T MINUS";N-R: NEXT R
100 IF Z=2 THEN B=1: E=N-6: S=1
160 LPRINT: NEXT P: NEXT Z
```

Now lines 80, 90, and 100 change the order in which the array is read. First (Z = 1) it's read upside down, then (Z=2) right side up, as before. This is the full listing:

```
10 DEFINT A: N=21: DIM A(N,N)
20 FOR R=1 TO N: FOR C=1 TO N
30 D=SQR(R^2+C^2)
40 IF INT(D+.5)=20 THEN A(R,C)=1
50 NEXT C
60 PRINT "T MINUS";N-R: NEXT R
70 LPRINT CHR$(27)"1"CHR$(7);
80 B=N: E=7: S=-1
90 FOR Z=1 TO 2
```

```
100 IF Z=2 THEN B=1: E=N-6: S=1
110 FOR P=B TO E STEP 7*S
120 PRINT "LOADING ROWS";P;"TO"; P+6*S
130 LPRINT CHR$(27) "*"CHR$(5)CHR$(2*N)CHR$(0);
140 FOR C=N TO 1 STEP -1: GOSUB 180: NEXT C
150 FOR C=1 TO N: GOSUB 180: NEXT C
160 LPRINT: NEXT P: NEXT Z
170 LPRINT CHR$(27)"@": END
180 F=0: FOR R=P TO P+6*S STEP S
190 IF A(R,C)=1 THEN F=F+2^ABS(P+6*S-R)
200 NEXT R
220 LPRINT CHR$(F);: RETURN
```

Go ahead and RUN it to see how it works.



There are two important points here: 1) instead of tracing the circle like a plotter, the program gathers the pattern in the computer's memory, then prints it line by line; 2) the program takes advantage of symmetry to print a figure four times the size of the original array.

Exploding galaxy

With a few more program changes, you can turn this mundane circle into a design for an exploding galaxy. First change the size so that you can see the full impact of the figure (note that 105 is a multiple of seven):

```
10 DEFINT A: N=105: DIM A(N,N)
```

Yes, that 105 means this will take even longer to print out than the circle did, but that's the price you pay for largeness.

If your computer system requires a WIDTH statement to prevent the printer from issuing a carriage return before the graphics line is complete, add it now:

```
7 WIDTH LPRINT 255
```

The format for this statement may be different for your BASIC; see your software documentation.

Next modify the distance formula slightly. Type:

```
30 D=SQR(R^2+C^2)/N: D=D*D
```

This adjustment makes it easier to compare the distance value with the value of the RND function (line 40, below).

Once the computer knows the distance of each cell from the upper-left corner, it can use the following test to determine which cells receive 1s and which cells continue to contain 0s.

```
40 IF D>RND(9) THEN A(R,C)=1
```

Line 40 compares the modified distance (D) of each cell to a random number between 0 and 1. If D is greater than the random number, a 1 goes in that cell.

Note: Use your computer system's version of RND(9) in line 40 [some systems use RND(X) or just RND].

By using a random number (line 40), you add a measure of uncertainty to the placement of the dots. Cells close to the upper-left comer of the square array have a high probability of containing a zero, while those far away have a high probability of containing a one. Since the program reads the array four times (normal, reversed, upside down, and upside down reversed), the upper-left comer of the array corresponds to the center of the large figure that the program will produce and the cells farther away from that point correspond to the comers of that figure. You will see the results when you run the program. Also, the use of randomness can yield a different pattern with each run of the program, but you may have to use a different number in the RND statement for each run, depending upon your version of BASIC.

Big bang

Are you ready to see what the program does? Change two lines:

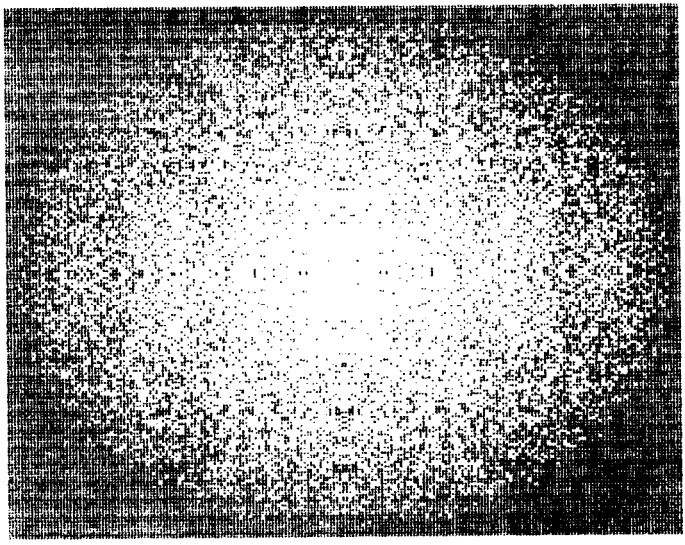
```
60 PRINT "COUNTDOWN TO BIG BANG: T MINUS";
   N-R: NEXT R
130  LPRINT  CHR$(27)"*"CHR$(0)CHR$(2*N)CHR$(0);
```

You may want to check your program against the full listing:

```
7 WIDTH LPRINT 255
10 DEFINT A: N=105: DIM A(N,N)
20 FOR R=1 TO N: FOR C=1 TO N
30 D=SQR(R^2+C^2)/N: D=D*D
40 IF D>RND(9) THEN A(R,C)=1
50 NEXT C
60 PRINT "COUNTDOWN TO BIG BANG: T MINUS"; N-R:
   NEXT R
```

```
70 LPRINT CHR$(27)"3"CHR$(20);CHR$(7);
80 B=N: E=7: S=-1
90 FOR Z=1 TO 2
100 IF Z=2 THEN B=1: E=N-6: S=1
110 FOR P=B TO E STEP 7*S
120 PRINT "LOADING ROWS";P;"TO"; P+6*S
130 LPRINT CHR$(27)"*"CHR$(0)CHR$(2*N)CHR$(0);
140 FOR C=N TO 1 STEP -1: GOSUB 180: NEXT C
150 FOR C=1 TO N: GOSUB 180: NEXT C
160 LPRINT: NEXT P: NEXT Z
170 LPRINT CHR$(27)"@": END
180 F=0: FOR R=P TO P+6*S STEP S
190 IF A(R,C)=1 THEN F=F+2^ABS(P+6*S-R)
200 NEXT R
210 IF F>8 AND F<14 THEN F=F-5
220 LPRINT CHR$(F);: RETURN
```

O.K., now RUN the program (you can have a few cups of coffee while you're waiting).



In the printout can you see white stars against the blackness of outer space?

The design above demonstrates that symmetry can increase the size of a complex figure produced by a two-dimensional array. The only problem is that the array for this figure uses over 20,000 bytes, which

is nearly all of the available memory on many personal computers. You are, therefore, not able to print significantly larger figures of this type with such computers.

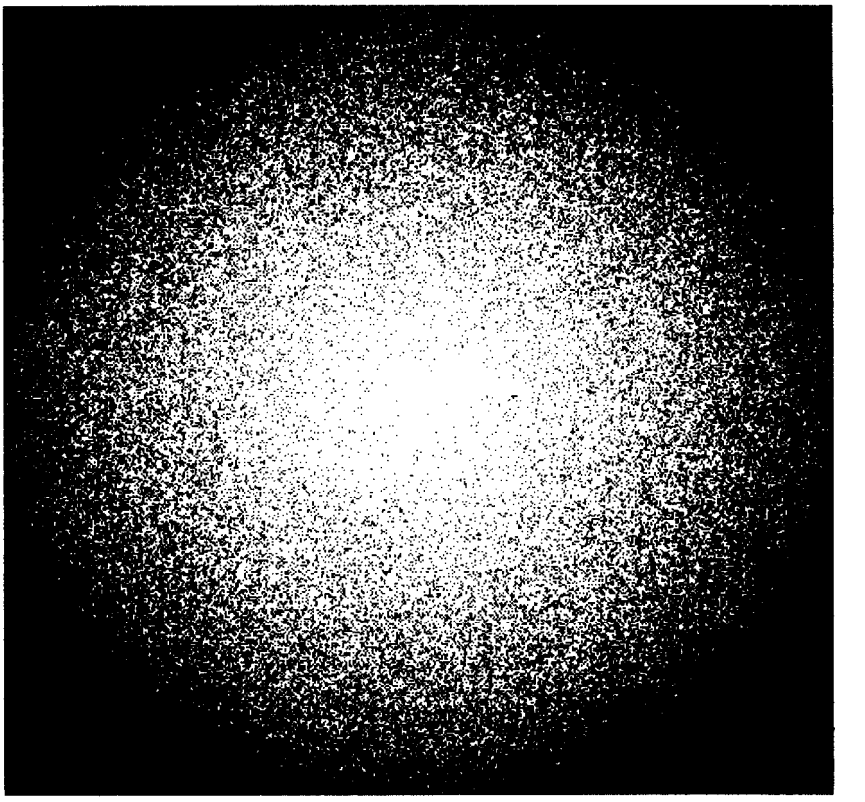
If you like the effect that is produced by this use of random numbers but would like a bigger printout without using more memory, there is a solution.

The alternate method used in the program listed below does not use symmetry and uses very little of the computer's memory because it does not store data in an array. It is, however, not a perfect solution. As you might expect, there is a trade-off. Once you enter this program and type RUN, you must wait an agonizingly long time before it is finished.

The program prints a figure that is over 36 square inches in area, but it takes seven to twelve hours to run. You will not, however, have to wait that long to see if the program is working correctly. After the second row starts printing (within 20 or 30 minutes), you can compare your partial printout with the figure on the next page to see whether or not your program is on the right track.

We don't want to sound less than enthusiastic; we just want you to understand the consequences of using less memory.

```
NEW
10 N=476: M=INT((N+1)/2): WIDTH LPRINT 255
20 N2=INT(N/256): N1=N-256*N2
30 LPRINT CHR$(27)"3"CHR$(20)CHR$(7);
40 FOR P=1 TO N-6 STEP 7
50 PRINT "PRINTING ROWS";P;"TO";P+6
60 LPRINT CHR$(27)"*"CHR$(5)CHR$(N1)CHR$(N2);
70 FOR C=1 TO N
80 F=0: PRINT C;
90 FOR R=P TO P+6
100 D=((R-M)^2+(C-M)^2)/M^2
110 IF D>RND(8) THEN F=F+2^ABS(P+6-R)
115 IF F=9 THEN F=10
120 NEXT R: LPRINT CHR$(F);
130 NEXT C: LPRINT: NEXT P
140 LPRINT CHR$(27)"@"
```



By changing the value of N to different multiples of seven, you can generate this pattern in different sizes. Just be prepared to let your computer cook for several hours.

Summary

We used this chapter to demonstrate the way you can use your FX as though it were a plotter. You can also use your computer system to design a symmetric pattern, applying mathematical principles to minimize the amount of data needed, and then store the data in an array. When you print the pattern, your system sends the data to the FX one line at a time.

Chapter 14
Symmetrical Graphics Patterns

In this chapter we continue to explore the generation of graphics patterns in memory. As in the last chapter, you will use ones and zeros in an array to generate pin patterns, but this time you will save memory by using a one-dimensional array to print a two-dimensional figure. You will construct one long program in which an array containing less than 300 elements will produce a pattern made up of many thousands of dots. Because of the length of the program we will only occasionally ask you to run it.

Begin by defining variables:

```
NEW
10 DIM A(480): X=1: C=0
20 MAX=5: MIN=1: RE=4: N=0
```

For easy reference, Table 14-1 lists, in alphabetical order, the variables you will use for this program. The array A, which is DIMensioned in line 10, will store the pattern. Program loops will use the variables in line 20 to control the size and shape of the figure. You can change these values later to create your own variations on the pattern.

Here are the loops:

```
30 FOR J=1 TO RE
40 N=N+1
50    GOSUB 300
60    IF N<MAX THEN 40
70    N=N-1
80    GOSUB 300
```

Table 14-1. Variables for SYMMETRY

Variable	Purpose
A	Array
C	Counter of array elements
DOT	Counter of dots; used to calculate P
H	Highest number used in calculating P
J	Loop counter
K	Loop counter
L	Loop counter
LAST	Last pass of the print head
MAX	Maximum number for the pattern
MIN	Minimum number for the pattern
N	Number of pins in the current pattern
N1	Length of the graphics line
N2	Length of the graphics line
P	Pin firing pattern
P0	Reverse pattern of P
PASS	Number of the current pass
R	Remainder
RE	Number of repeats of the pattern
X	0 or 1 to fill the array

```
90    IF N>MIN THEN 70
100 NEXT J: PRINT
```

The J loop will Repeat four times (RE = 4). It has two subloops, each of which depends on the value of N. Each time through the first loop (lines 40 to 60), N increases by one-to the value of MAX. Each time through the second loop (lines 70 to 90), N decreases by one-to the value of MIN. For each value of N, the program calls subroutine 300, and each time it is called, this subroutine adds more ones and zeros into the array.

Enter the program lines for the subroutine by typing:

```
290 LPRINT CHR$(27)"2": END
300 FOR K=0 TO MAX-N
310    FOR L=1 TO N
320       C=C+1: A(C)=X
330    NEXT L: X=1-X
340 NEXT K: PRINT N;: RETURN
```

Line 320 in the L loop stores the ones and zeros in the array. The end of line 330 makes X alternate between zero and one.

To print out the contents of the array at this point, type:

```
5 LPRINT CHR$(27)"Q"CHR$(44)
110 FOR K=1 TO C: LPRINT A(K);: NEXT K:
    LPRINT: LPRINT "C="C
```

and RUN your growing program. Figure 14-1 shows the first seven lines of the result:

```
 1  0  1  0  1  0  0  1  1  0  0  1  1  0  0
  0  1  1  1  0  0  0  1  1  1  1  0  0  0
0  1  1  1  1  1  0  0  0  0  1  1  1  1  0
 0  0  1  1  1  0  0  0  1  1  0  0  1  1  0
  0  1  0  1  0  1  0  0  1  1  0  0  1  1
0  0  0  1  1  1  0  0  0  1  1  1  1  0  0
 0  0  1  1  1  1  1  0  0  0  0  1  1  1  1
```

Figure 14-1. Printing the array contents

Your entire printout s just a one-line array; the ones and zeros wrap around when they meet the temporary right-margin that you set in line 5.

This program shows how FOR-NEXT loops can use variables to create patterns. The overall pattern gets made up of five sets of pattern 1, four sets of 2, three sets of 3, . . . as shown in Figure 14-2.

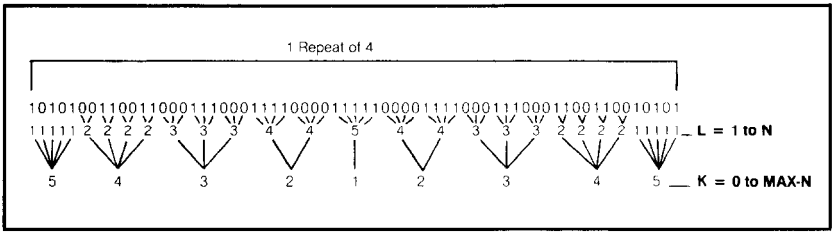


Figure 14-2. Pattern sets

Before proceeding, delete line 5 and modify line 110 so that it prints to the screen instead of to the printer:

```
110 FOR K=1 TO C: PRINT A(K);: NEXT K:
    PRINT' PRINT "C = "C
```

Pin Pattern Calculation

You will use the one-line array that you just created to generate a two-dimensional pattern. This technique results in a significant saving of memory compared to the method of generating an array that we used in the last chapter.

Two patterns are used in each pass of the print head. P is the pattern formed by the seven vertical dots at the start of each print line, and PO is its black/white reverse image. The program prints the pattern P in each column that is headed by a black dot. Pattern PO is printed in the other columns.

The pin patterns are created this way:

```
140 FOR PASS=0 TO LAST: P=0:
    PRINT  "PASS";PASS;"OF";LAST
160 FOR DOT=@ TO H
170 IF A(7*PASS+DOT+1)=1 THEN P=P+2^(6-DOT)
180 NEXT DOT
190 P0=127-P: IF PASS=LAST THEN P0=P0+1-2^(7-R)
```

For each pass of the print head (zero to LAST), the program calculates the pin patterns, seven dots at a time. Line 170 calculates P, and line 190 calculates its complement, PO.

Appendix F discusses problem codes and the P variable.

Adjust the line spacing to match the y-dot passes:

```
120 LPRINT CHR$(27)"1";
```

If this spacing produces slight gaps between rows, adjust it with CHR$(27)"3"CHR$(20).

The length of the array depends entirely on the variables in line 20. The following lines adjust the pin patterns (I' and PO) for the last pass of the print head if the array length is not a multiple of seven. Add:

```
130 LAST=INT(C/7):  R=C-7*LAST
150 H=6: IF PASS=LAST THEN H=R-1
```

On the last pass, R represents the number of pins used, and H is set to one less than the number of pins to be fired.

Graphics Width Settings

The required graphics width is C, the size of the array. If, however, C is greater than 255, the value n_2, in the graphics entry string must change from zero to one. With this in mind, add these three lines:

```
200 N1=C: N2=0
210 IF C>255 THEN N1=C-256: N2=1
220 LPRINT CHR$(27)"*"CHR$(5)CHR$(N1)CHR$(N2);
```

Introduced in Chapter 11, CHR$(27)" * "CHR$(5) is the one-to-one graphics density setting. It ensures a printout image that is square.

If your computer system requires a WIDTH statement to prevent the printer from issuing a carriage return before the graphics line is complete, add it now:

```
7 WIDTH LPRINT 255
```

The format for this statement may be different for your BASIC; see your software documentation.

Pattern Printout

Now that you've completed the groundwork, add the lines that actually print the pattern.

```
230 FOR K=1 TO C
240 IF A(K)=1 THEN LPRINT CHR$(P);
250 IF A(K)<>1 THEN LPRINT CHR$(P0);
260 NEXT K
270 LPRINT
280 NEXT PASS
```

These lines, which are part of the loop for each pass, cause the program to check each element of the array and then print either the pattern P (if the element is a one) or the pattern PO (if the element is not a one). Line 280 completes the PASS loop. After each pass is printed, the program recalculates the values of P and PO in lines 170 and 190.

Check your listing against the listing of Figure 14-3 to make sure it's all there:

```
10 DIM A(480): X=1: C=0
20 MAX=5: MIN=1: RE=4: N=0
30 FOR J=1 TO RE
40   N=N+1
50   GOSUB 300
60   IF N<MAX THEN 40
70   N=N-1
80   GOSUB 300
90   IF N>MIN THEN 70
100 NEXT J: PRINT
110 FOR K=1 TO C: PRINT A(K);:
    NEXT K: PRINT: PRINT    "C = "c
120 LPRINT CHR$(27)"1";
130 LAST=INT(C/7): R=C-7*LAST
140 FOR PASS=0 TO LAST: P=0: PRINT "PASS"; PASS;
    "0F";LAST
150 H=6: IF PASS=LAST THEN H=R-1
160 FOR DOT=0 TO H
170   IF A(7*PASS+DOT+1)=1 THEN P=P+2^(6-DOT)
180 NEXT DOT
190 P0=127-P: IF PASS=LAST THEN P0=P0+1-2^(7-R)
200 N1=C: N2=0
210 IF C>255 THEN N1=C-256: N2=1
220 LPRINT CHR$(27)"*"CHR$(5)CHR$(N1)CHR$(N2);
230 FOR K=1 TO C
240   IF A(K)=1 THEN LPRINT CHR$(P);
250   IF A(K)01 THEN LPRINT CHR$(P0);
260 NEXT K
270 LPRINT
280 NEXT PASS
290 LPRINT CHR$(27)"2": END
308 FOR K=0    TO MAX-N
310   FOR L=1 TO N
320     C=C+1: A(C)=X
330   NEXT L: X=1-X
340 NEXT K: PRINT N;: RETURN
```

Figure 14-3. Program for SYMMETRY

then RUN it to see if it looks like Figure 14-4.

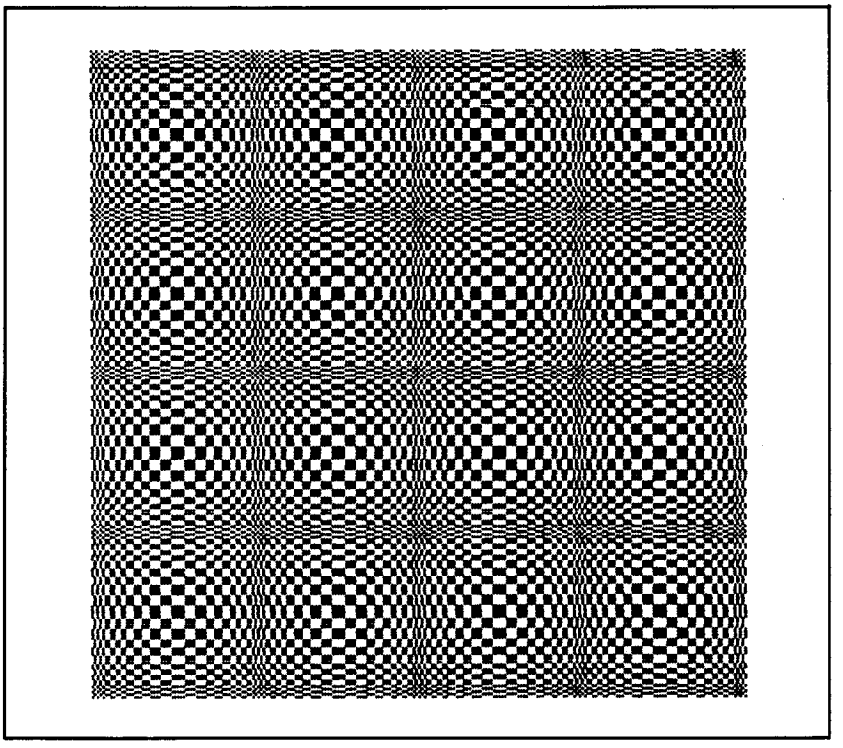


Figure 14-4. ***Symmetric pattern 1***

That's enough to knock your eyes right out of their sockets! And all **that** from a single one-dimensional array.

Variations

The computer completely controls your symmetric pattern, dot for dot. Small changes in the program can affect the pattern in a big way. For example, try this simple change in line 300:

```
300 FOR K=0 TO 0
```

And RUN the program again:

Notice in your printout (or in Figure 14-5) that each string of ones and zeros in the array prints only once. The K loop in line 300 controls the repetitions of these strings.

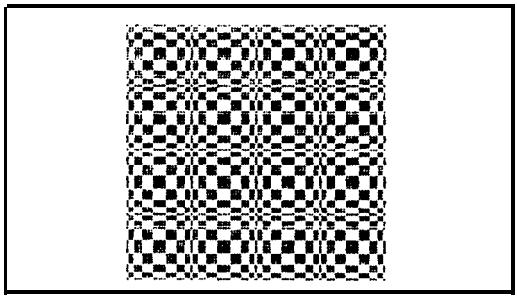


Figure 14-5. Symmetric pattern 2

Here's another interesting variation:

```
20 MAX=64: MIN=1: RE=1: N=1/2
40 N=N*2
70 N=N/2
```

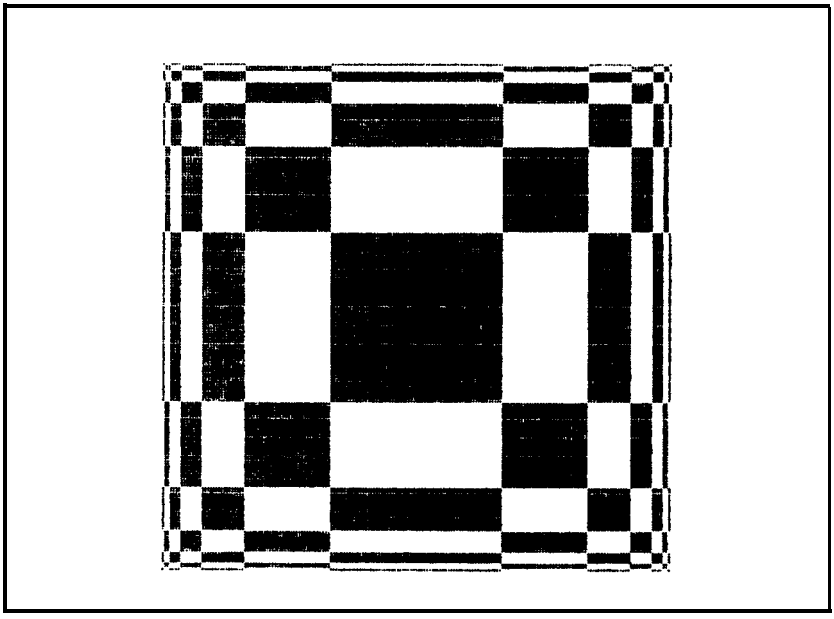


Figure 14-6. Symmetric pattern 3

Quite a difference! Instead of adding and subtracting one from N, lines ***40*** and ***70*** now double and halve it. This geometric progression creates a very different pattern.

Also notice that, because the variable RE is set to one, this pattern repeats only once.

Now's the time to experiment with some of your own changes to the variables and loops.

Summary

In this chapter you used a single string of ones and zeros to create a two-dimensional figure. You can use this technique to create large patterns with little drain on computer memory, but only with highly structured patterns.

You also used a graphics density of 72 dots per inch to produce a one-to-one aspect ratio of the dots.

Chapter 15
User-Defined Characters

If you've studied the program examples in this manual, you are quite adept at printing both graphics and text with the FX. In this chapter we're going to share the secrets of the ultimate in printer control-defining your own characters.

With the FX, you can create any number of new characters, graphics patterns to serve as building blocks for larger designs, or even whole type fonts. You can use these characters for any purpose as long as they fit into the same dot matrix as the ROM characters do-9 dots tall by **11** dots wide (6 main columns plus 5 intermediate columns). Figure **15-1** shows a comparison of a few sample characters and their ROM equivalents.

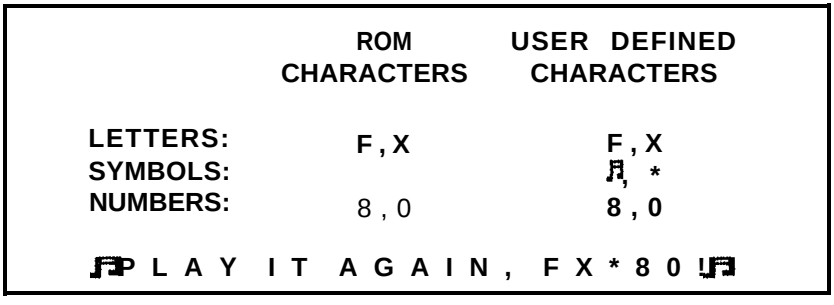


Figure 15-1. ROM and user-defined characters

Once you define your own characters, you can use them over and over, just as you use the FX's ROM characters. The FX prints user-defined characters the same way it does any other ASCII characters.

But before you can take advantage of this fantastic feature, you must first make a little preparation.

Preparation

DIP switch 1-4 controls the use of the FX's 2K RAM buffer. You can use this RAM memory as a large text buffer to smooth printer/computer communications, or you can store in it a set of user-defined characters. Unfortunately, it can't serve both purposes simultaneously. In this and succeeding chapters, we'll use this RAM area for user-defined characters. So set switch 1-4 off before proceeding.

Character Definition

Characters are defined with the ESCape "&" command sequence. The format is:

```
LPRINT  CHR$(27)"&"CHR$(r)CHR$(c1)CHR$(c2);
```

The r tells the printer in which RAM area the characters are to be stored. With a stock printer, there is only one area available: RAM area 0.

The notations c_1 and c_2 specify the range of characters to be defined. You can use the entire range of ASCII numbers from 0 to 255 (for which the ROM characters are shown in Appendix A), except for those areas where control codes reside (0 to 31, 127 to 159, and 255). You can also use some of the control-code locations, but only after special ESCape codes are issued. We'll get to that a bit later.

Here's how c_1 and c_2 work. Suppose you want to redefine the letters from A to E. The associated ASCII numbers are 65 to 69, so you simply let c_1 be 65(A) and c_2 be 69(E). Any of the keyboard characters can be redefined in a similar manner. For example, defining c_1 as 97 and c_2 as 101 selects lowercase letters a through e; defining c_1 as 33 and c_2 as 43 selects the symbols ! through + .

To simplify things a bit, the ASCII symbols can be used in place of CHR$($c_1$) and CHR$(c_2). For example, either:

```
LPRINT  CHR$(27)"&"CHR$(0)CHR$(65)CHR$(69);
```

or:

```
LPRINT  CHR$(27)"&"CHR$(0)"AE";
```

selects characters A through E. On some occasions you may wish to define only one character. That's O.K., too. Just use the same number or letter for c_1 and c_2:

```
130 LPRINT  CHR$(27)"&"CHRS(0)"EE";
```

The semicolon is very important. The CHR$(27)"&" sequence expects more data to follow (just as Graphics Mode does). The semicolon at the end of the line prevents an unwanted carriage-return code from disrupting the data.

For each character to be defined (determined by c_1 and c_2), the printer expects 12 data numbers to follow. The first of these numbers is called the attribute byte. It determines some special attributes or characteristics of the character being defined. The next 11 numbers contain the dot patterns of the symbol being defined-nothing fancy, just 11 standard graphics pin patterns.

Design

The first step in defining a new character is to lay out the dot pattern. Check Appendix A to see how the ROM characters are designed. Your characters share the same limitations as those found in the ROM. Characters can be a maximum of 8 dots tall (even though the matrix is 9 dots) and 11 dots wide. Most characters use only the top 7 pins of the print head; lowercase characters with descenders use the bottom 7. Also note that all characters leave the last two columns (one intermediate and one main) unused; this provides space between the letters when they are printed. Figure 15-2 shows the design of a letter E in an 8 by 11 matrix.

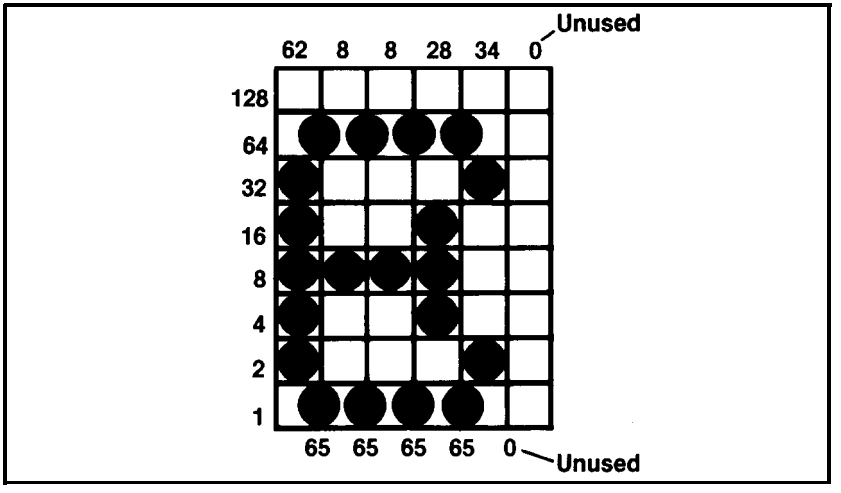


Figure 15-2. User-defined E

To be consistent with the ROM characters, we use only 7 rows. The character would normally go in the top 8 rows, but we shift all the dots down one row so that 7-bit systems can stay with the program.

Also note that two adjacent dots cannot be printed in the same row. Even in Half-Speed Mode, the printer simply refuses to print two overlapping dots. Figure 15-3 illustrates an E that is incorrectly designed because it uses overlapping dots:

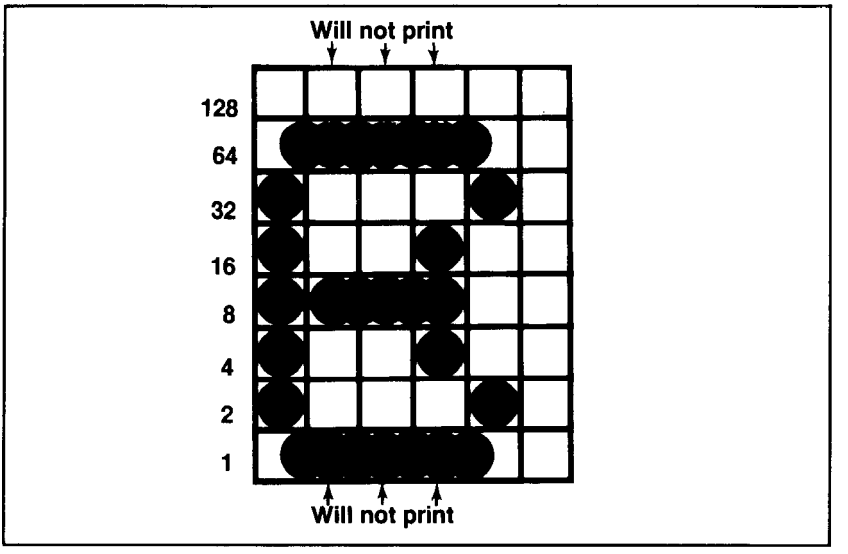


Figure 75-3. Incorrectly designed E

Dots into DATA

The data numbers for each column of Figure 15-2 are calculated in the same manner as the data for Graphics Mode. And the appropriate numbers can just as easily be stored in DATA statements. Type in the READ routine and data for the character in Figure 15-2:

```
150 FOR X=1 TO 11: READ C:
   LPRINT CHR$(C);: NEXT X
1170 DATA 62,65,8,65,8,65,28,65,34,0,0:  'My E
```

Notice that the DATA statement contains 11 numbers even though the design uses only 9 of the 11 columns. Unused columns must be coded as 0.

Attribute byte

The attribute byte is the first of the 12 data numbers required to define any character. At print time it controls two aspects of the way the character is printed. First, it determines which 8 pins of the print head are used to print the character. For most characters, the top 8 pins are used, but for lowercase characters with descenders (like g and p), the bottom 8 pins can be used.

So how does the attribute byte determine which 8 pins are used? At print time, the printer checks the attribute byte before each character is printed. If the high-order bit is on, the top 8 pins of the print head are used; if the high-order bit is off, the bottom 8 are used. To put it another way, if the attribute byte for a given character is 128 or greater, the top 8 pins are used; if it is 127 or less, the bottom 8 are used. Figure 15-4 demonstrates these choices.

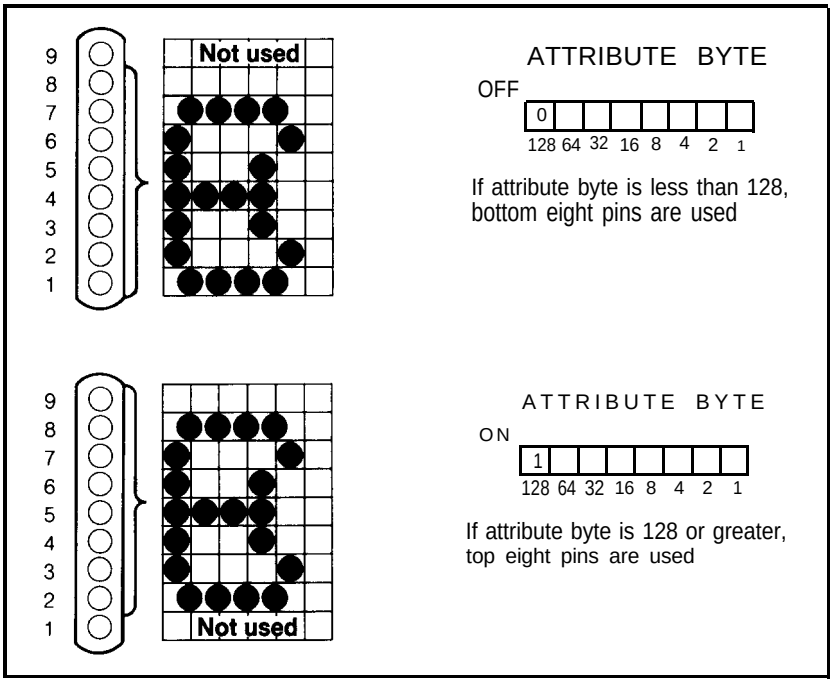


Figure 15-4. Pins chosen by attribute byte

Proportional print

The attribute byte also contains information used to print a character in Proportional Mode. It tells the printer in which columns to start and stop printing for each character. If you label the 11 columns deter-

mined by the data numbers as columns 0 to 10, then in Proportional Mode the minimum and maximum starting and stopping columns will be 0 and 11. Why 11 instead of 10? Column 11 is the maximum value because Proportional characters are always Emphasized; this makes each character wider by one intermediate column. So when defining your own characters for proportional printing, always reserve one extra column.

Suppose you want a character to start in column 1 and end in column 10. How do you put this information into the attribute byte? The starting column number (1) is converted to a 3-bit binary number (001) and stored in bits 4, 5, and 6 of the attribute byte. The ending column number (10) is converted to a 4-bit binary number (1010) and stored in bits 0 to 3. The conversions are shown in Figure 15-5.

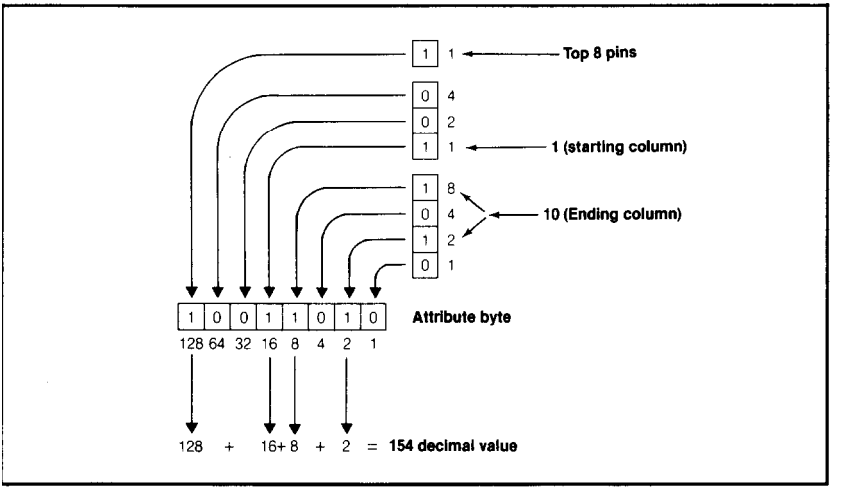


Figure 15-5. ***Attribute byte conversions***

The full 8-bit attribute byte, then, is composed of three parts:

1. Bit 7 determines which pins are used to print the character.

2. Bits 4, 5, and 6 determine the starting column number.

3. Bits 0, 1, 2, and 3 determine the ending column number.

So the attribute byte constructed in Figure 15-5—CHR$(154)—uses the top eight pins, starts printing in column 1, and ends in column 10.

Note that the proportional print information is used only when the character is printed in Proportional Mode. Otherwise-the full range of columns 0 to 11 is used. Also note that if 7-bit system users set the high-order bit with CHR$(27)">" before they use the CHR$(27)"&" sequence, it stays on for the attribute and character data bytes.

One final note. Even if you choose not to print the columns from 0 through 11, you must send the printer 11 data numbers plus the attribute byte. The printer expects 12 data numbers for each character, no matter what.

So much for the example. In the current program, set the attribute byte to 139 (139 in binary is 10001011). This value starts at column 0, ends at column 11, and uses the top 8 pins of the print head. Seven-bit systems can use 139 or 11. Either way, the printer will use the bottom 8 pins for 7-bit systems, Add:

```
140 LPRINT CHR$(139);
```

Printing User-Defined Characters

If you RUN the program at this point, it will define the character E in RAM area 0 (assuming switch 1-4 is off), but only the ROM version of the E will print. Try it. Add:

```
180 LPRINT "EEEEE"
200 LPRINT CHR$(27)"@": END
```

```
EEEEE
```

Sure enough, the FX prints a Roman Pica E. To print your newly defined character, you must tell the printer to ignore the ROM and print only RAM characters. The format for this instruction is:

```
LPRINT CHR$(27)"%"CHR$(n1)CHR$(n2);
```

The CHR$(27)"%" sequence determines the currently active character set. The n_1 selects either ROM (0) or RAM (l), while n_2 selects the area (0 is the only area available). The command to activate the RAM area is:

```
120 LPRINT CHR$(27)"%"CHR$(1)CHR$(0);
```

but before you print the user-defined E, make it more visible by adding:

```
170 LPRINT CHR$(27)"!8";
190 LPRINT  CHR$(27)"!@"
```

Line 170 uses the Master Select code to print Double-Strike, Expanded, Emphasized Pica characters. Line 190 uses the Master Select to return to Pica Mode.

Here are the lines you've typed so far:

```
120  LPRINT  CHR$(27)"%"CHR$(1)CHR$(0);
130 LPRINT  CHR$(27)"&"CHR$(0)"EE";
140 LPRINT  CHR$(139);
150 FOR X=1 TO 11: READ C: LPRINT CHR$(C);: NEXT X
170  LPRINT CHR$(27)"!8";
180  LPRINT "EEEEE";
190  LPRINT CHR$(27)"!@"
200  LPRINT CHR$(27)"@": END
1170   DATA 62,65,8,65,8,65,28,65,34,0,0: 'My E
```

Let's see what it looks like in Proportional Mode (without the last two columns printed). Add:

```
140 LPRINT  CHR$(137);
175  LPRINT CHR$(27)"p1";
```

See how the Es are packed closely together? Fine. Before proceeding, change back to monospacing by deleting line 175 and changing 140:

```
140 LPRINT CHR$(139);
```

While you are in the neighborhood, take a look at some of the other characters in RAM with:

```
180 LPRINT "EPSON"
```

Oops! Where is the rest of EPSON? All right, we confess-the only characters in the user-defined RAM are those you put there yourself. Characters that haven't been defined print as blank spaces. So the

RAM area is like a big blank chalk board waiting for you to fill it up. At this point, because you have only defined an E, that's all you get from RAM.

Downloading Command

Wouldn't it be nice if you could magically transport some of the ROM characters over to the RAM area so you wouldn't have to switch back and forth or define an entire character set each time you use the RAM area? In plenty of applications you only need to define a few special characters to be used with the standard alphabet and numbers. That's why the FX provides the option of copying (sometimes called downloading) the entire ROM set into the user-defined RAM area.

The downloading command has the format:

```
LPRINT CHR$(27)":"CHR$(n1)CHR$(n2)CHR$(n3);
```

This command is designed with possible future expansion in mind. For now, set all three numbers to 0:

```
110 LPRINT CHR$(27)":"CHR$(0)CHR$(0)CHR$(0);
```

EPSON

Now you get your custom designed E plus four of the normal characters copied over from ROM. Notice that the E is lower on the page than the other characters even though the high-order bit of the attribute byte is on. In order to save 7-bit users from total frustration, we designed the character to use the bottom seven rows. If you do not have a 7-bit system, you can use the top seven rows for all but lowercase characters with descenders.

Caution: Be very careful about using the Reset Code after defining your own characters in RAM. This code wipes out the entire contents of RAM . . . goodbye user-defined characters!

Defining More Characters

Once the ESCape sequences are in place, adding more characters is a breeze. To see how much of a breeze, simply add more data:

```
1150 DATA 7,8,16,36,64,36,16,8,7,0,0: 'My A
1160 DATA 127,0,72,0,72,0,76,2,121,0,0: 'My R
```

and make these changes:

```
130 LPRINT CHR$(27)"&"CHR$(0)"rt";
140 FOR Y=1 TO 3: LPRINT CHR$(139);
160 NEXT Y
180 LPRINT "rst"
```

A R E

Line 130 controls the reading of the data. It expects data for three characters: r, s, and t. This example uses lowercase characters. If necessary, you can use CHR$(114) and CHR$(l16) in place of the "rt".

The attribute byte for each character is sent in line 140 and the other 11 bytes are read from DATA lines. This method is nice for quick and easy character definition.

If you intend to print your characters in Proportional Mode, you'll want to add a different attribute byte to the start of each DATA line and adjust the READ routine for 12 numbers. Don't forget the attribute byte or you'll end up with results you won't like.

Redefining Control Codes

For some of you dedicated users, the range from 32 to 126 and 160 to 254 may not be large enough to accommodate all the characters you want. Perhaps you have a passion for Egyptian hieroglyphics, or maybe you need a complete set of mathematics symbols. And what about the entire Japanese character set (it has some 4000 symbols)?

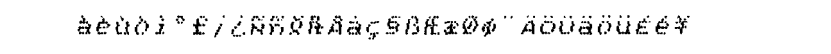
If you get carried away with user-defined characters, you may end up searching for more storage. Anticipating this need, Epson provides commands that will allow you to define and print certain control codes in the same way that you treat other characters. (Remember that the low-order control codes are the ASCII codes 0 through 31 plus 127, and the high-order control codes are 128 through 159 plus 255.)

These codes do not normally print symbols on paper, rather they cause the printer to change modes. To make them print as normal symbols requires an extra command. For example, the command to "normalize" the high-order control codes is:

```
LPRINT CHR$(27)"6"
```

Let's use this command to see how the ROM control codes can print. Add:

```
2 LPRINT CHR$(27)"6"
4 FOR X=128 TO 159: LPRINT CHR$(X);: NEXT X
6 LPRINT CHR$(27)"7"
8 STOP
```



That's where the international characters (Italic version) have been stored. So the CHR$(27)"6" command without the CHRS(27)"R" gives you access to the international characters. The CHR$(27)"7" turns these characters back into control codes. See how the CHR$(27)"6" and CHR$(27)"7" commands work with the user-defined characters in RAM by deleting lines 2 through 8, and adding the following (unless yours is a 7-bit system, in which case this won't work):

```
130 LPRINT CHR$(27)"&"CHR$(0)CHR$(128)CHR$(131);
140 FOR Y=1 TO 4: LPRINT CHR$(139);
165 LPRINT CHR$(27)"6"
180 LPRINT CHR$(128)CHR$(129)CHR$(l30)CHR$(131)
1140 DATA 0,l26,1,2,4,8,4,2,1,126,0: 'My W
```

W A R S

The low-order control codes **can** also be defined, but not all of them can be printed with ease. CHR$(27)"I1" makes **them** printable, and CHRS(27)"I0" returns them to normal.

Just as the higher control codes hide the Italic international characters, the lower control codes hide the Roman international characters.

Let's see how the CHR$(27)"I1" and CHR$(27)"I0" codes work with user-defined characters. Change:

```
130 LPRINT CHR$(27)"&"CHR$(0)CHR$(1)CHR$(3);
140 FOR Y=1 TO 3: LPRINT CHR$(139);
165 LPRINT CHR$(27)"I1"
180 LPRINT CHR$(1)CHR$(2)CHR$(3)
```

And add:

```
1100 DATA 0,121,0,73,0,73,0,73,0,79,0: 'My S
1110 DATA 0,127,0,65,0,65,0,65,0,127,0: 'My Oh
```

S O W

The program now contains six DATA lines, but it uses only the first three. The three characters are stored in ASCII codes 1, 2, and 3 in RAM; they are printed by line 180.

Not all of the low-order (O-31) control codes can be changed to print as normal characters-nor would you want them to. Imagine, if you changed code 27 to print as a normal character . . . no more ESCape codes. You would have a hard time getting anything done.

Codes that currently activate special modes or actions by the printer cannot be printed as normal characters. These include 7 to 15, 17 to 20, 24, and 27. It is, however, possible to print the characters stored in these locations with the CHR$(27) "R" command.

Here's how it works. Suppose you choose to define the ASCII code 8 (normally a backspace). The CHR$(27) "&" command will work fine, but printing CHR$(8) still produces a backspace, even after a `CHR$(27)"I1". CHR$(27)"R"` to the ***rescue.*** `CHR$(27)"R"` lets you print the character stored in location 8 with another ASCII code. The `CHR$(27)"R"` transports the character to an easily printable location. To find out what is stored where, use Table 15-1.

Table 15-1. International character locations

Dec. Code	USA	France	Germ.	Eng.	Denm.	Sweden	Italy	Spain	Japan
35				6				12	
36						11			
64		0	16			29			
91		5	23		18	23	5	7	
92		15	24		20	24		9	31
93		16	25		13	13	30	8	
94						25			
96						30	2		
123		30	26		19	26	0	22	
124		2	27		21	27	3	10	
125		1	28		14	14	1		
126		22	17			28	4		

Find 8 in the table; it is in the CHR$(93) row under the Spain heading. To print the character stored in 8, use CHR$(27)"R"CHR$(7) to activate the Spanish character set, and print CHRS(93). Ole! This same technique can be used to access any of the normally unprintable control codes.

Using an international character set while defining characters can be a two-edged sword. If you are currently using one of the international sets (other than USA), then defining any of the codes 35, 36, 64, 91 to 94, 96, or 123 to 126 gets a bit tricky. These codes are merely pointers to the control-code areas in which the international characters are really stored.

To define any of these characters while using an international set, the true location of the character must be used. For instance, if the printer is in the Spanish set and you wish to define character 93, you must use CHR$(8) in the CHR$(27)"&" sequence to define the character, but CHR$(93) to print it.

To make sure you understand this thoroughly try answering this one: how would you redefine the ESCape code? First find 27 on the chart. It occurs in two places, one of which is in the column labelled Sweden and the row labelled 124. So you could store a user-defined character at 27 and print it in the Swedish set as character 124.

Mode Strings

For some applications, you may wish to use all 256 RAM locations for your own special symbols. In that case, there is no need to download the ROM into RAM. But you will need a quick and easy way to switch back and forth between the two character sets. One easy way to do this is to define two character strings:

```
80  RAM$=CHR$(27)+"%"+CHR$(1)+CHR$(0)
90  ROM$=CHR$(27)+"%"+CHR$(0)+CHR$(0)
```

Add these lines to the current program. To demonstrate their effect, try:

```
138 LPRINT  CHR$(27)"&"CHR$(0)"18";
14p FOR Y=1 TO 8: LPRINT CHR$(139);
180  LPRINT  ROM$"12345678"RAM$"12345678"
1120 DATA 0,63,64,8,64,8,64,28,64,32,0: 'My F
1130 DATA 0,32,64,0,64,63,64,0,64,32,0: 'My T
```

If you find yourself defining characters in small groups, the same technique can be used to store part of the CHR$(27) " &" command:

```
Z$=CHR$(27)+"&"+CHR$(0)
```

Z$ can be used to define each new string of characters with a simple command such as either of these:

```
LPRINT Z$"AZ";
LPRINT  Z$CHR$(l28)CHR$(159);
```

STRATA

Your current program uses eight user-defined characters, which will be used again in a later chapter. For now, delete lines 80, 90 and 165 and change line 180:

```
180, LPRINT "ø:147646 12345678"
```

S T R A T A S O F T W A R E

Save the current program as STRATA.

Summary

`CHR$(27)"&"CHR$(`n_1`)CHR$(`n_2`)CHR$(`n_3`);`

Defines characters, where n_1 selects the RAM buffer (0), n_2 is the starting character, and n_3 is the ending character

For each character in the CHR$(27)"&" sequence from n_2 to n_3, the printer expects 12 data numbers. The first number, called the attribute byte, determines the height and width characteristics of the character at print time. The other 11 numbers determine the pin patterns used to print the character

`CHR!$(27)"%""CHR$(`n_1`)CHR$(`n_2`)`

Activates a given character set, where n_1 indicates ROM (0) or RAM (1) and n_2 is 0

CHR$(27)“: “CHR$(n_1)CHR$($n_2$)CHR$(n_3)	Downloads ROM characters into RAM. All three numbers are 0
CHR$(27)“6”	Enables printing of codes 128 to 159 and 255
CHR$(27)”7”	Disables printing of codes 128 to 159 and 255
CHR$(27)“11”	Enables printing of the codes 0 to 31 except those used as control codes. The control codes can be printed with CHR$(27)“R”
CHR$(27)“I0”	Disables printing of codes 0 to 31

Chapter 16
Combining User-Defined Characters

In this chapter we'll explore the technique of combining user-defined characters to make large letters and symbols.

Large Letters: Double Wide

We'll start by placing two characters next to each other to form a double-width letter. Enter this new program, being careful to enter the line numbers as written.

```
NEW
20 LPRINT CHR$(27)" :"CHR$(0)CHR$(0)CHR$(0);
25 'Copies ROM to RAM
30 LPRINT CHR$(27)"%"CHR$(1)CHR$(0); 'Activates RAM
60 LPRINT CHR$(27)"&"CHR$(0)"AB";
65 'Defines characters A & B
70 FOR Y=1 TO 2: LPRINT CHR$(139); 'Attribute byte
80 FOR X=1 TO 11: READ N: LPRINT CHR$(N);: NEXT X
90 NEXT Y
180 LPRINT CHR$(27)"@";: END
```

The ESCape sequences in line 20, 30, and 60 are the commands from the last chapter. This program prepares the printer to define the two characters A and B. Enter the DATA lines:

```
200 DATA 0,2,5,0,11,16,3,32,70,0,84
210 DATA 68,32,66,49,8,21,14,5,2,0,0
```

And how do they look side by side? Enter:

```
100 LPRINT "AB"
```

Very nice. Using two characters side by side provides a larger matrix and therefore gives more flexibility in character design. But there is one problem. When two user-defined characters are placed side by side, there is one intermediate column that, unless the first character is printed in Emphasized Mode, cannot contain any dots. As you can see in Figure 16-1, that is the shared intermediate column.

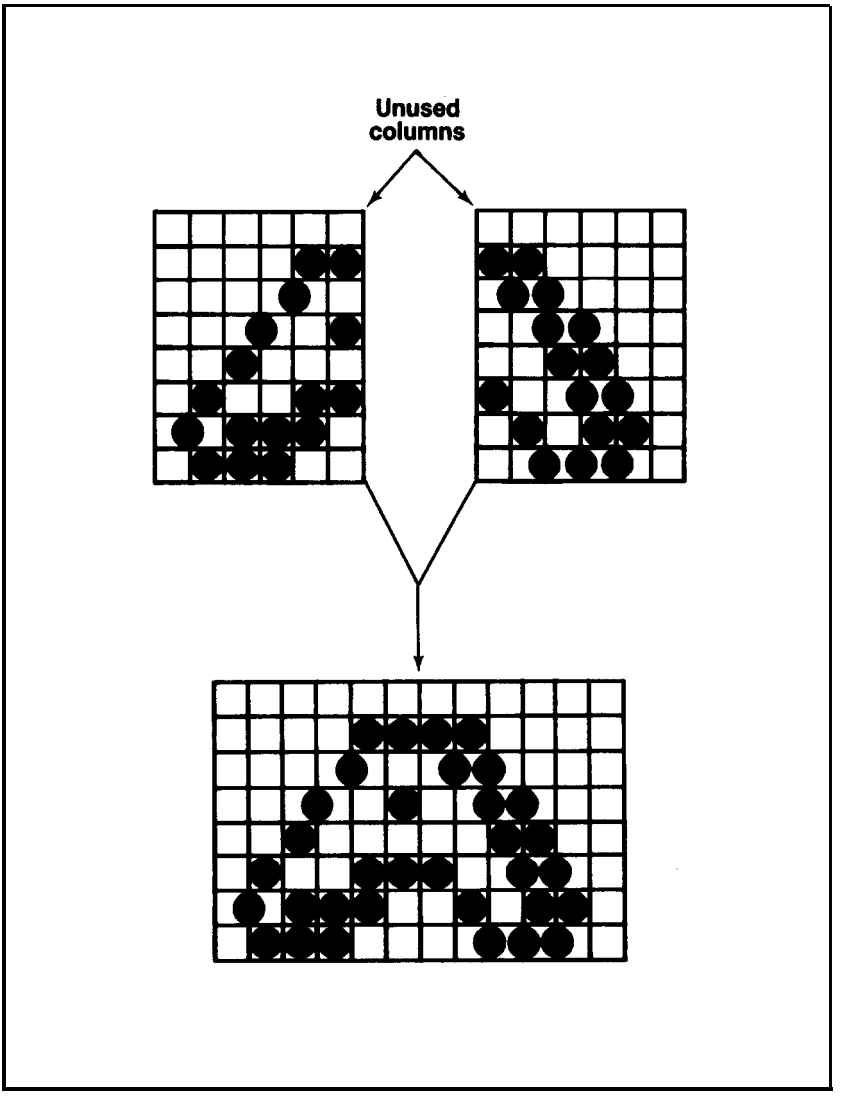


Figure 16-1. Side-by-side characters

Large Letters: Double High

Let's stack two characters, one on top of the other, with these changes:

```
10 LPRINT CHR$(27)"1"CHR$(27)"U1";
100 LPRINT "A"
110 LPRINT "B"
200 DATA 16,32,95,0,64,0,127,0,63,0,0
210 DATA 14,0,123,0,3,0,123,0,1,127,0,15
```

Line 10 changes the lines spacing to 7-dot and turns on Unidirectional Mode for precise alignment of the two lines. If there are slight gaps between rows, change the line spacing to 6-2/3-dot with CHR$(27)"3"CHR$(20).

With a little imagination, you can create some dynamite patterns by combining characters.

Giant Letters: Double High and Double Wide

For even larger type styles, you can design letters that are both two characters tall and two characters wide. This gives you an 18 by 22 matrix. If you have a 7-bit system, you will have to skip to "Core Sets," below.

Which ASCII numbers can be use to store the four characters that will make up each letter? A quick glance at the ASCII chart (Appendix A) shows that there are four symbols that readily relate to each letter of the alphabet. They are the upper- and lowercase versions of each letter in its Roman and Italic typefaces. For example, the letter G could be designed using the following four ASCII codes:

CHR$(71) Uppercase Roman G
CHR$(103) Lowercase Roman g
CHR$(199) Uppercase Italic G
CHR$(231) Lowercase Italic g

Such usage is shown in Figure 16-2.

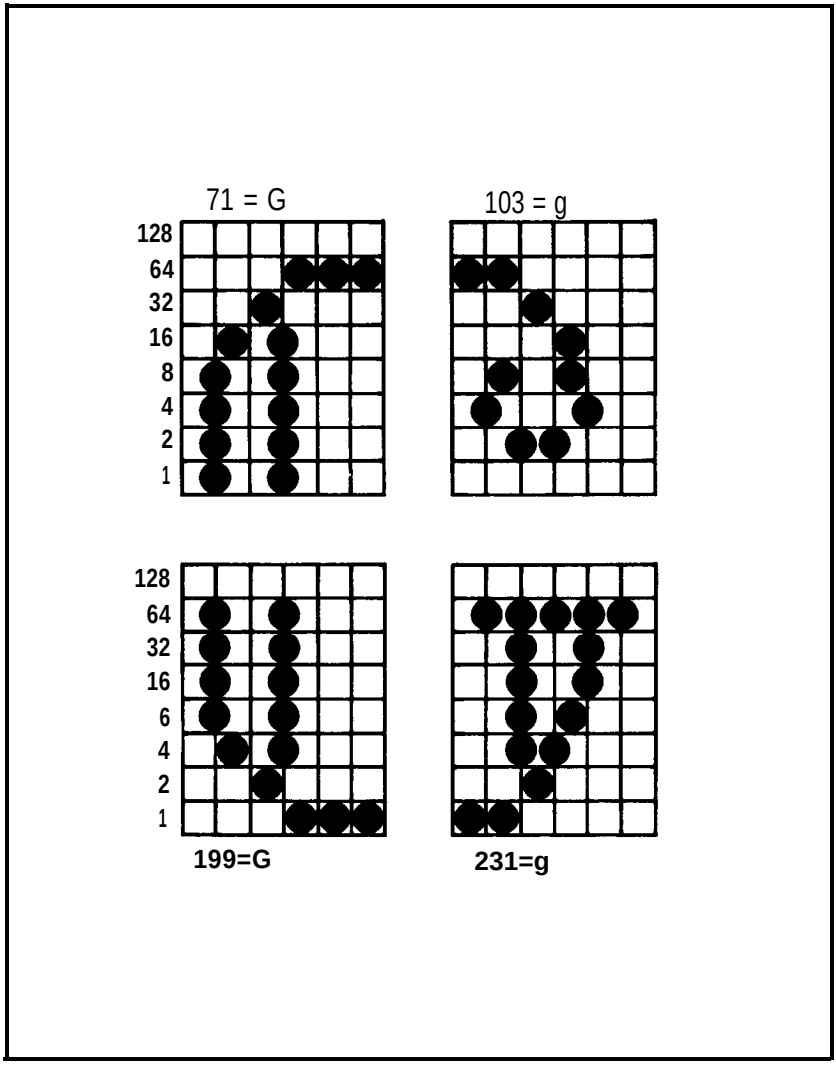


Figure 16-2. Double high and wide character

In order to define letters in groups of four, you'll have to modify the definition process. Fortunately, the ASCII numbers that represent the four versions of each character have a consistent pattern. That pattern in shown in Table 16-1.

Table 16-1. ASCII pattern

Pattern	Example
Roman letter = L	G = 72
lowercase letter = L + 32	g = 72 + 32 = 110
Italic letter = L + 128	G = 72 + 128 = 200
Italic lowercase letter = L + 160	g = 72 + 160 = 232

With this in mind, add these lines:

```
40 READ L: PRINT CHR$(L) 'Print to screen
50 FOR Y=0 TO 1: FOR Z=0 TO 1: A=L+128*Y+32*Z
```

and make these changes:

```
60 LPRINT CHR$(27)"&"CHR$(0)CHR$(A)CHR$(A);
70 LPRINT CHR$(139);
90 NEXT Z: NEXT Y
```

Line 50 calculates the code (A), to be defined in line 60, by adding the appropriate amount to the base letter L. Line 60 is the `CHR$(27)"&"` defining sequence, and line `70` sets the attribute byte to 139.

The code for the letter to be defined and the data for its four components are stored in DATA statements. Delete lines 200-210 and type:

```
250 'G
260 DATA 71
270 DATA 0,15,16,0,32,31,64,0,64,0,64
280 DATA 64,4,72,2,32,2,24,4,0,0,0
290 DATA 0,120,4,0,2,124,1,0,10,1
300 DATA 1,64,0,124,2,68,8,120,0,64,0
```

Here's the printing routine:

```
100 A$="":        INPUT "ENTER A STRING ø'", A$:
    IF A$="" THEN 180
110 INPUT "ENTER A MASTER PRINT MODE NUMBER ø",M
120 LPRINT CHR$(27)"!"CHR$(M);
130 FOR Y=0 TO 1: FOR X=1 TO LEN(A$)
140 A=ASC(MID$(A$,X,1))+128*Y
150 LPRINT CHR$(A)CHR$(A+32);
160 NEXT X: LPRINT: NEXT Y
170 LPRINT
```

Some BASIC systems do not support the MID$ statement-instead they use subscripts to isolate portions of a string. To designate the

characters in positions 6 and 7 of string A$, for example, MID$(A$,6,2) would be coded as A$(6,7). If your system uses this scheme, change line 140 to:

```
140  A=ASC(A$(X,X))+128*Y
```

This program automatically prints all four parts of each letter. You type just a single letter; it does the rest. Before you RUN, check it against Figure 16-3.

```
10  LPRINT  CHR$(27)"1"CHR$(27)"U1";
20 LPRINT CHR$(27)" :"CHR$(0)CHR$(0)CHR$(0);
30 LPRINT  CHR$(27)"%"CHR$(1)CHR$(0);
40 READ L: PRINT CHR$(L)
50 FOR Y=0 TO 1: FOR Z=0 TO 1: A=L+128*Y+32*Z
60  LPRINT  CHR$(27)"&"CHR$(0)CHR$(A)CHR$(A);
70 LPRINT CHR$(139);
80 FOR X=1 TO 11: READ N: LPRINT CHR$(N);: NEXT X
90 NEXT Z: NEXT Y
100 A$="": INPUT "ENTER A STRING ø", A$:
    IF A$="" THEN 180
110 INPUT "ENTER A  MASTER PRINT MODE NUMBER ø ", M
120 LPRINT CHR$(27)"!"CHR$(M);
130 FOR y=0 To 1: FOR X=1 TO LEN(A$)
140  A=ASC(MID$(A$,X,1))+128*Y
150 LPRINT CHR$(A)CHR$(A+32);
160 NEXT X: LPRINT: NEXT Y
170 LPRINT
180 LPRINT CHR$(27)"@": END
250 'G
260 DATA 71
270 DATA 0,15,16,0,32,31,64,0,64,0,64
280 DATA 64,4,72,2,32,2,24,4,0,0,0
290 DATA 0,120,4,0,2,124,1,0,1,0,1
300 DATA 1,64,0,124,2,68,8,120,0,64,0
```

Figure 16-3. Program for giant G

When you RUN it, you should see line 100's prompt:

ENTER A STRING

You can respond with any string of letters, but for now type GO, with

no space after the 0 and then press RETURN. The next prompt on the screen is:

ENTER A MASTER PRINT MODE NUMBER

For now, enter a 24. Remember, all codes from 0 to 255 produce a combination of print modes, but there are only 16 unique combinations. You may want to refer to Figure 5-2 for the other possibilities.

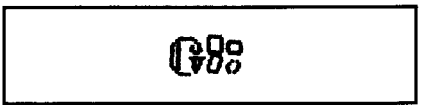


Figure 16-4. Giant G

The four characters defined in lines 270 to 300 combine to print a giant G. The 0 is printed as four small characters. Figure 16-4 illustrates the way the program arranges the four versions of each character to make a larger letter.

Line 140 examines the A$ string, character by character, and determines its ASCII value. Line 150 prints what's now stored in the locations for the Roman upper- and lowercase versions of each character on the first pass of the print head. On the second pass, Y is set to 1 and 128 is added to A in line 140. Thus line 150 then prints what has replaced the Italic versions of the character.

To get a better idea of what this program can do, you'll need to add more data. The DATA lines below supply data for the letters A-M-E-S and for the space character. (It is necessary to redefine the space character since two of its four components print as @ signs.)

Be careful. Although the space character is user-definable, you should avoid using it as anything other than a space character. No matter how it is defined, the space character will never print at the beginning or end of a line.

Enter the data lines as shown in Figure 16-5. Now add a loop to READ the new data by changing these lines:

```
40 FOR W=1 TO 6: READ L: PRINT CHR$(L)
90 NEXT Z: NEXT Y: NEXT W
```

```
190 'SPACE
200 DATA 32
210 DATA 0,0,0,0,0,0,0,0,0,0
220 DATA 0,0,0,0,0,0,0,0,0,0
230 DATA 0,0,0,0,0,0,0,0,0,0
240 DATA 0,0,0,0,0,0,0,0,0,0
310 'A
320 DATA 65
330 DATA 0,0,1,0,1,0,6,24,32,92,0
340 DATA 67,32,24,4,3,0,0,0,0,0
350 DATA 0,65,32,7,24,33,64,32,16,0,8
360 DATA 8,97,24,7,0,97,24,7,0,1,0
370 'M
380 DATA 77
390 DATA 0,64,0,127,0,32,16,7,8,0,4
400 DATA 4,0,8,7,16,32,0,127,0,64,0
410 DATA 0,1,0,127,0,1,0,127,0,1,0
420 DATA 0,1,0,127,0,1,0,127,0,1,0
430 'E
440 DATA 69
450 DATA 0,64,0,127,0,64,0,62,65,0,65
460 DATA 65,0,67,0,64,0,64,32,0,0,0
470 DATA 0,1,0,127,0,1,0,126,1,0,1
480 DATA 1,0,69,0,11,0,11,0,6,0,0
490 'S
500 DATA 83
510 DATA 0,0,0,24,36,0,66,16,105,0,68
520 DATA 74,0,69,0,68,0,40,16,0,0,0
530 DATA 0,8,22,0,33,0,33,16,1,0,65
540 DATA 33,0,17,0,75,4,33,0,22,8,0
```

Figure 16-5. Data for AMES

And RUN. This time respond to the prompts by typing:

```
GAMES SEEM SAME
49
```

Figure 16-6. Games seem same

When characters are printed side by side in any mode that compresses the matrix columns, gaps appear at the intermediate column

positions. That includes Elite and Compressed Modes. For a comparison of the three print pitches, RUN the program three more times and enter:

MESSAGES, 48
MESSAGES, 49
MESSAGES, 52

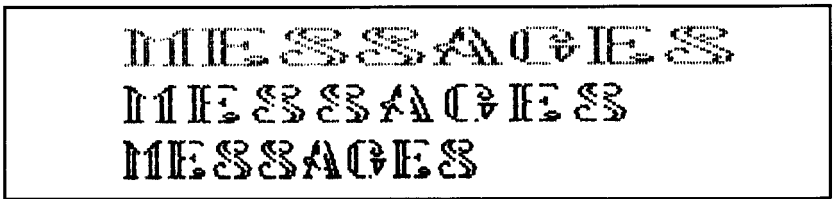


Figure 16-7. Messages in three pitches

All three mode combinations include Double-Strike and Expanded print; the only difference between them is the pitch. The first pitch is Pica, the second is Elite, the third is Compressed.

Despite this limitation, you should have a good time adding the rest of the alphabet or defining your own character set. By the way, the Introduction at the beginning of this manual shows a few more of these Double Wide and Double High letters.

You may want to SAVE the current program before proceeding.

Core Sets

Combining user-defined characters is a great way to create frequently used logos or fancy headings. But as you saw, defining an entire alphabet of oversized letters uses up ASCII codes rather quickly.

Fortunately, there is an alternative. In some cases, you may be able to define a handful of core characters that can be combined to make any letter in the alphabet. This requires a bit of imagination; we present an example here to lubricate those creative gears.

Prepare for the program changes by deleting lines 20, 40, 50, and 100 to 540. Now change:

```
60 LPRINT CHR$(27)"$"CHR$(0)"16";
70 FOR Y=1 TO 6: LPRINT CHR$(139);
90 NEXT Y
```

Deleting line 20 ensures that the printer does not download the ROM characters. That makes your defined characters the only ones around-no funny stuff on the printer. Here is the data:

```
100'SIX
110 DATA 7,8,16,0,32,3,68,0,72,0,73
120 DATA 73,0,72,0,68,3,32,0,16,8,7
130 DATA 73,0,9,0,17,96,2,0,4,8,112
140 DATA 112,8,4,0,2,96,17,0,9,0,73
150 DATA 127,0,0,0,0,127,0,0,0,0,127
160 DATA 73,73,73,73,73,73,73,73,73,73,73
```

That's right, there are only six characters, but it is a very powerful set of characters. With them, you can print an entire alphabet and more. To see the magnificent SIX, type:

```
180 LPRINT "1 2 3 4 5 6"
200 LPRINT CHR$(27)"@": END
```

Now try printing them in a different order. Type:

```
170 FOR Y=1 TO 5
180 READ P$: LPRINT P$
190 NEXT Y
210 ' Tracks
220 DATA "626626201620162620501 66"
230 DATA "050050505050505050500"
240 DATA "050056305650500056304 62"
250 DATA "050050505050505050005"
260 DATA "050050463046363050466 63"
```

Give it a RUN to get the pattern shown in Figure 16-S.

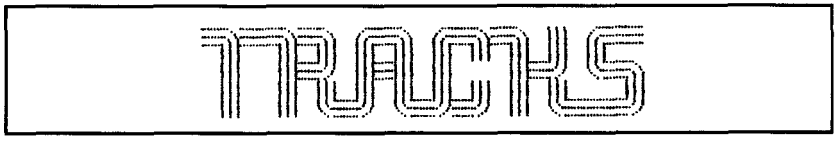


Figure 16-8. Tracks

Those little characters can do some amazing tricks. Try another; begin by deleting lines 210 to 260, then change:

```
170 FOR Y=1 TO 8
210 ' Pattern
```

```
210  ' Pattern
220 DATA "00012000", "01665620"
230 DATA "05055050", "16634652"
240 DATA "45621663", "05055050"
250 DATA "04656630", "00043000"
```

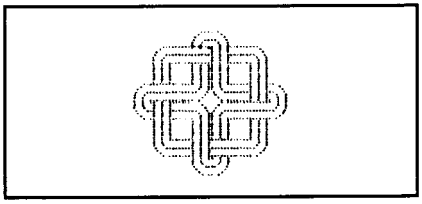


Figure 16-9. Interlace

Have fun creating your own designs with these characters. You may wish to SAVE the program before proceeding.

Line Graphics

The RX series printers have a set of line-graphics characters stored in ROM. In the following program, we define a similar set.

What are line-graphics characters? They are a set of characters that fit together to make borders and outlines for all kinds of forms. Since they are so useful, we use them again in the next chapter; be sure to save the next program when you are through entering it.

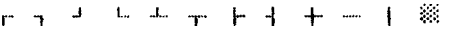
We will add to the program that you built and saved as STRATA in the last chapter. LOAD in the STRATA program and delete lines 180 and 190. Then add these lines:

```
130 LPRINT CHR$(27)"&"CHR$(0)"al";
140 FOR Y=1 TO 12: LPRINT CHR$(139);
170 LPRINT "a b c d e f g h i j k l"
```

Remember, if your computer cannot send lowercase letters, use the ASCII values for the letters you need (See Appendix A). Add:

```
899 ' <<< LINE GRAPHICS AND SHADING >>>
900 DATA 0,0,0,0,15,0,8,0,8,0,8: ' a
910 DATA 8,0,8,0,15,0,0,0,0,0,0: ' b
920 DATA 8,0,8,0,120,0,0,0,0,0,0: ' c
930 DATA 0,0,0,0,120,0,8,0,8,0,8: ' d
940 DATA 8,0,8,0,120,0,8,0,8,0,8: ' e
950 DATA 8,0,8,0,15,0,8,0,8,0,8: 'f
960 DATA 0,0,0,0,127,0,8,0,8,0,8: ' g
```

```
970 DATA 8,0,8,0,127,0,0,0,0,0,0: 'h
980 DATA 8,0,8,0,127,0,8,0,8,0,8:  ' i
990 DATA 8,0,8,0,8,0,8,0,8,0,8: ' j
1000 DATA 0,0,0,0,127,0,0,0,0,0,0: 'k
1010 DATA 84,0,170,0,84,0,170,0,84,0,170: ' l
```



You can put the line-graphics characters to work like this:

```
100 LPRINT CHR$(27)"1"
170 LPRINT "ajjjjjjfjjjjjjjb"
175 LPRINT "k NAME k PHONE k"
180 LPRINT "fgjjjjjjijjjjjjh"
185 LPRINT "kbbbbbbbkbbbbbbbbk"
190 LPRINT "djjjjjjejjjjjjjc"
```

RUN the program to produce pair of boxes as shown below.

NAME	PHONE

Note the the text lines use only uppercase letters since you've replaced the lowercase versions with your own characters.

Make sure you SAVE this program as LINE-but before you do, delete lines 170 to 1%.

Summary

In this chapter we've shown you how to combine user-defined characters horizontally, vertically, and both ways at once. We've also given you two core sets of characters, SIX and LINE.

Chapter 17
Business Applications

In this chapter we turn our attention to business applications. First we program a sample barchart. Then we use designs from previous chapters to develop a program that puts the FX through its paces.

The programs pull together many of the programming techniques that you've used in the course of this manual. The second one is considerably longer than the programs in previous chapters, and the length provides two benefits. First, you'll have a chance to really test your own understanding of the FX printer. Second, a longer program shows you how versatile your printer can be.

Preparation

Load the LINE program you saved at the end of Chapter 16 and delete lines 1100-1170. You will use the line graphics characters to print a sales chart in this chapter.

Barchart

This program creates the barchart shown as Figure 17-1. It uses the line graphics characters from the LINE program as well as three new user-defined characters. Change the following lines:

```
100 LPRINT CHR$(27)"3"CHR$(10)CHR$(27)"U1";
130 LPRINT CHR$(27)"&"CHR$(0)CHR$(94)CHR$(107);
140 FOR Y=1 TO 14: LPRINT CHR$(139);
160 NEXT Y: LPRINT CHR$(27)"C"CHR$(33);
```

Line 100 sets up line spacing and Unidirectional print. You print in Unidirectional Mode to be sure the line graphics characters line up precisely.

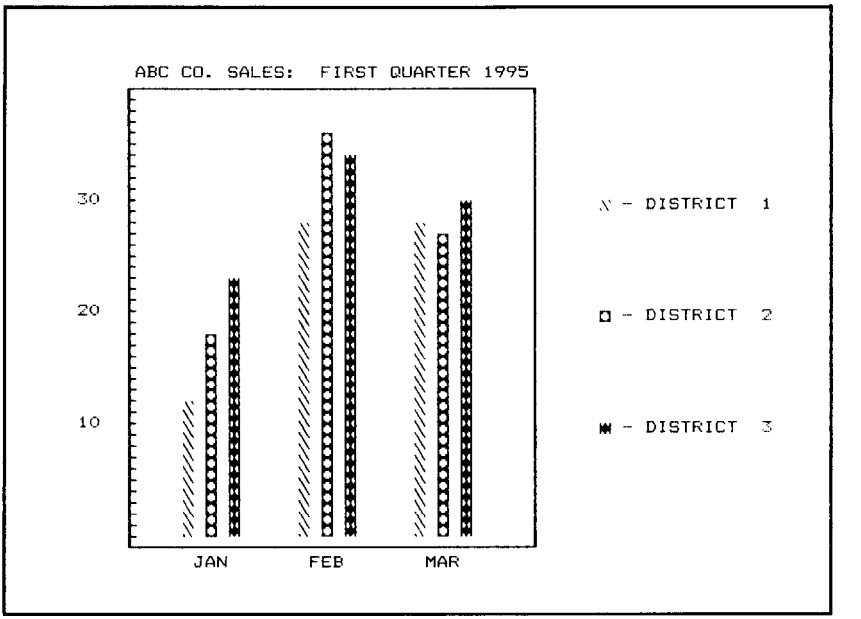


Figure 17-1. ***Barchart***

In line 130 the parameters for "&" specify that our 14 user-defined characters will be stored at positions 94 to 107 (ASCII symbols ^ to k).

Line 140 sets up a READ loop (Y) for the characters and sends the attribute byte—CHRS(139). Line 160 closes the loop and sets the form length to 33 lines so that, while you are setting the chart up, you can print two charts to a page. When you finish making experimental printouts, you will probably want to change this specification to 66 to conform to the usual page size. Check that you like the positioning of your paper since "C" also sets top of form to the current position of the print head. Now add these lines:

```
10 FOR J=1 TO 3: FOR K=1 TO 3
20     READ MAX(J,K)
30 NEXT K: NEXT J
40 DATA  12,18,23,28,36,34,28,27,30
```

Line 20 READs the maximum heights for the chart's vertical bars from line 40 and stores them in the array MAX. The next data lines define three new characters:

```
70  DATA  2,1,64,32,16,8,4,2,1,64,32
80  DATA  127,0,99,0,65,0,65,0,99,0,127
90  DATA  127,0,28,0,62,65,62,0,28,0,127
```

Since you will need to switch two features-line feeds and Emphasized Mode-on and off within the program, you can store their commands as shorter strings:

```
50 B$=CHR$(27)+"E"
60 D$=CHR$(27)+"J"+CHR$(11): C$=CHR$(27)+"F"
```

If your system won't send an 11, change both instances to either 10 or 12. The next 18 lines print the barchart.

```
190 LPRINT CHR$(27)"D"CHR$(14)CHR$(19)CHR$(24)
    CHR$(34)CHR$(44)CHR$(54);
200 LPRINT CHR$(60)CHR$(1);: H$=CHR$(137): Z=1
```

Lines 190 and 200 set horzontal tab stops and store the horizontal tab character in H$.

```
210 LPRINT H$;H$;"ø.ABC CO. SALES: FIRST QUARTER
    1995": LPRINT: LPRINT
220 LPRINT H$;H$;B$;"a";: N=34: A$="j": GOSUB 800:
    LPRINT "b"
799 ' *** STRING$ ROUTINE ***
800 FOR J=1 TO N: LPRINT A$;: NEXT J: RETURN
```

Lines 210 and 220 start the printing, using horizontal tabs and special characters. For systems without a STRING$ function, the GOSUB in line 220 can print a string of characters.

```
230 LPRINT H$;H$;"k";H$;H$;H$;H$;"k"
240 FOR R=39 TO 1 STEP -1: LPRINT H$;: F=0
250 IF R/10=INT(R/10) THEN LPRINT R;: F=1
260 LPRINT B$;H$;"g";D$;C$;
```

Line 250 prints the value of R every tenth line and sets a flag (F) to print DISTRICT

```
270 FOR M=1 TO 3: LPRINT H$;
280 FOR P=1 TO 3
290 IF R>MAX(M,P) THEN LPRINT  "øø"; ELSE LPRINT
    CHR$(93+P);  " ø";
```

Line 290 compares the current row (R) with the array (MAX) to determine whether to print a character or a blank space.

```
300 NEXT P: NEXT M: LPRINT B$;H$;"k";C$;: IF F=0
    THEN LPRINT: GOTO 320
310 LPRINT H$;CHR$(93+Z)"  ø- DISTRICTø ";Z: Z=Z+1
320 NEXT R: LPRINT H$;H$;B$;"g";H$;H$;H$;H$;"k"
325 LPRINT H$;H$;B$j"k";H$;H$;H$;H$;H$;"k"
```

```
330 LPRINT H$;H$;"d";: A$="j": GOSUB 800: LPRINT "c"
335 LPRINT: LPRINT
340 LPRINT C$;H$;H$;H$; "     ø.JAN"; H$; ø1"  FEB";H$;~~MAR":
    LPRINT
390 LPRINT CHR$(27)"@": END
```

Line 300 closes the P and M loops, prints the right-hand border, and sends control to either 310 or 320. Line 310 prints the districts. Lines 320 through 340 print the bottom portion of the chart.

Check your listing against the one shown as Figure 17-2. Now it's time to RUN your program and see if it looks like the printout near the beginning of this chapter.

You can change the heights of the bars by changing the data in line 40. Go ahead and SAVE the program as BARCHART if you like.

```
10 FOR J=1 TO 3: FOR K=1 TO 3
20 READ MAX (J,K)
30 NEXT K: NEXT J
40 DATA  12,18,23,28,36,34,28,27,30
50 B$=CHR$(27)+"E"
60  D$=CHR$(27)+"J"+CHR$(11):  C$=CHR$(27)+"F"
70  DATA  2,1,64,32,16,8,4,2,1,64,32
80  DATA  127,0,99,0,65,0,65,0,99,0,127
90  DATA  127,0,28,0,62,65,62,0,28,0,127
100 LPRINT CHR$(27) "3"CHR$(10) CHR$(27)"U1";
110 LPRINT CHR$(27) ":"CHR$(0)CHR$(0)CHR$(0);
120  LPRINT  CHR$(27)"%"CHR$(1)CHR$(0);
130  LPRINT  CHR$(27)"&WR$(0)CHR$(94)CHR$(107);
140 FOR Y=1 TO 14: LPRINT CHR$(139);
150 FOR X=1 TO 11: READ C: LPRINT CHR$(C);: NEXT X
160 NEXT Y: LPRINT CHR$(27)"C"CHR$(33);
190  LPRINT  CHR$(27)"D"CHR$(14)CHR$(19)CHR$(24)
    CHR$(34)CHR$(44)CHR(54);
200  LPRINT  CHR$(60)CHR$(1);: H$=CHR$(137): Z=1
210 LPRINT H$;H$; "ø/  ABC CO. SALES: FIRST QUARTER
    1995": LPRINT: LPRINT
220 LPRINT H$;H$;B$;"a";: N=34: A$="j": GOSUB 800:
    LPRINT "b"
230  LPRINT  H$;H$;"k";H$;H$jH$;H$;"k"
240 FOR R=39 TO 1 STEP -1: LPRINT H$;: F=0
250 IF R/10=INT(R/10) THEN LPRINT R;: F=1
260 LPRINT B$;H$;"g";D$;C$;
```

Figure 17-2. Program for BARCHART

```
270 FOR M=1 TO 3: LPRINT H$;
280 FOR P=1 TO 3
290 IF R>MAX(M,P) THEN LPRINT     "ØØ" ;ELSE LPRINT
    CHR$(93+P);  "Ø";
300 NEXT P: NEXT M: LPRINT B$;H$;"k";C$;: IF F=0
    THEN LPRINT: GOT0 320
310 LPRINT H$;CHR$(93+Z)"  Ø- DISTRICT 'Ø'";Z: Z=Z+1
320 NEXT R: LPRINT H$;H$;B$;"g";H$;H$;H$;H$;"k"
325 LPRINT H$;H$;B$;"k";H$;H$;H$;H$;"k"
330 LPRINT H$;H$j"d";: A$="j": GOSUB 800: LPRINT "c"
335 LPRINT: LPRINT
340 LPRINT C$;H$;H$;H$; "  'Ø JAN";H$; 'ØI FEB";H$; ØMAR":
    LPRINT
390 LPRINT CHR$(27)"@": END
799 ' *** STRING$ ROUTINE ***
800 FOR J=1 TO N: LPRINT A$;: NEXT J: RETURN
899 '  <<< LINE GRAPHICS AND SHADING >>>
900 DATA 0,0,0,0,15,0,8,0,8,0,8:'a
910 DATA 8,0,8,0,15,0,0,0,0,0,0: 'b
920 DATA 8,0,8,0,120,0,0,0,0,0,0: 'c
930 DATA 0,0,0,0,120,0,8,0,8,0,8: 'd
940 DATA 8,0,8,0,120,0,8,0,8,0,8: 'e
950 DATA 8,0,8,0,15,0,8,0,8,0,8: 'f
960 DATA 0,0,0,0,127,0,8,0,8,0,8: 'g
970 DATA 8,0,8,0,127,0,0,0,0,0,0: 'h
980 DATA 8,0,8,0,127,0,8,0,8,0,8: 'i
990 DATA 8,0,8,0,8,0,8,0,8,0,8: 'j
1000 DATA 0,0,0,0,127,0,0,0,0,0,0,: 'k
1010 DATA 84,0,170,0,84,0,170,0,84,0,170:'1
```

Figure 17-2. ***Program for BARCHART (concluded)***

Statement Form

For the last application of this manual, you will produce, for a hypothetical business-Strata Software, the statement form that is shown as Figure 17-3. Feel free to adapt the form or any of its elements to your own purposes.

Do you recognize figures from previous chapters within this form? We have you use the logo we developed in Chapter 12, user-defined characters from Chapter 1.5, and line-graphics characters from Chapter 16. By combining figures from previous chapters with new material, this large program demonstrates the way you can use several of the techniques you have learned within a single application.

STRATA SOFTWARE
80 TRACK DRIVE
DATA TOWN, U.S.A. 01248
PHONE FX1-0080

STATEMENT

ACCOUNT NO.	DATE

AMOUNT REMITTED

PLEASE DETACH AND RETURN WITH YOUR PAYMENT

DATE	INVOICE NO.	DESCRIPTION	CHARGES	PAYMENTS	BALANCE

CURRENT	30 DAYS	60 DAYS	90 DAYS	AMOUNT DUE

STRATA SOFTWARE

THANK YOU

Figure 17-3. Statement form

Since this program uses many of the routines from the BARCHART program above, begin by loading that program. Many of its lines need no changes, including:

```
110, 120, 150 and 799-1010
```

Delete lines 10 to 90. Make small changes to four lines:

```
100 DIM A(18): LPRINT
    CHR$(27)"3"CHR$(20)CHR$(27)"U1";
130 LPRINT CHR$(27)"&"CHR$(0) "at";
140 FOR Y=1 TO 20: LPRINT CHR$(139);
160 NEXT Y: LPRINT CHR$(27)"C"CHR$(66);: GOSUB 700
```

Lines 130 and 140 specify the number (20) and locations (ASCII a - t) of user-defined characters. Line 160 sets the form length to 66 lines and top of form to the current position of the print head, and sends the program to line 700, where the logo subroutine begins.

You can check your changes against the complete listing of STATEMENT that appears as Figure 17-4. That listing also gives you the lines that are new; such lines fall into two long sections:

```
170 to 770
1099 to 1370
```

Enter them one at a time, replacing the corresponding lines from the BARCHART program.

If your computer system requires a WIDTH statement to prevent the printer from issuing a carriage return before the graphics line is complete, add it now:

```
7 WIDTH LPRINT 255
```

The format for this statement may be different for your BASIC; see your software documentation.

If you are using an FX-100, add this line to set the right margin:

```
90 LPRINT CHR$(27)"QP";
```

```
7 WIDTH LPRINT 255
90 LPRINT CHR$(27)"QP";
100 DIM A(18): LPRINT
    CHR$(27)"3"CHR$(20)CHR$(27)"U1";
110 LPRINT CHR$(27)":"CHR$(0)CHR$(0)CHR$(0);
120 LPRINT CHR$(27)"%"CHR$(1)CHR$(0);
130 LPRINT CHR$(27)"&"CHR$(0)"at";
140 FOR Y=1 TO 20: LPRINT CHR$(139);
150 FOR X=1 TO 11: READ C: LPRINT CHR$(C);: NEXT X
160 NEXT Y: LPRINT CHR$(27)"C"CHR$(66);: GOSUB 700
170 LPRINT    CHR$(27)"!8";"ø¡ mpsrpr mnopqrst";
    CHR$(27)"!@";
180 LPRINT CHR$(27)"B"CHR$(18)CHR$(25)CHR$(1);
190 LPRINT CHR$(27)"D"CHR$(13)CHR$(17)CHR$(57)
    CHR$(69)CHR$(1);
200 H$=CHR$(137): LPRINT H$;CHR$(14) "ø¤STATEMENT"
210 GOSUB 700: LPRINT: GOSUB 700
220 LPRINT H$;H$;CHR$(27)"!A;"80 TRACK DRIVE":
    GOSUB 700
230 LPRINT: GOSUB 700: LPRINT H$;"DATA TOWN, U.S.A.
    01248": GOSUB 700
240 LPRINT: LPRINT H$;H$;CHR$(27)"!Q";
    "PHONE FX1-0080"
250 LPRINT CHR$(27)"!@";
260 LPRINT CHR$(27)"D"CHR$(57)CHR$(72)CHR$(1);
270 C=2: H=2: F=0: FT=1: GOSUB 500: LPRINT
280 LPRINT CHR$(11);H$;"&"CHR$(8);: A$=CHR$(95):
    N=21: GOSUB 800
290 LPRINT: LPRINT H$;CHR$(27)  "S1";øøø¸AMouNT
    REMITTED"
300 LPRINT CHR$(27)"D"CHR$(3)CHR$(13)CHR$(29)
    CHR$(46)CHR$(55)CHR$(68)CHR$(1);
310 LPRINT CHR$(11);H$;H$;CHR$(27)"S0";          "øøøøøøøPLEASE
    DETACH AND RETURN WITH YOUR PAYMENT"
320 LPRINT CHR$(27)"T";: N=80: A$="-": GOSUB 800:
    LPRINT
330 C=6: H=12: F=1: FT=0: GOSUB 500
340 LPRINT CHR$(27)"D"CHR$(8)CHR$(26)CHR$(38)
    CHR$(50)CHR$(65)CHR$(1);
350 C=5: H=2: F=0: GOSUB 500: LPRINT
360 LPRINT CHR$(27)"D"CHR$(8)CHR$(57)CHR$(0)
370 LPRINT CHR$(27)"!T";H$;"mpsrpr mnopqrst"H$;
380 LPRINT CHR$(27)"!1";CHR$(27)"4";"THANK YOU"
390 LPRINT CHR$(27)"@": END
```

Figure 17-4. ***Program* for *STATEMENT***

```
499 ' *** BOX SUBROUTINE ***
500 FOR K=1 TO 5: READ  L$(K),M$(K),N$(K),R$(K):
    NEXT K
510 FOR K=1 TO C: READ W(K): NEXT K
520 FOR L=1 TO 5: IF L=4 THEN FOR G=1 TO H
530   IF FT=1 THEN LPRINT H$;
540   LPRINT L$(L);: FOR K=1 TO C-1
550   FOR J=1 TO W(K): LPRINT M$(L);: NEXT J
560   LPRINT N$(L);: NEXT K
570   N=W(C): A$=M$(L): GOSUB 800: LPRINT R$(L);
580   IF L<>2 THEN 640
590  LPRINT  CHR$(27)"!H";CHR$(27)"A"CHR$(0)
600   FOR Q=1 TO C: READ T$: LPRINT H$;T$;: NEXT Q
610   LPRINT CHR$(27)"!@";
620   IF F=1 THEN LPRINT CHR$(27)"A"CHR$(0): N=80:
      A$="l" : GOSUB 800
640   IF L<5 THEN LPRINT CHR$(27)"1"
650   IF L=4 THEN NEXT G
660 NEXT L: RETURN
699 ' *** DRAW LOGO ***
700  LPRINT  CHR$(27)"L"CHR$(60)CHR$(0);
710 READ N: IF N=128 THEN 770
720 IF N>=0 THEN LPRINT CHR$(N);: GOT0 710
730 READ P,R: FOR J=1 TO -N: LPRINT CHR$(P)
    CHR$(R);: NEXT J
740 GOT0 710
770 RETURN
799 ' *** STRING$ ROUTINE ***
800 FOR J=1 TO N: LPRINT A$;: NEXT J: RETURN
899 ' <<< LINE GRAPHICS AND SHADING >>>
900 DATA 0,0,0,0,15,0,8,0,8,0,8: 'a
910 DATA 8,0,8,0,15,0,0,0,0,0,0: 'b
920 DATA 8,0,8,0,120,0,0,0,0,0,0: 'c
930 DATA 0,0,0,0,120,0,8,0,8,0,8: 'd
940 DATA 8,0,8,0,120,0,8,0,8,0,8: 'e
950  DATA 8,0,8,0,15,0,8,0,8,0,8: 'f
960 DATA 0,0,0,0,127,0,8,0,8,0,8: 'g
970  DATA 8,0,8,0,127,0,0,0,0,0,0 'h
980 DATA 8,0,8,0,127,0,8,0,8,0,8: 'i
990  DATA 8,0,8,0,8,0,8,0,8,0,8:  'j
1000 DATA 0,0,0,0,127,0,0,0,0,0,0: 'k:
1010 DATA 84,0,170,0,84,0,170,0,84,0,  170: 'l
1099 ' <<< STRATA SOFTWARE >>>
```

Figure 17-4. ***Program*** *for* ***STATEMENT (continued)***

```
1100 DATA 0,121,0,73,0,73,0,73,0,79,0: 'm - S
1110 DATA 0,127,0,65,0,65,0,65,0,127,0: 'n - 0
1120 DATA 0,63,64,8,64,8,64,28,64,32,0: 'o - F
1130 DATA 0,32,64,0,64,63,64,0,64,32,0: 'p - T
1140 DATA 0,126,1,2,4,8,4,2,1,126,0: 'q - W
1150 DATA 0,7,8,16,36,64,36,16,8,7,0: 'r - A
1160 DATA 0,127,0,72,0,72,0,76,2,121,0: 's - R
1170 DATA 0,62,65,8,65,8,65,28,65,34,0: 't - E
1199 ' <<< LOGO DATA >>>
1200 DATA 0,1,2,4,11,18,36,72,-16,16,64,8,
     64,8,32,16,0,-7,0,0,128
1210 DATA 0,126,1,0,126,1,-5,0,0,1,2,4,11,18,36,-16
     8,32,4,32,4,16,8,0,128
1220 DATA 0,0,0,64,32,16,72,36,-3,16,4,34,
     65,0,0,65,34,-8,16,4,18,11,4,2,1,0,-9,0,0,128
1230 DATA -8,0,0,64,32,16,72,36,16,-7,4,16,36,
     67,0,0,1,66,36,-4,16,4,18,11,4,2,1,-2,0,
     0,128
1240 DATA 0,32,16,64,8,64,-15,8,32,72,16,32,64,-6,
     0,0,0,127,0,0,127,0,0,0,128
1250 DATA -7,0,0,0,8,4,16,2,16,-15,2,
     8,18,36,72,16,32,64,-2,g,0,128
1299 ' <<< BOX DATA >>>
1300 DATA a,j,f,b,k,  "ø",k,k,g,j,i,h,k,  "ø",k,k,
     d,j,e,c
1310 DATA 11,8,"ø ACCOUNT NO.","DATE"
1320 DATA j,j,f,j,  "ø","ø" k,"ø ",j,j,i,j,
     "ø","ø" ,k,"ø" ,j,j,e,j
1330 DATA 11,11,20,8,8,15,"DATE","INVOICE NO."
1340 DATA "DESCRIPTION", "CHARGES",
     "PAYMENTS","BALANCE"
1350 DATA j,j,f,j,  "ø","ø",k,"ø", j,j,i,j, "ø","ø", k,
     "ø",j,j,e,j
1360 DATA 22,11,11,12,18
1370 DATA "CURRENT", "30 DAYS", "60 DAYS", "90
     DAYS", "AMOUNT DUE"
```

Figure 174. ***Program for STATEMENT (concluded)***

You may prefer to work out what each line does on your own—three cheers if you do. But if you want a little guidance, here is a brief program overview, followed by a line-by-line description of the main portion of the program.

STATEMENT breaks down into several large blocks of routines and data:

Lines	*Routine*
100-150	Defines characters and does housekeeping
160-390	Prints the statement form
500-660	The box subroutine
700-770	The logo subroutine
800	The STRING$ subroutine
900-1010	Data for line graphics
1100-1170	Data for the STRATA SOFTWARE letters
1200-1250	Data for the logo
1300-1370	Data for the box routine

Line 170 prints STRATA SOFTWARE using Master Select to define **the** mode.
Lines 180 and 190 set vertical and horizontal tab stops.
Line 200 stores the tab command in H$ and prints STATEMENT in Expanded print.
Line 210 prints the second and third lines of the logo.
Lines 220-250 print more of the logo and the address in various mode combinations.
Line 260 sets new horizontal tab stops.
Line 270 sets some variables for the box subroutine at 500. That routine prints the box in the upper-right comer of the sheet.
Line 280 does a vertical tab, then prints a string of 21 underline characters (ASCII 95) via subroutine 800.
Line 290 prints **the** Subscript AMOUNT REMITTED.
Line 300 sets new horizontal tab stops.
Line 310 tabs vertically twice, then prints a Superscript message.
Line 320 cancels Scripts and prints 80 hyphens with subroutine 800.
Line 330 calls the box routine.
Line 340 sets new horizontal tabs.
Line 350 calls the box routine.
Line 360 ses new horizontal tabs.
Line 370 prints STRATA SOFTWARE in a different print mode.
Line 380 thanks us in Italic characters.
Line 390 resets all modes and ends the program.

Deciphering the box routine in lines 500 through 660 is left as an exercise for you. The following hints will get you started.

The subroutine at 500 is used to create three boxes of different sizes and characteristics, using the line-graphics characters. The data stored in lines 1300 to 1370 determine which line-graphics characters are used to print the boxes, the width of each cell, and the headings.

The variables sent to **the** subroutine are:

C the number of cells
H the height of the cells
F a flag for shading the headings
FT **a flag that** allows a horizontal tab to adjust the left margin of the box.

999 REM: The End

In this chapter we developed a program that uses many of the features of the FX printer. The program demonstrates the tremendous potential of **the** powerful tool you have at your beck and call. We hope it inspires you to use the FX printer in many creative ways to enhance your own programming applications.

INDEX

Note: Refer to Table of Contents and List of Figures for specific programs. Also, the chapter summaries are not indexed.

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V